

Silicon Graphics, Inc.



Scaling Linux® to 512 Processors and Beyond

John Hawkes (hawkes@sgi.com)

Linux Software Engineering

April 2006



Overview

- SGI® Altix® hardware overview
- What does **scaling** really mean?
- How has Linux® evolved to support many hundreds of processors and terrabytes of main memory?
- Selected Benchmark Results
- Where can we make future improvements?

SGI® Altix®: Overview

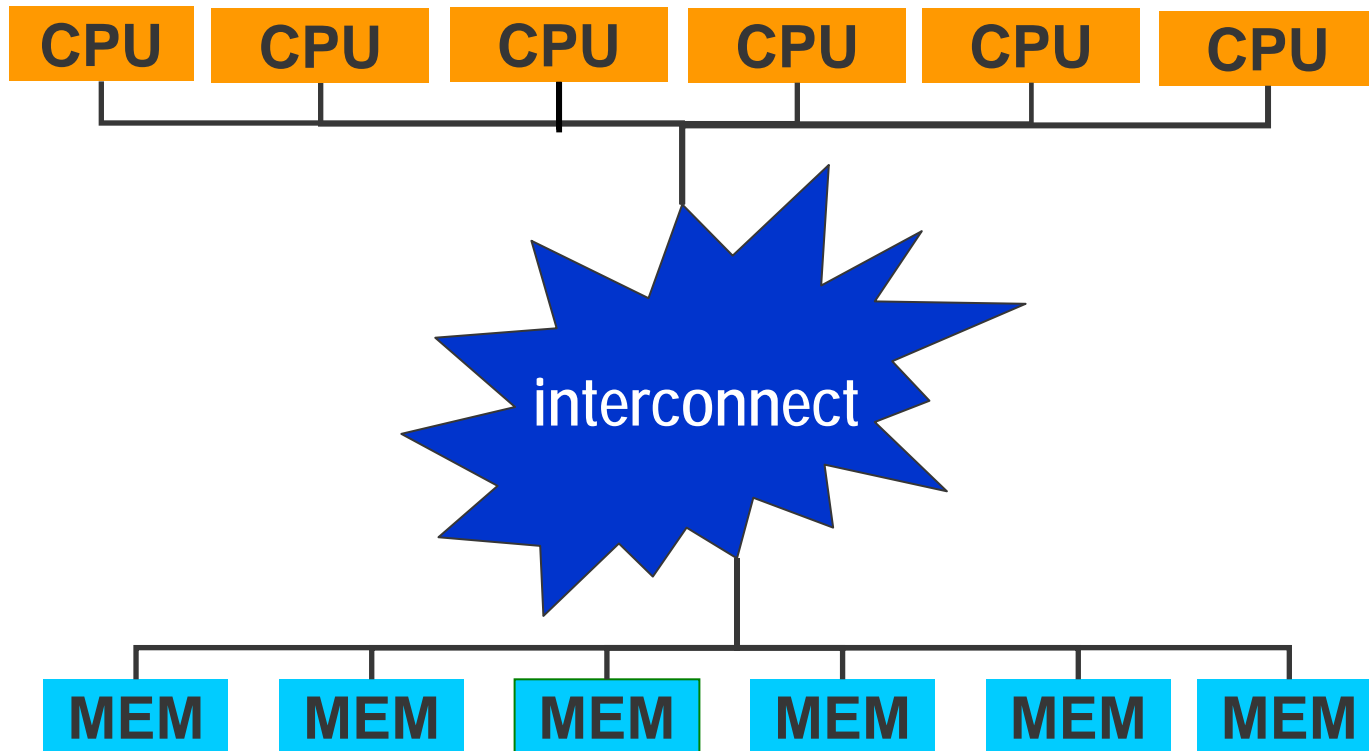
- The SGI® Altix® server family introduced in January 2003
 - Up to 64 processors and 512GB of main memory in a Single System Image (SSI)
 - SGI ProPack® distribution with customized Linux® 2.4.19-based kernel
 - Performance and scaling improvements to Altix over time
 - Improvements to Intel® Itanium® 2 processors, SGI hardware, denser memory
 - Single System Images up to 512 processors and 4TB of main memory
 - And clusters of these systems, such as NASA's 20x512p Columbia supercluster
 - SGI ProPack® 2.4.19-based kernel evolves to 2.4.21-based kernel
 - includes selective backports from 2.5.x and 2.6.x
 - Additionally supported by SuSE® SLES8® standard distribution
 - Today, Altix supported by native 2.6.x
 - and supported by SuSE® SLES9® and Red Hat® RHEL® standard distributions
 - Substantially larger configurations are on the horizon
-

SGI® Altix® Hardware: Overview

- NUMA (non-uniform memory access) machine
 - globally addressable and globally shared memory
- SGI® NUMAflex™ interconnect
 - same architecture as SGI® Origin® 3000
 - high bandwidth, low latency
- Intel® Itanium® 2 processors
- Targeted at HPC applications
 - although also performs and scales well on non-HPC workloads, e.g., file servers, graphics (albeit in smaller configurations than discussed here)

