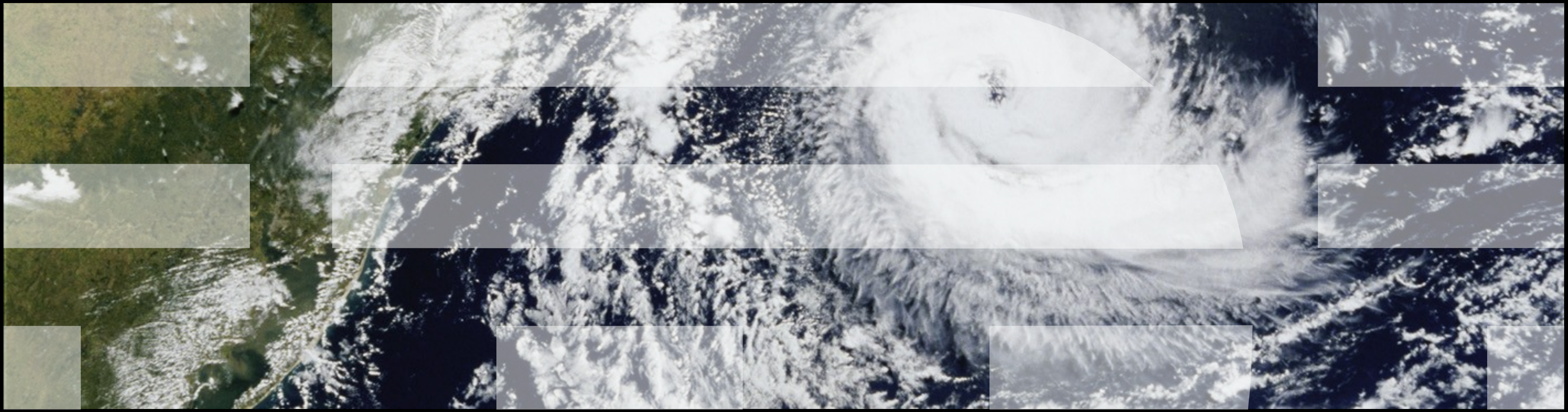
Paul E. McKenney, IBM Distinguished Engineer, Linux Technology Center (Linaro)

29 August 2012

Real-Time Response on Multicore Systems: *It is Bigger Than You Think*



2012 Linux Plumbers Conference, Scaling Microconference

The -rt Patchset Was Used in Production Early On

- 2006: aggressive real-time on 64-bit systems
  - Real-time Linux kernel (x86_64, 4-8 processors, deadlines down to 70 microseconds, measured latencies less than 40 microseconds)
    - I only did RCU. Ingo Molnar, Sven Dietrich, K. R. Foley, Thomas Gleixner, Gene Heskett, Bill Huey, Esben Nielsen, Nick Piggin, Lee Revell, Steven Rostedt, Michal Schmidt, Daniel Walker, and Karsten Wiese did the real work, as did many others joining the project later on.
    - Plus a huge number of people writing applications, supporting customers, packaging distros, …
- But some were not inclined to believe it, so...

The Writeup

“SMP and Embedded Real Time”

- Five Real-Time Myths:
  - Embedded systems are always uniprocessor systems
  - Parallel programming is mind crushingly difficult
  - Real time must be either hard or soft
  - Parallel real-time programming is impossibly difficult
  - There is no connection between real-time and enterprise systems

Source: Paul E. McKenney “SMP and Embedded Real Time”, Linux Journal, Feb 2007, http://www.linuxjournal.com/article/9361

“SMP and Embedded Real Time”

