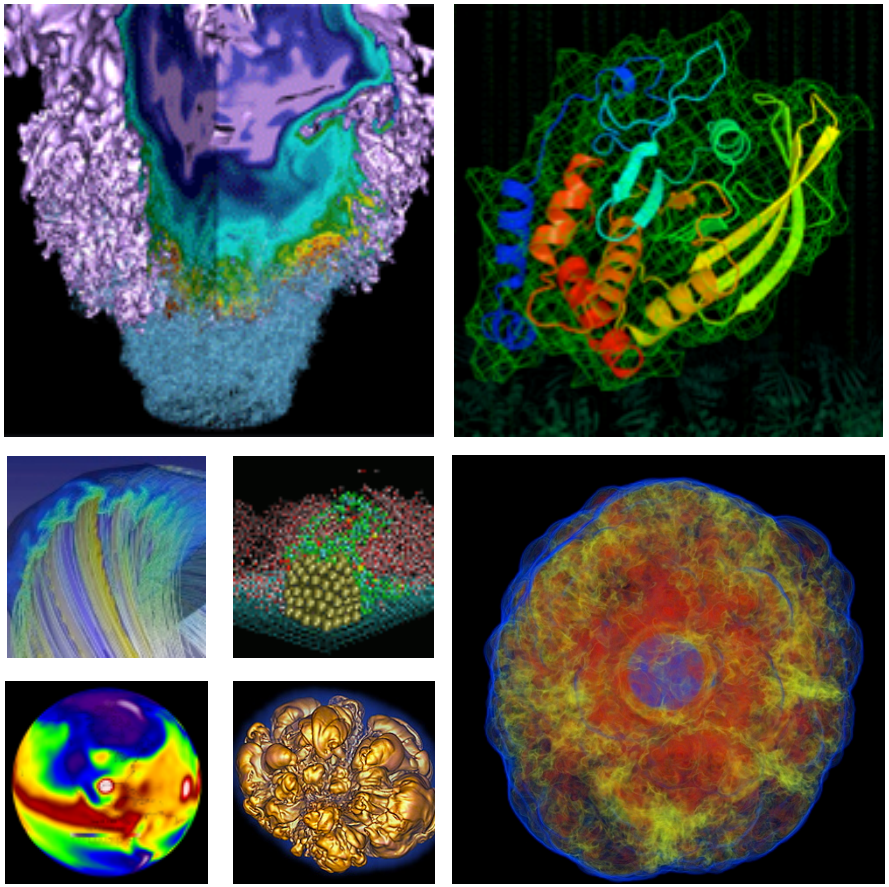
NESAP Update



NERSC Application Readiness Team

March 31, 2015

NERSC Dungeon Session Completed

- Wednesday – Friday March 18-20, Hillsboro, OR
- Codes: MFDn, BerkeleyGW, CESM, EMGeo + 1
- 7 NERSC staff + 1 Cray + 1 additional LBNL staff
- Nathan (Cray) + [Max/Ruchira/John/Kartik/Gaurav/DmitryR/DmitryP/Martyn/SarahK/Nadezhda/Rakhi/MikeG/ChrisC/Anthony/Rezaur/BBurroughs/Avinash/Jeongnim]: 18+ Intel
- Significant Intel preparation in advance

General Observations

- Having several codes at once worked well
- Need code expert present
- Detailed processor architecture exploration!
- Code kernel useful for fast repeats but needs to be weighed against work involved and context lost
- Might not get immediate performance boost but learn a lot about code, tools, and Intel architecture
- Next DS: may have KNL; expect to choose codes that can make best use of it; solid understanding of performance required + reasonably good scaling

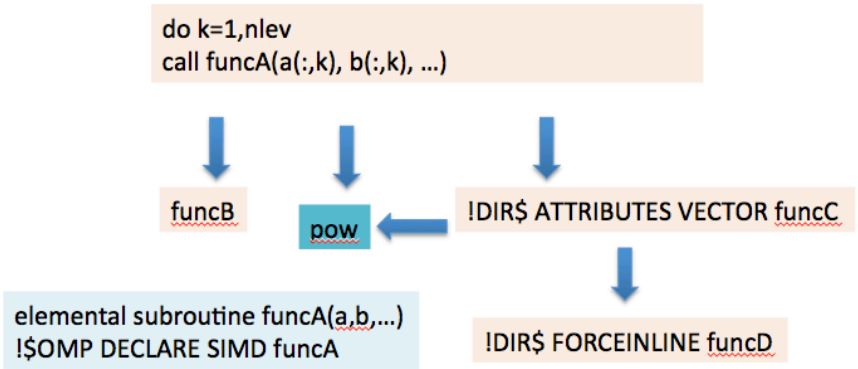
More Specific Examples

- What you can learn from the compiler
- What you can learn from VTune
- How you can run VTune at NERSC on Edison

