You are here

linux.conf.au
SYDNEY 2007

Automatic Page Migration for Linux
[A Matter of Hygiene]

Lee T. Schermerhorn
HP/Open Source & Linux Organization

"All architectural design involves teasing apart a problem by looking at the needs from as many directions as possible, until it reveals the structure within itself that the system designer can use to defeat it."

-- Alan Carter, "The Programmer's Stone"

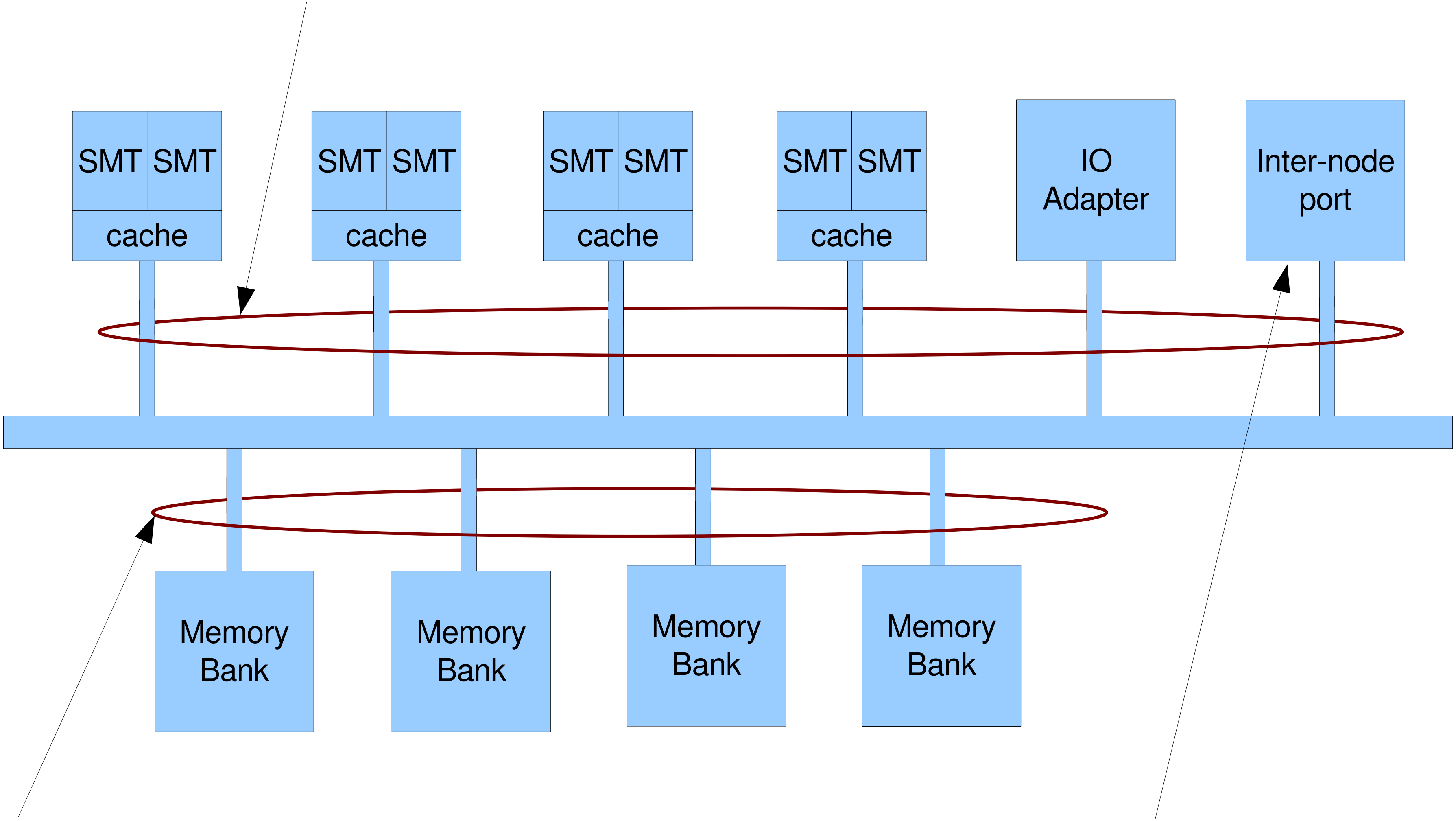
http://www.reciprocality.org/Reciprocality/r0/index.html

Why NUMA?

- To provide sufficient memory/IO/interconnect bandwidth to achieve scaling for systems based on modern, high-performance processors
  - multiple "low-cpu count" SMP [SMT] nodes or cells with "sufficient bandwidth" for local cpus + local IOA + inter-node interconnect
  - Low latency inter-node interconnect of sufficient bandwidth to handle off-node traffic.
    - Possibly hierarchy of these for larger systems
  - "It's the bandwidth, stupid!"
    - latency contributes to decrease in effective bandwidth
    - under contention, latency = f(load) ⇨ low or even inverse scaling
  - Heavily dependent on locality for a "win", similar to processor caches

