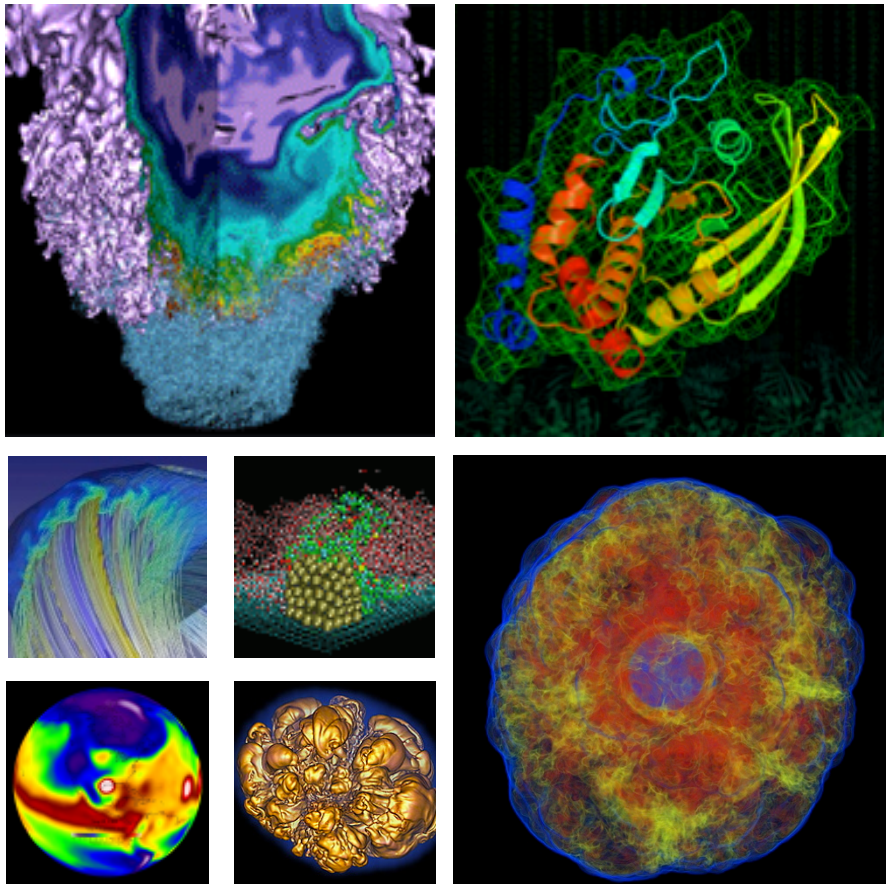
Babbage: the Intel Many Core (MIC) Testbed System at NERSC



Helen He

NERSC User Services Group

July 10, 2014

NERSC 40 YEARS at the FOREFRONT 1974-2014

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

First Message

- **Babbage can help you to prepare for Cori regarding thread scalability (hybrid MPI/OpenMP implementation) and vectorization.**

- **Performance on Babbage will be significantly worse than Cori. However, using Babbage can expose bottlenecks and weaknesses in your code for improvement.**

Outline

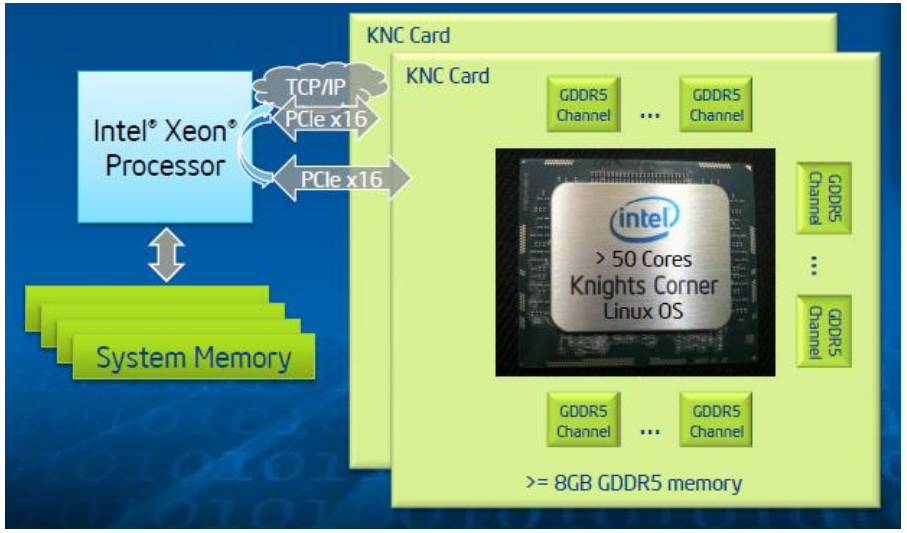
- Knights Corner (KNC) architecture and programming considerations
- System configurations and programming environment
- How to compile and run
- Examples of tuning kernels and applications

Basic Terminologies

- **MIC:** Intel Many Integrated Cores architecture
- **Xeon:** Intel Processors. Various product names include Nehalem, Westmere, Sandy Bridge (SNB) etc.
- **Xeon Phi:** Intel's marketing name for MIC architecture.
  - Some code names are: Knights Ferry (KNF), Knight Corner (KNC), Knights Landing (KNL)
- **Knights Corner (KNC):** first generation product of MIC Xeon Phi implementation
  - Co-processors connected to host via PCIe
  - Validate programming models
  - Prepare for next generation production hardware

Babbage Nodes

- **1 login node: bint01**
- **45 compute nodes, each has:**
  - Host node: 2 Intel Xeon Sandybridge EP processors, 8 cores each. 2.6 GHz, AVX 256-bit. Peak performance 166 GB/sec
  - 2 MIC cards (5100P) each has 60 native cores, connected by a high-speed bidirectional ring, 1053 MHz, 4 hardware threads per core.
  - Peak performance 1 TB/sec
  - 8 GB GDDR5 memory, peak memory bandwidth 320 GB/sec
  - 512-bit SIMD instructions. Holds 16 SP or 8 DP floating point numbers.