Make Sure Your Code is Vectorized

- **Vectorization is another important level of parallelism after MPI and OpenMP for KNL.**
  - https://www.nersc.gov/users/computational-systems/edison/programming/vectorization/
  - https://software.intel.com/en-us/articles/explicit-vector-programming-in-fortran
- **Use compiler report to check and make sure key functions are vectorized (and all functions on the call stack are vectorized too)**
  - Elemental functions need to be inlined
  - "-ipo" is needed if functions are in different source codes.
  - "-qopt-report=5" reports highest level of details: loop optimization, alignment, ipo, inline, dependency, vectorization, etc.
- **Add !$OMP DECLARE SIMD, !DIR$ ATTRIBUTES VECTOR, and $!DIR$ FORCELINE when needed.**

Example call stack for vectorization and inlining



Questions Answered at The Dungeon

Is your code memory bandwidth/latency or compute bound?

VNC config

<no current project> - Intel VTune Amplifier

Welcome | test_ge

General Exploration General Exploration viewpoint (change)

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up | Top-down Tree | Tasks and Frames | gppkernel.d...

Grouping: OpenMP Region / Function / Call Stack

OpenMP Region / Function / Call Stack	Clockticks	Instructions Retired	CPI Rate	MUX Reliability	Filled Pipeline Slots		Unfilled Pipeline Slots (Stalls)			Pot... Gain	Pot.. Gain (% of Col... Tim..	Elaps... Time	Nu. of Ope. thr..	Ins.. Cou.	Mod.	Fun.. (Full)	Sou.. File	Start Address
							Back-End Bound											
					Retiring	Bad Specul...	Memory Bound	Core Bound	Front-... Bound									
MAIN_ompparallel:18@unknown:281:529	1,757,836,636,751	2,159,187,238,776	0.814		0.328	0.001	0.141	0.731	0.003	0.926s	1.9%	39.849s	18	512				0
MAIN_ompparallel:18@unknown:67:74	3,946,005,919	4,100,006,150	0.962		0.245	0.004	0.105	0.357	0.004	0.233s	0.5%	0.234s	18	1				0
MAIN_ompparallel:18@unknown:206:260	297,620,446,430	194,362,291,543	1.531		0.162	0.001	0.621	0.305	0.010	0.160s	0.3%	6.751s	18	512				0
[Serial - outside any region]	13,782,020,673	11,316,016,974	1.218		0.258	0.000	0.392	0.302	0.021		0.0%	1.465s						0

Learned that Vtune can tell you, for each OpenMP region, the percent of cycles retired that are either memory bound or core bound. It can also tell you which level of memory you are bound by.

- **The heaviest loop in the above code is core-bound (due to divisions), not memory bound.**

Questions Answered at The Dungeon

Why is complex division so slow and what can be done about it?

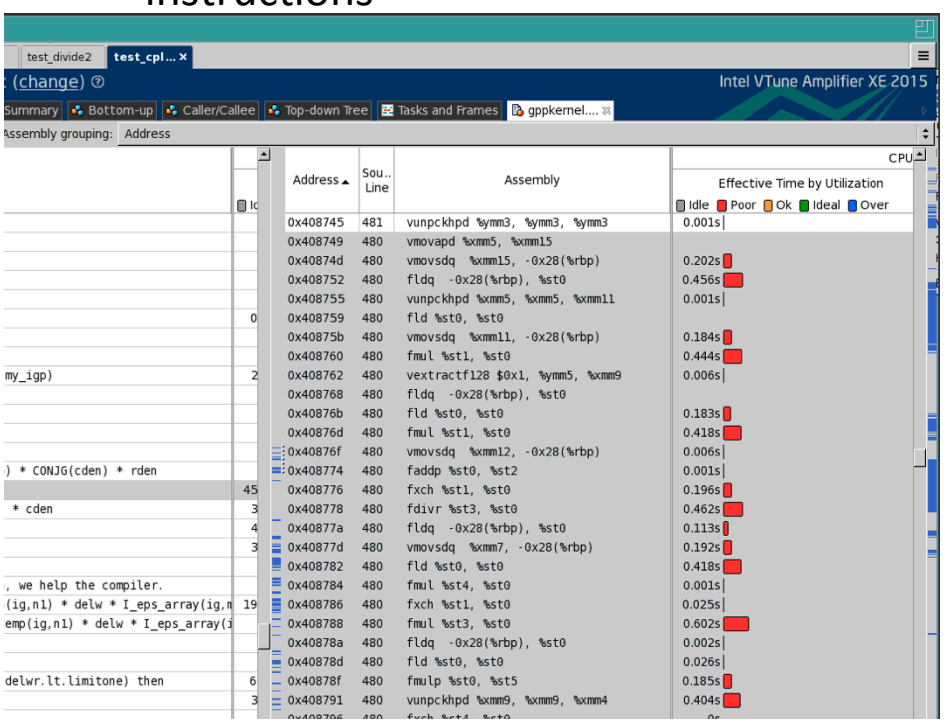
Why is real division so much faster?

```
c = 1 / c
```