A Node/Cell

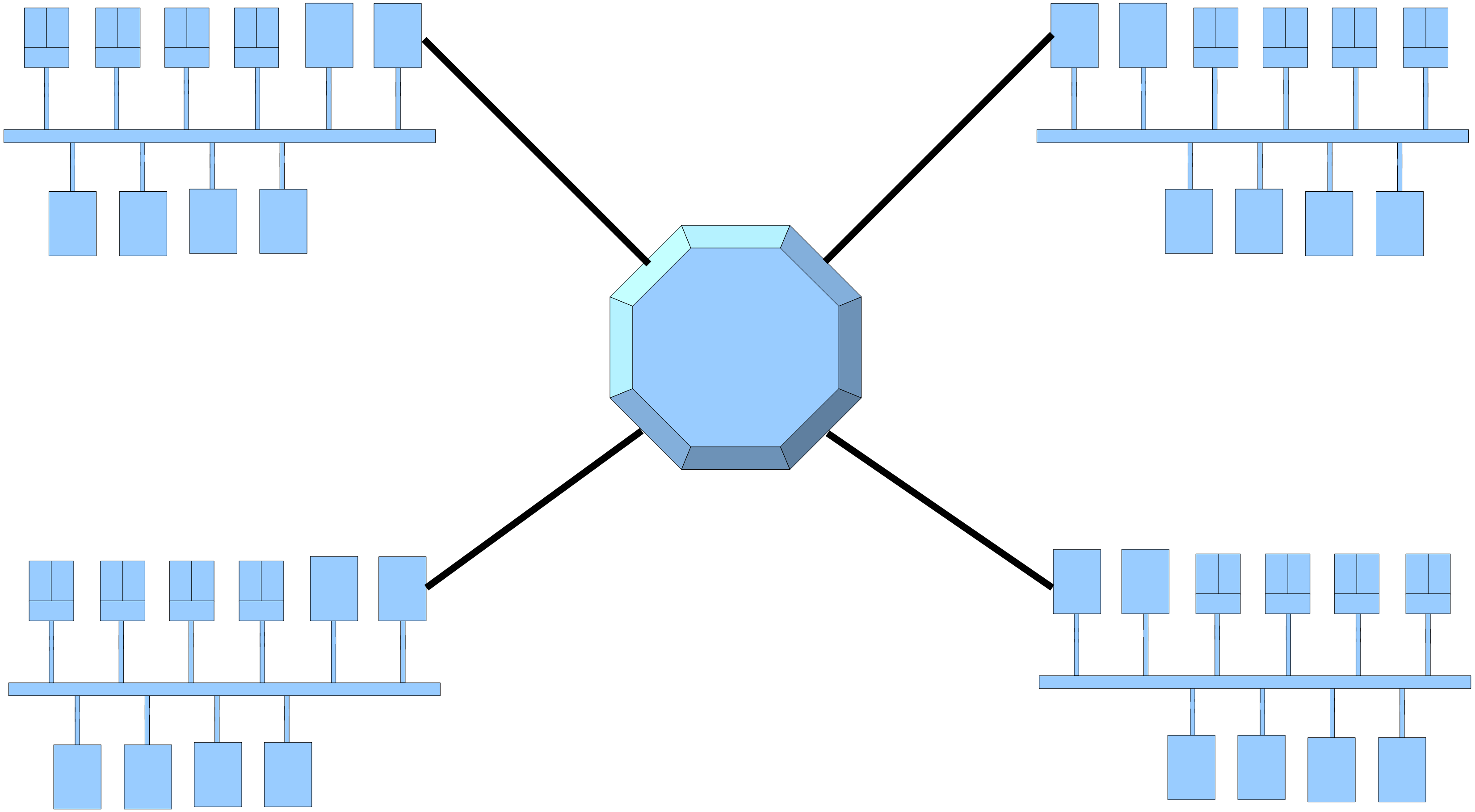
Memory Consumer Bandwidth:
CPUs + DMA +
Incoming remote accesses [CPU + DMA]



