Lustre Performance on KNL

Single Shared File I/O

Locking primitives

KNL: Slow.

- What does slow mean?
- Single process throughput ~300 MB/s
- Limited by copy rate between userspace and kernel (same limit as on Xeon, but Xeon is ~1.3 Gb/s)
- “full packed” nodes also very slow due to contention
- Moderate process counts are OK

KNL: Slow.

- What can we do?
- Many processes: Address contention
- Single process: ‘copy’ operation already in assembly
- Very difficult to parallelize in kernel, ask me for details of early attempts
- Parallelize outside of kernel – Split I/O between multiple processes in userspace

KNL: Slow.

- Quick note on goals:
  Cray compute nodes are limited to ~5.5 GB/s, so we don't overwhelm downstream network in big systems
- At ~4 GB/s here, possible NUMA issues
- Gregoire is aiming higher (8-10 GB/s), sees some different problems

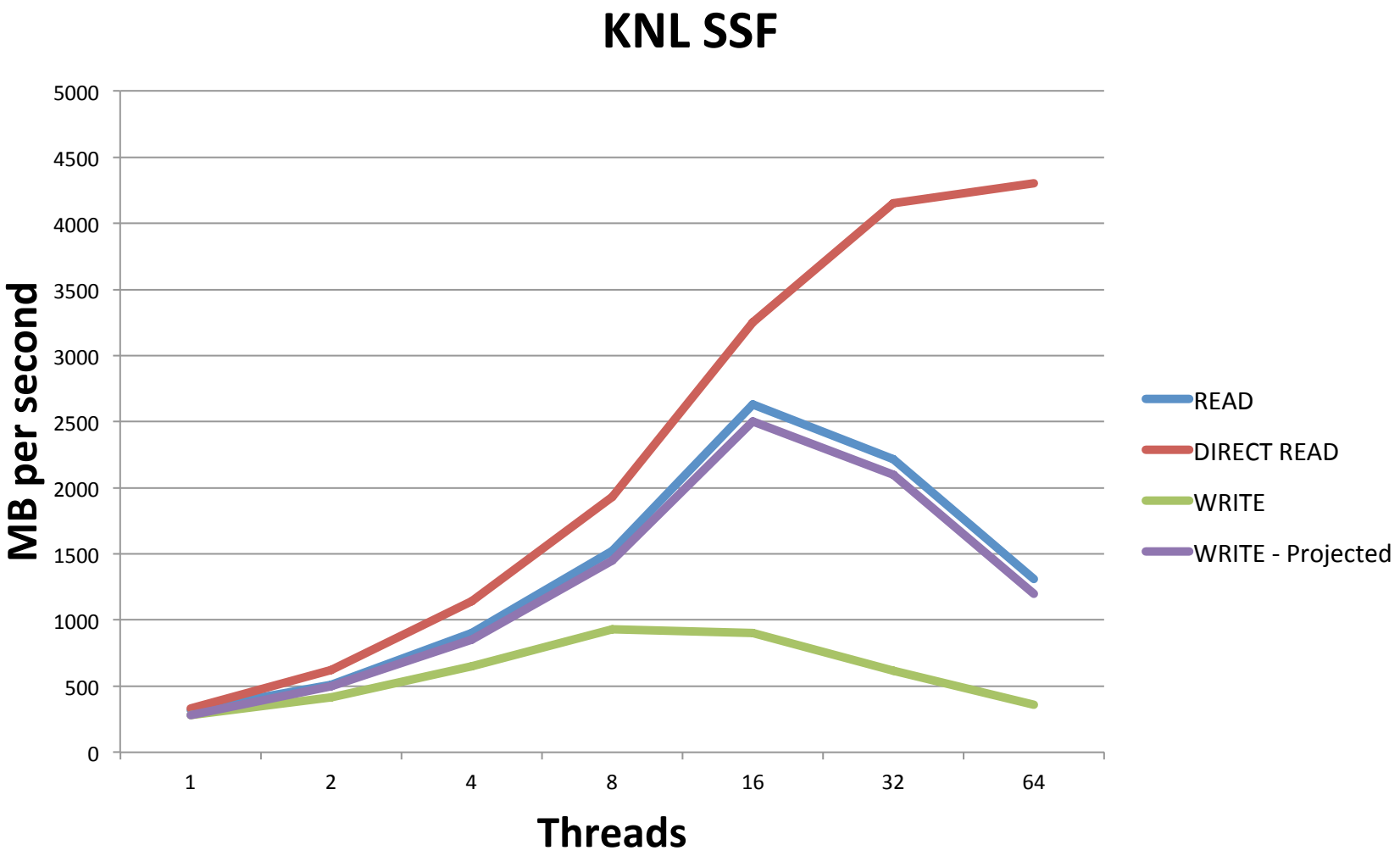
Single shared file I/O

- Always important; particularly interesting on KNL because individual cores are so slow
- All of this is single node
- Current master: Reading is OK until higher thread counts, writing is bad at > 1 rank
- A quick digression on that…