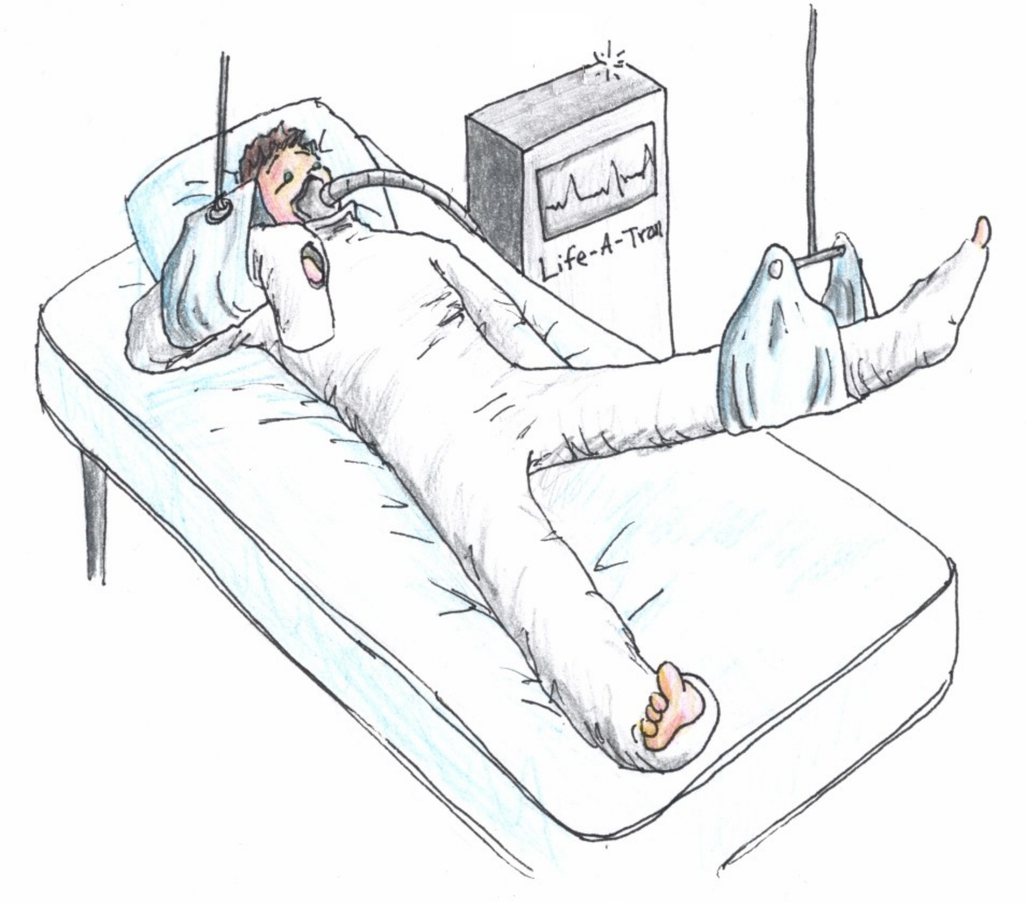
- Five Real-Time Myths:
  - Embedded systems are always uniprocessor systems
  - Parallel programming is mind crushingly difficult
  - Real time must be either hard or soft
  - Parallel real-time programming is impossibly difficult
  - There is no connection between real-time and enterprise systems
- This message was not well-received in all quarters
  - Despite cute cartoons...

Source: Paul E. McKenney “SMP and Embedded Real Time”, Linux Journal, Feb 2007, http://www.linuxjournal.com/article/9361

The Limits of Hard Real Time in the Hard Real World

You show me a hard real-time system,
and I will show you a hammer that will cause it to miss its deadlines.

But Do Hardware Failures Count?



Rest assured, sir, that should there be a failure,
it will not be due to software!

I Believe That “SMP and Embedded Real Time” Has Stood the Test of Time

However, I Did Make One Big Error in
“SMP and Embedded Real Time”

Large Error in “SMP and Embedded Real Time”

- February 8, 2012
  - Dimitri Sivanic reports 200+ microsecond latency spikes from RCU
  - My initial response, based on lots of experience otherwise:
    - “You must be joking!!!”

Large Error in “SMP and Embedded Real Time”

- February 8, 2012
  - –Dimitri Sivanic reports 200+ microsecond latency spikes from RCU
  - –My initial response, based on lots of experience otherwise:
    - “You must be joking!!!”
  - –Further down in Dimitri's email: NR_CPUS=4096

