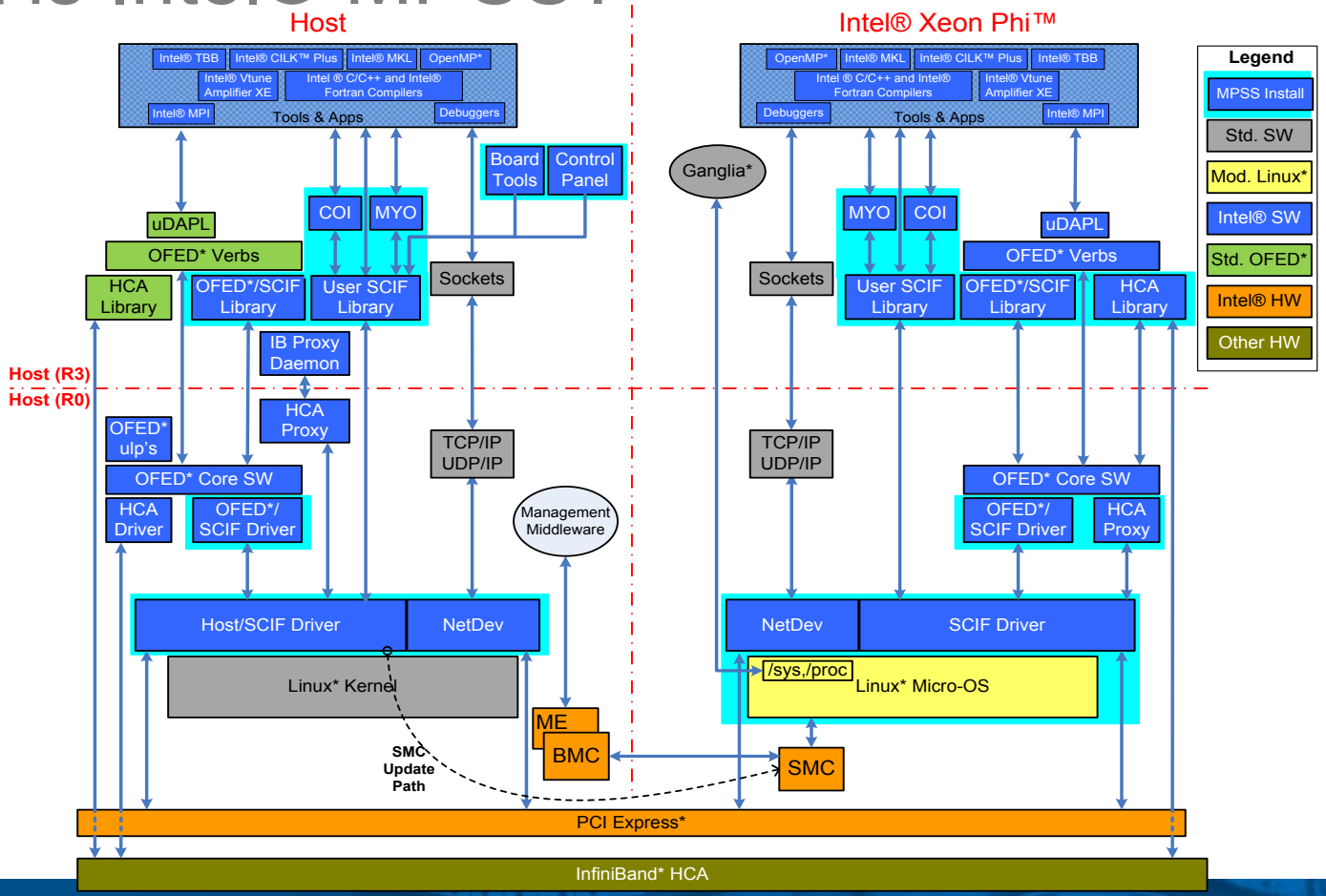
Manycore Platform Software Stack KNC Enhancements, Future Plans and KNL

Ravi Murty, Intel

July 8, 2014; IXPUG; Austin, TX

What is Intel® MPSS?