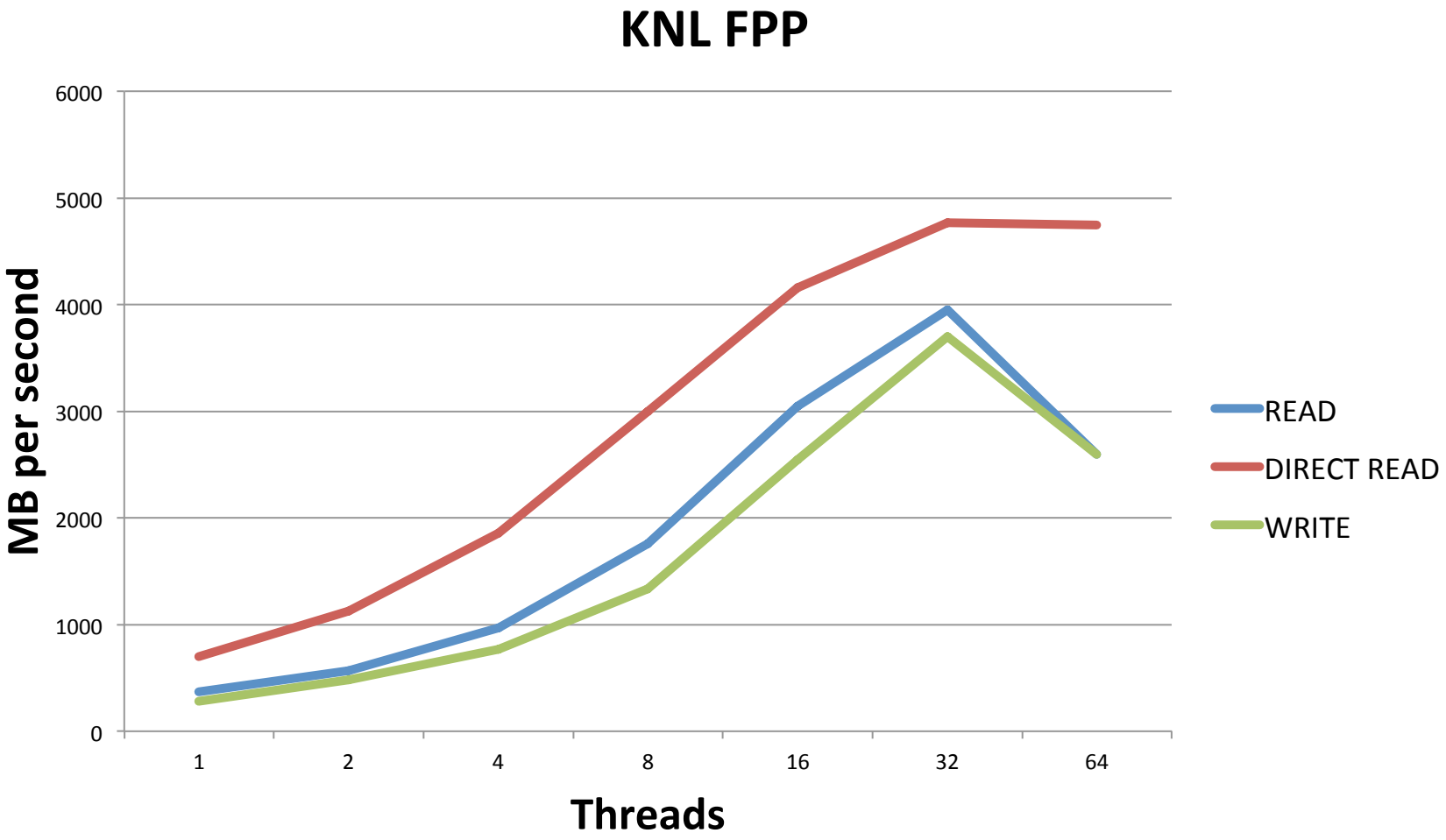
Single shared file I/O - Writing

- Much better in earlier versions (2.5-)
- Found RPC sizes were often small, even with large, well formed I/O from IOR
- LU-8515 – Do a better job of picking extents to send
- Results in well formed I/O, good performance, details in ticket
- All benchmarks are with that patch.

Performance: KNL SSF



Performance: KNL FPP



Single shared file I/O

- Scales better, still not great
- Contention first appears on cl_object_attr_lock, ~16 processes
- Page cache at 32-64 processes, mapping->tree_lock
- Disastrous, most of CPU time spent spinning there

Cl_object_attr_lock

- cl_object_attr_lock can be converted to an rwlock
- CPU spinning drops, perf record looks good
- Performance gets worse by 40%
- Fairness issues: Prior to ~3.15 kernel, rwlocks are badly unfair, no queueing

Cl_object_attr_lock

- Osc_page_touch_at (a writer to the attrs) appears in the perf traces
- Less time spent spinning, but writer is blocked, which kills performance
- Newer rwlocks (3.15+) are queued & fair, should help… Though we might not need them.