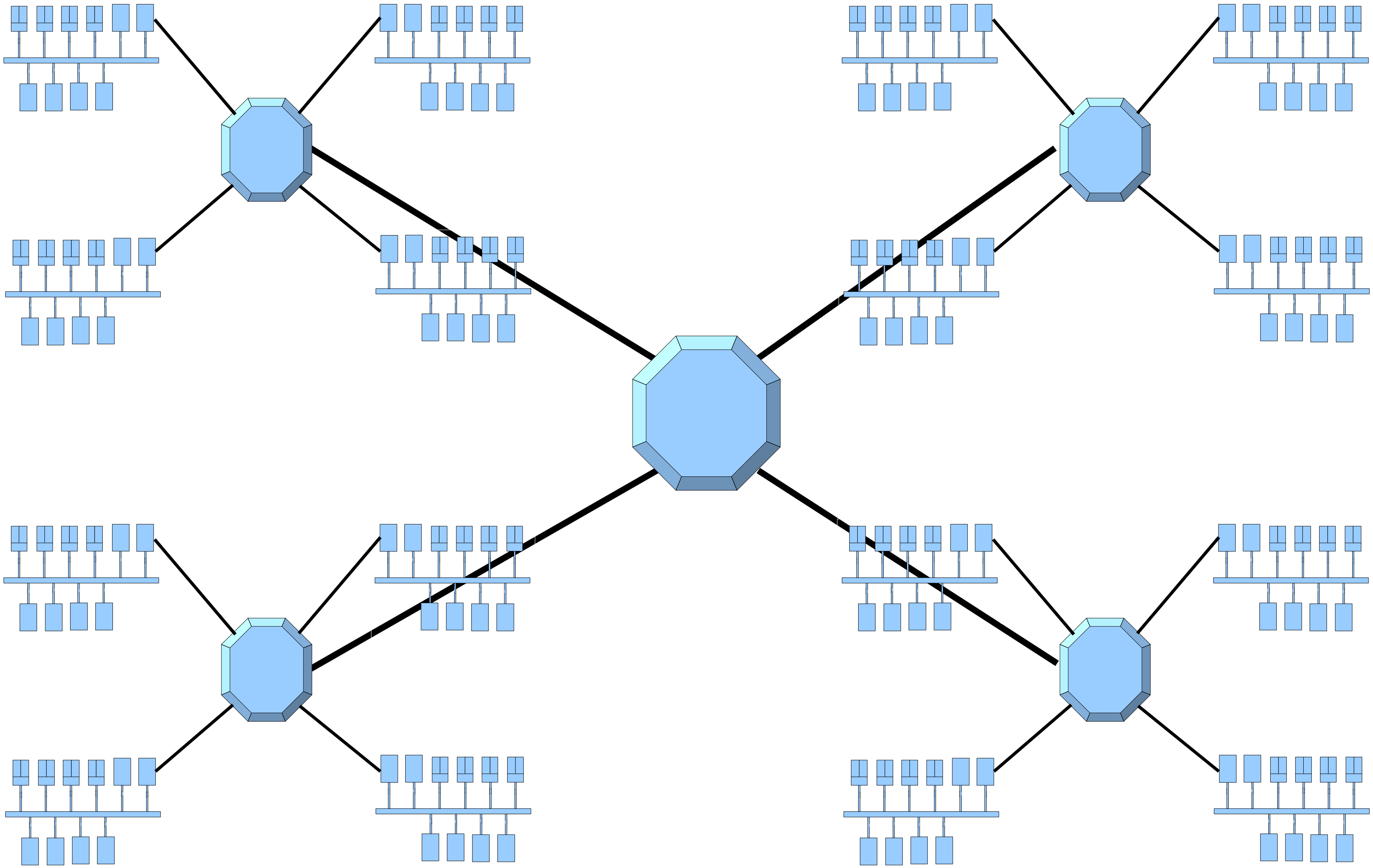
Total Memory Bandwidth

Both a consumer and provider of Memory Bandwidth

Single Level Interconnect



Multi-Level Interconnect



An Approach to NUMA Support

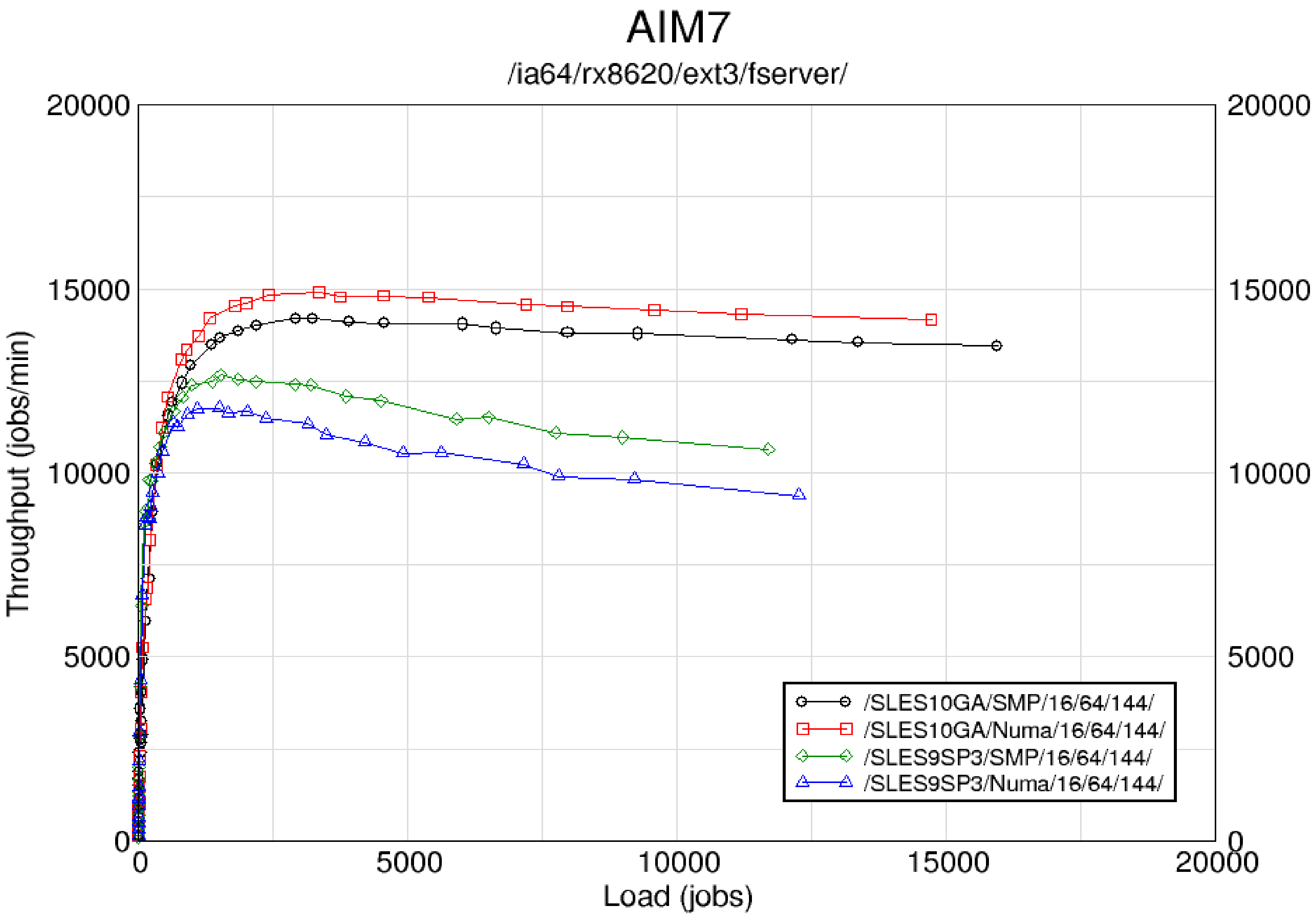
- Enhance kernel to "do the right thing":
  - reasonable [non-pathological] default behavior
  - Tunable behavior for various applications/loads
- Provide hints, application specific information to kernel:
  - Inheritable or "other-directed" behavior/options, settable by command line tools – e.g., numactl – for unmodified applications
  - APIs [system calls] for NUMA-aware applications and language run-time environments. Requires source modification or, at least, recompile/relink.