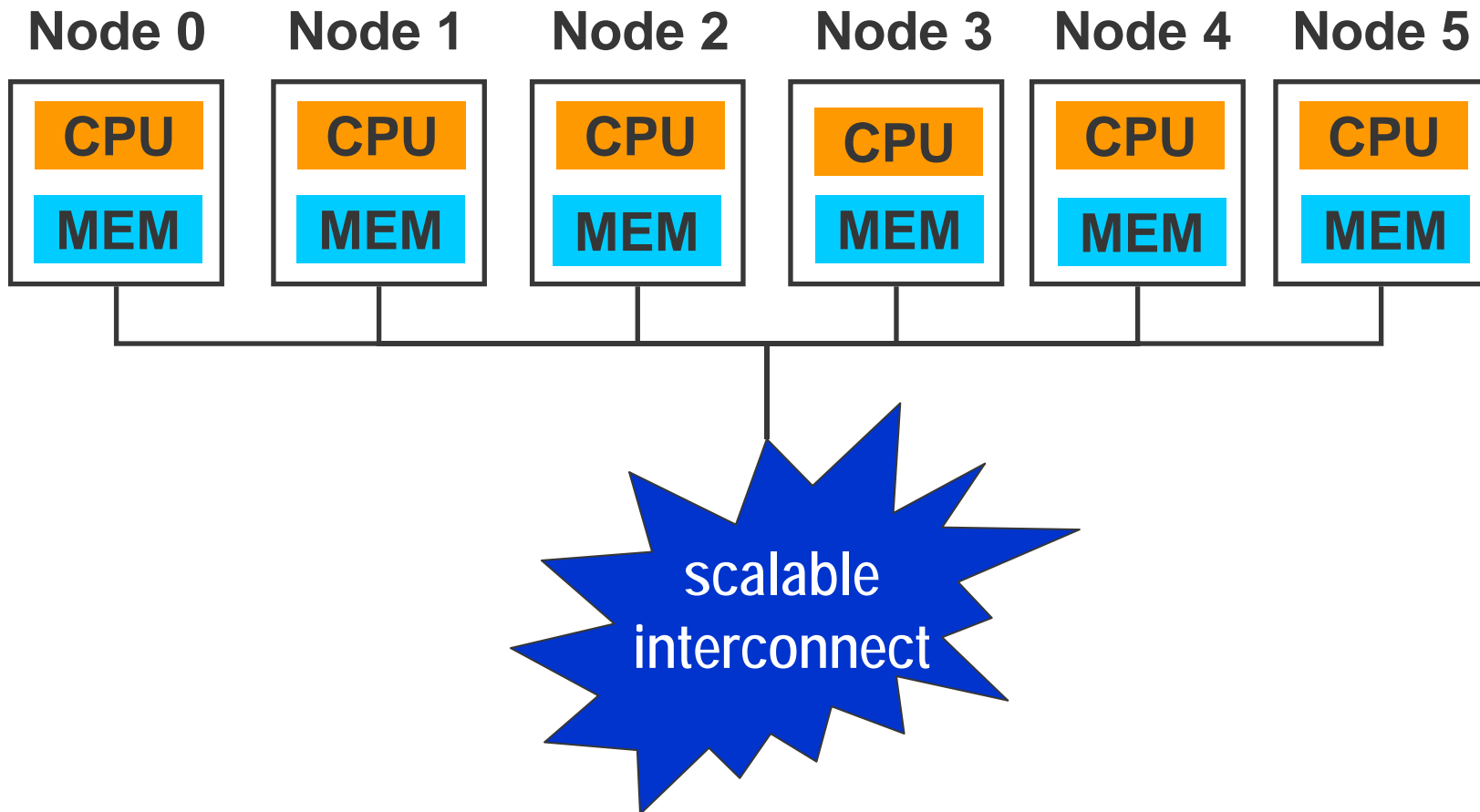
SGI® Altix® Hardware: SMP versus NUMA

- Most small multiprocessor systems are SMP
- A simple, generic Symmetric Multiprocessor architecture:

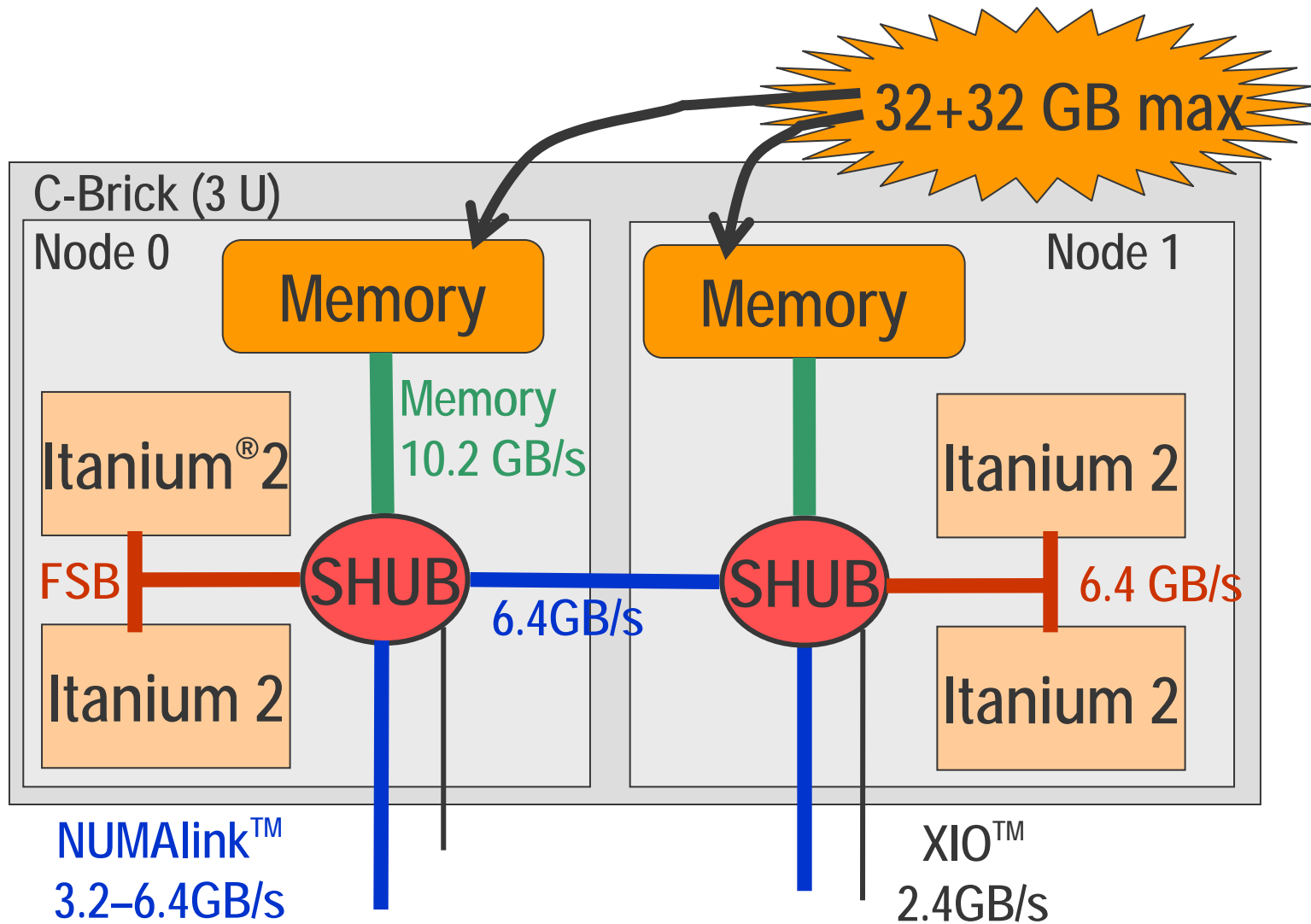


SGI® Altix® Hardware: SMP versus NUMA

- Generic NUMA machine

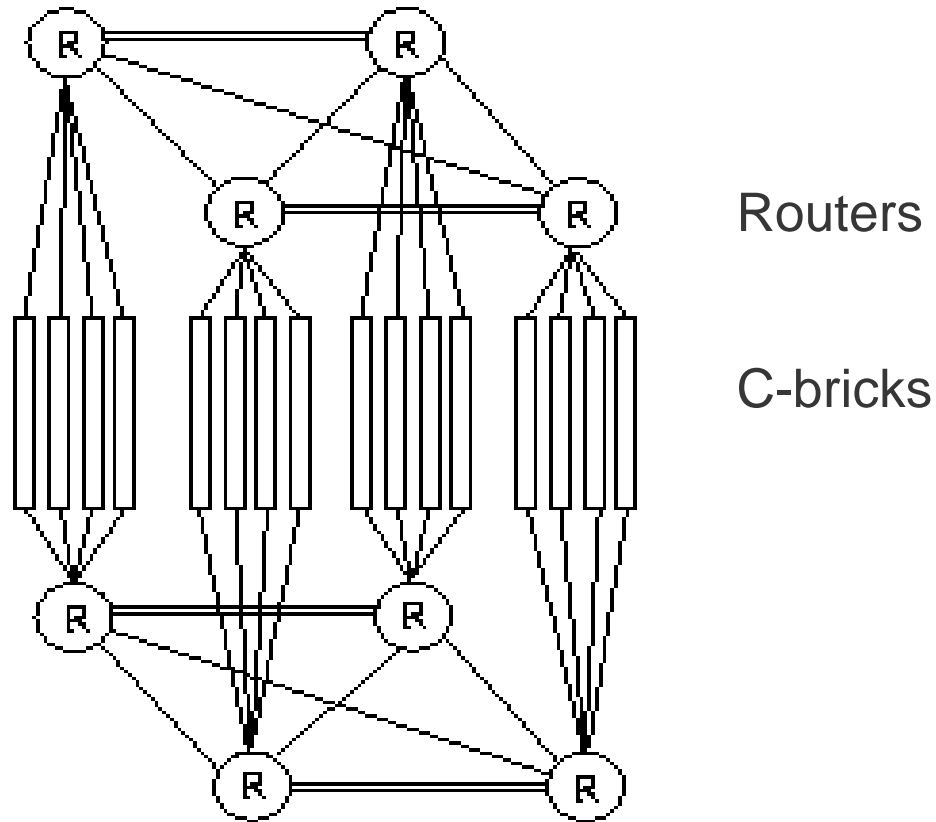


SGI® Altix® 3000 Hardware: Compute Brick



SGI® Altix® Hardware: Processor Interconnect

- Interconnect is a 2-plane, Fat-Tree with “Express Links” added:



64 Processor System

What is *scaling*?

Scaling is a characteristic of how a system performs as resources (e.g., processors, main memory, I/O) are added

- *Scaling* only makes sense in the context of specific workloads
 - Different workloads exhibit different scaling characteristics
- *Perfect scaling* is an ideal linear 1-to-1 improvement in performance
 - E.g., 2x the number of processors doubles performance
- Next best thing: positive improvements (even though deltas may get smaller)
- And next best thing: deltas decline to zero, but don't go negative
- And sometimes: deltas decline to zero, then go negative
 - E.g., adding a processor makes performance **worse**

What limits scaling?

- Contention on a shared resources become bottlenecks
 - Main memory, cachelines, locks protecting data structures
 - I/O channels, devices
 - Processors
- Algorithms that get linearly (or worse) slower as resources increase
 - Searching increasingly larger data structures
 - Needing each processor to execute some service
 - Terrabytes of memory potentially becomes terrabytes of dirty pages that cannot quickly be written to disk or swap
- NUMA memory nonlinearities
 - Local vs. remote memory access speeds and throughputs
 - Nonlinear performance falloff when contention worsens
 - Software threads that migrate away from optimal local memory

Memory bottlenecks: lock contention

- kernel_flag (*BKL*) \Rightarrow reduce usage; replaced with finer-grained locks
- Global runqueue and runqueue_lock \Rightarrow per-CPU runqueues and locks
- lru_list, lru_list_lock \Rightarrow per-CPU lists, and lockless
- page_cache_lock \Rightarrow recoded as fine-grained, then redesigned locking
- dentry hash table and dcache_lock \Rightarrow recoded with ref counts, RCU
- Concurrent page-faults of anon pages in shared addr space are serialized
 \Rightarrow redesigned locking to allow parallel faults