Intel® MPSS Roadmap for KNC

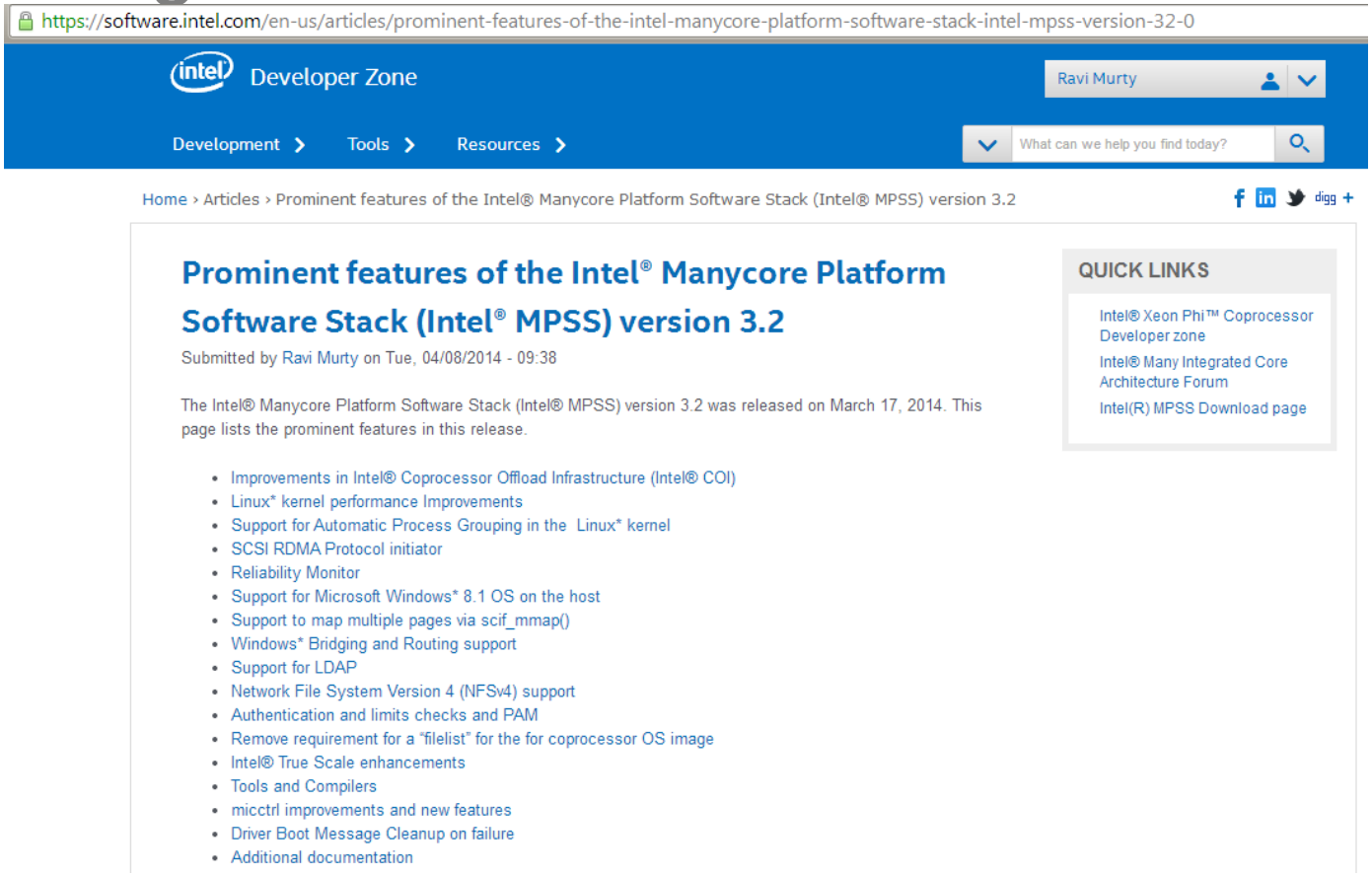
- MPSS 3.2
  - Cluster improvements based on feedback received at SC'13 BoF session
  - NFSv4 and LDAP support
  - - RHEL 6.0
- MPSS 3.3
  - OFED 3.5-2, MLNX 2.1 OFED
  - IPoIB with Mellanox* HCAs
  - Kernel mode RDMA clients – for cluster and storage traffic
  - New options to micctrl for verbose output – e.g. file system changes
  - Improvements in micmgmt API in terms of error reporting for RAS
  - - RHEL 6.1, + RHEL 7.0
- COI (offload compiler runtime) improvements
  - Bi-directional DMA, Multi-D data transfers
- MPSS 3.4 (Late Q3) and MPSS 3.5 (Late Q4)

Coprocessor RPMs

- Newer Intel® MPSS versions have a number of pre-built RPMs for the coprocessor
  - Common packages of interest include – gcc toolchain, python, perl etc
  - Environment on MIC can be customized and made permanent via several methods
  - ~50 RPMs available in MPSS 3.2

See Intel® MPSS “Prominent Features” blogs per release

Tracking New Features in Intel® MPSS



Performance Improvements in MPSS (OS improvements for Many Core Architectures)

Parallel AP Boot

Physical (HW) IDs
SW IDs (kernel assigned)

- Standard Linux kernel boots the bootstrap processor (BSP) first
  - Send INIT-IPI/SIPI to each CPUs (HW thread), *one at a time*
  - CPUs wake up, join the kernel and “ack” their presence
- Why can’t we do this in parallel by broadcasting the SIPI?
  - How do we prevent the APs from stomping on top of each other?

Remarkably faster coprocessor boot times (105 sec -> 15 sec)

File IO Improvements

https://software.intel.com/en-us/blogs/2014/01/07/improving-file-io-performance-on-intel-xeon-phi

intel Developer Zone

Ravi Murty

Development › Tools › Resources ›

What can we help you find today?

Home › Blogs › Improving File IO performance on Intel® Xeon Phi™ Coprocessors

Improving File IO performance on Intel® Xeon Phi™ Coprocessors

Submitted by Rajesh Sudarsan... on Wed, 03/05/2014 - 08:57

Authors: Ravi Murty and Rajesh Sudarsan

Introduction

The importance of file IO in HPC applications is well known. Large HPC applications read their inputs from files and write their outputs to files throughout the lifetime of their execution. In some instances the output of an application is reused as input in subsequent runs of the same or different application and the data is shared via files. Additionally, applications running on large number of nodes in a cluster store a snapshot of their execution state periodically as *checkpoint* data. This checkpoint data can be used to restart the application in the event that the application is somehow interrupted, for example because of a node failure in the cluster. Other simple usage models that demand good file IO performance include simple system administration activities like installation of packages (e.g. RPMs).

These types of usage models demand good file IO performance from the underlying operating system. In this article we describe our investigation of system calls used for file IO, like *read(2)* and *write(2)*, metrics used to measure their performance and the optimizations implemented that improved their performance on the Intel® Xeon Phi™ coprocessor.

The Virtual File System (VFS) layer

Before we get into the details of our investigations and describe the improvements made on the Intel® Xeon Phi™ coprocessor, let's take a quick look at the VFS layer in Linux*. As shown in Figure 1, the VFS is the generic file system layer in the kernel that provides the file system interface to application

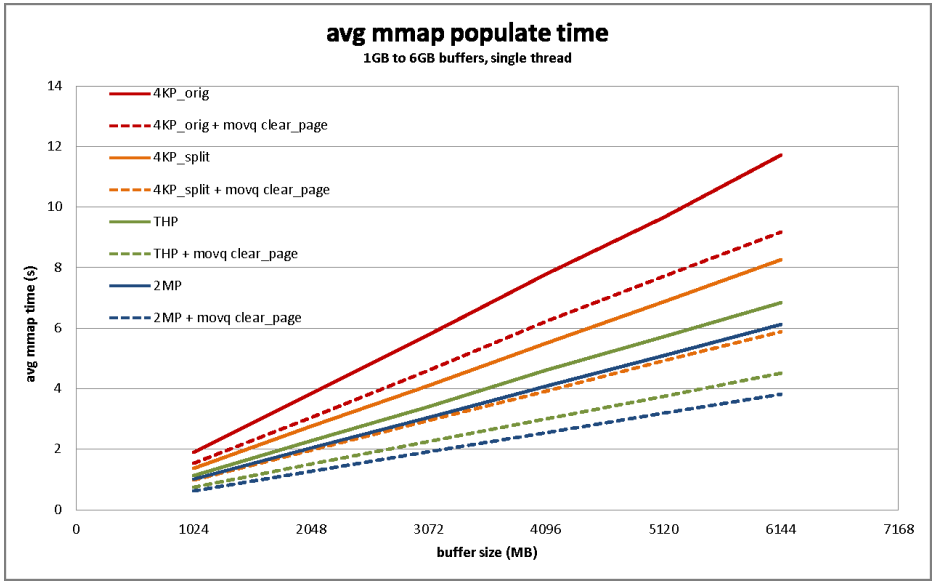
Key Optimizations:

