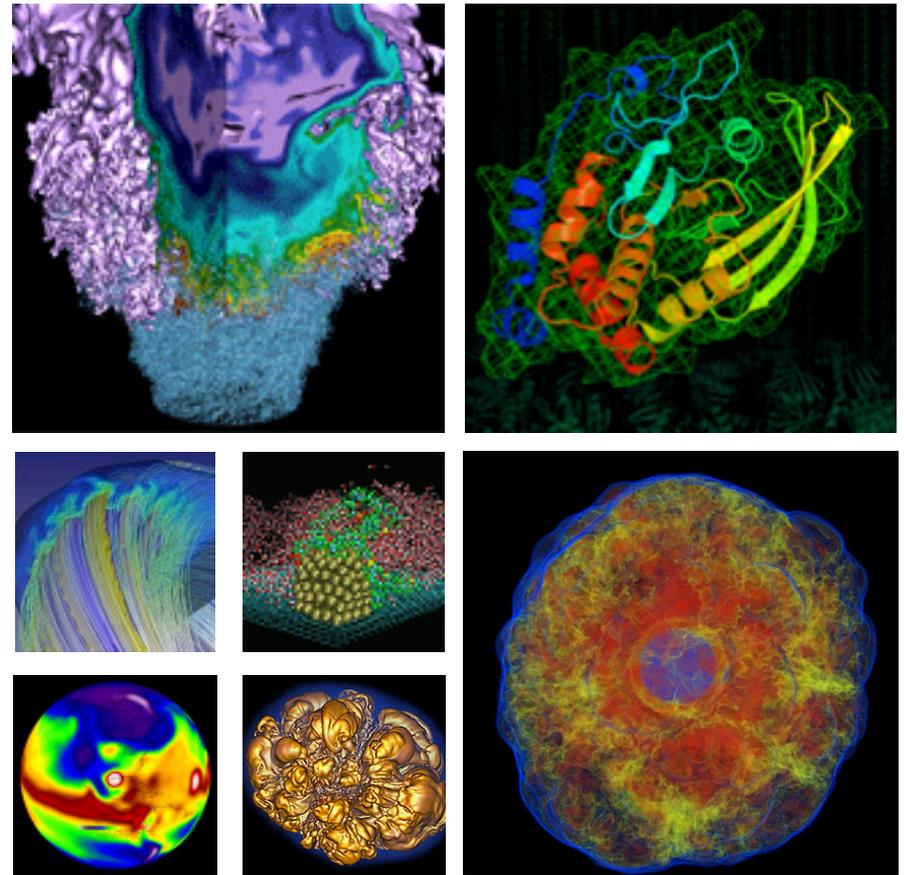


NERSC Application Readiness



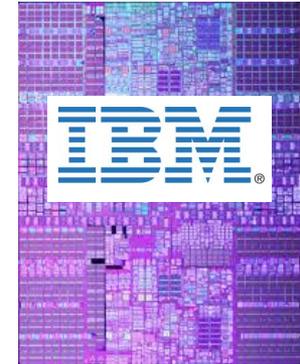
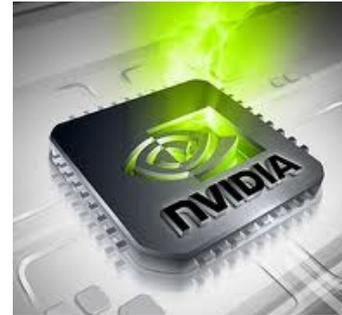
NERSC **40** YEARS
at the
FOREFRONT
1974-2014

Katie Antypas
NERSC-8 Project Manager
NERSC Services Dept. Head

App Readiness Across Labs
March 25, 2014

NERSC-8 architecture will be announced soon

- Regardless of processor architecture, users will need to modify applications to achieve performance
 - Expose more on-node parallelism in applications
 - Increase application vectorization capabilities
 - For co-processor architectures, locality directives must be added
 - Hierarchical memory



*NERSC workload has hundreds of codes.
How will application teams make the transition?*

Application Readiness Resources



- **We have a number of resources devoted to application readiness**
 - ~8 person NERSC App Readiness Team
 - ~8 Post-docs
 - Deep dive sessions with the chip vendor
 - Support from NERSC-8 Integrator

We're very interested in learning about the experiences from ALCF and OLCF Application Readiness from Titan and Mira

NERSC App Readiness Team



Katerina Antypas
(Co-Lead)



Nick Wright (Co-Lead)
Amber (Proxy for
NAMD, LAMMPS)



Harvey Wasserman
SNAP (S_N transport
proxy)



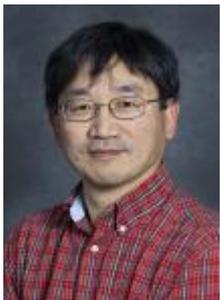
Brian Austin
Zori (Proxy for
QWalk etc.)