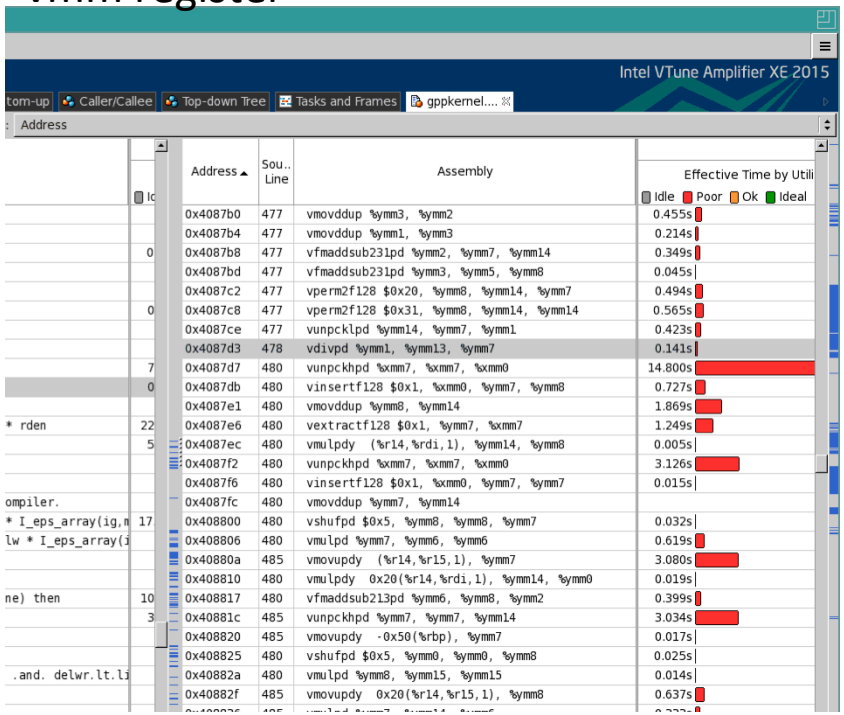
vs.

```
r = c * conjg(c)
r = 1 / r
c = conjg(c) * r
```

Complex division yields many x87 instructions



Real division yields vector instruction vmm register



Insight into Memory Subsystem (1)

- **VTune General Exploration: Expand collected metrics to see more detail on what levels of the memory hierarchy you are bound (if at all)**

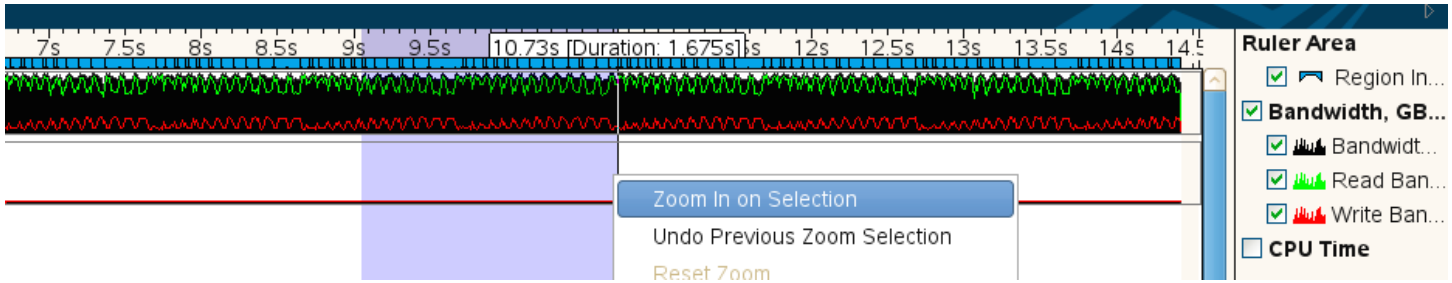
Clockticks	Instructions Retired	CPI Rate	MUX Reliability	Filled Pipeline Slots		Unfilled Pipeline Slots (Stalls)	
				Retiring	Bad Speculation	Back-End Bound	Front-end Bound
43,126,064,689	62,090,093,135	0.695					
28,364,042,546	32,782,049,173	0.865					
23,518,035,277	28,584,042,876	0.823					
38,104,057,156	22,110,033,165	1.723					
9,134,013,701	5,266,007,899	1.735					