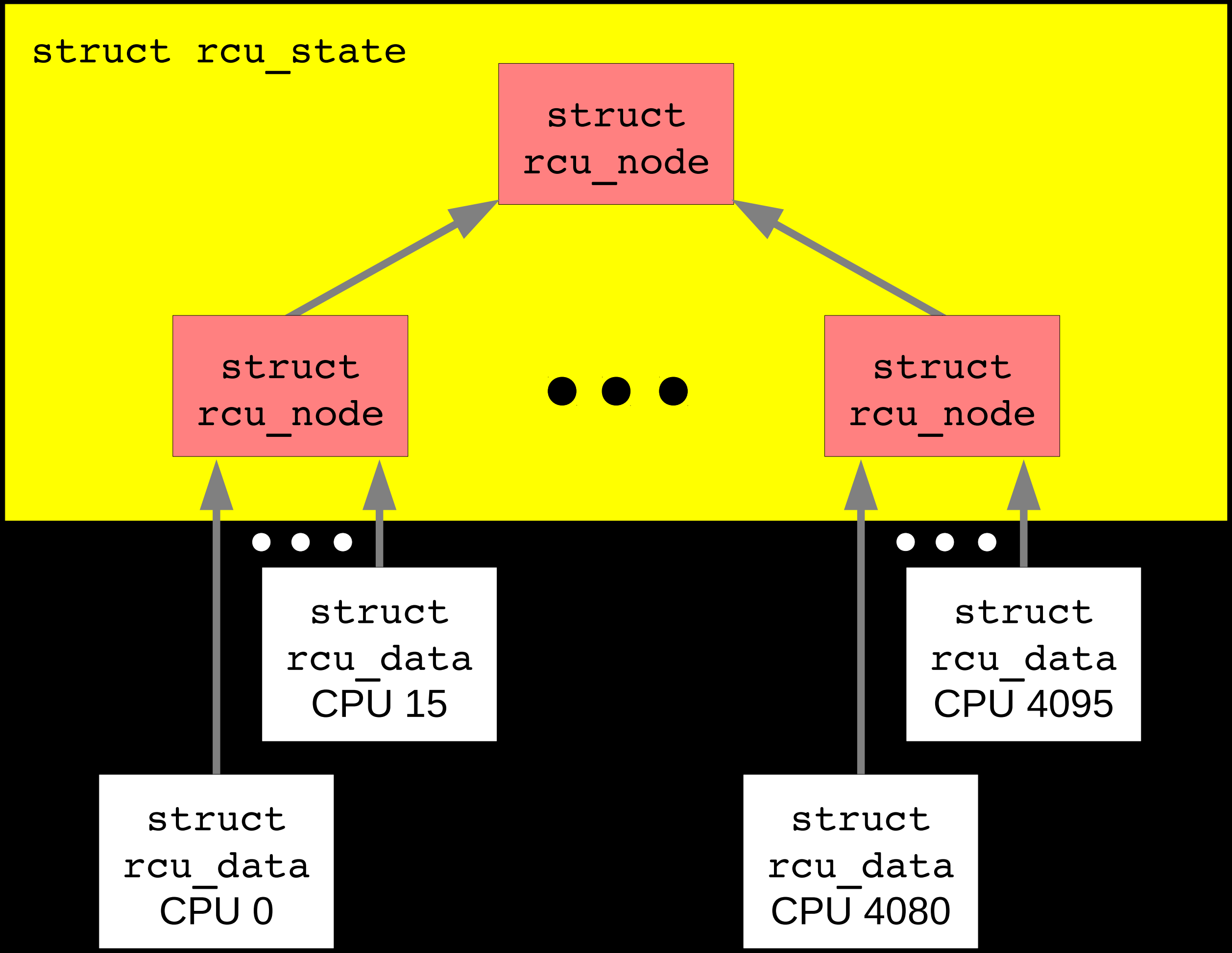
Large Error in “SMP and Embedded Real Time”

- February 8, 2012
  - Dimitri Sivanic reports 200+ microsecond latency spikes from RCU
  - My initial response, based on lots of experience otherwise:
    - “You must be joking!!!”
  - Further down in Dimitri's email: NR_CPUS=4096
    - “You mean it took only 200 microseconds?”

Large Error in “SMP and Embedded Real Time”

- February 8, 2012
  - Dimitri Sivanic reports 200+ microsecond latency spikes from RCU
  - My initial response, based on lots of experience otherwise:
    - “You must be joking!!!”
  - Further down in Dimitri's email: NR_CPUS=4096
    - “You mean it took only 200 microseconds?”
- The large error: I was thinking in terms of 4-8 CPUs, maybe eventually as many as 16-32 CPUs
  - More than two orders of magnitude too small!!!