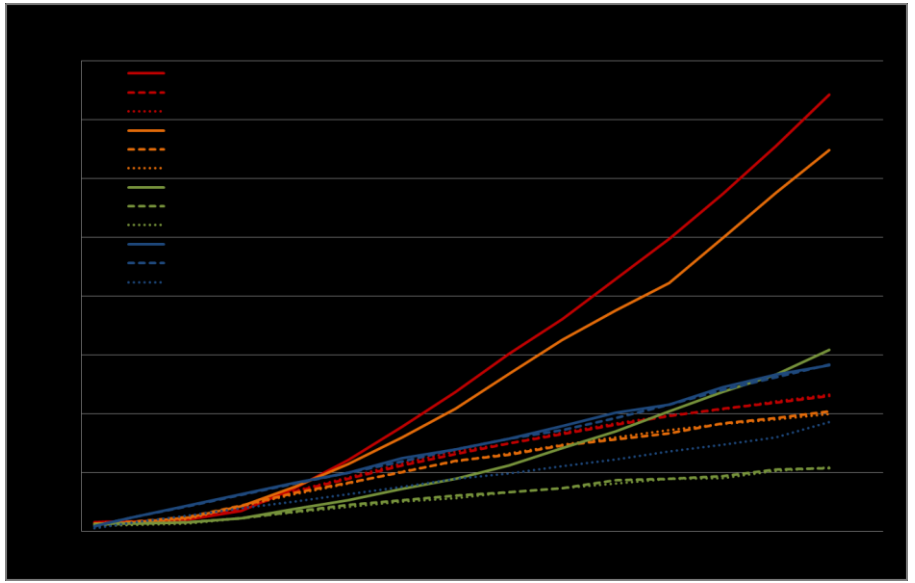
- Speeding up "memcpy"
- Pre-computing page wait-queue hash
- Use of high order pages for page cache allocations

2.5X – 3.8X Improvement AMB IOZone

mmap() improvements

- THP support was added with 2.6.38 for KNC
- ST performance of mmap(MAP_POPULATE) isn't very good.





Memory allocation is critical to application performance: MPI and offload

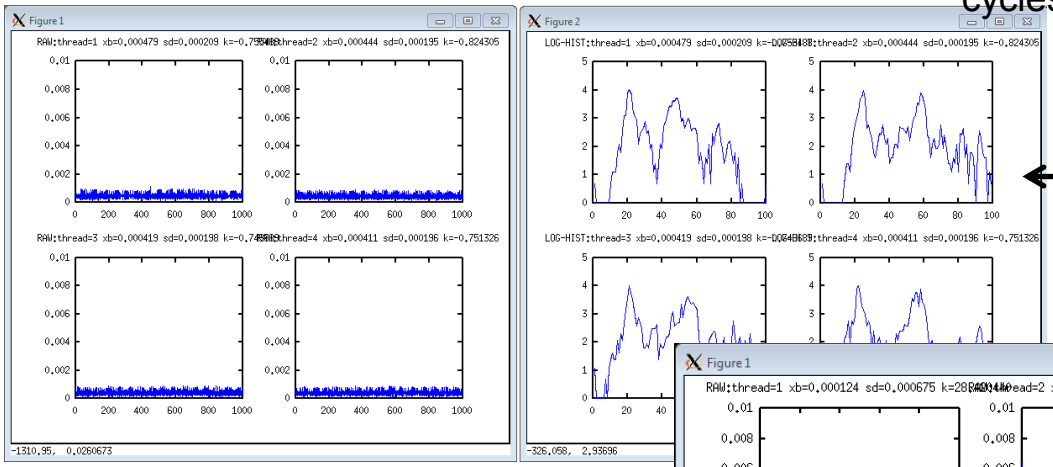
Is OS Jitter a problem for you?

- We understand the impact of OS noise on large clusters
  - Run-to-run variability during debug matters too!
- Platform vs. OS noise
  - Cache misses, TLB are important. But we're interested in OS noise
- FWQ/FTQ – simple, single node benchmark
- PSNAP – PAL System Noise Activity Program from Los Alamos
  - Depends on MPI but doesn't depend on its performance (this is perfect)
- HOMME initial results
- Improvements will be applicable to KNC, KNL and future many core arch

Do you have OS jitter sensitive apps that you can share with us?

Fixed Work Quantum

- Work quantum (w) is something simple like inc; cmp; jnz
- On KNC, with w = 20: 2^20 * 36 + 13 = **37748749** cycles



Linux kernel
2.6.38 + KNC patches

vmstat and load balancer disabled

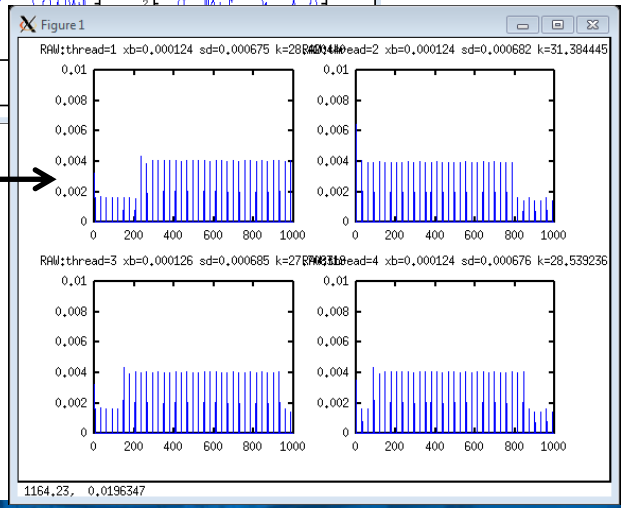
37797772
37784277
37795635
37796806
37796501
37786017
37797071
37793456
37785946