- Hugetlb pages get allocated at mapping time and serialized
 \Rightarrow allocation at first-touch becomes parallelized

Memory bottlenecks: lock contention (cont'd)

- Reductions in RCU rcu_ctrlblk lock contention
- Replaced some rwlocks with seqlock (e.g., xtime_lock)
- do_gettimeofday() uses cmpxchg
 - 4p is 10x slower than 1p; 32p is 100x slower than 1p

⇒ platform-specific avoidance of cmpxchg, using cheap global RTC timebase
- spin_unlock() tweak: use nta store to speed acquisition by another CPU
- write_unlock() tweak: denote write-lock with a byte flag instead of a bit, and clear with nta store instead of load+cmpxchg

Memory bottlenecks: cacheline ping-pong'ing

- Eliminate or reduce false cacheline sharing
 - Unintended: unrelated frequently-read and -written data in same cacheline
- Eliminate or reduce intensely written cachelines
 - `rwlock read_lock()` reader count isn't free: dirties cacheline at lock, unlock
 - Mapping `/dev/zero` was unnecessarily keeping counts & limits
 - ⇒ eliminate, and get 150x speedup for 256p concurrent mapping
- Problems with global cache lines that are updated on every clock tick
 - 512 processors making updates, each needing ~ 2usec, 1024 times/second
 - means no forward progress
 - (booting the kernel becomes a stress test of global cache coherency protocol)
 - E.g., `readprofile` concurrent incrementings of global idle-loop buckets
- Fix: moved to per-CPU storage and aggregate global value when needed

Memory inefficiencies: NUMA nonlinearities

- The “Catch-22” of a big NUMA machine:
 - There is lots of memory, but practically none of it is local
 - e. g., on a 512 CPU system, 0.4% of the available memory is local
- Local vs. remote memory access speeds and throughputs
 - cpusets help keep software threads close to memory (and CPUs)
 - Explicit CPU locality helps (*runon, taskset*)
 - First-touch memory allocation helps
 - but needs sticky CPU locality
 - and node-local allocation
 - per-CPU or per-node memory management
 - nonlinear performance falloff when contention worsens
- It is easy to fill local memory with page-cache pages
 - Resulting in only remote memory allocations being available