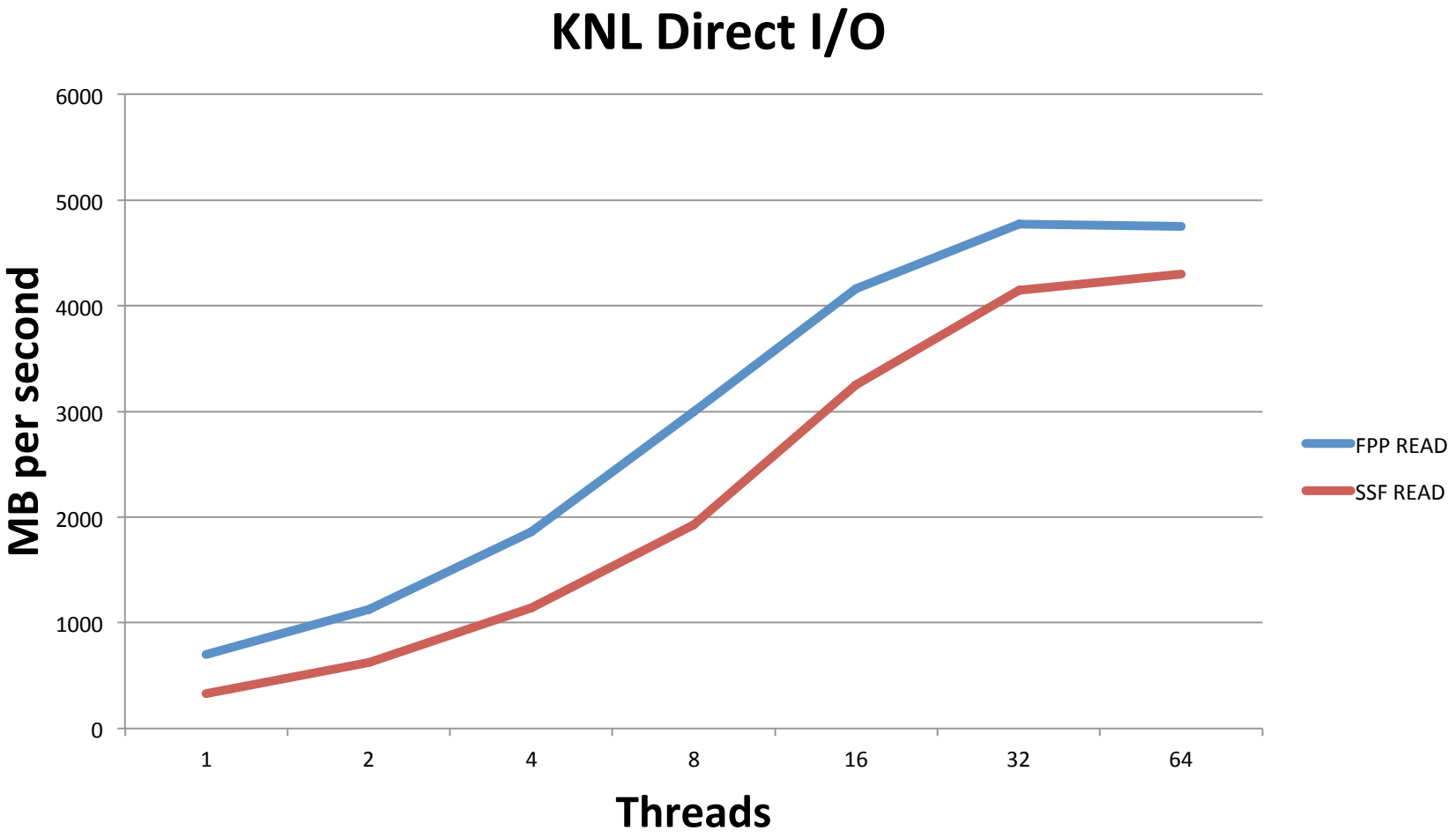
Mapping->tree_lock

- Tree_lock protects the radix tree for the page mapping for our file (not a problem in FPP, since there is a mapping per file)
- Nothing we can do directly, deep in the page cache
- Direct I/O!

Direct I/O

- Direct I/O writing is terrible, since we cannot do the usual asynchronous writing