Evolution of Linux NUMA Support

- Node-aware mm; per node page pools and daemons, ...
  - "reasonable default behavior" – local allocation
- Mempolicy: by Andi Kleen, et al
  - APIs – mbind(), et al – and command line tool – numactl
- Task Migration: NUMA awareness in Scheduler Domains
  - all that remains of Eric Focht's early work on "Node Affine NUMA Scheduler"
- cpusets: resource/behavior "container" mechanism by Simon Derr and Paul Jackson
- Direct, synchronous page migration by Christoph Lameter
- NUMA-aware kernel memory allocations
  - SLAB infrastructure by Christoph Lameter
  - increasing kernel subsystem use thereof

Performance Impact of NUMA Features

- AIM7 benchmark:
  - many [very many] tasks
  - exercise kernel scalability more than user-space NUMA effects
- HP rx8620 platform:
  - 16 cpu [ia64], 4 node system
  - hardware memory interleaving across all nodes at cache-line granularity [SMP mode] vs all "cell local memory" [NUMA mode]
- 2.6.5+ kernel [SLES9] vs 2.6.16+ kernel [SLES10]
  - "NUMA Penalty" ⇨ "NUMA Benefit"

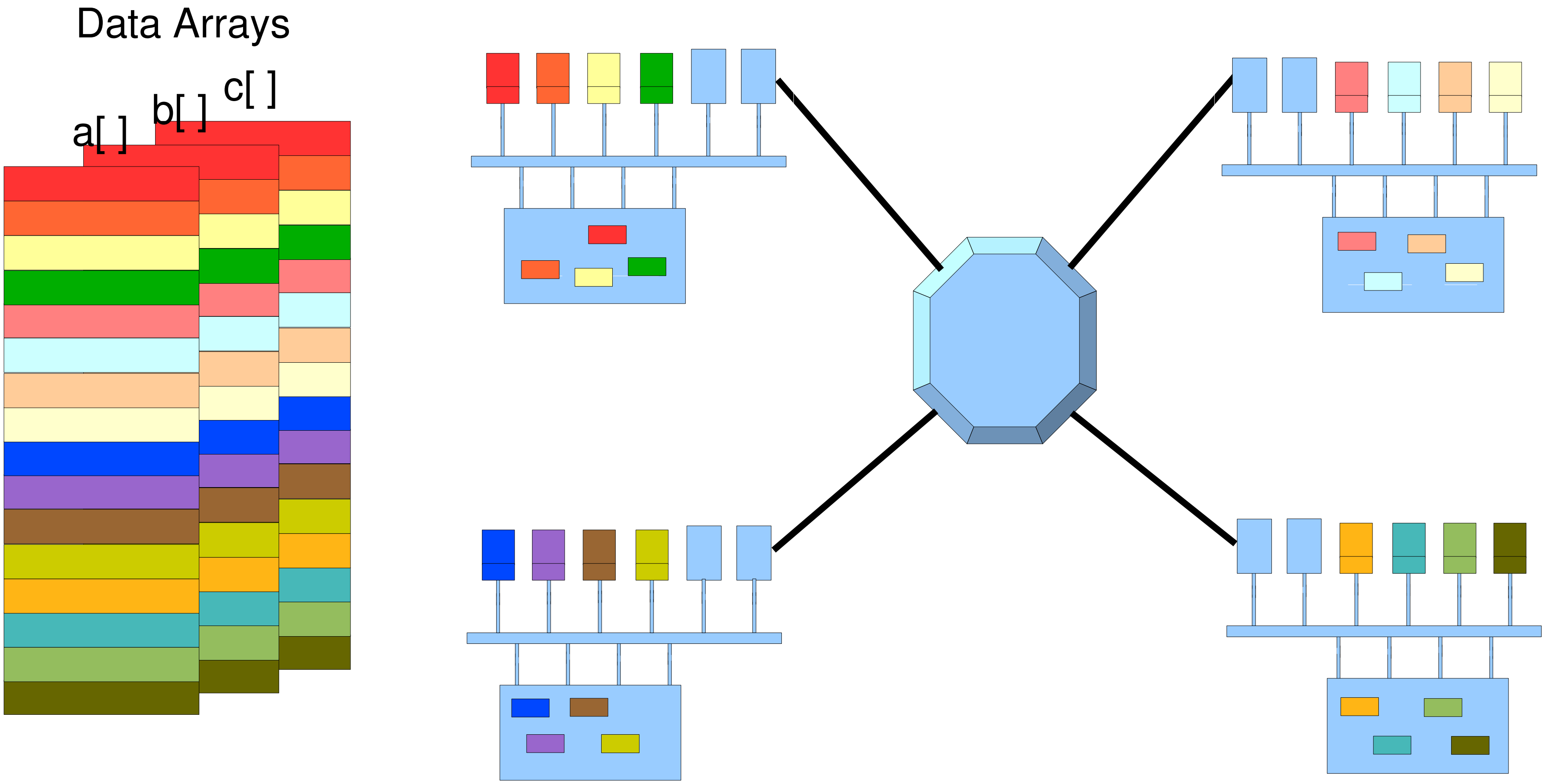

AIM7
/ia64/rx8620/ext3/fserver/
Throughput (jobs/min)
Load (jobs)
0
5000
10000
15000
20000
/SLES10GA/SMP/16/64/144/
/SLES10GA/Numa/16/64/144/
/SLES9SP3/SMP/16/64/144/
/SLES9SP3/Numa/16/64/144/