Clockticks	Instructions Retired	CPI Rate	MUX Reliability	Filled Pipeline Slots		Unfilled Pipeline Slots (Stalls)						
				Retiring	Bad Speculation	Back-End Bound						Front-end Bound
						Memory Bound				Core Bound		
						L1 Bound	L3 Bound	DRAM Bound	Store Bound	Divider	Port Utilization	
43,126,064,689	62,090,093,135	0.695										
28,364,042,546	32,782,049,173	0.865										
23,518,035,277	28,584,042,876	0.823										
38,104,057,156	22,110,033,165	1.723										
9,134,013,701	5,266,007,899	1.735										

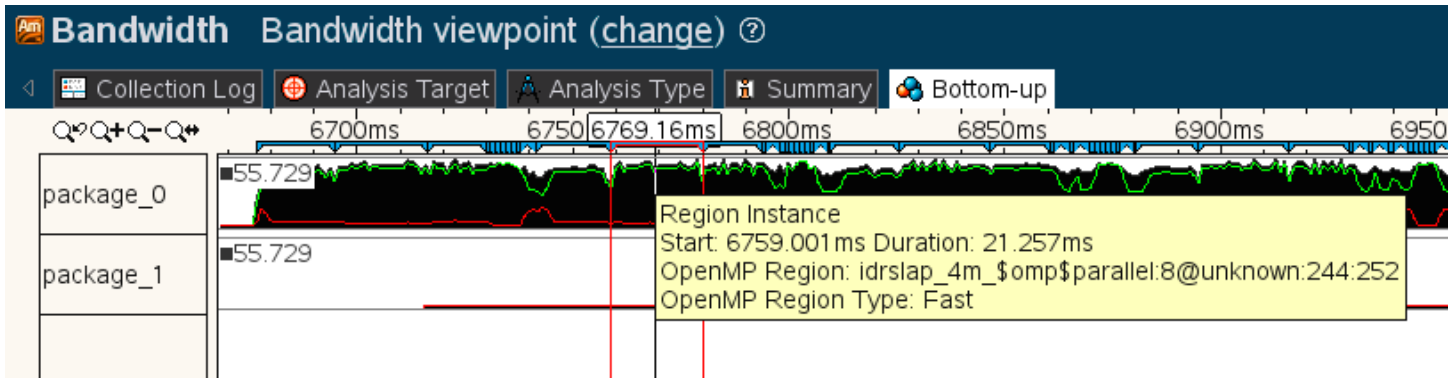
- **Switch to Hardware Event Counts viewpoint to see the underlying event counts (cache misses, etc.)**

Insight into Memory Subsystem (2)

- **VTune Bandwidth: Utilization as a function of time; Per-package (NUMA node) read and write, zoom-able**



- **Mouse-over to trace back to *specific OMP regions* in your code**
  - Helpful in discovering which computations might benefit from MCDRAM for working set (but remember to also check for locality issues affecting BW)



VTune Tips and Tricks Learned at Dungeon

- **Want to profile a single code region?**

```
Example:
  #include "ittnotify.h"
  __itt_resume();
    ... code to profile ...
  __itt_pause();

Compile with: -I$(VTUNE_AMPLIFIER_XE_2015_DIR)/include
Link with: $(VTUNE_AMPLIFIER_XE_2015_DIR)/lib64/libittnotify.a
Run with: amplxe-cl -start-paused ...
```

- **Want a list of all available counters?**
  - **Option 1**: Create new project / analysis in VTune GUI (New Hardware Event-based Sampling Analysis); Click Add Event and search
  - **Option 2**: Read the db file directly for the relevant architecture:

```
See: $VTUNE_AMPLIFIER_XE_2015_DIR/config/sampling3/<arch>_db.txt

Example on Edison:
  less $VTUNE_AMPLIFIER_XE_2015_DIR/config/sampling3/ivytown_db.txt
```