- Direct I/O reading is great
- Bonus: avoids cl_env contention in FPP because it doesn't call ll_invalidate_page

Direct I/O


KNL Direct I/O
MB per second
6000
5000
4000
3000
2000
1000
0
1
2
4
8
16
32
64
Threads
FPP READ
SSF READ

KNL vs. Xeon: Locking Primitives

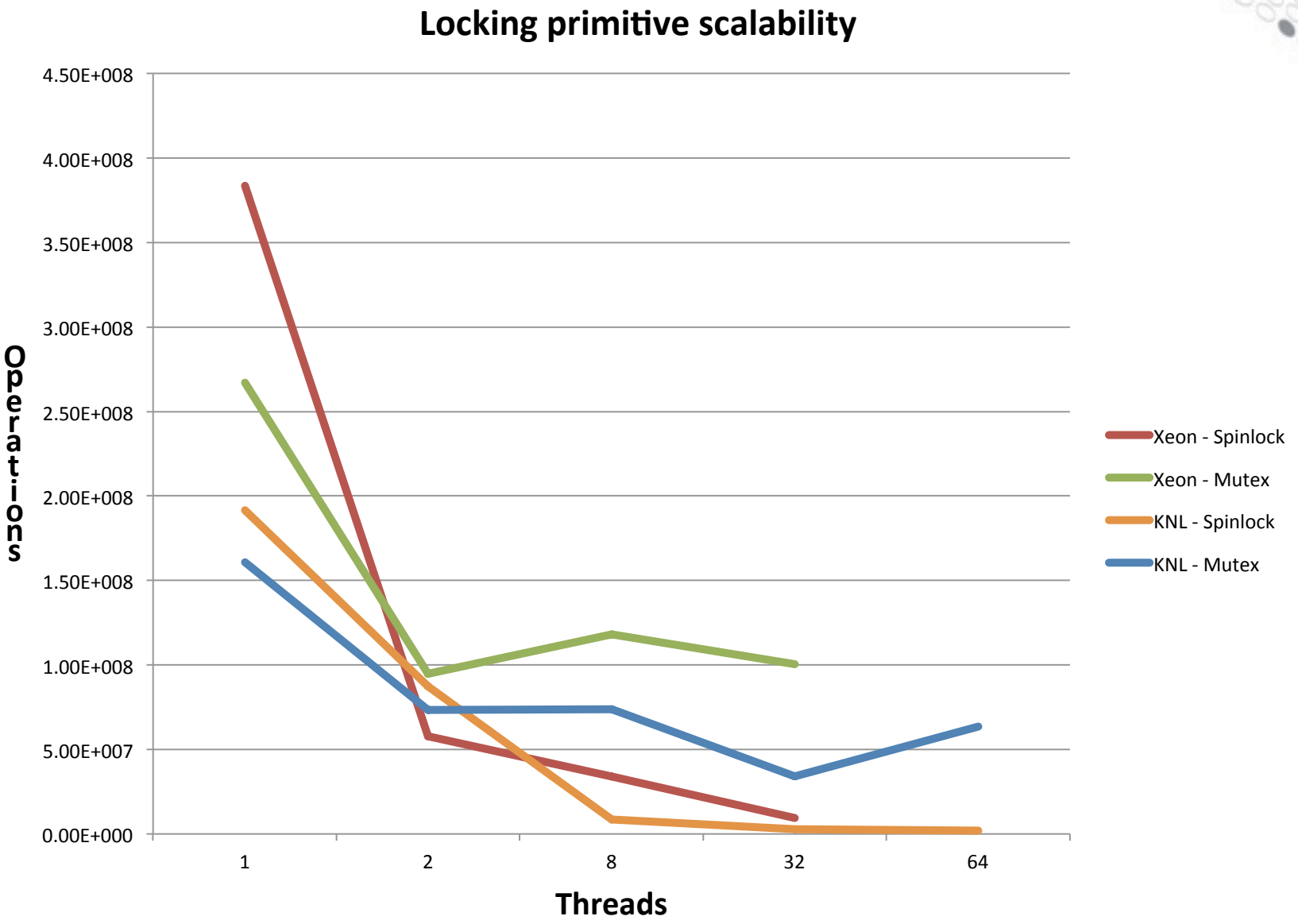
- That's it for Lustre directly.
- Let's talk locking primitives…