Babbage KNC *vs.* Cori KNL

- **Similarities**
  - Many integrated cores on a node (>60), 4 hardware threads per core
  - MPI+OpenMP as main programming language
  - 512 bits vector length for SIMD

- **Significant improvements in KNL**
  - Self-hosted architecture (not a co-processor!)
  - 3X single thread performance than KNC
  - Deeper out-of-order execution
  - High bandwidth on-package memory
  - Improved vectorization capabilities
  - And more …

Even with these differences, Babbage can still be helpful preparing applications for Cori. (Note: Edison can be used as well, future presentation)

Programming Considerations

- Use “native” mode on KNC to mimic KNL, which means ignore the host, just run completely on KNC cards.
- Encourage single node exploration on KNC cards with problem sizes that can fit.
- Simple to port from multi-core CPU architectures, but hard to achieve high performance.
- Need to explore high loop level parallelism via threading and SIMD vectorization
- Available threading models: OpenMP, pthreads, etc.

User Friendly Test System

- **Configured with ease-of-use in mind**
  - All production file systems are mounted
  - System SSH configuration allows password-less access to the host nodes and MIC cards smoothly
  - Modules created and loaded by default to initiate Intel compiler and MPI libraries
  - Batch scheduler installed for allocating nodes
  - MIC cards having access to system libraries allows multiple versions of software to co-exist
    - No need to copy system libraries or binaries to each MIC card manually as pre-steps for running jobs
    - Created scripts and wrappers to further simplify job launching commands
  - User environment very similar to other production systems

Intel Linux Studio XE Package

- **Intel C, C++ and Fortran compilers**
- **Intel MKL:** math kernel libraries
- **Intel Integrated Performance Primitive (IPP):** performance libraries
- **Intel Trace Analyzer and Collector:** MPI communications profiling and analysis
- **Intel Vtune Amplifier XE:** advanced threading and performance profiler
- **Intel Inspector XE:** memory and threading debugger
- **Intel Advisor XE:** threading prototyping tool
- **Intel Threading Building Blocks (TBB) and Intel Cilk Plus:** parallel programming models

Available Software

Loaded by default:

-bash-4.1$ module list

Currently Loaded Modulefiles:

1) modules	3) torque/4.2.6	5) intel/14.0.0	7) usg-default-modules/1.1
2) nsg/1.2.0	4) moab/7.2.6	6) impi/4.1.1	

Modules Available:

-bash-4.1$ module avail

<omit system software modules …>

---------- /usr/common/usg/Modules/modulefiles ----------

advisor/4.300519	impi/4.1.0	szip/host-2.1
allineatools/4.2.1-36484(default)	impi/4.1.1(default)	szip/mic-2.1(default)
fftw/3.3.4-host	inspector/2013.304368	totalview/8T.12.0-1(default)
fftw/3.3.4-mic(default)	intel/13.0.1	usg-default-modules/1.0
hdf5/host-1.8.10-p1	intel/13.1.2	usg-default-modules/1.1(default)
hdf5/host-1.8.13	intel/14.0.0(default)	vtune/2013.update16(default)
hdf5/mic-1.8.10-p1	intel/14.0.3	zlib/host-1.2.7
hdf5/mic-1.8.13(default)	itac/8.1.3	zlib/host-1.2.8
hdf5-parallel/host-1.8.10-p1	netcdf/host-4.1.3	zlib/mic-1.2.7
hdf5-parallel/host-1.8.13	netcdf/host-4.3.2	zlib/mic-1.2.8(default)
hdf5-parallel/mic-1.8.10-p1	netcdf/mic-4.1.3	
hdf5-parallel/mic-1.8.13(default)	netcdf/mic-4.3.2(default)	

How to Compile on Babbage

- Only Intel compiler and Intel MPI are supported.
- Compile on the login node "bint01" directly to build an executable to run on the host or on the MIC cards.
- You can also compile on a host node. Do not "ssh bcxxxx" directly from "bint01", instead, use "qsub -I -l nodes=1" to get a node allocated to you.
- Use "ifort", icc" or "icpc" to compile serial Fortran, C, or C++ codes.
- Use "mpiifort", "mpiicc", or "mpiicpc" to compile parallel Fortran, C, or C++ MPI codes. (NOT mpif90, mpicc, or mpiCC)
- Use the "-openmp" flag for OpenMP codes.
- Use the "-mmic" flag to build an executable to run on the MIC cards.
- Example:
  Build a binary for host: % mpiicc -openmp -o xthi.host xthi.c
  Build a binary for MIC: % mpiicc -mmic -openmp -o xthi.mic xthi.c

Spectrum of Programming Models

	Host Only	Offload	Symmetric (Host and MIC)	Native (MIC only)
Xeon (Host)	Program foo call bar() End	Program foo call bar() End	Program foo call bar() End	--
Xeon Phi (MIC)	--	bar()	Program foo call bar() End	Program foo call bar() End

- **Knights Landing (KNL) will be in self-hosted mode, thus eliminates the host and the need to communicate between host and MIC.**
- **We encourage users to focus on optimizing in the Native mode and explore on-node scaling on a single KNC card.**