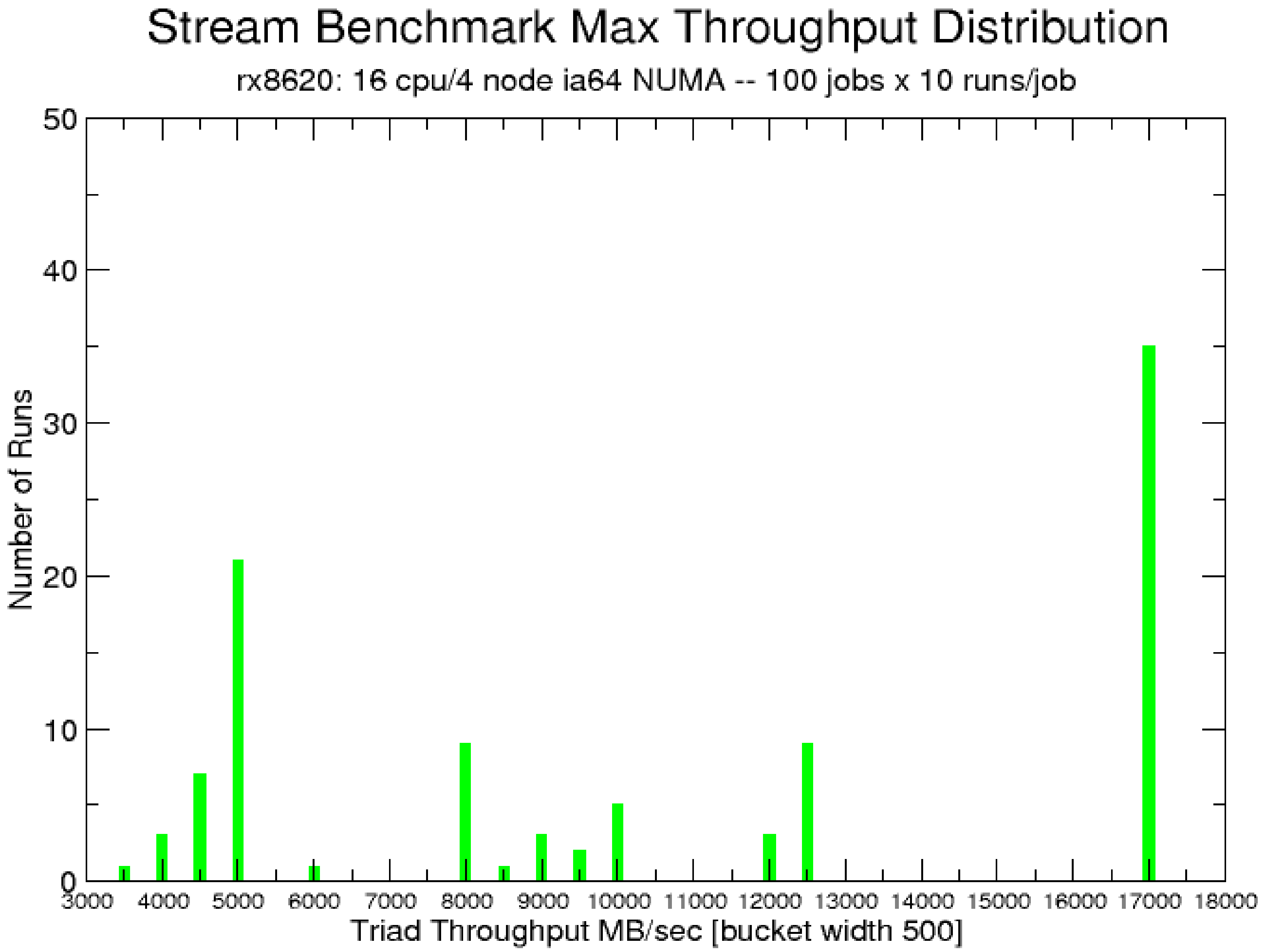
Application Performance: Room for Improvement

- Applications that are sensitive to NUMA locality effects:
  - large memory footprint with [relatively] high cache miss rate ⇨ sensitive to effective memory bandwidth
  - long lived; subject to load balancing
- Simulate using McAlpin Stream benchmark
  - specifically designed to measure platform memory bandwidth = bytes_moved / elapsed_time
  - build with OpenMP extensions [Intel Compiler] to run on all nodes
  - If all threads access local memory, we get to claim the sum of all nodes' bandwidths
- Let the kernel have it's way – no affinity, no cpusets, ...
  - run-to-run variability—apparently due to locality
  - disruption of initial locality by inter-node load balancing

Stream Benchmark Threads & Memory

[OpenMP Decomposition]