Memory inefficiencies: improve NUMA awareness

- NUMA-aware slab allocator
- NUMA-aware (and cpuset-aware) hugetlb allocation
- NUMA-aware pageset allocation
- Node-specific vmalloc
- NUMA-aware device memory allocation

Linear Search Algorithms May Scale Poorly

- `exit_notify()` scans entire tasklist, holding `tasklist_lock`: interrupts disabled for hundreds of msecs; livelocks system in extreme cases
 - ⇒ better management of parent-sibling-child links does selective scans
- `tcp_ehash[]` with 32+ million entries (Gbytes) severely impacts `tcp_get_info()`
 - ⇒ limit table to a rational size
- `del_timer_sync()` searches every per-CPU `timer_list`
 - ⇒ do targeted search
- `timer_bh()` `__run_timers()` on `cpu0` searches every per-CPU `timer_list`
 - ⇒ use per-CPU softirq handlers
- `show_mem()` (e.g., OOM killer) does massive page-faulting on big configs
 - ⇒ intelligent page table scanning gets 1500x speedup

Linear Search Algorithms May Scale Poorly (cont'd)

Searching all per-CPU structs will, in general, encounter cache and tlb limits.

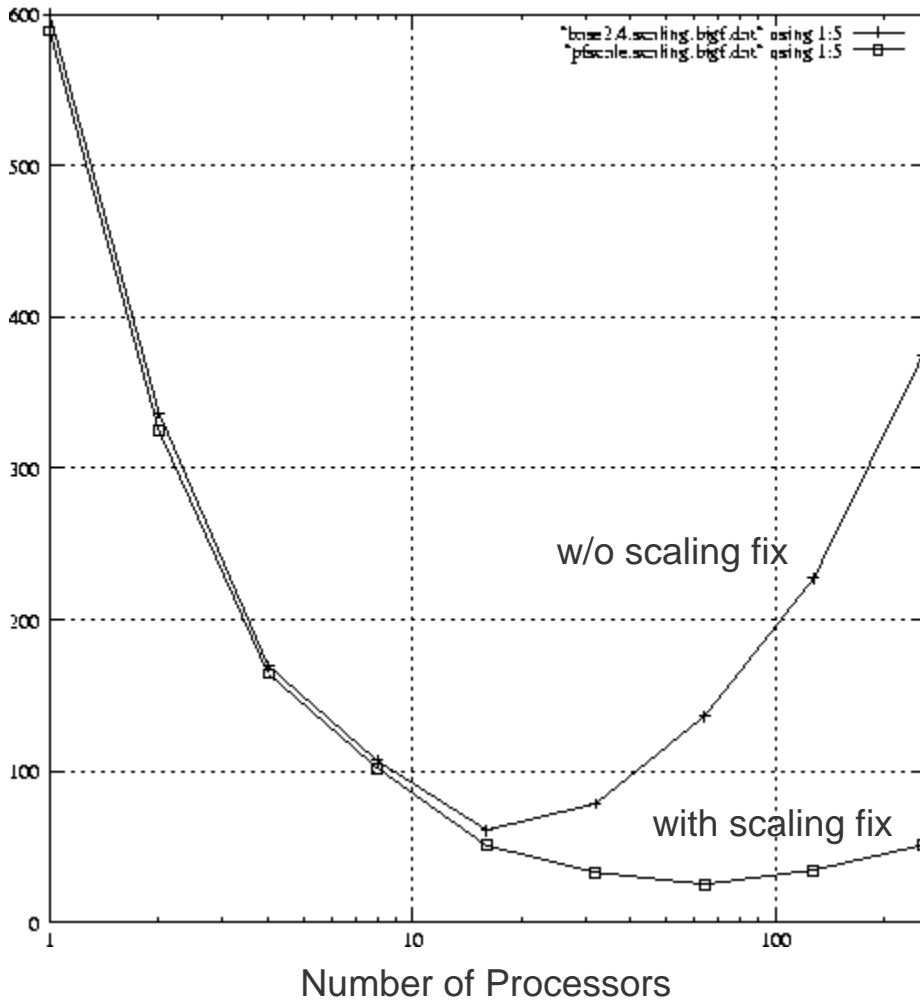
- Each per-CPU access requires TLB miss, which eventually consumes DTLB and slows down thereafter
- Each per-CPU access likely fills same cacheline color, which eventually consumes L3 N-way limit and requires main memory access

Various remedies may apply:

- Selectively move some per-CPU fields into global arrays
- For Altix, perform some frequent operations (e.g., writing to remote SHUB registers) using physical addressing mode, instead of virtual, to avoid using TLB
- For Altix, instead of for-all-CPU's tlb flush, flush only CPUs where task executed