Hongzhang Shan
NWChem (Proxy for
qchem, GAMESS)



Aaron Collier
Madam-Toast /
Gyro



Woo-Sun Yang
CAM (Proxy for
CESM)



Jack Deslippe
Quantum
ESPRESSO /
BerkeleyGW (Proxy
for VASP, Abinit)



Helen He
WRF



Matt Cordery
MPAS



Kirsten Fagnan
Bio-Informatics



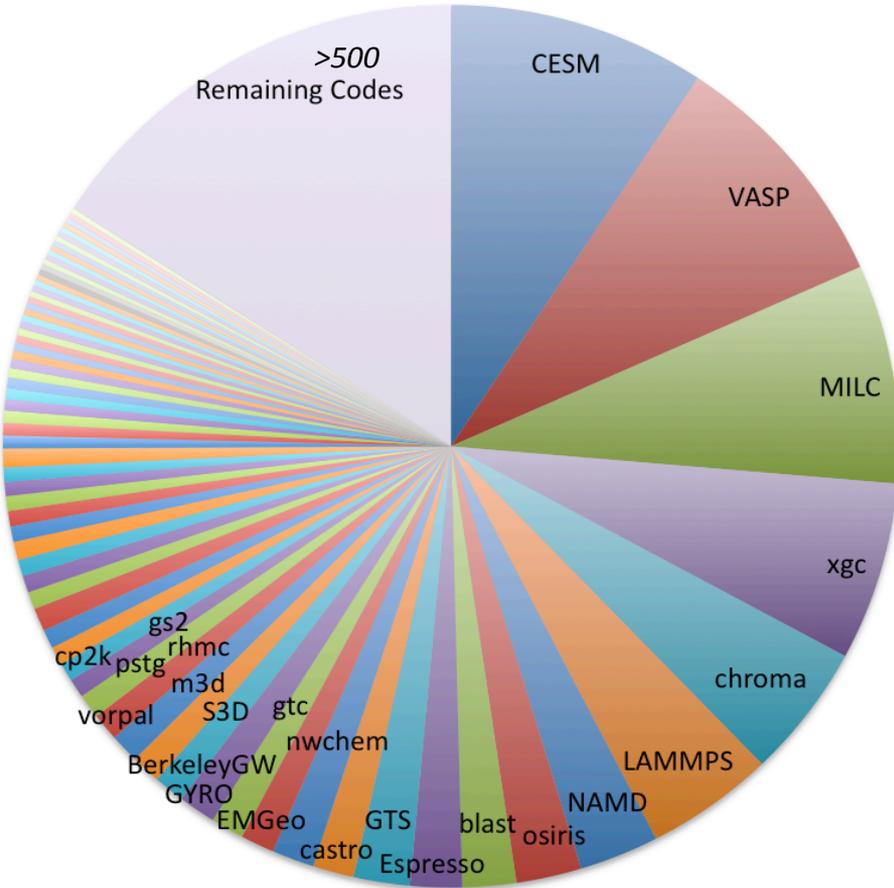
Christopher Daley
FLASH

- If you do nothing, your MPI-only code may run poorly on future machines.
- NERSC is here to help