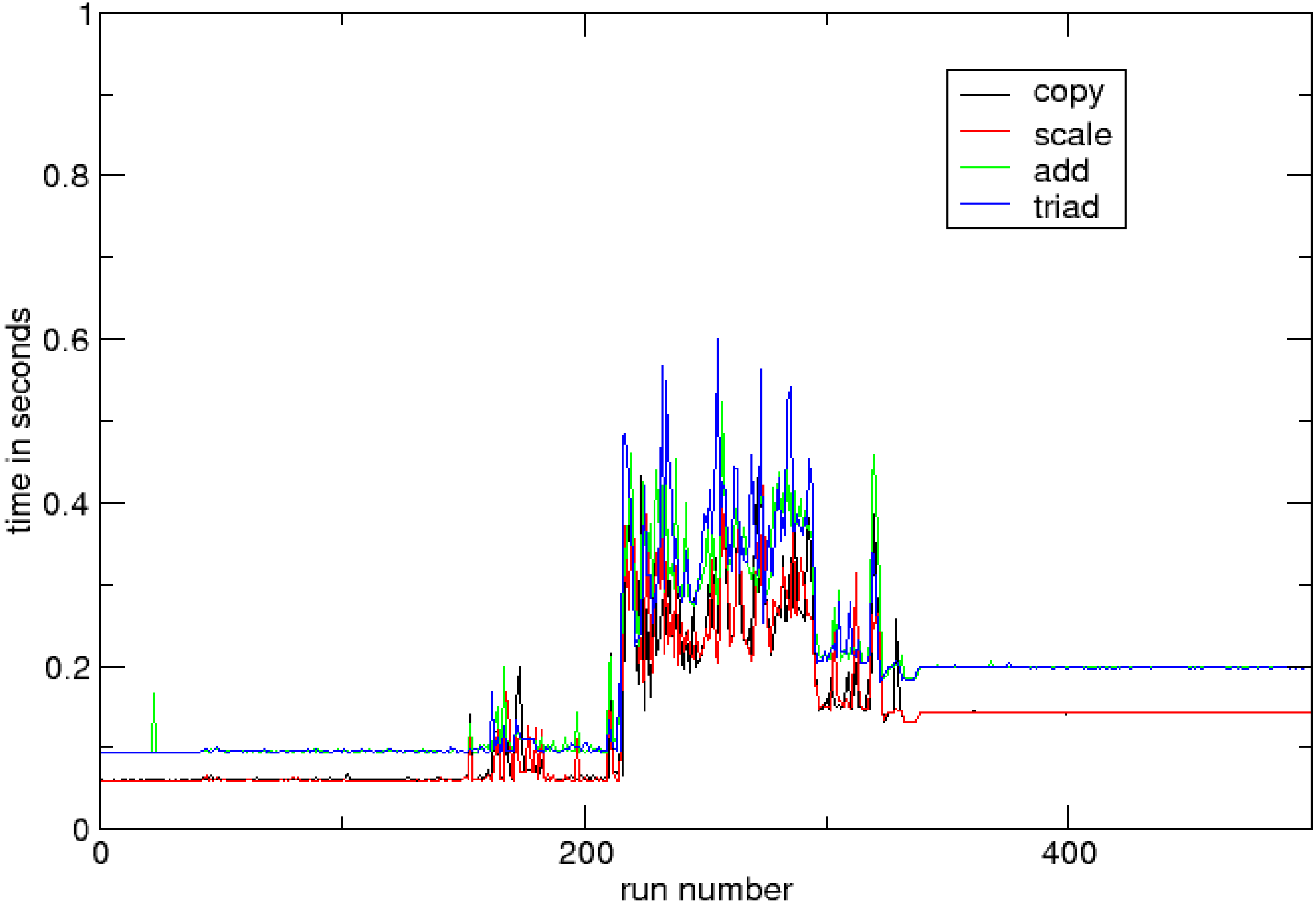

Stream Benchmark Max Throughput Distribution
rx8620: 16 cpu/4 node ia64 NUMA -- 100 jobs x 10 runs/job
Number of Runs
0
10
20
30
40
50
3000
4000
5000
6000
7000
8000
9000
10000
11000
12000
13000
14000
15000
16000
17000
18000
Triad Throughput MB/sec [bucket width 500]

Dynamic Effects of Load Balancing

- Allow kernel to balance cpu loads:
  - i.e., no explicit binding of OpenMP threads to cpus
- Run benchmark for sufficient time to allow a transient work load to run
- Apply transient work load:
  - 32-job [2 x # cpus] parallel kernel build
- Plot completion time for each "run" of stream benchmark

500 run stream with 32job kernel build

rx8620: 16cpu/4node IA64 NUMA -- No Auto-Page Migration



What's Happening?

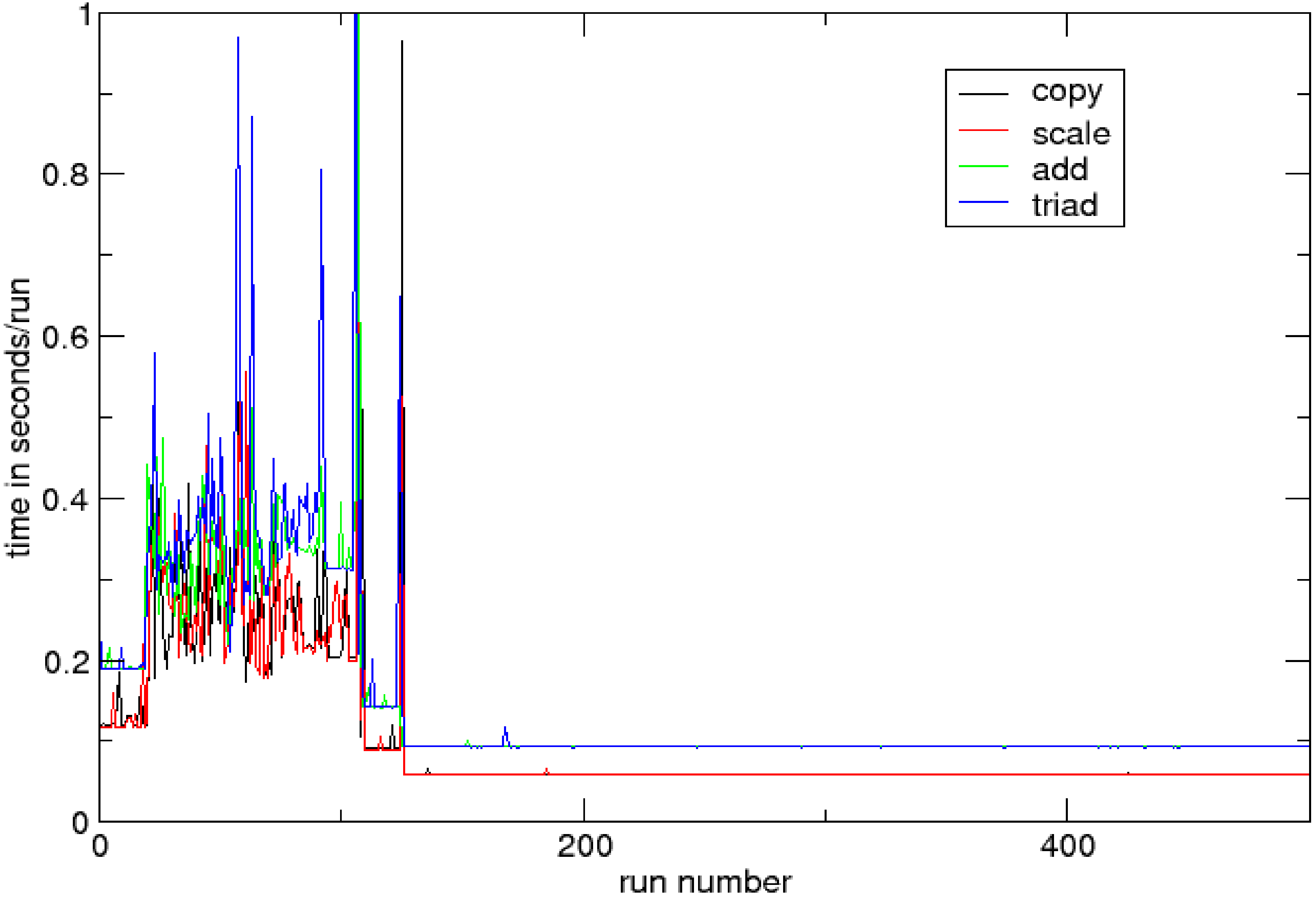
- Without "manual intervention", optimal placement has relatively low probability
  - that probability decreases with increasing node count
- Even with near optimal initial placement, transient workloads can destroy locality
- But, why not just bind tasks/threads to cpus/nodes?
  - Standard practice in dedicated HPC environments, but:
  - "absence of easy, fine-grained load balancing is a real barrier to getting good scalability"
    - "OpenMP in the Real World", Rob Thacker, Dept. of Physics, Queens University
    - http://www.sharcnet.ca/events/fw2004/slides/FW2004_Thacker.pdf
  - Less acceptable in some environments ["enterprise"]

Can the Kernel Help?