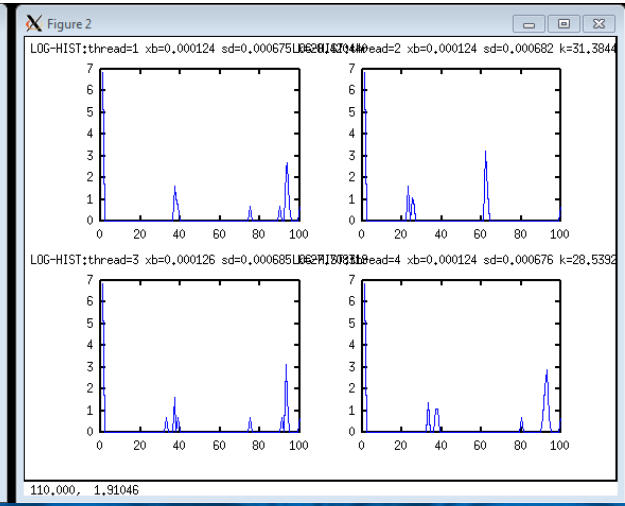
37748749
37748749
37748749
37748749
37748749
37748749
37748749
37811964
37748751
37748749

Linux kernel
3.13 + KNC patches

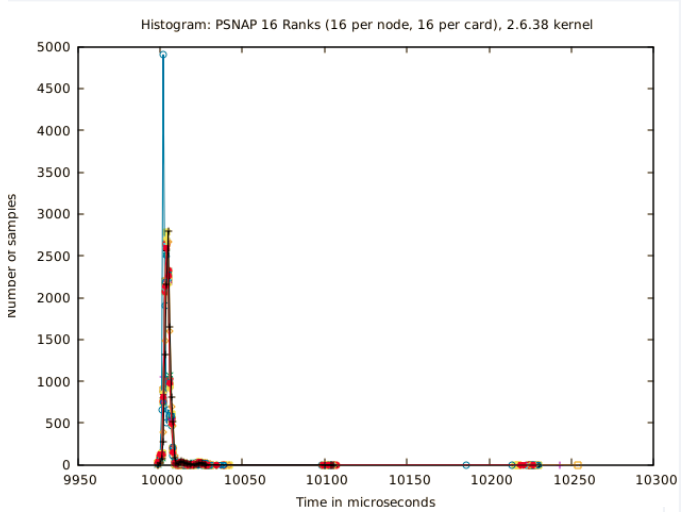
CONFIG_NO_HZ_FULL

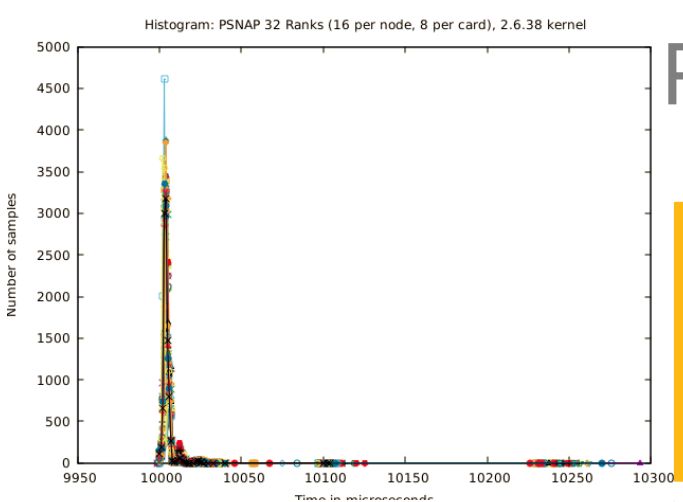
highres =on nohz_full = 1-239, isolcpus = 1-239 rcu_nocbs=1-239