Estimating memory BW on <arch>

- **This can be done using the ratio of bytes-loaded (last-level cache misses) to retired uOps (measured on Edison Ivybridge, for example):**

```
estimated_BW = 64 bytes per cache line *
               (OFFCORE_RESPONSE.ALL_READS.LLC_MISS.LOCAL_DRAM_0 /
                UOPS_RETIRED.ALL) *
               arch_freq *
               arch_ops_per_clock *
               arch_concurrency
```

- **This provides an upper bound on BW utilization: Assumes every clock cycle retires** `arch_ops_per_clock` **uOps**
  - Likely fewer in practice when waiting on memory subsystem

VTune is Available on Edison at NERSC

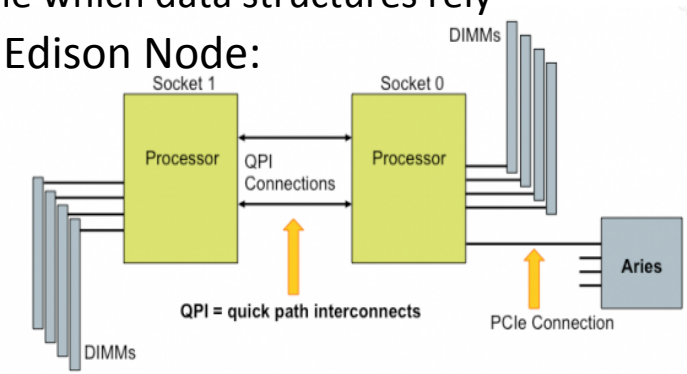
- **VTune is a performance analysis tool that can provide rich performance insight into hotspots, threading, locks & waits, bandwidth and more. Use powerful analysis to sort, filter and visualize results on the execution timeline and on your source.**
  - For serial and parallel codes – on-node performance
  - There are both GUI and command line interfaces.

- **To get started with VTune on Edison, see**
  - https://www.nersc.gov/users/software/debugging-and-profiling/vtune/#toc-anchor-4
- **A guide to optimizing codes for Ivy Bridge processors (Edison)**
  - https://software.intel.com/sites/default/files/Using_Intel_VTune_Amplifier_XE_on_Xeon_E5v2_or_E7v2_Family_1.0.pdf
- **Intel VTune website**
  - https://software.intel.com/en-us/intel-vtune-amplifier-xe

HBM Tools are Available on Edison

- **The memkind library is a user extensible heap manager. It can be used to test/simulate the benefit of high bandwidth memory (HBM or HBW) on the dual-socket Edison compute nodes today.**
  - This is not an accurate model of the bandwidth and latency characteristics of the KNL on package memory, but is a reasonable way to determine which data structures rely critically on bandwidth.
  - Use QPI bus to simulate slow memory
  - Code change is required:
    !DIR$ ATTRIBUTES FASTMEM directive

Edison Node:



- **To get started on Edison, see**
  - https://www.nersc.gov/users/computational-systems/cori/preparing-for-cori/using-high-performance-libraries-and-tools/#toc-anchor-1
- **Reference**
  - http://memkind.github.io/memkind
  - http://memkind.github.io/memkind/memkind_arch_20150318.pdf
  - http://ihpcc2014.com/pdf/100_KNL_HPC_Developer_Forum_SC_14.pdf

More HBM Tools Will be Available Soon

- **New features available in VTune – identify candidate arrays for HBM**
  - available in May, 2014
  - This can be used with the memkind library

- **Auto HBMT – automatically move arrays of a certain size to the HBM at run time**
  - No code change is required

SDE Available on Edison Under CCM

- **Intel Software Developer Emulator (SDE)**
  - Supports the extended instruction sets for KNL (AVX512)
  - Allows developers to use currently available compilers and assemblers on the currently available processors (Edison) to gain insights about whether applications are ready to take advantage of the opportunities created by the new instructions available on the future architecture (KNL)
- **To get started on Edison, see**
  - https://www.nersc.gov/users/computational-systems/cori/preparing-for-cori/using-high-performance-libraries-and-tools/#toc-anchor-2
- **Reference:**
  - https://software.intel.com/en-us/articles/intel-software-development-emulator
- **A method to evaluate the flops of your application codes:**
  - https://software.intel.com/en-us/articles/calculating-flop-using-intel-software-development-emulator-intel-sde

Some Background Info

- **Important Intel tutorial**
  - https://software.intel.com/en-us/articles/how-to-tune-applications-using-a-top-down-characterization-of-microarchitectural-issues
  - "How to Tune Applications Using a Top-Down Characterization of Microarchitectural Issues"
  - Applies to Xeon processors (not KNC or KNL)

National Energy Research Scientific Computing Center