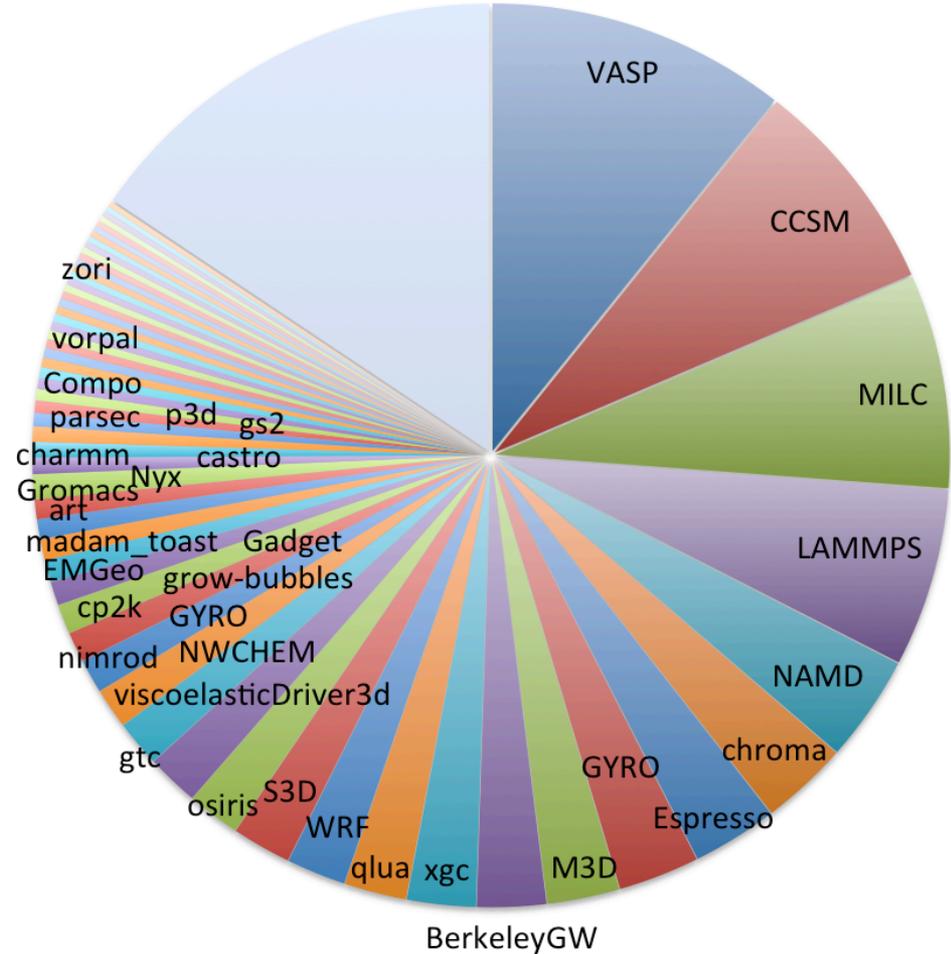
NERSC's highly concentrated workload



Breakdown of Application Hours on Hopper 2012



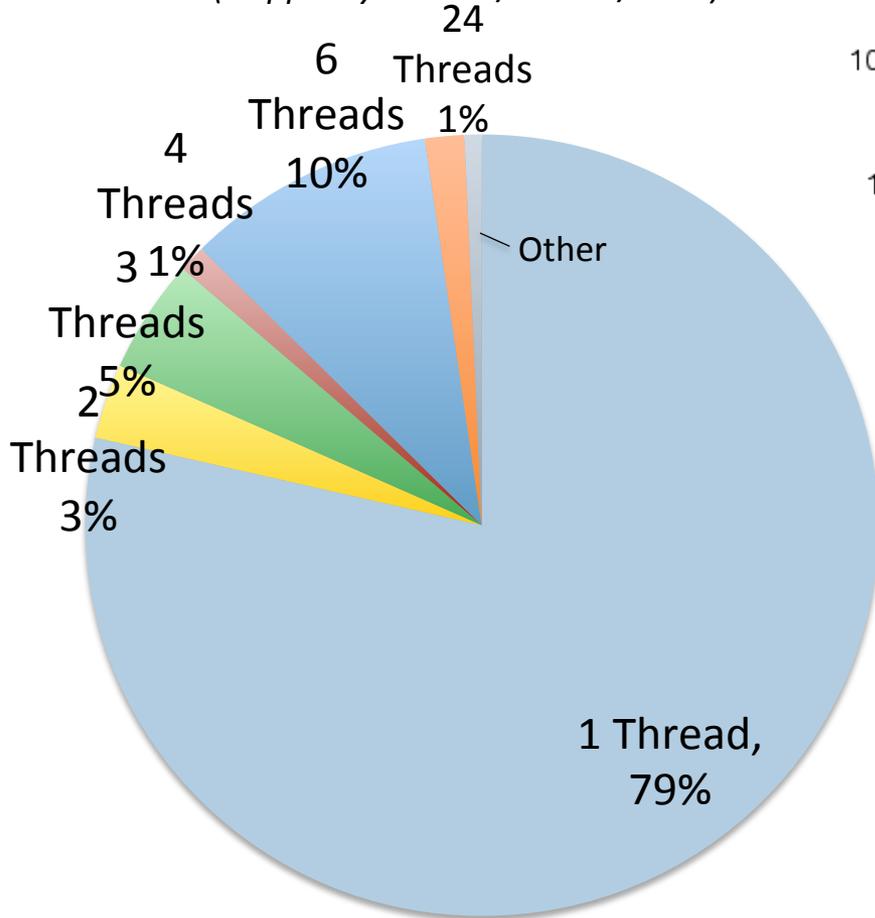
Breakdown of Application Hours on Hopper and Edison 2013



Over 20% of hours used on Hopper run with more than a single thread per task

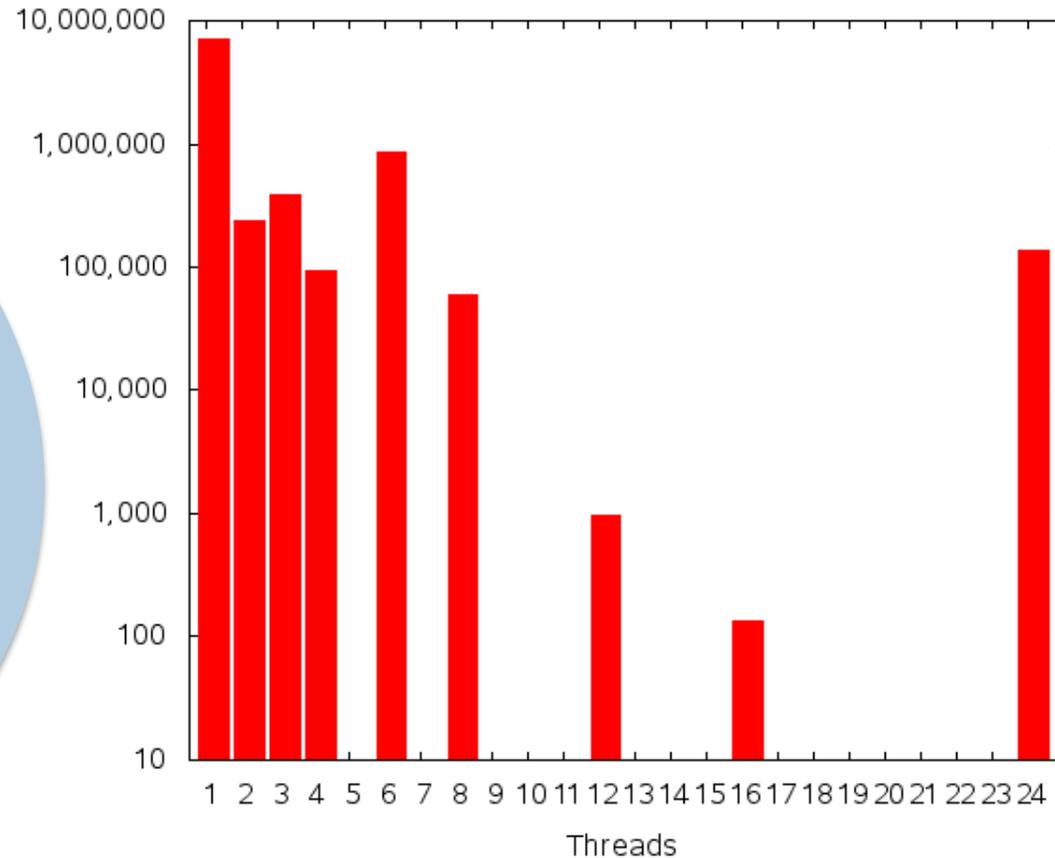
% of Hours Used by Thread Count

(Hopper system 11/2012-2/2013)