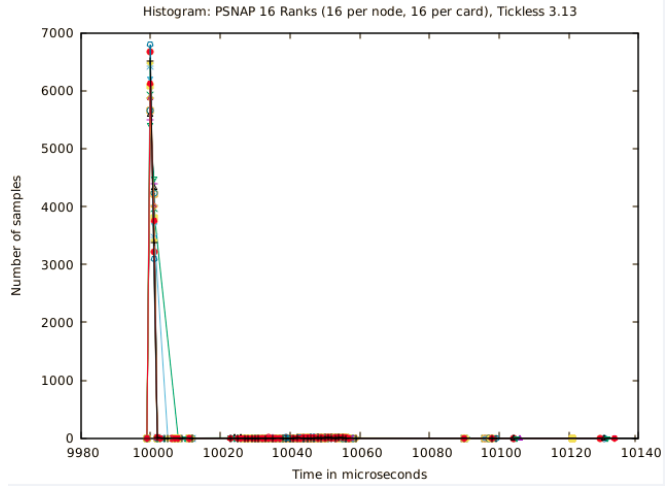


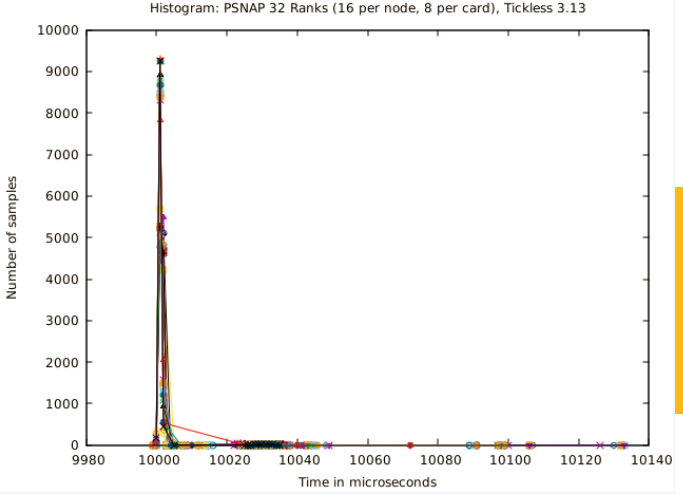
PSNAP 1.2





2.6.38:
Only about 30% of samples are at 10010us



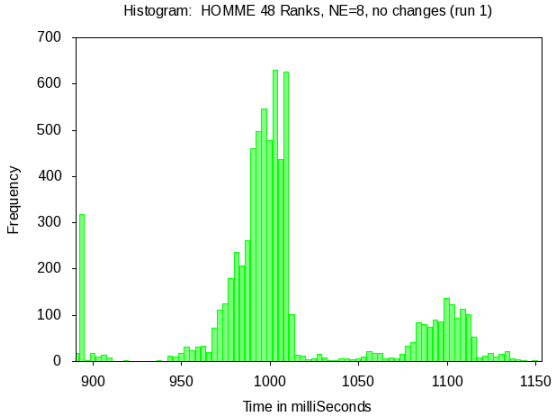


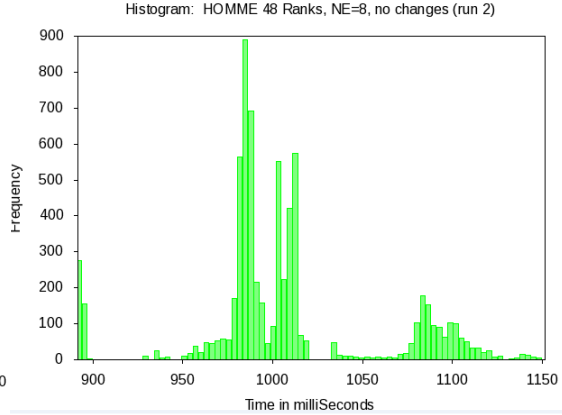
3.13:
Most samples are at 10000 - 10002us

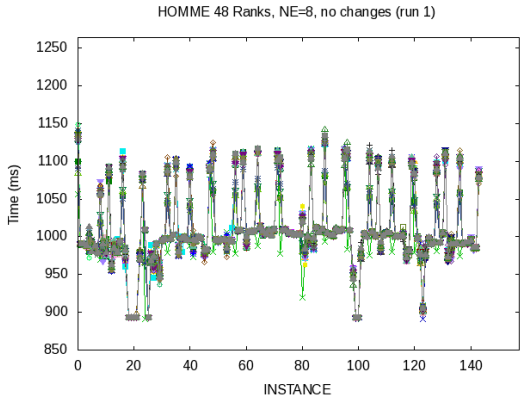
HOMME

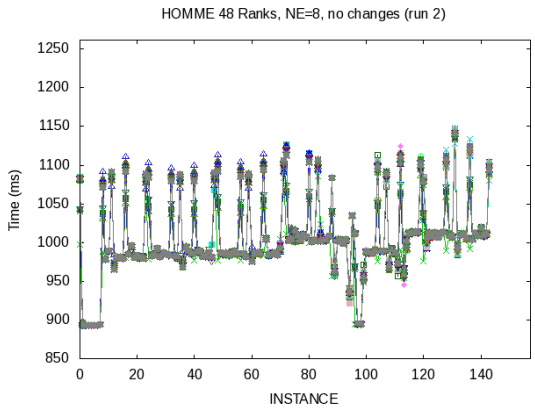
Large Variability, run-to-run and iteration-to-iteration

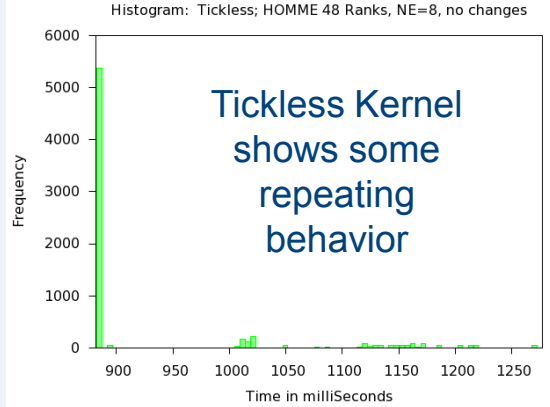
NCAR expected tall skinny Histogram



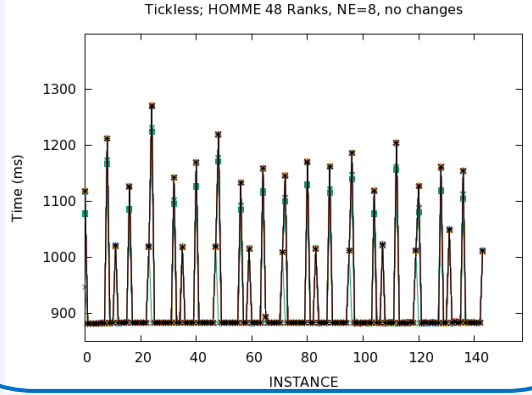








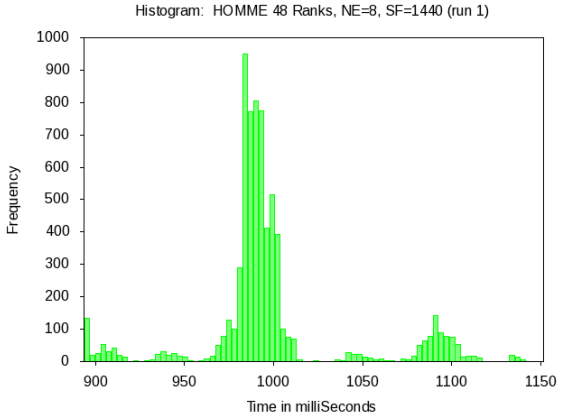
Tickless Kernel shows some repeating behavior