RCU Initialization



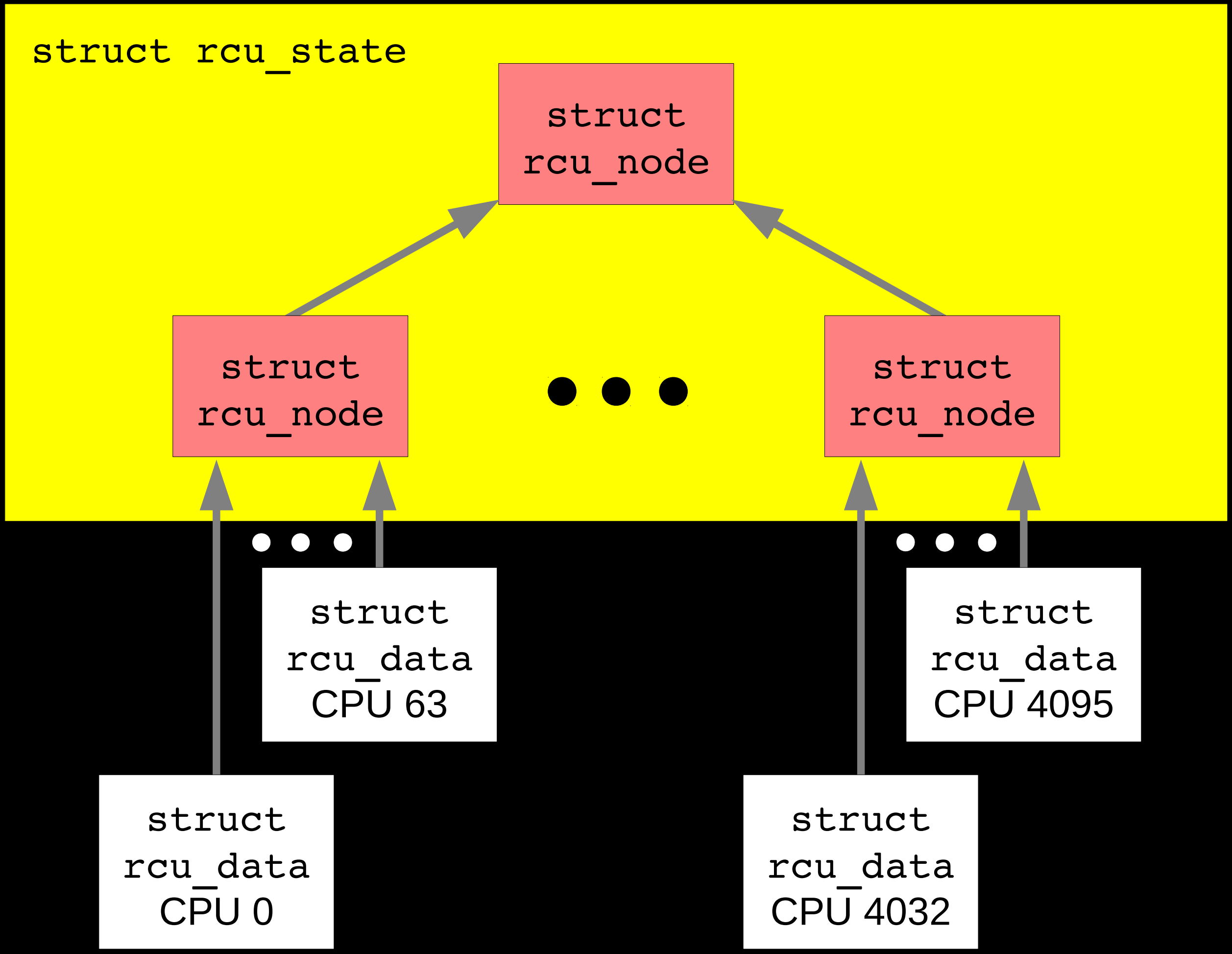
Level 0: 1 rcu_node

Level 1: 4 rcu_nodes

Level 2: 256 rcu_nodes

Total: 261 rcu_nodes

RCU Initialization, CONFIG_RCU_FANOUT=64



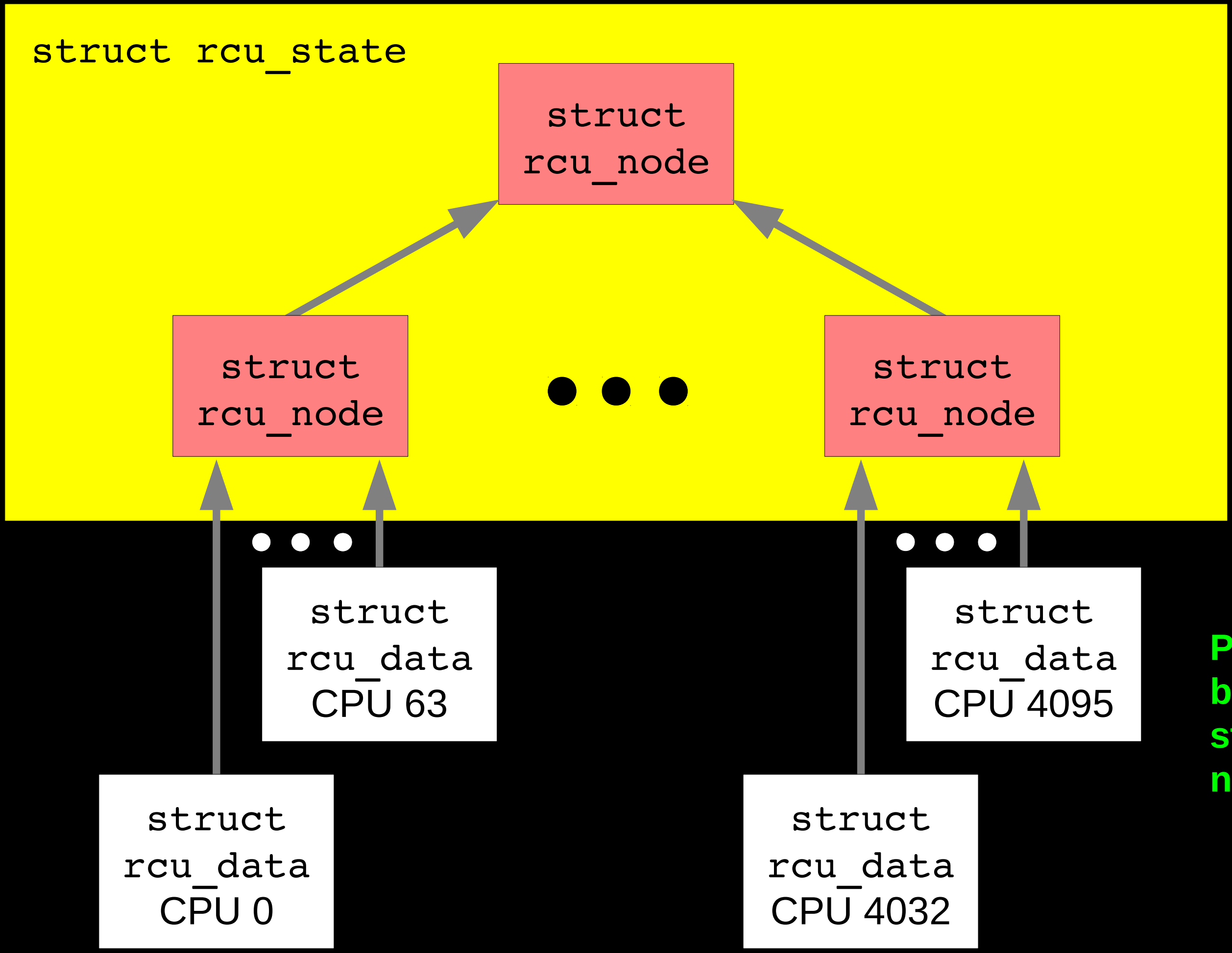