- The kernel is performing inter-node load balancing
  - it knows when it's moving a task to a new node
- However, without extensive [expensive?] page reference tracking, the kernel can't know which pages a task will reference after migration
  - especially true for multi-threaded programs
- Still, if the kernel could arrange for a task to self [auto] migrate the pages it touches after inter-node migration, locality can be restored
  - at some cost, of course. Migration doesn't come for free.
- Enter automatic, lazy page migration

500 run stream with 32job kernel build

rx8620: 16cpu/4-node IA64 NUMA -- Auto-Page Migration Enabled



Automatic/Lazy Page Migration

- So, what is it? How does it work?
- Two major components [patch sets]:
  - "migrate-on-fault" [a.k.a. "lazy page migration"]: migrates a page in the fault path if:
    - migrate-on-fault enabled [per cpuset]
    - page has no ptes referencing it
    - page's mapping has a "migratepage" op
    - page location does not match policy.
  - "auto-migrate": if enabled [per cpuset]
    - marks task with "migrate pending" when load balancer migrates task to a new node
    - task "unmaps" [removes pte references from] all pages controlled by "default" policy with "mapcount" < N [default 1] .
    - migrate-on-fault pulls them local on next touch

Page Migration in 2.6.19

- mbind() -- add MPOL_MF_MOVE[_ALL] flags
  - migrate pages in range to match specified policy
  - w/o '_ALL moves only pages with mapcount == 1 – i.e., currently referenced only by calling task's page table.
- migrate_pages(pid, maxnodes, srcnodes[], destnodes[])
  - new syscall to move self or others if allowed by permissions and cpuset constraints
  - internally: migrate_to_node() for each "source node"
- More internals:
  - check_range() used by mbind() and migrate_to_node() to scan task's page table for pages eligible to migrate
  - migrate_pages(): direct, synchronous migration of list of pages collected by check_range()

More 2.6.19 Migration Internals

- migrate_pages() – for each page in list:
  - unmap_and_move() using get_new_page() allocation function, passed in as arg to migrate_pages()
- unmap_and_move()
  - allocate new page; abort if fails
  - try_to_unmap() with 'migration flag'
  - move_to_new_page()
- move_to_new_page()
  - if no mapping, invoke migrate_page() directly,
  - else call mapping's migratepage operation, if any, else use fallback_migrate_page()
  - buffer_migrate_page() for pages with file system private data attached.

Migrate on Fault Migration Additions

- mbind() – add MPOL_MF_LAZY modifier to '_MOVE*
  - Don't actually move pages; just "remove translations" [pte references]
  - migration occurs in fault path on next touch if page is misplaced w/rt policy in faulting context
- Internally, after collecting pages via check_range(), call new function migrate_pages_unmap_only() instead of migrate_pages()
  - push anon pages to swap cache if not already there
  - try_to_unmap with "MIGRATE_DEFERRED" flag
    - migrate_pages() now uses "MIGRATE_DIRECT" flag

Migration in Fault Path

- E.g., for anon pages: do_swap_page()
  - under page lock, if migrate_on_fault enabled for faulting task, call check_migrate_misplaced page()
  - static, in-line function if configured; else NULL macro
  - if page_mapcount(page) == 0 and page's mapping has a migratepage op,
    - call mpol_misplaced() to check page location against policy; returns target node if misplaced
    - call migrate_misplaced_page(), if misplaced
  - migrate_misplaced_page()
    - allocate new page on target node: ! GFP_WAIT + GFP_THISNODE; if fails just return old page
    - set up new page and call mapping's migratepage op with new 'faulting' flag
    - adjust ref counts; free old page; return new page

Automatic Page Migration Additions

- In most places where scheduler calls set_task_cpu(), call new check_internode_migration()
  - if automigration enabled for this task, and new cpu is on different node, set current->migration_pending and TIF_NOTIFY_RESUME thread_info flag
- On return to user space, in "notify_resume" handler, call check_migrate_pending():
  - no-op if SIGKILL pending
  - if current->migrate_pending, call new function auto_migrate_task_memory()
  - calls migrate_to_node() with additional flags:
    - 'AUTOMIGRATE – only scan vma's with default policy
    - optionally, 'LAZY – 'unmap_only() instead of migrate_pages()

Configurable Behavior

- At kernel build:
  - MIGRATE_ON_FAULT: depends on MIGRATION, selects CPUSETS & SWAP
  - AUTO_MIGRATION: depends on MIGRATION, selects CPUSETS
- Run-time, per cpuset:
  - **migrate_on_fault**: enable/disable; default == disable
  - **auto_migration**: enable/disable; default == disable
  - **auto_migrate_interval**: [seconds] – don't migrate task to new node more often than this; default == 30 sec
  - **auto_migrate_lazy**: default ==direct migration
  - **migrate_max_mapcount**: threshold for selecting pages for migration; default == 1

Performance of Kernel Builds

- By itself, kernel build [-j32] does not see a win from automatic migration:

2.6.19-rc6; avg & std devn for 10 runs

	Real	User	System
No Patches	92.87	1099.97	65.15
	2.18	1.01	0.26
with AutoMigration patches – disabled	92.42	1100.55	65.36
	1.68	1.03	0.44
Migrate on Fault enabled	92.55	1099.64	75.01
	1.79	0.73	0.61
Both Migrate on Fault and Automigration enabled	93.47	1098.05	76
	1.83	0.86	0.28

What's Happening?