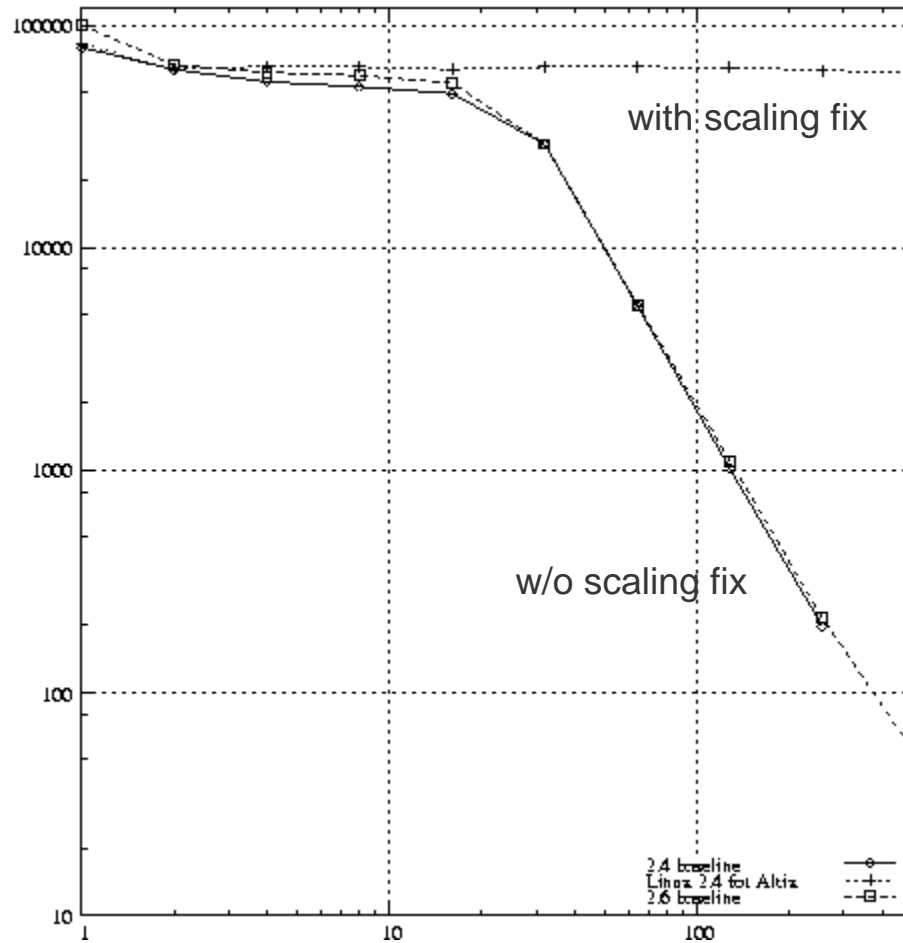
Changes to Linux[®] for Altix[®]: Page Fault Scaling