How to Run on Host

- **Useful for comparing performance with running natively on the MIC cards.**

```
bint01% qsub -I -l nodes=2
<wait for a session>
% cd $PBS_O_WORKDIR

% cat $PBS_NODEFILE
 bc1012
 bc1011

% get_hostfile
% cat hostfile.$PBS_JOBID
 bc1012-ib
 bc1011-ib
```

```
% export OMP_NUM_THREADS=4

% mpirun -n 2 -hostfile hostfile.$PBS_JOBID -ppn 1 ./xthi.host
Hello from rank 0, thread 0, on bc1012. (core affinity = 0-15)
Hello from rank 0, thread 2, on bc1012. (core affinity = 0-15)
…
Hello from rank 1, thread 3, on bc1011. (core affinity = 0-15)
Hello from rank 1, thread 0, on bc1011. (core affinity = 0-15)
```

How to Run on MIC Cards Natively

```
bint01% qsub -I -l nodes=2
<wait for a session>
% cd $PBS_O_WORKDIR
% cat $PBS_NODEFILE
 bc1012
 bc1011
% get_micfile
% cat micfile.$PBS_JOBID
 bc1011-mic0
 bc1011-mic1
 bc1010-mic0
 bc1010-mic1
% export OMP_NUM_THREADS=12
% export KMP_AFFINITY=balanced
```

```
% mpirun.mic -n 4 -hostfile micfile.$PBS_JOBID -ppn 1 ./xthi.mic | sort
Hello from rank 0, thread 0, on bc1011-mic0. (core affinity = 1)
Hello from rank 0, thread 1, on bc1011-mic0. (core affinity = 5)
Hello from rank 0, thread 10, on bc1011-mic0. (core affinity = 41)
Hello from rank 0, thread 11, on bc1011-mic0. (core affinity = 45)
…
Hello from rank 3, thread 6, on bc1010-mic1. (core affinity = 25)
Hello from rank 3, thread 7, on bc1010-mic1. (core affinity = 29)
Hello from rank 3, thread 8, on bc1010-mic1. (core affinity = 33)
Hello from rank 3, thread 9, on bc1010-mic1. (core affinity = 37)
```

Thread Affinity: KMP_AFFINITY

- **none:** default option on host
- **compact:** default option on MIC. Bind threads as close to each other as possible

Node	Core 1				Core 2				Core 3			
	HT1	HT2	HT3	HT4	HT1	HT2	HT3	HT4	HT1	HT2	HT3	HT4
Thread	0	1	2	3	4	5						

- **scatter:** bind threads as far apart as possible

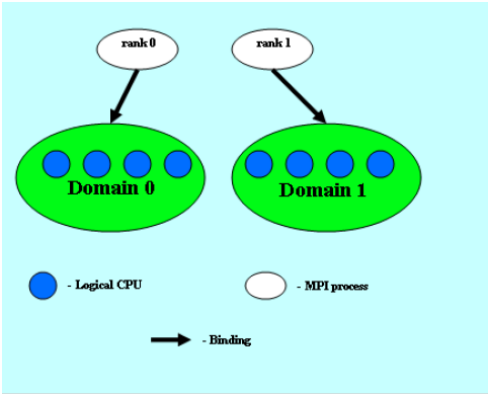
Node	Core 1				Core 2				Core 3			
	HT1	HT2	HT3	HT4	HT1	HT2	HT3	HT4	HT1	HT2	HT3	HT4
Thread	0	3			1	4			2	5		

- **balanced:** only available on MIC. Spread to each core first, then set thread numbers using different HT of same core close to each other.

Node	Core 1				Core 2				Core 3			
	HT1	HT2	HT3	HT4	HT1	HT2	HT3	HT4	HT1	HT2	HT3	HT4
Thread	0	1			2	3			4	5		

- **explicit**: example: setenv KMP_AFFINITY "explicit, granularity=fine, proclist=[1:236:1]"
- New env on coprocessors: KMP_PLACE_THREADS, for exact thread placement