Level 0: 1 rcu_node

Level 2: 64 rcu_nodes

Total: 65 rcu_nodes

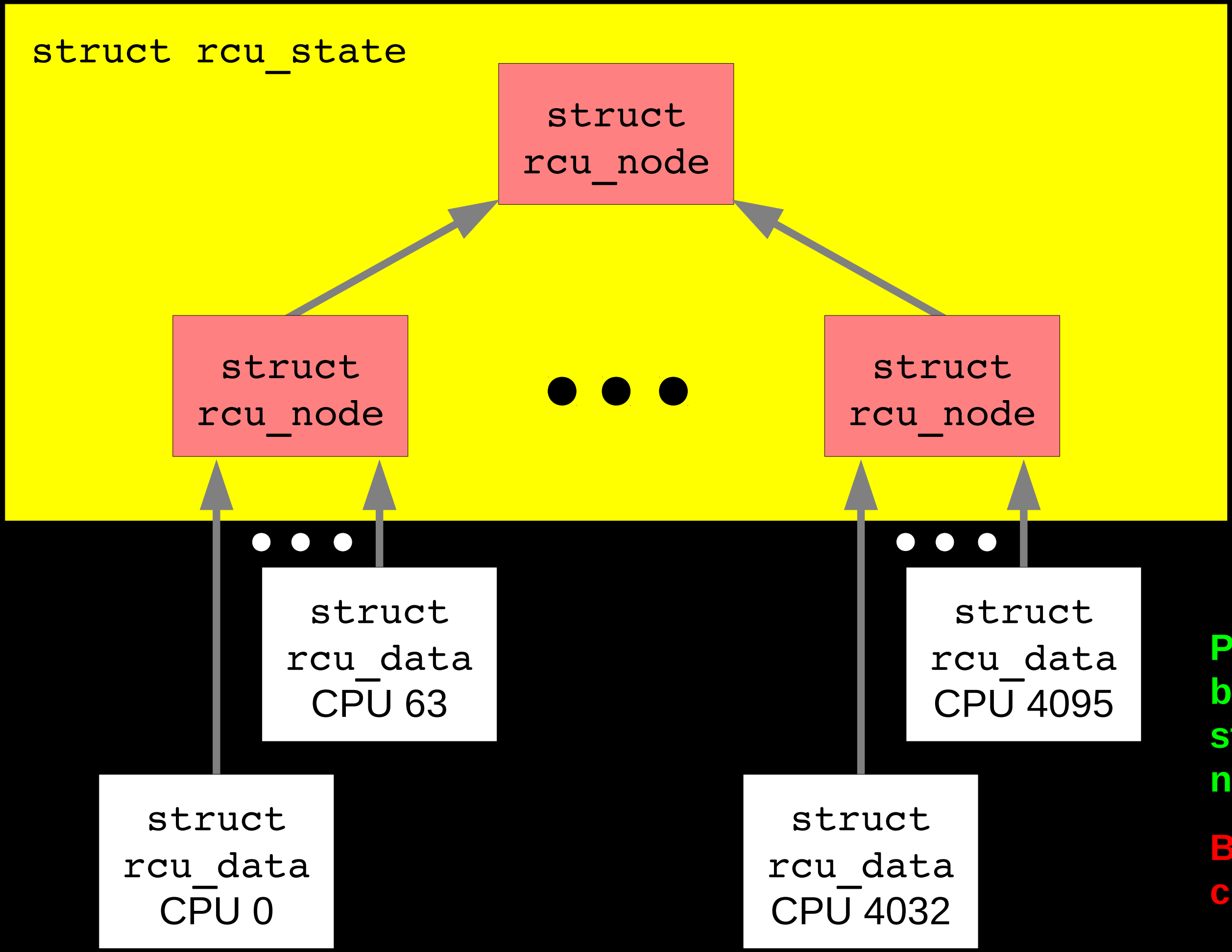
Decreases latency from 200+ to 60-70 microseconds. “Barely acceptable” to users. But we can do better...

