Kernel Support for 4096 Processors

Mike Travis <travis@sgi.com>

SiliconGraphics

Disclaimer

- This presentation contains forward-looking statements regarding the SGI Altix® server family and roadmap, other SGI® technologies, and third-party technologies that are subject to risks and uncertainties. These risks and uncertainties could cause actual results to differ materially from those described in such statements. The viewer is cautioned not to rely unduly on these forward-looking statements, which are not a guarantee of future or current performance. Such risks and uncertainties include long-term program commitments, the performance of third parties, the sustained performance of current and future products, financing risks, the impact of competitive markets, the ability to integrate and support a complex technology solution involving multiple providers and users, the acceptance of applicable technologies by markets and customers, and other risks. These forward-looking statements are subject to risks and uncertainties as set forth in the company's most recent SEC reports on Form 10-Q, 8K and Form 10-K. Silicon Graphics is under no obligation to publicly update or revise any forward-looking statements, whether to reflect new information, future events or otherwise.
-
-
-
- Product plans, descriptions, and dates are estimates only and subject to change without notice. SGI may choose not to announce or make generally available any products or programs discussed in this presentation. Therefore, you should not make changes in your business operations on the basis of the information presented here.
- Carlsbad, Dixon, Ultraviolet, Santa Fe, Oro Valley, Chama and Taos are internal project code names.
-

SiliconGraphics

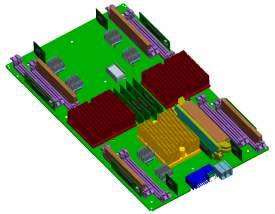
Overview

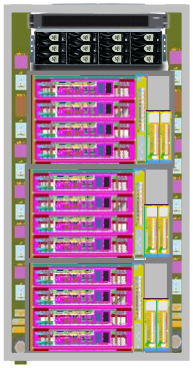
- Why 4096 cpus?: sneak preview of “UV”
- Supporting UV in Linux Kernel
- Stack Overflow: early work
- New cpumask API
- Future considerations
- Reducing wasted static memory
- Questions

SGI® Project Ultraviolet Architecture Highlights

- **Scalability**
  - 4096 cpu threads in single x86-64 system
  - Memory to 16TB in one OS image
  - Direct user access to 4PB of memory using the GRU
  - Aggregate IO per system to over 1 TB/s
- **Hardware-enabled Performance and Reliability**
  - Advanced fault detection, prevention, containment
  - Enhanced monitoring and serviceability
  - Special features accelerate compute or data-intensive workloads
- **Low TCO (total cost of ownership)**
  - x86 64 and Linux economics
  - Industry-leading rack-level energy efficiency
  - Easy to deploy, develop, administer and productively use

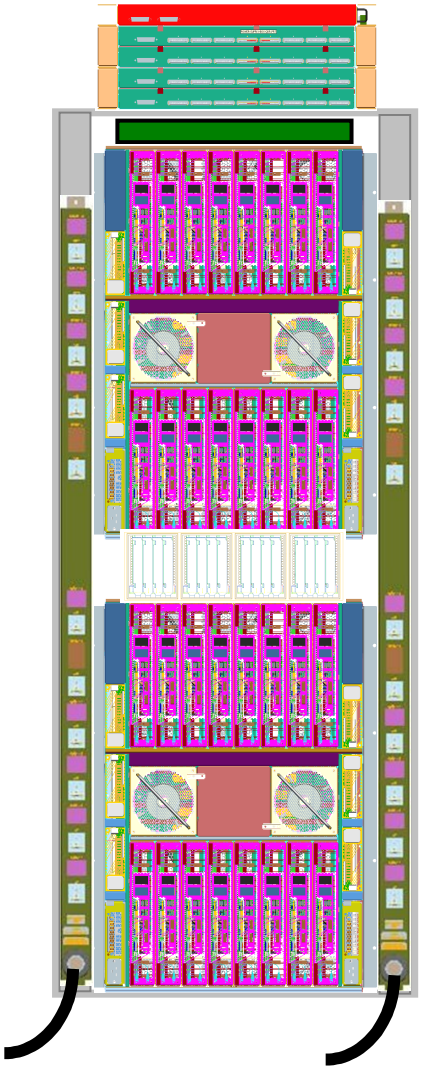
UV Technologies



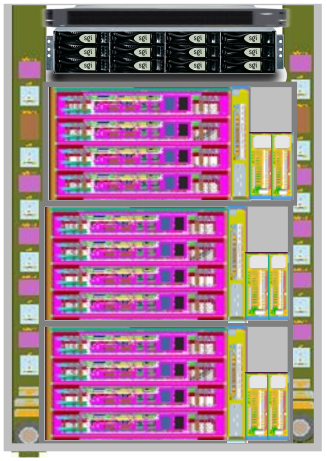


- UV Node Controller
  - Supports data-efficient scale, reliability and performance
- UV NUMAlink® 5 Interconnect
  - Very high BW/low latency system fabric
- SGI's Advanced Blade Packaging
  - SMP in a blade form factor
  - Power efficient and flexible
  - Advanced water cooling solutions available for large systems