MPI Process Affinity: I_MPI_PIN_DOMAIN

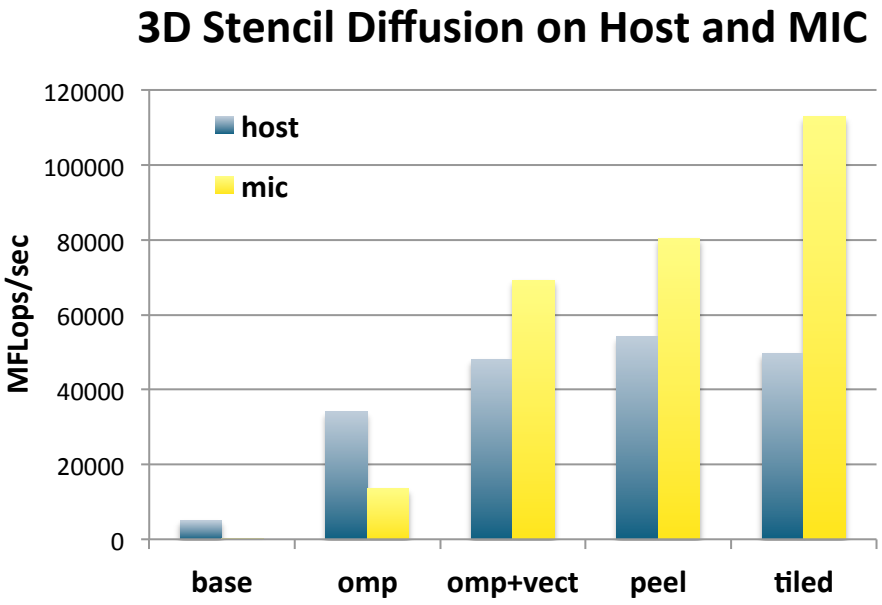


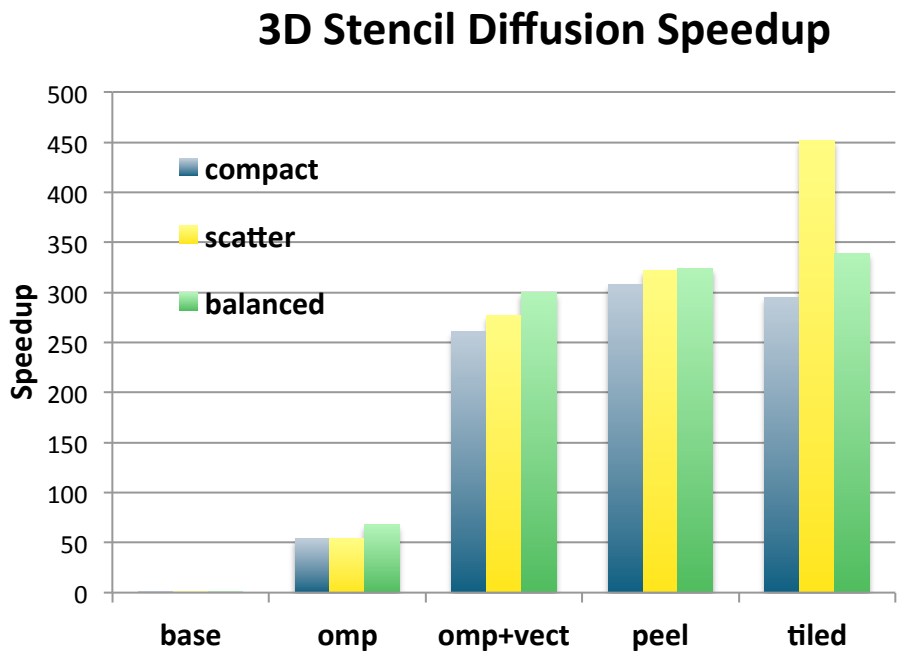
- Map CPUs into non-overlapping domains
  - 1 MPI process per domain
  - OpenMP threads pinned inside each domain
- **I_MPI_PIN_DOMAIN**=<size>[:<layout>]

<size> = omp adjust to OMP_NUM_THREADS
auto #CPUs/ #MPI procs
<n> a number

<layout> = platform according to BIOS numbering
compact close to each other
scatter far away from each other