Move Grace-Period Initialization Into a kthread



Preemption opportunity between each rcu_node structure's initialization, negligible latency

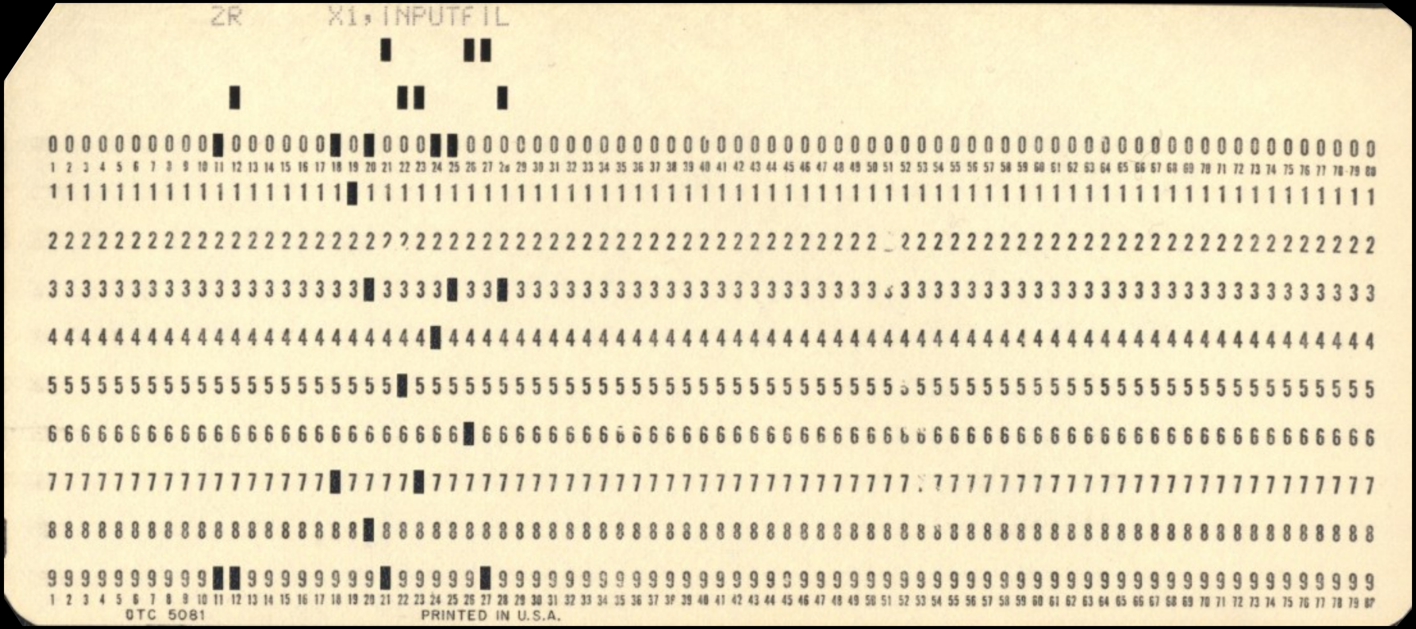
Move Grace-Period Initialization Into a kthread