Node Hours used by Application Thread Count

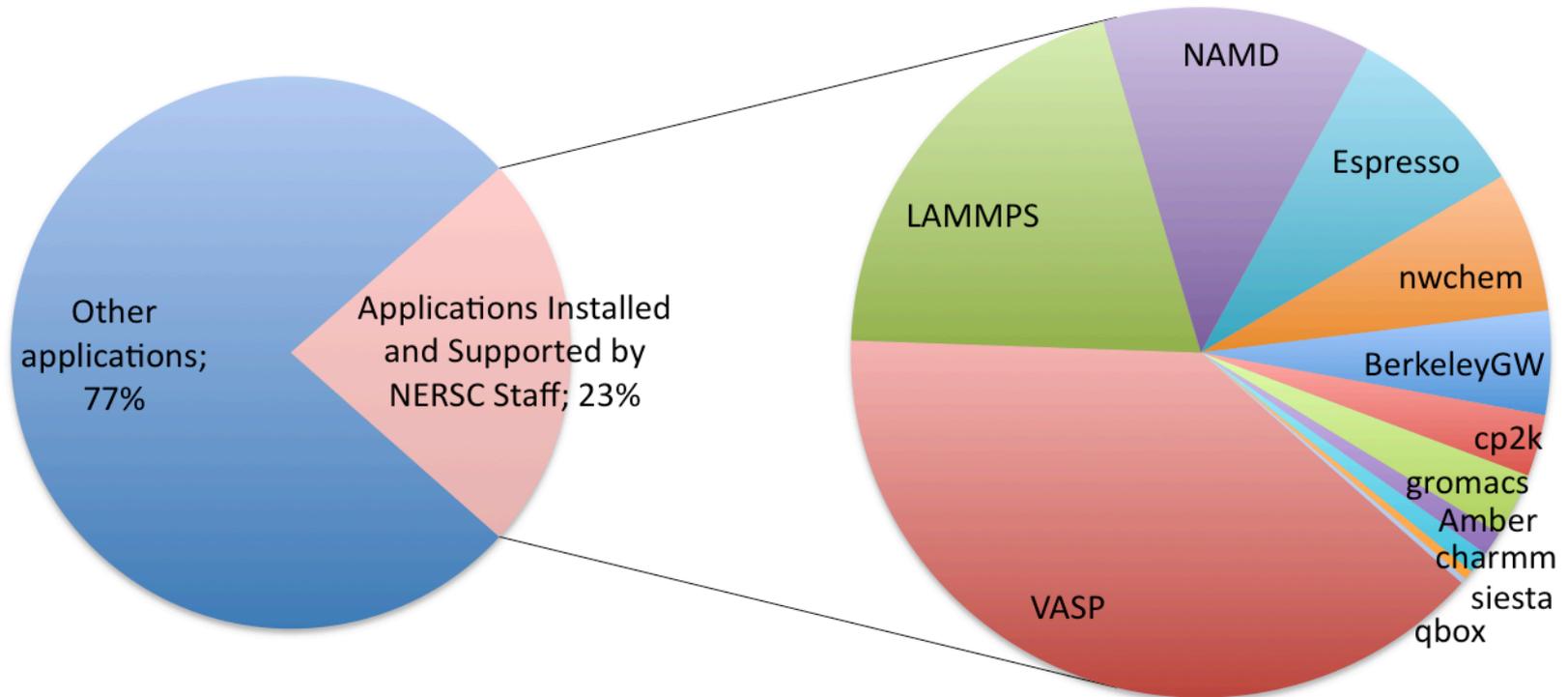
(Hopper system 11/2012-2/2013)



NERSC installs and directly supports software used by over 20% of workload



Percent of workload using NERSC installed and supported application codes
(On Hopper by Number of Hours)