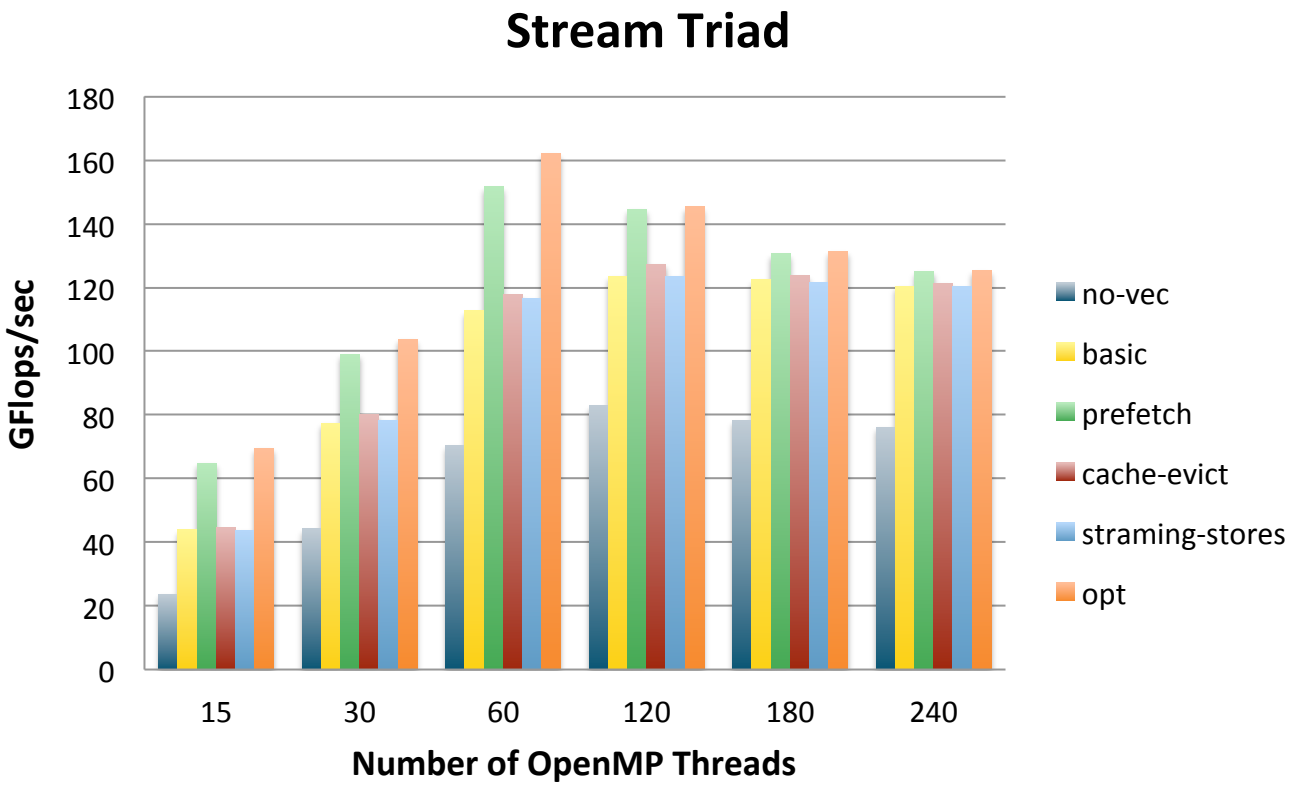
3D Stencil Diffusion Algorithm





-- On host: use 16 threads, KMP_AFFINITY=scatter
-- On MIC: Tested with different number of threads: 60, 120, 180, 236, 240, combined with various KMP_AFFINITY options.
-- The best speedup on MIC is obtained via 180 threads with scatter affinity.
-- Runs faster on host with base option and OpenMP only option.
-- Faster on MIC when vectorization is introduced with OpenMP.
-- OpenMP and Vectorization both play significant roles on MIC
-- More advanced loop optimization techniques (loop peel and tiling) can improve further.

STREAM



-- Best rate is 162 GFlops/sec with 60 OpenMP threads on 1 MIC card.
-- 60% improvement with vectorization.
-- Software prefetch helps significantly. (35% improvement)
-- Intel reports best performance of 174 GFlops/sec on Xeon Phi 7100P, 61 core

Tuning Lessons Learned

- Some code restructuring and algorithm modifications are needed to take advantage of the KNC architecture.
- Some applications won't be able to fit into memory with pure MPI due to the small memory size on KNC cards.
- It is essential to add OpenMP at as high a level as possible to explore loop level parallelism, and make sure large, innermost, computational extensive loops are vectorized.
- Explore the scalability of OpenMP implementation.
- Try various MPI and OpenMP affinity options.
- Special compiler options on KNC also helps.
- Memory alignment is important.
- Optimizations targeted for KNC can help performance for other architectures: Xeon, KNL.

Summary

- **Performance on Babbage does not represent what will be on Cori.**
- **Babbage can help you to prepare for Cori regarding thread scalability (hybrid MPI/OpenMP implementation) and vectorization.**
- **Please contact consult@nersc.gov for questions.**

Further Information

- **Babbage web page:**
  - https://www.nersc.gov/users/computational-systems/testbeds/babbage
- **Intel Xeon Phi Coprocessor Developer Zone:**
  - http://software.intel.com/mic-developer
- **Programming and Compiling for Intel MIC Architecture**
  - http://software.intel.com/en-us/articles/programming-and-compiling-for-intel-many-integrated-core-architecture
- **Optimizing Memory Bandwidth on Stream Triad**
  - http://software.intel.com/en-us/articles/optimizing-memory-bandwidth-on-stream-triad
- **Interoperability with OpenMP API**
  - http://software.intel.com/sites/products/documentation/hpc/ics/impi/41/win/Reference_Manual/Interoperability_with_OpenMP.htm
- **Intel Cluster Studio XE 2013**
  - http://software.intel.com/en-us/intel-cluster-studio-xe/
- **Intel Xeon Phi Coprocessor High-Performance Programming.** Jim Jeffers and James Reinders, Published by Elsevier Inc. 2013.

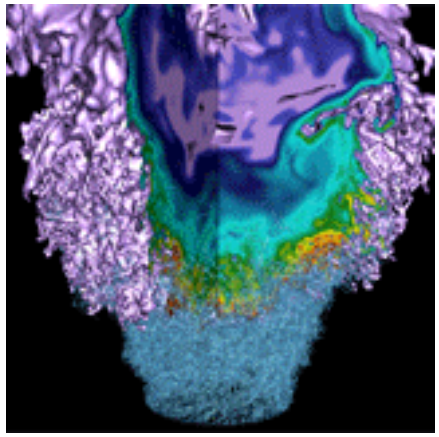
Acknowledgement

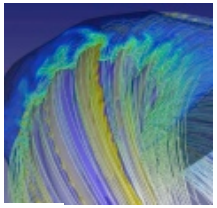
- **Babbage system support team, especially Nick Cardo, for configuring the system and solving many mysteries and issues.**

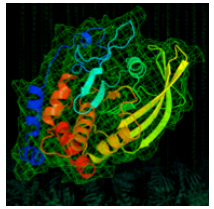
- **NERSC Application Readiness Team for testing, providing ideas, and reporting problems on the system.**

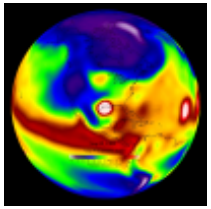
Thank you.

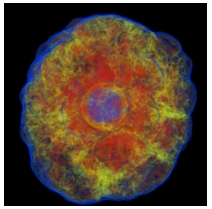
Extra Slides

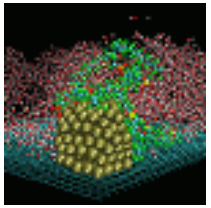












Babbage Compute Nodes

- **45 nodes, etc., bc09xx, bc10xx, bc11xx**
- **Host node: 2 Intel Xeon Sandybridge ES-2670 processors**
  - Each processor has 8 cores, with 2 hardware threads (HT not enabled), 2.6 GHz, peak performance 166.4 GFlops
  - 128 GB memory per node
  - Memory bandwidth 51.2 GB/sec
  - AVX 32 byte aligned: AVX on host, 256-bit SIMD
- **2 MIC cards (5110P, bc09xx-mic0, bc09xx-mic1) each with:**
  - 60 native cores, connected by a high-speed bidirectional ring, clock speed is 1053 MHz, Error Correcting Code (ECC) enabled
  - 4 hardware threads per core
  - 8 GB GDDR5 memory, effective speed 5 GT/s, peak memory bandwidth 320 GB/sec
  - L1 cache per core: 32 KB 8-way associative data and instruction cache
  - L2 cache per core: 512 KB 8-way associative inclusive cache with hardware prefetcher
  - Peak performance 1011 GFlops
  - Vector Unit.
    - 512-bit SIMD instructions,
    - 32 512-bit registers, holds 32 DP and 64 SP …