Seconds to Touch Data



Changes to Linux[®] for Altix[®]: /dev/zero mmap() scaling

Page faults/second wall clock time



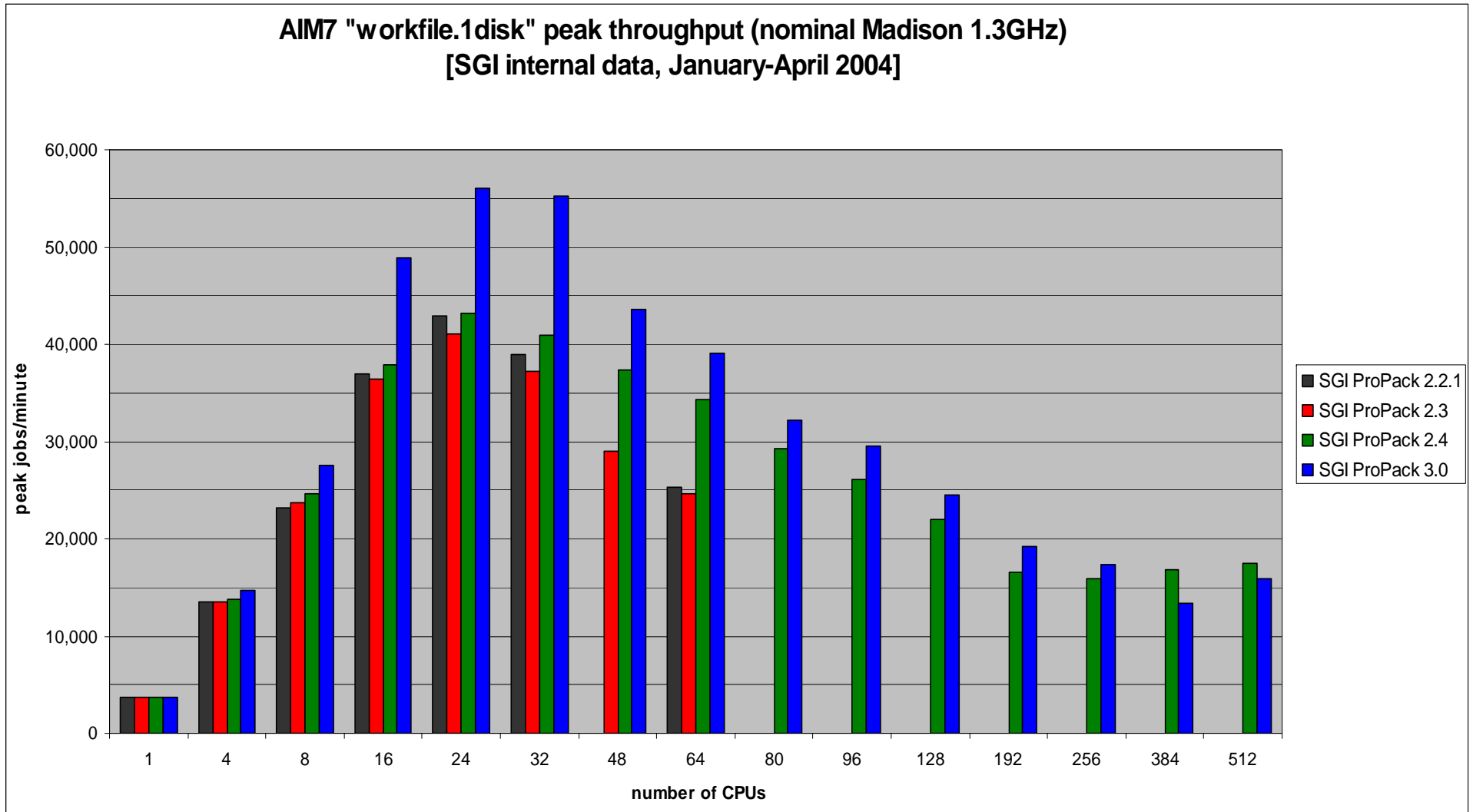
Note: Scaling problem in both Linux 2.4 & 2.6

Number of Processors

SGI ProPack™ scaling progress - AIM7 workfile.1disk

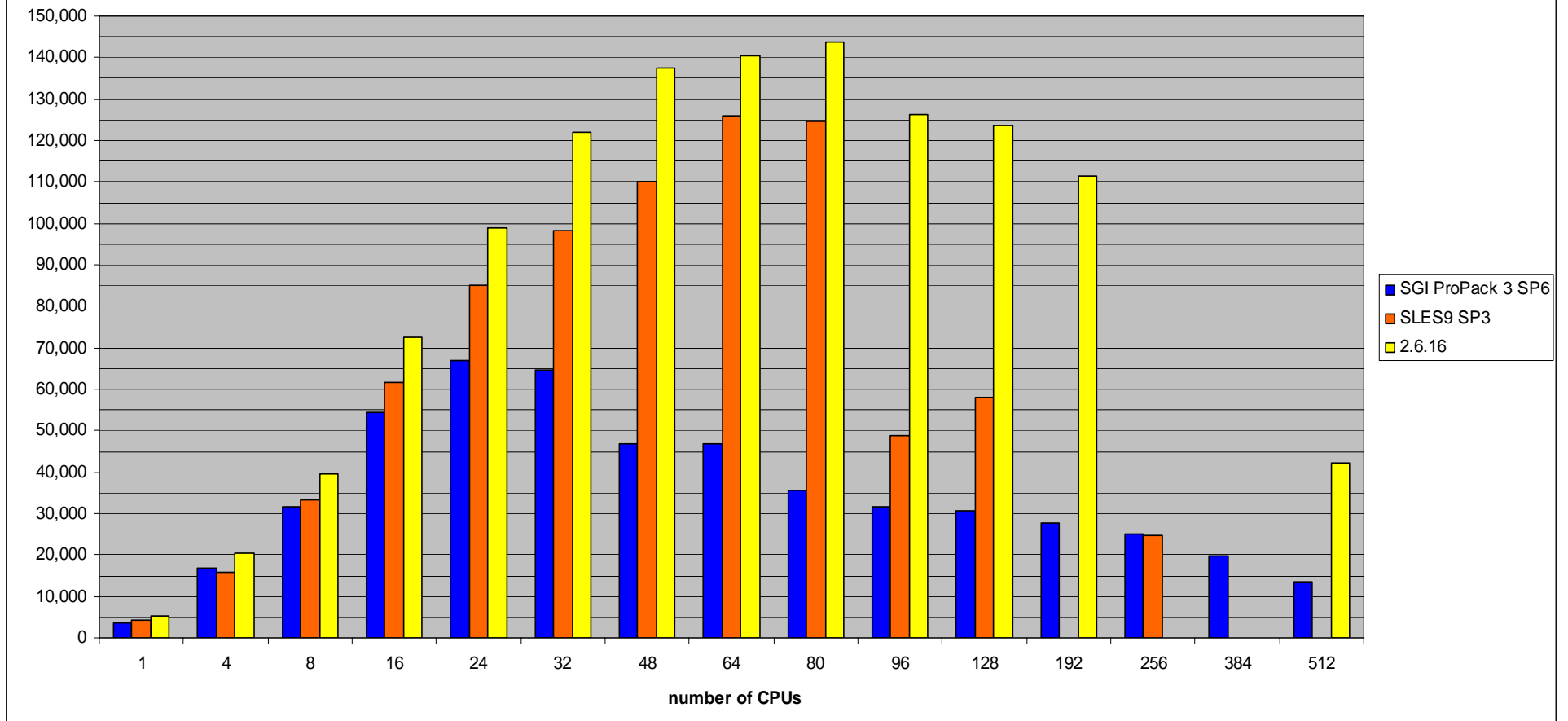
- AIM7 is multitasking, multifunctional benchmark that illuminates a range of performance problems
 - high CPU counts and high task counts illuminates scaling bottlenecks
- “workfile.1disk” is a variation of the “workfile.shared” workload
 - eliminates various I/O-bound subtests (sync_*, creat-clo, disk_src, link_test)
 - can run computebound with 1 disk drive
- Linux® 2.4 on Altix® steadily improved throughput from release to release
 - throughput peaks at 24 CPUs for this workload
 - bottlenecks are various, e.g., tasklist_lock, BKL, dcache_lock
- Linux® 2.6 on Altix® is markedly superior in performance and scaling to 2.4
 - throughput peaks at 64-80 CPUs for this workload
 - bottlenecks above 64 are various, e.g., locks and cacheline conflicts