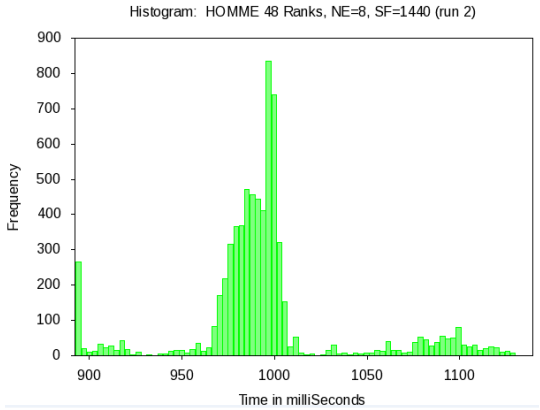


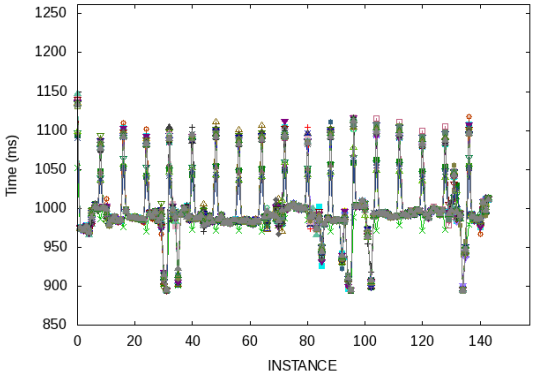
HOMME - II

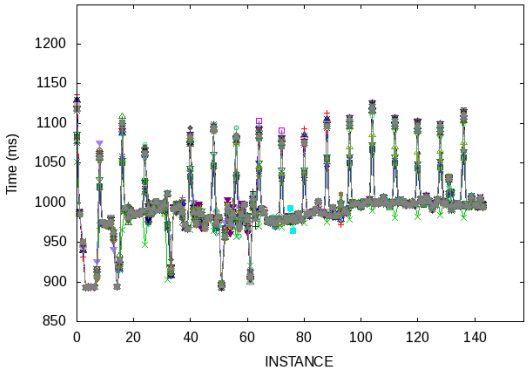
Analysis Found Diagnostics Enabled during run, removed via SF=1440 in namelist

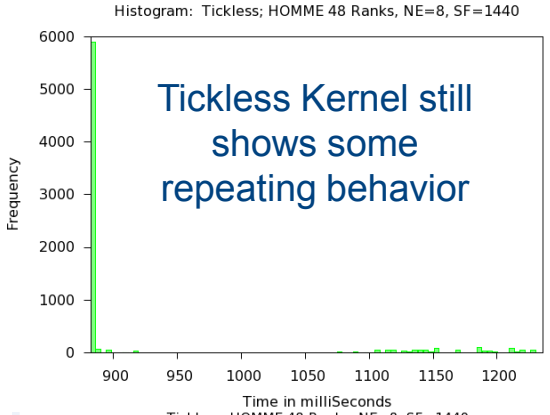
Still Large Variability, run-to-run and iteration-to-iteration

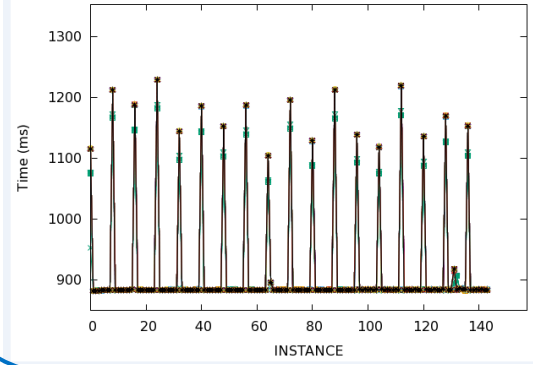








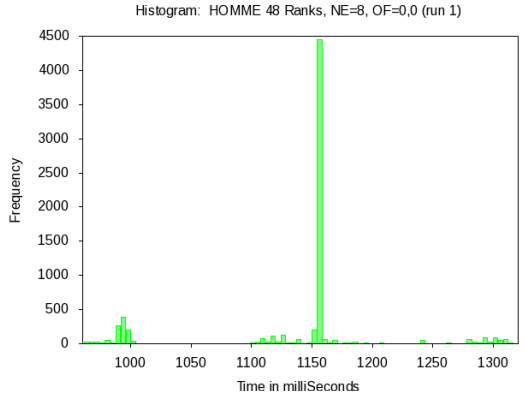


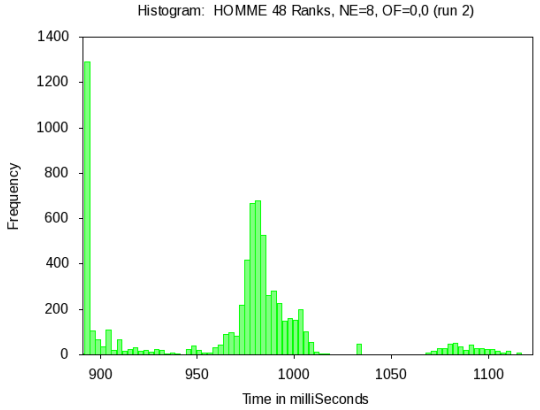


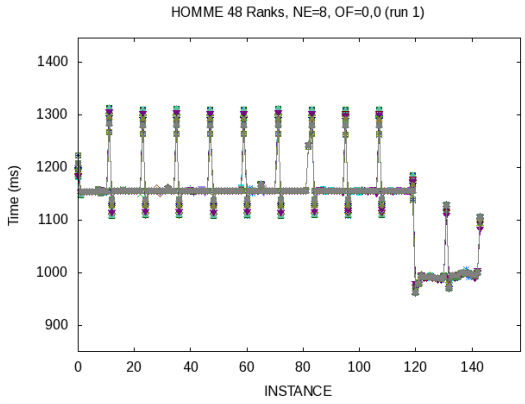
HOMME - III

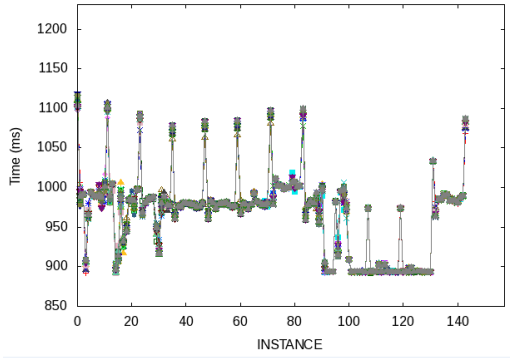
Analysis Found I/O was contributing to variation, removed via OF=0,0 in namelist

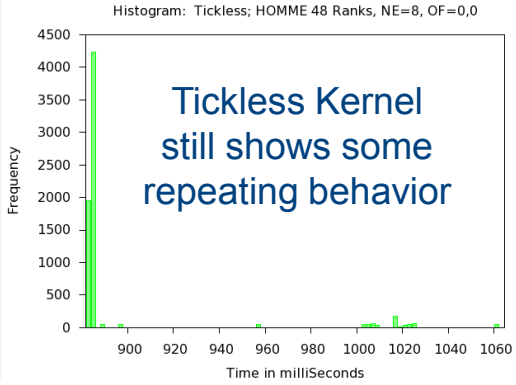
Still Large Variability, run-to-run and iteration-to-iteration