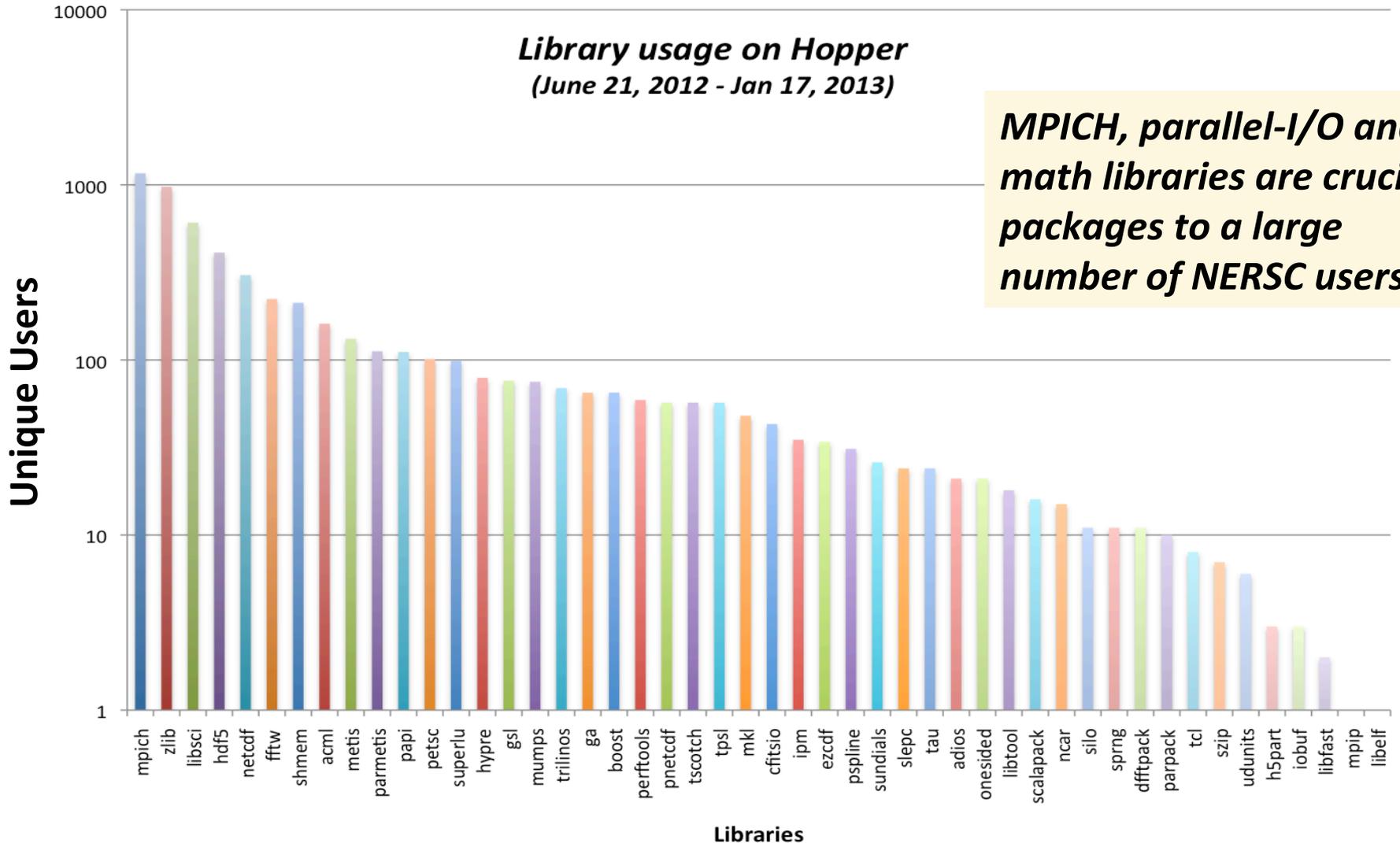


Application	Assessment of readiness	Comments on advanced architecture application readiness	% Cum
CESM		CamSE port to Titan, vendor interest, large complex code	9%
VASP		GPU solver for exchange correlation, closed code	18%
MILC		GPU version, MIC version in progress, BG/Q port, OpenMP enabled	26%
XGC		SciDAC funded project, vendor interest	33%
Chroma		GPU Version, MIC version, SciDAC project	38%
LAMMPS		GPU version, Part of Titan App Readiness	42%
NAMD		GPU Version, MIC version, BG/Q port, highly engaged team	45%
Osiris		2D GPU version, 3D in progress, OpenMP enabled, BG/Q port, SciDAC funded	48%
BLAST		Engaged NERSC staff, vendor interest but high throughput application	50%
Espresso		Funded NERSC App Readiness staff, Ported to GPU, MIC port in progress	51%
GTS		Running on BG/Q, SciDac project, OpenMP Enabled, GPU port	53%
CASTRO		OpenMP enabled	55%
NWChem		SciDAC project, NERSC funded App Readiness staff, Intel MIC port, vendor interest	57%
EMGeo		GPU implementation	58%
GTC		OpenMP enabled, GPU version, Starting MIC port, BG/Q port	59%
GYRO		NERSC funded app readiness code, OpenACC directives added, BG/Q port, SciDAC funded	60%
BerkeleyGW		NERSC funded app readiness code, OpenMP enabled, GPU enabled, MIC port	61%

NERSC users require an array of software to conduct their research



Library usage on Hopper
(June 21, 2012 - Jan 17, 2013)