Memory Alignment

- **Always align at 64 byte boundaries to ensure data can be loaded from memory to cache optimally**
  - 20% performance penalty without memory alignment for
  - DGEMM (matrix size 6000x6000)
- **Fortran: compile with "-align array64byte" option to align all static array data to 64 memory address boundaries**
- **C/C++: declare var**
  - static: float var[100] __attribute__((aligned(64)));
  - dynamic: __mm_aligned_malloc(buf, 64)
- **More options with compiler directives**

SIMD and Vectorization

- **Vectorization:** the process of transforming a scalar instruction (SISD) into vector instruction (SIMD)
- **To tell compiler to ignore potential dependencies and vectorize anyway:**
  - Fortran directive: !DIR$ SIMD
  - C/C++ directive: #pragma simd
- **Example:** a,b,c, are pointers, compiler does not know they are independent

Not vectorized:

```
for (i=0; i<n; i++)
   a[i]=b[i]+a[i-1]
```

Not vectorized:

```
for (i=0; i<n; i++)
   a[i]=b[i]+c[i]
```

Vectorized:

```
#pragma simd
for (i=0; i<n; i++)
   a[i]=b[i]+c[i]
```

Wrapper Script for mpirun on MIC Cards

- **In all module files:**
  - Set $LD_LIBRARY_PATH for libraries needed on host
  - Set $MIC_LD_LIBRARY_PATH for libraries needed on MIC card
- **% cat mpirun.mic**

```
#!/bin/sh
mpirun -env LD_LIBRARY_PATH $MIC_LD_LIBRARY_PATH $@
```

- **Sample execution line:**

```
% mpirun.mic -n 4 -hostfile micfile.$PBS_JOBID -ppn 2./xthi.mic
```

-DMIC Trick for Configure on MIC Card

- **Sometimes when install software libraries on MIC cards, a test program needs to be run. Due to cross-compile, the test program will fail.**
- **The trick is to define "-DMIC" for the the compiler options such as CC, CXX, FC, etc. used in "configure":** export CC="icc –DMIC", …
- **Replace all "-DMIC" in Makefile with "-mmic", then compile and build.**

```
files=$(find ./* -name Makefile)
perl –p –i –e 's/-DMIC/-mmic/g' $files
```

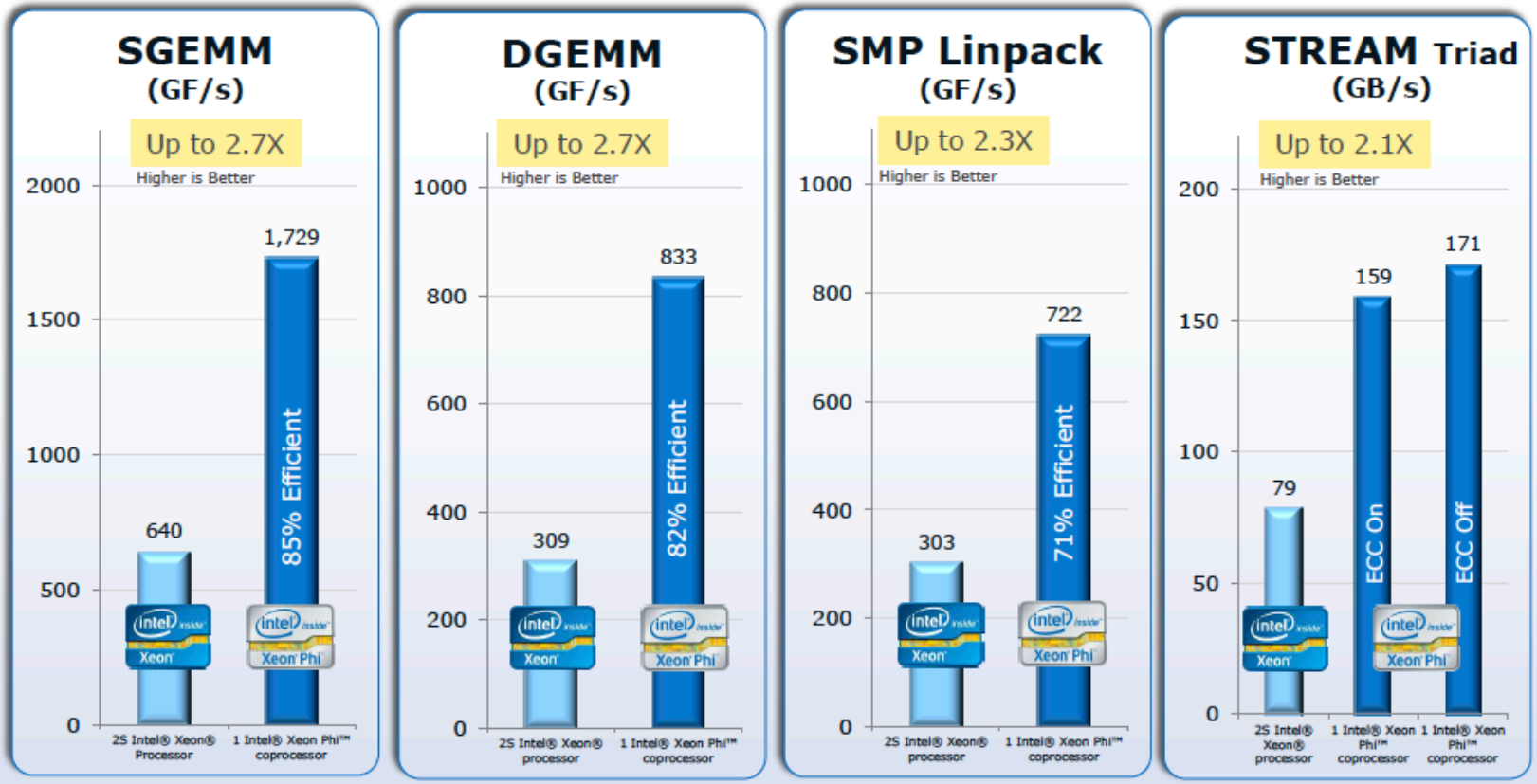
Sample Batch Script

```
#PBS -q regular
#PBS -l nodes=2:mics=2
#PBS -l walltime=02:00:00
#PBS -V

cd $PBS_O_WORKDIR
export OMP_NUM_THREADS=60
export KMP_AFFINITY=balanced

mpirun.mic -n 4 –hostfile $PBS_MICFILE -ppn 1 ./myexe.mic      (# sometimes the full path to the executable is needed, otherwise you may see a "no such file or directory" error).

----------------------------------
# Can use custom hostfile with -host option:
% mpirun.mic –n 4 -host bc1013-mic0,bc1012-mic1 –ppn 2 ./myexe.mic

# Can pass env with –env option:
% mpirun.mic -n 16 -host bc1011-mic1 -env OMP_NUM_THREADS 2 -env KMP_AFFINITY balanced ./
myexe.mic
```