Project Ultraviolet Product Design



- Bladed Node Package
- SMP Architecture
- Many configuration Options
- Unmatched Scalability



Project Ultraviolet: Architecture Limits

- Blade
  - 2 CPU Sockets (16 cores, 32 threads, 128GB w/8GB DIMMS)
- IRU
  - 16 Blades (32s, 256c, 512t, 2TB)
- Rack
  - 2 IRU's (64s, 512c, 1024t, 4TB)
- SSI System
  - 128 Blades (256s, 2048c, 4096t, 16TB [current X86_64 limit])
- System (NUMAlinked partitions)
  - 1024 Blades (2048s, 16384c, 32768t, 8PB)

Supporting UV in Linux Kernel

- Large installations require “standardized” linux distros that are verified for both quality and security. Most will not rebuild the kernel using a special configuration.
- This requires CONFIG_NR_CPUS = 4096 in the distro build even for 2 cpu systems.
- How to support this large configuration while minimizing the impact to small systems.

Minimizing Impact of NR_CPUS=4096

Largest Resource Hogs:

- cpumask_t's on the stack
  - cpumask_t arguments passed to functions
  - temporary cpumask_t variables
  - most annoying because this causes stack overflow panics

- Static arrays sized by “NR_CPUS”
- Static structures with embedded cpumask_t's
  - consumes static memory that's not being used

Stack overflow: the early days

- Add alternate functions that take cpumask_t pointers.
  Ex: set_cpus_allowed → set_cpus_allowed_ptr
- cpumask_of_cpu() heavily used, replaced with cpumask_of_cpu_map[]
- Add functions that removed need for temporary cpumask_t variables.
  Ex: cpu_mask_to_apicid_and() in struct apic
- Add dynamically allocated cpumask_t's
  Ex: build_sched_domain() had 7 temp masks!

Stack Overflow: the reality

- But... code contributions from others resulted in cpumask_t's being put back on the stack.
- This resulted in what Rusty called the whole process as the “whack a mole” approach. :-)
- “There must be a better way”...

Solution: New cpumask API

- Introduce parallel cpumask API to eventually remove the “cpumask_t” typedef.
- Always reference cpumasks via pointers (const pointer when possible.)
- Provide facility to dynamically allocate cpumasks when NR_CPUS is large, and declare static cpumasks when NR_CPUS is small.

New cpumask API: basic philosophy

- Use “struct cpumask” instead of “cpumask_t”
- Explicitly pass pointers to cpumasks (const wherever possible.)
  - cpus_or → cpumask_or, etc.
- Limit cpumask traversal to “nr_cpu_ids”
  - for_each_cpu_mask() → for_each_cpu()
    - Afterwards >= nr_cpu_ids not == NR_CPUS!

New cpumask API: temporary cpumasks

- cpumask_var_t:
  - struct cpumask[1], OR
  - struct cpumask * (`CPUMASK_OFFSTACK`)
- alloc_cpumask_var(&var, gfp)
  - Nop unless `CPUMASK_OFFSTACK`
- Only use for decls. Always pass (const) struct cpumask *.

New cpumask API: helper functions

- Pointer (often const!) versions of old friends:
  - cpumask_of_cpu() → cpumask_of()
  - cpu_online_map → cpu_online_mask
- Accessors for now-const masks:
  - set_cpu_possible(int, bool)
  - init_cpu_possible(const struct cpumask *)
- New convenience iterators:
  - cpumask_any_and(a,b)
  - cpumask_any_but(a, cpu)
  - for_each_cpu_and(a, b)
- work_on_cpu() instead of mugging current->cpus_allowed

New cpumask API: Future Considerations

- 2.6.30 aim is to remove all cpumasks and old operators from everywhere on x86.
- alloc_cpumask_var will only allocate nr_cpu_ids bits, not NR_CPUS
  - Means you can't use old iterators on them!
  - (Currently zeroes extra bits to be sure).
- struct cpumask will be undefined if `CONFIG_CPUMASK_OFFSTACK`=n
  - Stops obvious on-stack declaration
  - Stops assignment
  - Requires complete cpumask_var_t conversion.

Reducing wasted static memory

- Remove arrays sized by “NR_CPUS”
  - Use per_cpu variables where possible.
  - Use alloc_percpu which only allocates enough memory for possible cpus.
  - Build static array in .init.data section and copy to a dynamically allocated array based on “nr_cpu_ids”.
  - Dynamically allocate IRQ's based on need.

Questions?

Thanks!

- Many thanks to Rusty Russell for his help and guidance during this cpumask development (and letting me “cut and paste” from his LCA slides!)

  More history of the evolution of the new cpumask API can be found at:

  – http://ozlabs.org/~rusty/index.cgi/tech/2009-01-07
- Thanks to Christoph Lameter for his help in getting me “up to speed” in interacting with the Linux kernel community.
- And finally thanks to Ingo, Yinghai, Andrew and others for their critical reviews and often very clever solutions.