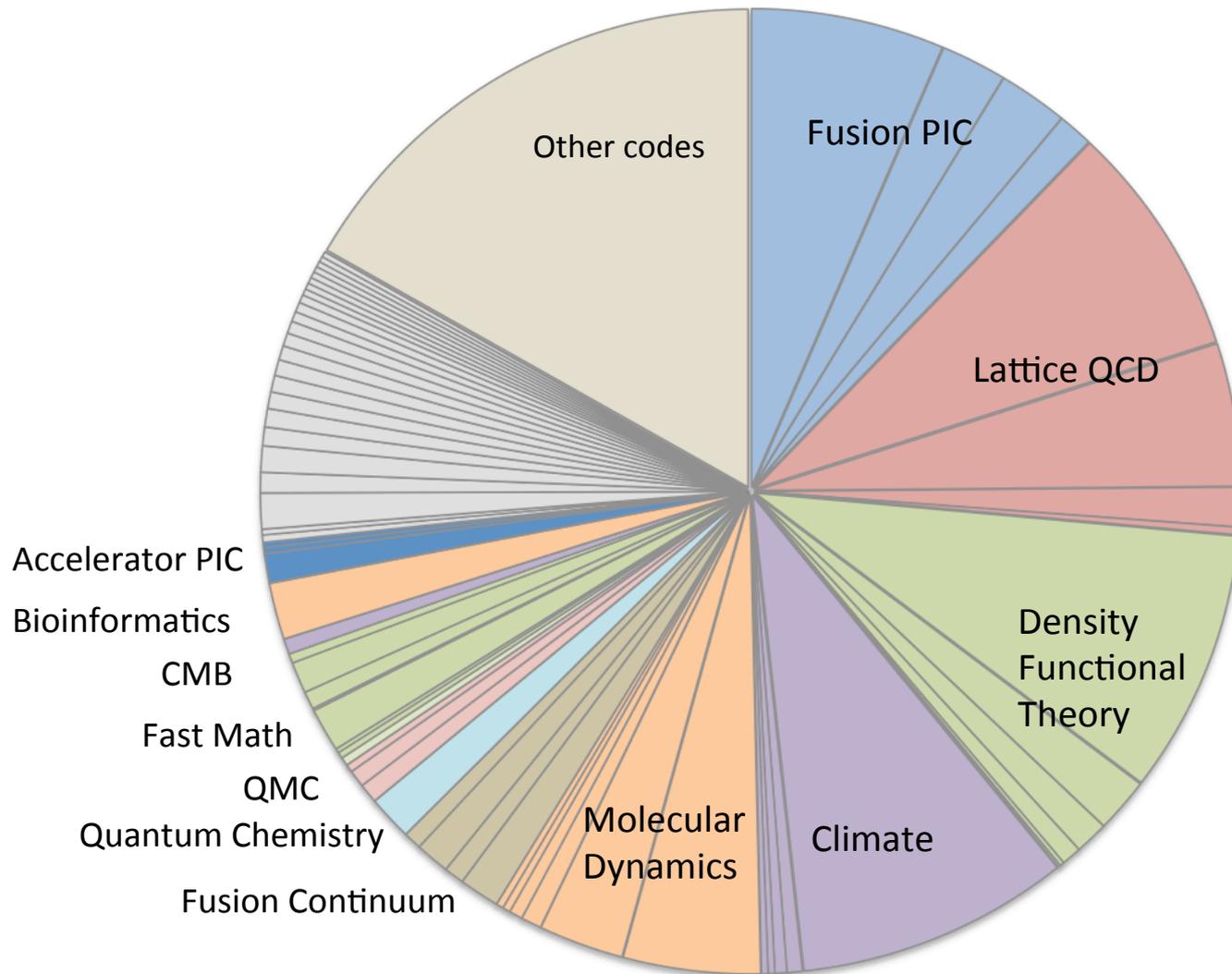
MPICH, parallel-I/O and math libraries are crucial packages to a large number of NERSC users



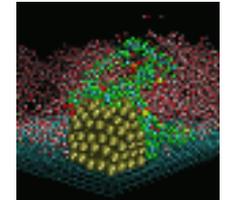
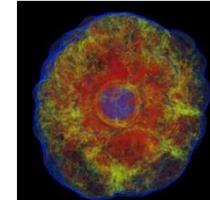
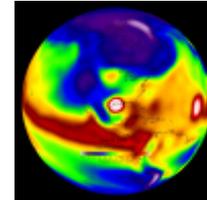
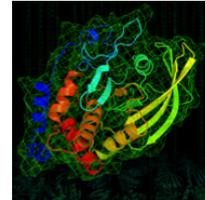
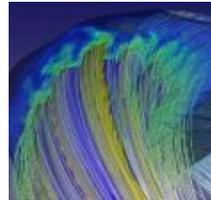
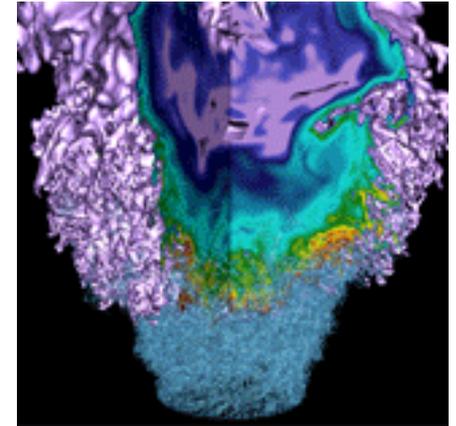
Thank you

In addition to science area and code diversity, NERSC supports a broad range of algorithms