Preemption opportunity between each rcu_node structure's initialization, negligible latency

But this represents a large change, so validating...

Coping With 4096-CPU System Scarcity



Other Possible Issues

Other Possible Issues

- The synchronize_*_expedited() primitives loop over all CPUs
  - Parallelize? Optimize for dyntick-idle state?
- The rcu_barrier() primitives loop over all CPUs
  - Parallelize? Avoid running on other CPUs?
- Should force_quiescent_state() make use of state in non-leaf rcu_node structures to limit scan?
  - This actually degrades worst-case behavior
- Grace-period initialization and cleanup loops over all rcu_node structures
  - Parallelize?
- NR_CPUS=4096 on small systems (RCU handles at boot)
- Interactions with scheduler (remember 3.0?)

Conclusions

Conclusions

- They say that the best way to predict the future is to invent it

Conclusions

- They say that the best way to predict the future is to invent it
  - I am here to tell you that even this method is not foolproof

Conclusions

- They say that the best way to predict the future is to invent it
  - I am here to tell you that even this method is not foolproof
- SMP, real time, and energy efficiency are each well known
  - The real opportunities for new work involve combinations of them
- Some need for 10s-of-microseconds latency on 4096 CPUs
  - Translates to mainstream need on tens or hundreds of CPUs
    - Supporting this is not impossible

Conclusions

- They say that the best way to predict the future is to invent it
  - I am here to tell you that even this method is not foolproof
- SMP, real time, and energy efficiency are each well known
  - The real opportunities for new work involve combinations of them
- Some need for 10s-of-microseconds latency on 4096 CPUs
  - Translates to mainstream need on tens or hundreds of CPUs
    - Supporting this is not impossible
- There is still much work to be done on the Linux kernel
  - But even more work required for open-source applications
- The major large-system challenges are at the design level

Conclusions

- They say that the best way to predict the future is to invent it
  - I am here to tell you that even this method is not foolproof
- SMP, real time, and energy efficiency are each well known
  - The real opportunities for new work involve combinations of them
- Some need for 10s-of-microseconds latency on 4096 CPUs
  - Translates to mainstream need on tens or hundreds of CPUs
    - Supporting this is not impossible
- There is still much work to be done on the Linux kernel
  - But even more work required for open-source applications
- The major large-system challenges are at the design level
  - Pity that design issues receive little emphasis in the CS curriculum!!!

Legal Statement

- This work represents the view of the author and does not necessarily represent the view of IBM.
- IBM and IBM (logo) are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product, and service names may be trademarks or service marks of others.

Questions?

For more information:
http://www2.rdrop.com/users/paulmck/realtime/paper/bigrt.2012.07.10a.pdf

Backup

About That Single Global Lock...