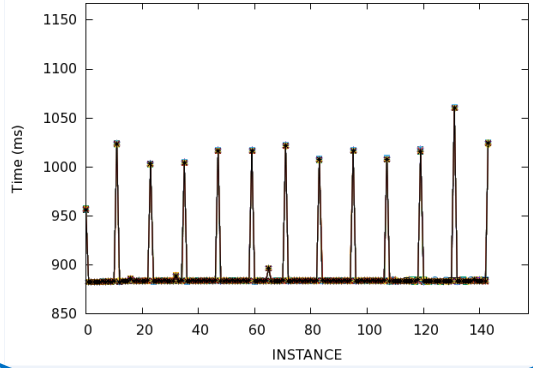








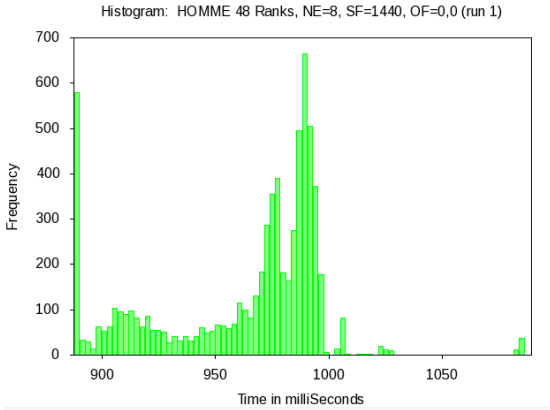
Tickless Kernel still shows some repeating behavior

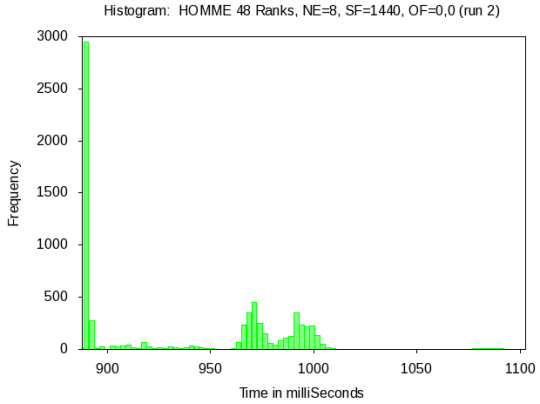


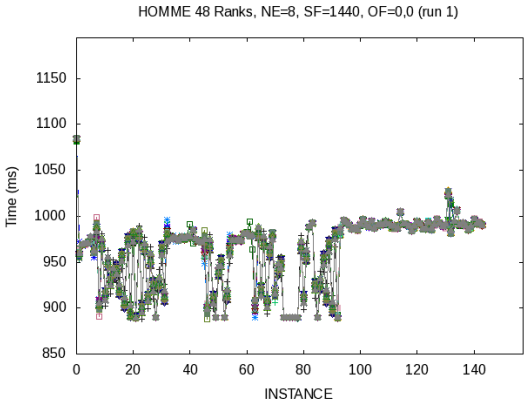
HOMME - IV

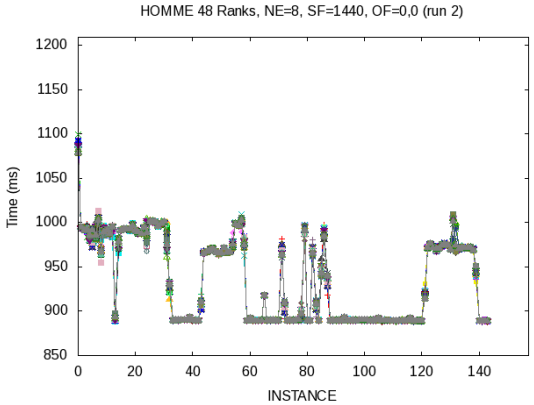
Removed both source of variation (diagnostics + I/O) via SF=1440, OF=0,0 in namelist

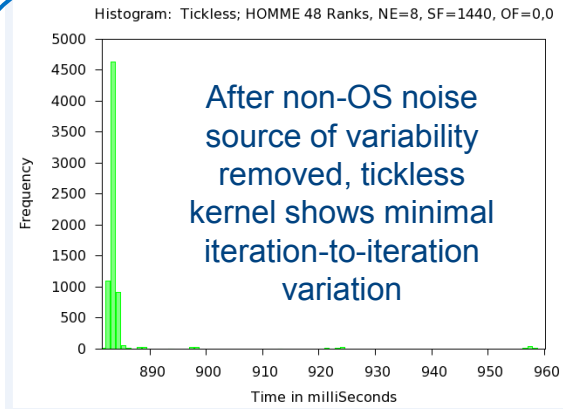
Still Large Variability, run-to-run and iteration-to-iteration (OS noise)



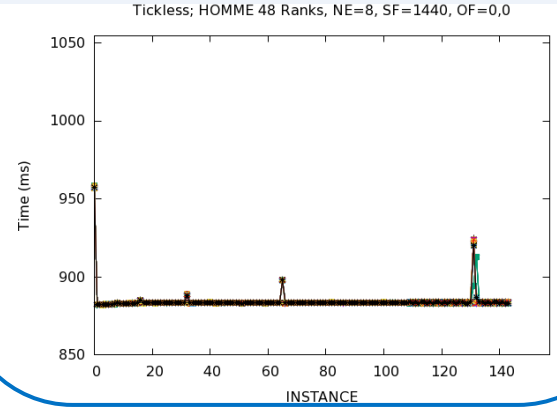








After non-OS noise source of variability removed, tickless kernel shows minimal iteration-to-iteration variation



Knights Landing (KNL)

- Many core architecture with greater than 60 cores
  - Based on Intel® Silvermont microarchitecture with many enhancements
- 3X single-thread performance compared to KNC
- Up to 16GB of on-package high BW memory – 5x memory BW of DDR4
- Integrated high-speed fabric called Intel® Omni Scale

Knights Landing (KNL) Software

- No MPSS on KNL bootable processor!
  - Compilers, Debuggers, vtune etc.
  - MCDRAM library for allocating from HBW memory.
- Goal: Standard “shrink wrap” OS (RHEL 7.1, Suse 12.1)
  - Intel®AVX-512 state save/restore among other things
  - X2APIC mode
  - ACPI support for MCDRAM NUMA nodes
- Binary compatibility with Xeon (except TSX)
- Adaptive tickless kernel will need CONFIG_NO_HZ_FULL

Code modernization (MPI, OMP) effort spent on KNC will carry forward:

- Thread parallelization, Vectorization, cache-blocking
- Additional tuning will be required to take advantage of KNL

Feedback and Q/A

- Are there areas you think we need to focus on?
- What would you like to see on KNL?