Top Codes by Algorithm



FLASH Case Study Christopher Daley



Case Study on the Xeon-Phi (MIC) Architecture

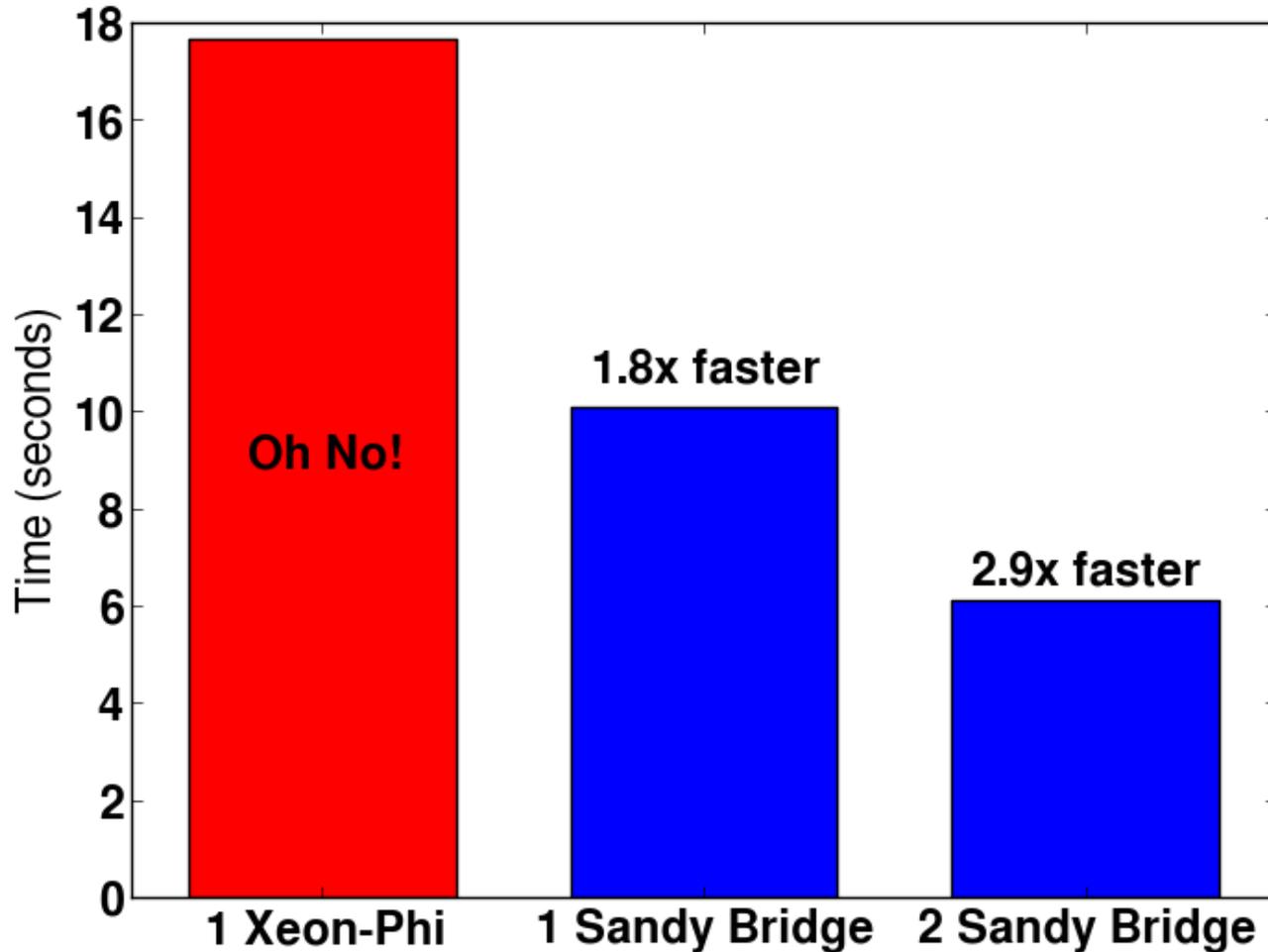
- **NERSC Testbed Babbage**
 - 45 Sandy-bridge nodes with Xeon-Phi Co-processor
 - Each Xeon-Phi Co-processor has
 - 60 cores
 - 4 HW threads per core
 - 8 GB of memory
 - Multiple ways to program with co-processor
 - As an accelerator
 - As a self-hosted processor (ignore Sandy-bridge)
 - Reverse accelerator
- We chose to test as if the Xeon-Phi was a stand alone processor because Intel has announced next generation will be self-hosted

FLASH application readiness

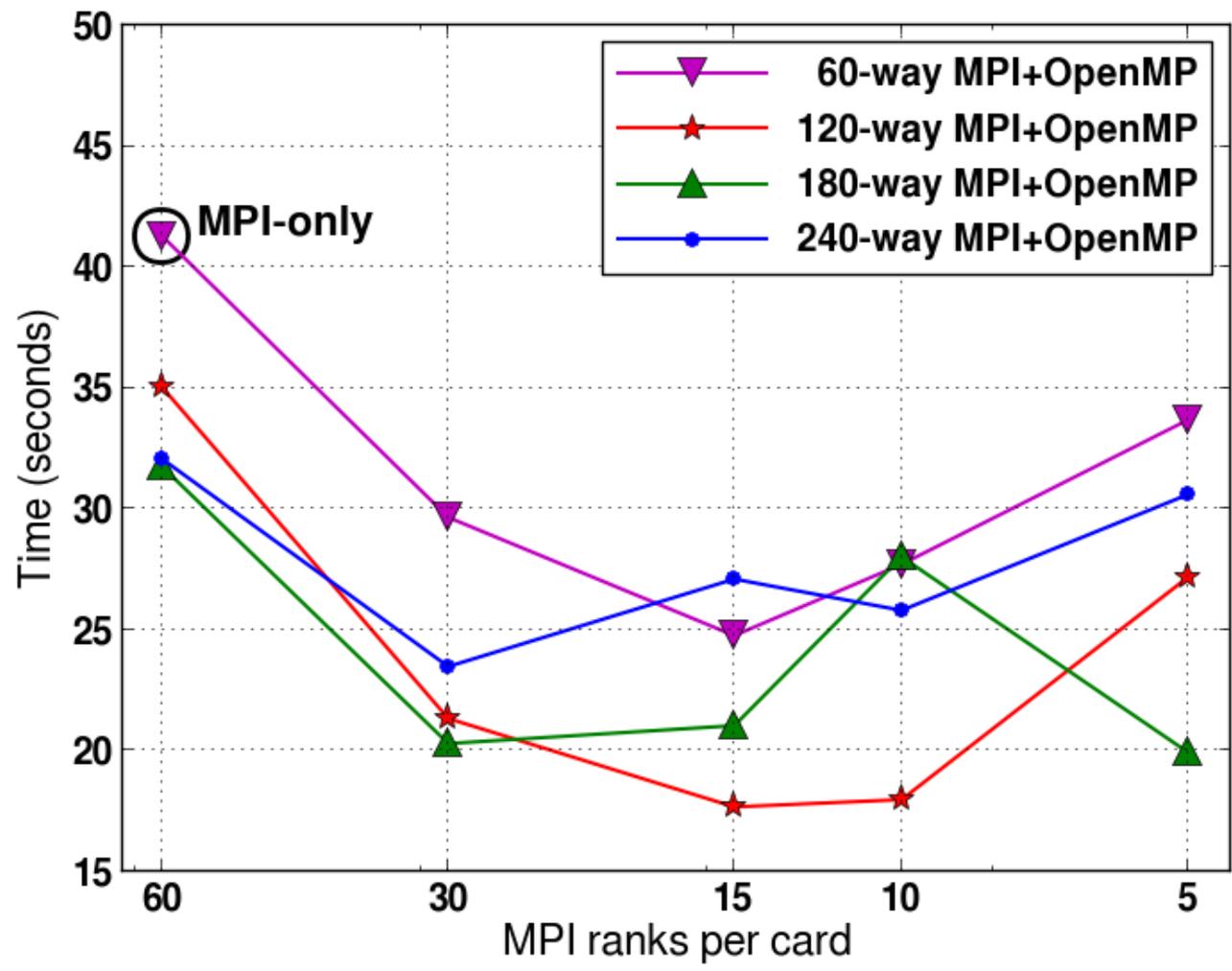
- **FLASH is an Adaptive Mesh Refinement (AMR) code with explicit solvers for hydrodynamics and magneto-hydrodynamics**
- **Parallelized using**
 - MPI domain decomposition AND
 - OpenMP multithreading over either local domains or over cells in each local domain
- **Target application is a 3D Sedov explosion problem**
 - A spherical blast wave is evolved over multiple time steps
 - We test a configuration with a uniform resolution grid (and not AMR) and use 100^3 global cells
- **The hydrodynamics solvers perform large stencil computations.**