- Statistics from [mmtrace] instrumented kernel— 32 job parallel kernel build:

	Migrate-on-Fault Only	Migrate-on-Fault + Automigration	MoF + AutoMig No Pg Cache Mig
Automatic task memory migrations	0	2714	2789
Pages scanned for migration	0	7065	13415
Pages selected for lazy mig	0	2525	7413
PTE Faults	7.26M	7.27M	7.27M
"No Page" Faults	3.76M	3.76M	3.76M
"Swap Page" Faults	0	2100	4266
"Anon Page" Faults	3.0M	3.0M	3.0M
Pages checked for misplacement	502K	504K	2749
Misplaced page migration attempts	346K	343K	2632
Misplaced page migrations successful	345K	342.5K	2632

What's Really Happening?

- Out of 3.76M mapped file/page cache faults, ~0.5M were found to have zero mapcount, and so were checked for misplacement
- 346K of these were deemed to be "misplaced" and the kernel attempted a migration. 99.7% of these attempts succeeded.
- Examination of the traces showed many page cache pages bouncing between nodes thousands of times.
- My conclusions:
  - unconditionally migrating unreferenced file backed pages may not be such a good idea
  - consider replicating shared, read-only pages as future work item [old patches exist from Virtual Iron]

Migrating Anon Pages only

- Removed file page migration patch
  - not run time configurable [yet]

2.6.19-rc6; avg & std devn for 10 runs

	Real	**User**	**System**
No Patches	92.87	1099.97	65.15
	2.18	1.01	0.26
with AutoMigration patches – disabled	92.42	1100.55	65.36
	1.68	1.03	0.44
Migrate on Fault enabled	92.55	1099.64	75.01
	1.79	0.73	0.61
Both Migrate on Fault and Automigration enabled	93.47	1098.05	76
	1.83	0.86	0.28
MoF and Automigration enabled -- anon pages only	91.27	1098.69	66.5
	1	0.46	0.42

Kernel Builds on Heavily Loaded System

- Run kernel build [-j32] with parallel Stream benchmark [16 threads on 16cpu system]:
  - migrate anon pages only

2.6.19-rc6; avg & std devn for 10 runs

	Real	User	System
with AutoMigration patches – disabled	92.42	1100.55	65.36
	1.68	1.03	0.44
MoF and Automigration enabled -- anon pages only	91.27	1098.69	66.5
	1	0.46	0.42
Stream running; MoF/AutoMigration disabled	147.33	1179.71	93.76
	2.6	4.02	1.91
Stream running; MoF/AutoMigration enabled	120.94	1130.24	73.17
	2.06	4.96	1.84

Status of the Patches

- Two separate, mostly independent, patch sets:
  - migrate-on-fault
  - automatic page migration
    - "lazy" option depends on migrate-on-fault patch set
  - both currently atop "mapped file policy" patch set
- Two related patch sets:
  - memory policy for shared, mmap()ed files
  - migration cache: pseudo-swap cache for lazy migration of anon pages without backing storage
- Maintained "out-of-tree", up to date with -mm tree and mainstream releases and release candidates
  - little support for acceptance, even into -mm, when last posted
- Available at: http://free.linux.hp.com/~lts/Patches/PageMigration

Size of Patches

- Including Mapped File Policy and Migration Cache Patch Sets

ia64	**text**	**Δtext**	**data**	**Δdata**	**bss**	**Δbss**	**total**	**Δtotal**
No patches	8174117		1600644		1472245		11247006	
+file policy	8178629	4512	1600900	256	1472253	8	11251782	4776
+mig on fault	8184613	5984	1601044	144	1472261	8	11257918	6136
+automig	8190189	5576	1601788	744	1472261	0	11264238	6320
+migcache	8197781	7592	1602588	800	1472261	0	11272630	8392
Total		23664		1944		16		25624

x86_64	**text**	**Δtext**	**data**	**Δdata**	**bss**	**Δbss**	**total**	**Δtotal**
No patches	3818668		1116316		1419912		6354896	
+file policy	3820703	2035	1116548	232	1420040	128	6357291	2395
+mig on fault	3823040	2337	1116660	112	1420040	0	6359740	2449
+automig	3825182	2142	1117492	832	1420040	0	6362714	2974
+migcache	3828221	3039	1118228	736	1419912	-128	6366361	3647
Total		9553		1912		0		11465

Testing Status

- Load/stress testing with Dave Anderson's [Red Hat] "Unix System Exerciser" a.k.a. usex
- Overnight, over [long] weekends, ...
  - 16cpu/4node and 64cpu/16node ia64 systems; 2 cpu/node x86_64 system
- Patches hold up well:
  - tests continue to run
  - some issues in logs:
    - floating point and alignment issues on ia64 from /usr/bin tests
      - may not be related to patches
    - race[s] in page cache migrate-on-fault
      - fixed
      - maybe moot?

Some References

- Ottawa Linux Symposium Linux Presentations:
  - Matthew Dobson, Patricia Gaughen, Michael Hohnbaum, Erich Focht, "Linux Support for NUMA Hardware", *Proceedings of the Ottawa Linux Symposium,* Ottawa, Ontario, Canada, July 2003
    - http://archive.linuxsymposium.org/ols2003/Proceedings/All-Reprints/Reprint-Gaughen-OLS2003.pdf
  - Ray Bryant and John Hawkes, "Linux Scalability for Large NUMA Systems", *Proceedings of 2003 Ottawa Linux Symposium,* Ottawa, Ontario, Canada, July 2003.
    - http://archive.linuxsymposium.org/ols2003/Proceedings/All-Reprints/Reprint-Bryant-OLS2003.pdf
  - Ray Bryant, Jesse Barnes, John Hawkes, Jeremy Higdon, and Jack Steiner, "Scaling Linux to the Extreme", *Proceedings of the 2004 Ottawa Linux Symposium*, Ottawa, Ontario, Canada, July 2004
    - http://www.linuxsymposium.org/proceedings/LinuxSymposium2004_V1.pdf, p133
  - Christoph Lameter, "Local and Remote Memory: Memory in a Linux/NUMA System",
    - http://kernel.org/pub/linux/kernel/people/christoph/pmig/numamemory.pdf

Any [time for] questions?