SGI ProPack™ scaling progress - AIM7 workfile.1disk



SGI ProPack™ vs. Linux® 2.6 - AIM7 workfile.1disk

AIM7 "workfile.1disk" peak throughput (nominal Madison 1.5GHz)
[SGI internal data, 2004-2006]



Tools

- SGI Open Source tools

- Lockmeter <http://oss.sgi.com/projects/lockmeter>
- Kernprof <http://oss.sgi.com/projects/kernprof>
- Open SpeedShop <http://oss.sgi.com/projects/openspeedshop>
- Performance Co-Pilot <http://oss.sgi.com/projects/pcp>
- kdb <http://oss.sgi.com/projects/kdb>

- Community tools

- readprofile, oprofile
- perfmon, perfmon2, pfmon
- HP Caliper
- q-tools

Future Work

- Continued reduction of lock contention
 - dcache_lock
 - radix tree locking
 - XFS internal hash tables locking
 - Continued improvements to CPU Scheduler
 - sched domains tracking create/delete of cpusets
 - load-balancing tradeoffs: some threads benefit greatly being sticky to local memory, vs. other threads which benefit from non-sticky migration
 - Continued improvements to memory allocation & reclaim algorithms
 - Will a single policy be optimal for all workloads?
 - Better integration of allocation constraints with allocation locality
 - Per-node statistics to help allocators make better memory balancing decisions
 - Make networking code & data structures more NUMA-aware
-

Future Work (cont'd)

- Device scaling issues for 100xTB and petabytes of data
- Kernel text replication? 40% improvement on some microbenchmarks.
- Read-only data replication? Needs investigation.

Concluding Remarks

- Scaling is workload-dependent
 - In general, the greater the kernel component, the greater the likelihood of encountering a bottleneck
 - Some poor scaling behavior is due to poorly scaling usermode algorithms, not kernel
- Very large SSI configurations encourage large multithreaded applications
 - But threads of a common application tend to use common kernel functionality, which increases the likelihood of encountering a bottleneck
- Scaling bottlenecks are often nonlinear
 - A workload scales well to N CPUs, then performance improvements flatten out quickly, or even deteriorates
 - NUMA nonuniformities contribute to this nonlinear behavior
- As processor count increases, different bottlenecks may dominate
 - Behavior of N CPUs isn't always a reliable predictor of 2xN CPUs

The Future Is Never Guaranteed

This presentation contains forward-looking statements regarding 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 detailed from time to time in the company's most recent SEC reports, including its reports on Form 10-K and Form 10-Q.

© 2006 Silicon Graphics, Inc. All rights reserved. Silicon Graphics, SGI, IRIX, Altix, Origin, XFS, the SGI logo and the SGI cube are registered trademarks, and SGI ProPack, NUMALink, NUMAflex, XIO, OpenMP, and The Source of Innovation and Discovery are trademarks of Silicon Graphics, Inc., in the U.S. and/or other countries worldwide. R12000 is a registered trademark and R14000A is a trademark of MIPS Technologies, Inc., used under license by Silicon Graphics, Inc. Linux is a registered trademark of Linus Torvalds in several countries. Intel and Itanium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. All other trademarks mentioned herein are the property of their respective owners. (06/04)

Questions? Comments?