KNL vs. Xeon: Locking Primitives

- KNL is slower than Xeon (~30% of Xeon speed in scalar/control code)
- Solution: Add processes
- Lock contention is worse with > number of processes, kills performance
- “Just parallelize it” – Actual quote from someone at Cray about Lustre and KNL

KNL vs. Xeon: Locking primitives

- Important question:
  KNL is slower, but is it any worse at contention than Xeon?
- Trivial benchmark:
  Lock, increment, unlock, repeat until time is up. Kernel space.
- Implemented as Lustre patch to proc
  (Available on request, code quality… .....)


Locking primitive scalability
4.50E+008
4.00E+008
3.50E+008
3.00E+008
2.50E+008
2.00E+008
1.50E+008
1.00E+008
5.00E+007
0.00E+000
Operations
1
2
8
32
64
Threads
Xeon - Spinlock
Xeon - Mutex
KNL - Spinlock
KNL - Mutex

Locking Primitives

- Answer: No, KNL isn't any worse than Xeon
- Observations:
- Mutexes aren't hurt too much by contention >2, but have problems this doesn't show
- Standard spin locks are terrible under contention
- Atomic increment – Highly inconsistent results, seemed to be flat after 2 threads
- Rwlocks – not graphed – have the major problem of unfairness

Qspinlocks!

- Ideally, we'd remove all points of contention, but that's not practical
- Better locking primitives offer hope!
- Qspinlock, new spinlock implementation added in early 4.x kernel
- Very clever, avoids most contention on a single memory location
- Might make rwlock change irrelevant
- See https://lwn.net/Articles/590243/

Where's the beef?

- No KNL specific improvements except for cl_env change.
- Can't get newer kernels running on Cray hardware yet (Porting work ongoing)
- Tests in VMs with current kernels showed totally different results from hardware; not worth testing there
- Use newer kernels if you can

What's left?

- Contention is still bad in the page cache, can we do anything?
- Lockless page cache proposed ~8 years ago, never happened
- Some intriguing documents about lockless page cache on HPDD JIRA
- Why are 2 threads SSF not as fast as 2 threads FPP? (No visible contention client side)
- Adding threads other places – Cray gnilnd needed another worker thread on KNL

Finally:

- **Any questions?**
- Happy to answer questions later or by email (paf@cray.com)