Thread Affinity: KMP_PLACE_THREADS

- New setting on coprocessors only. In addition to KMP_AFFINITY, can set exact but still generic thread placement.
- **KMP_PLACE_THREADS**=<n>Cx<m>T,<o>O
  - <n> Cores times <m> Threads with <o> of cores Offset
  - e.g. 40Cx3T,1O means using 40 cores, and 3 threads (HT2,3,4) per core
- OS runs on logical proc 0, which lives on physical core 59
  - OS procs on core 59: 0,237,238,239.
  - Avoid use proc 0, i.e., use max_threads=236 on Babbage.

Synthetic Benchmark Summary (Intel MKL) (5110P)



Coprocessor results: Benchmark run 100% on coprocessor, no help from Intel® Xeon® processor host (aka native)

Notes

1. Intel® Xeon® Processor E5-2670 used for all SGEMM Matrix = 13824 x 13824 , DGEMM Matrix 7936 x 7936, SMP Linpack Matrix 30720 x 30720
2. Intel® Xeon Phi™ coprocessor 5110P (ECC on) with "Gold Release Candidate" SW stack SGEMM Matrix = 11264 x 11264, DGEMM Matrix 7680 x 7680, SMP Linpack Matrix 26872 x 28672

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Source: Intel Measured results as of October 26, 2012 Configuration Details: See backup
For more information go to http://www.intel.com/performance

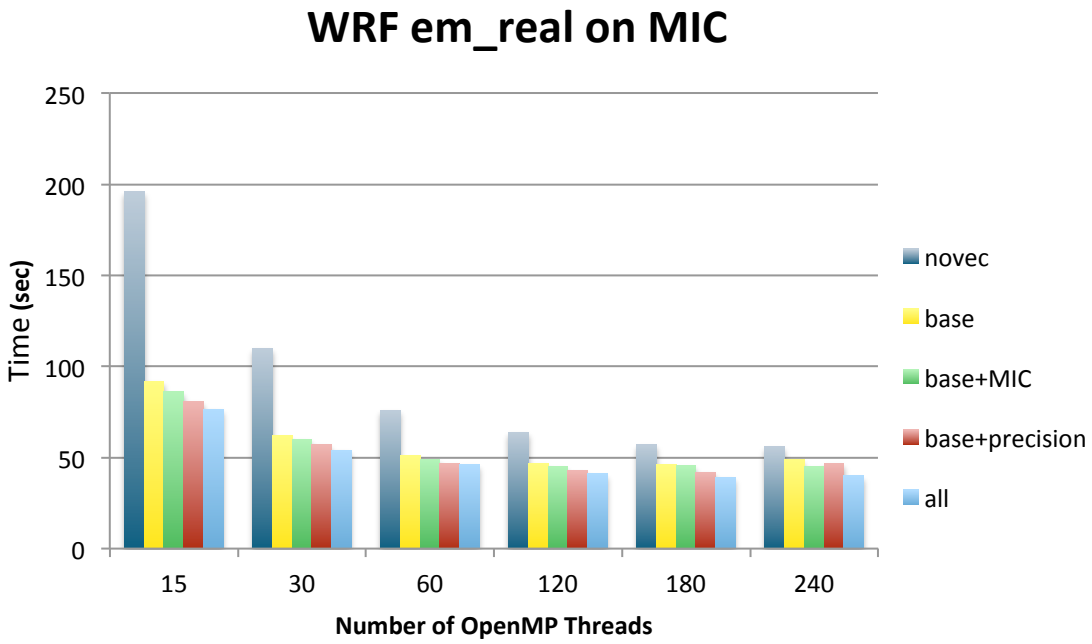
11

STREAM Compiler Options

- **-no-vec:**
  - -O3 -mmic –openmp -no-vec -DSTREAM_ARRAY_SIZE=64000000
- **Base**
  - -O3 -mmic -openmp -DSTREAM_ARRAY_SIZE=64000000
- **Base plus -opt-prefetch-distance=64,8**
  - Software Prefetch 64 cachelines ahead for L2 cache
  - Software Prefetch 8 cachelines ahead for L1 cache
- **Base plus -opt-streaming-cache-evict=0**
  - Turn off all cache line evicts
- **Base plus -opt-streaming-stores always**
  - Enable generation of streaming stores under the assumption that the application is memory bound
- **Opt**:
  - Use all above flags (except –no-vec)
- No more huge pages needed since it is now default in MPSS