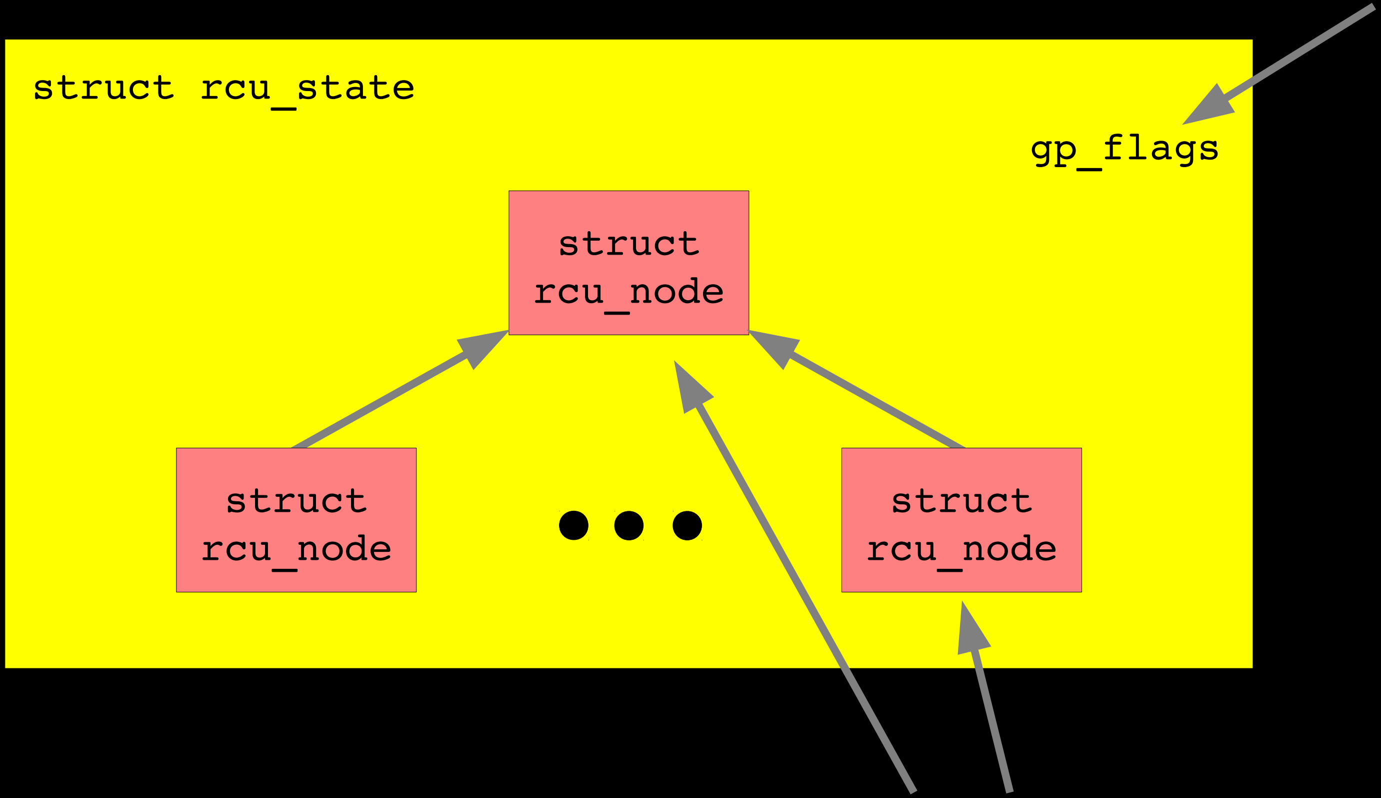
About That Single Global Lock...

- Grace-period operations are global events
  - So if already running or being awakened, no action required
- This situation can be handled by a variation on a tournament lock (Graunke & Thakkar 1990)

About That Single Global Lock...

- Grace-period operations are global events
  - So if already running or being awakened, no action required
- This situation can be handled by a variation on a tournament lock (Graunke & Thakkar 1990)
  - A variation that does not share the poor performance noted by Graunke and Thakkar

Conditional Tournament Lock



Checked at each level

spin_trylock() at each level, release at next level

Conditional Tournament Lock Code

```
rnp = per_cpu_ptr(rsp->rda, raw_smp_processor_id())->mynode;
for (; rnp != NULL; rnp = rnp->parent) {
  ret = (ACCESS_ONCE(rsp->gp_flags) & RCU_GP_FLAG_FQS) ||
        !raw_spin_trylock(&rnp->fqslock);
  if (rnp_old != NULL)
    raw_spin_unlock(&rnp_old->fqslock);
  if (ret) {
    rsp->n_force_qs_lh++;
    return;
  }
  rnp_old = rnp;
}
```

Conditional Tournament Lock Code

```
rnp = per_cpu_ptr(rsp->rda, raw_smp_processor_id())->mynode;
for (; rnp != NULL; rnp = rnp->parent) {
  ret = (ACCESS_ONCE(rsp->gp_flags) & RCU_GP_FLAG_FQS) ||
        !raw_spin_trylock(&rnp->fqslock);
  if (rnp_old != NULL)
    raw_spin_unlock(&rnp_old->fqslock);
  if (ret) {
    rsp->n_force_qs_lh++;
    return;
  }
  rnp_old = rnp;
}
```

Effectiveness TBD