- Questions? email ravi.murty@intel.com

ravi.murty@intel.com

Backup

RPMs for coprocessor

- Newer Intel MPSS versions have a number of pre-built rpms available for the coprocessor to extend native coprocessor user environment
- Common packages that may be of interest
  - Native gcc toolchain, Python, Perl, Bash, Ganglia
  - *~50 RPMS available with MPSS 3.2 (*see “Software for Coprocessor OS” section on MPSS download page*)*
- You can customize your environment temporarily, or make the associated RPM installs permanent via several methods:
  - Use micctrl to add desired rpms at boot time
  - Snapshot the initrd image and reuse on desired Intel Xeon Phi coprocessors
- In both cases, you will likely want to create a local repository first to provide automatic dependency resolution

Coprocessor Side RPM customization

Since gcc is a common request, here's a quick example using packages from the Intel MPSS 3.2 release:

1. Boot a Coprocessor and make the K1OM RPMs available locally (via NFS or scp to coprocessor)
2. Install coreutils and dependencies:

```
[root@mic0]# cd <dir-where-you-have-all-k1om-rpms>
[root@mic0]# rpm -ihv coreutils*.rpm libgmp*.rpm
```

3. Build local repo to handle dependencies:

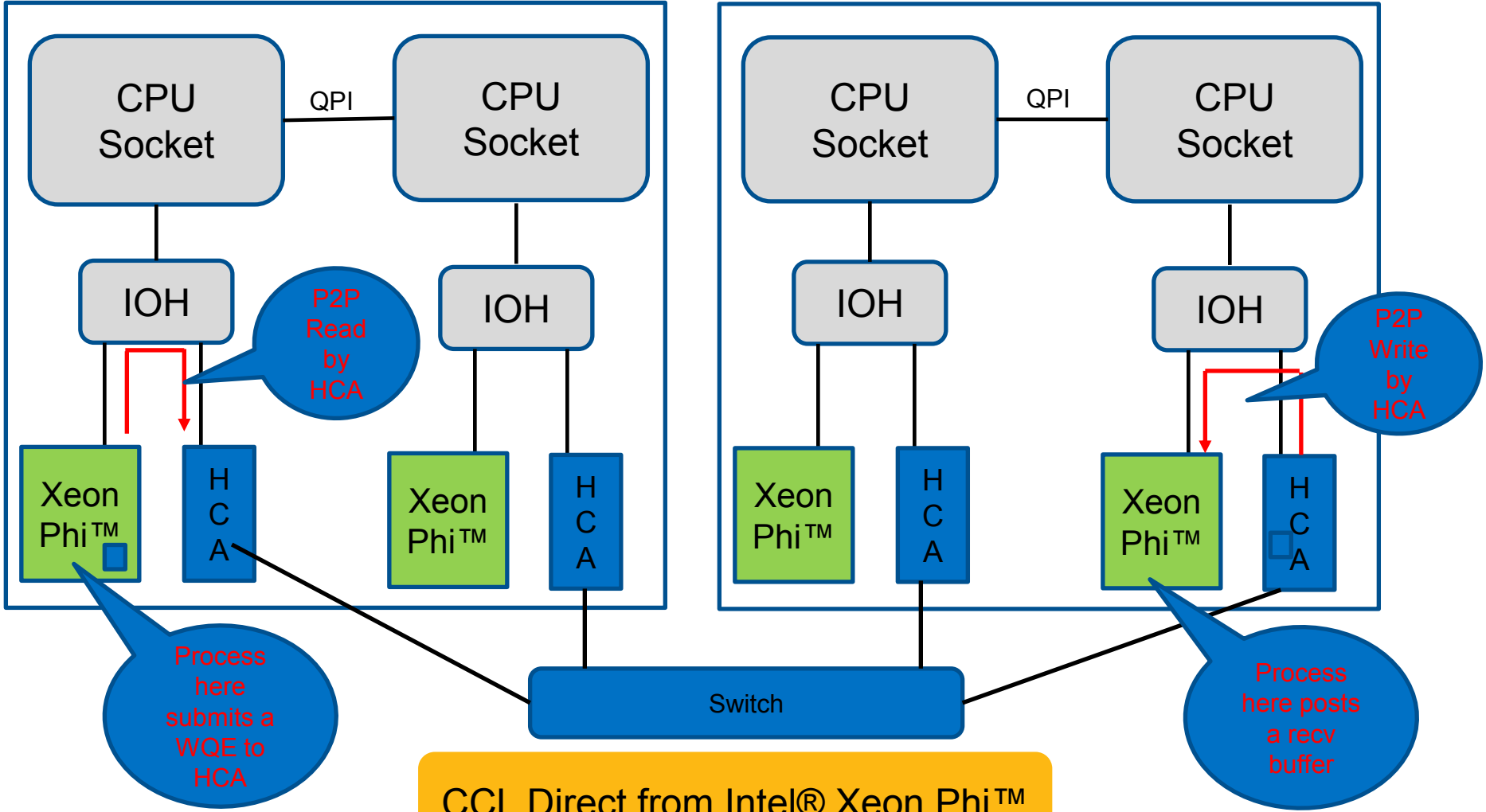
```
[root@mic0]# zypper ar . Mpss
```

4. Install desired components (using gcc as an example):

```
[root@mic0]# zypper install gcc-symlinks-4.7.0+mpss3.2-1.k1om.rpm
```

5. [root@mic0]# gcc --version

   gcc (GCC) 4.7.0 20110509 (experimental)



CCL Direct from Intel® Xeon Phi™

CCL-Proxy Cross-Socket (SNB)