WRF (Weather Research and Forecasting Model)

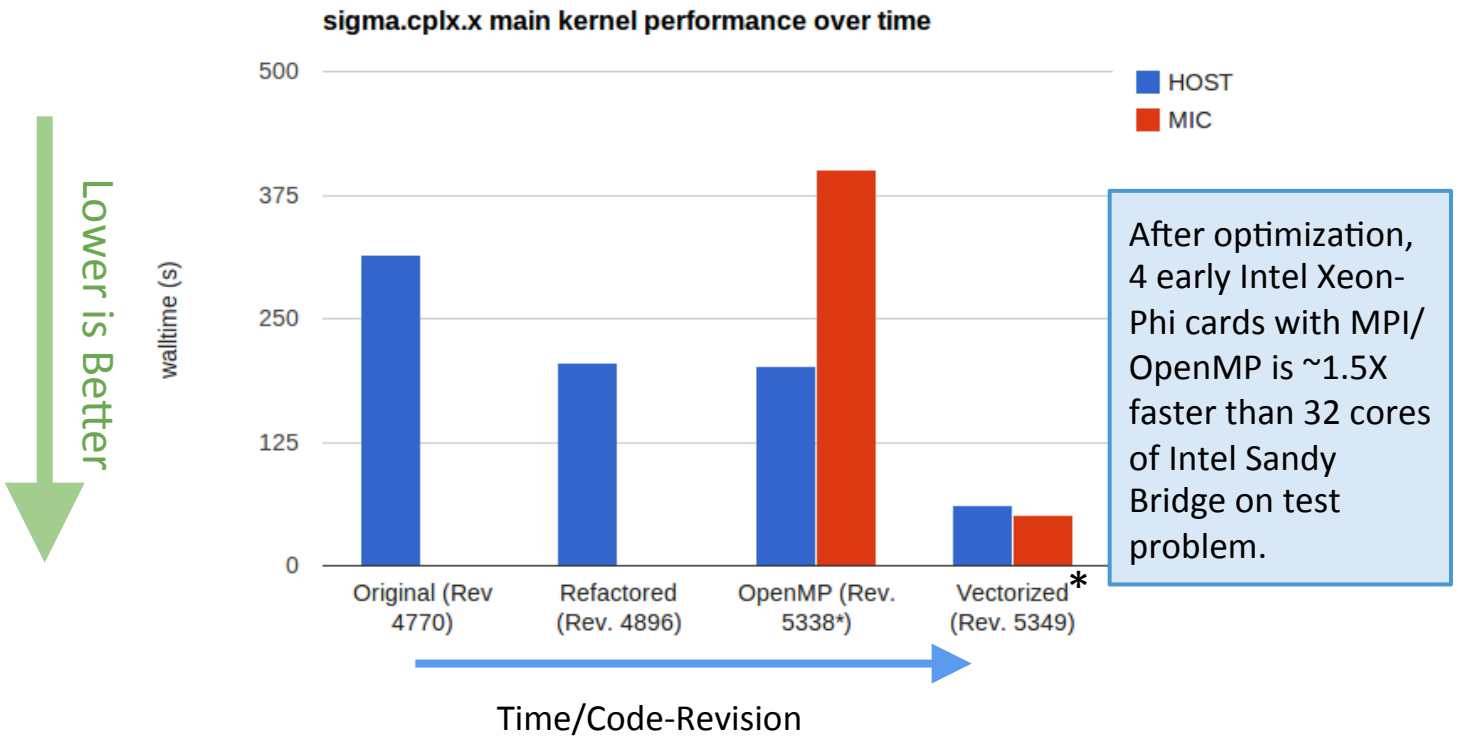
- **novec: -**O3 -novec-w -ftz -fno-alias -FR -convert big_endian -openmp -fpp –auto
- **base:** -mmic -O3 -w -openmp -FR -convert big_endian -align array64byte -vec-report6
- **precision:** -fimf-precision=low -fimf-domain-exclusion=15 -fp-model fast=1 -no-prec-div -no-prec-sqrt
- **MIC:** -opt-assume-safe-padding -opt-streaming-stores always -opt-streaming-cache-evict=0 -mP2OPT_hlo_pref_use_outer_strategy=F



- Best time is 39.26 sec with 180 OpenMP threads.
- 15.1% improvement with all optimization flags compared with base options.

Steps to Optimize BerkeleyGW

Courtesy of Jack Deslippe



After optimization, 4 early Intel Xeon-Phi cards with MPI/ OpenMP is ~1.5X faster than 32 cores of Intel Sandy Bridge on test problem.

1. Refactor to create hierarchical set of loops to be parallelized via MPI, OpenMP and Vectorization and to improve memory locality.
2. Add OpenMP at as high a level as possible.
3. Make sure large innermost, flop intensive, loops are vectorized

   * - eliminate spurious logic, some code restructuring simplification and other optimization