Best MIC performance vs host

Lower is Better



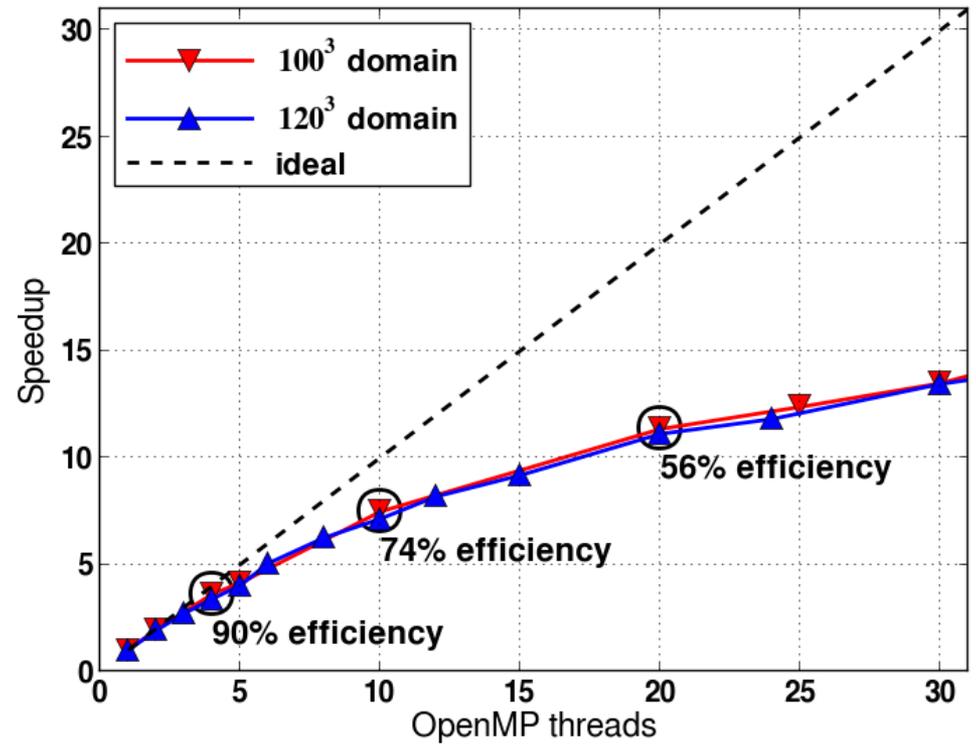
Best configuration on 1 MIC card



Lower is Better

MIC performance study 1: thread speedup

Higher is Better



- 1 MPI rank per MIC card and various numbers of OpenMP threads
- Each OpenMP thread is placed on a separate core
- 10x thread count ideally gives a 10x speedup

- Speedup is not ideal
 - But it is not the main cause of the poor MIC performance
 - ~70% efficiency @ 12 threads (as would be used with 10 MPI ranks per card)

Vectorization is another form of on-node parallelism

```
do i = 1, n  
  a(i) = b(i) + c(i)  
enddo
```



$$\begin{pmatrix} a_1 \\ \dots \\ a_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \dots \\ b_n \end{pmatrix} + \begin{pmatrix} c_1 \\ \dots \\ c_n \end{pmatrix}$$

Intel Xeon Sandy-Bridge/Ivy-Bridge:	4 Double Precision Ops Concurrently
Intel Xeon Phi:	8 Double Precision Ops Concurrently
NVIDIA Kepler GPUs:	32 SIMT threads

Things that Kill Vectorization

Compilers want to “vectorize” your loops whenever possible. But sometimes they get stumped. Here are a few things that prevent your code from vectorizing:

Loop dependency:

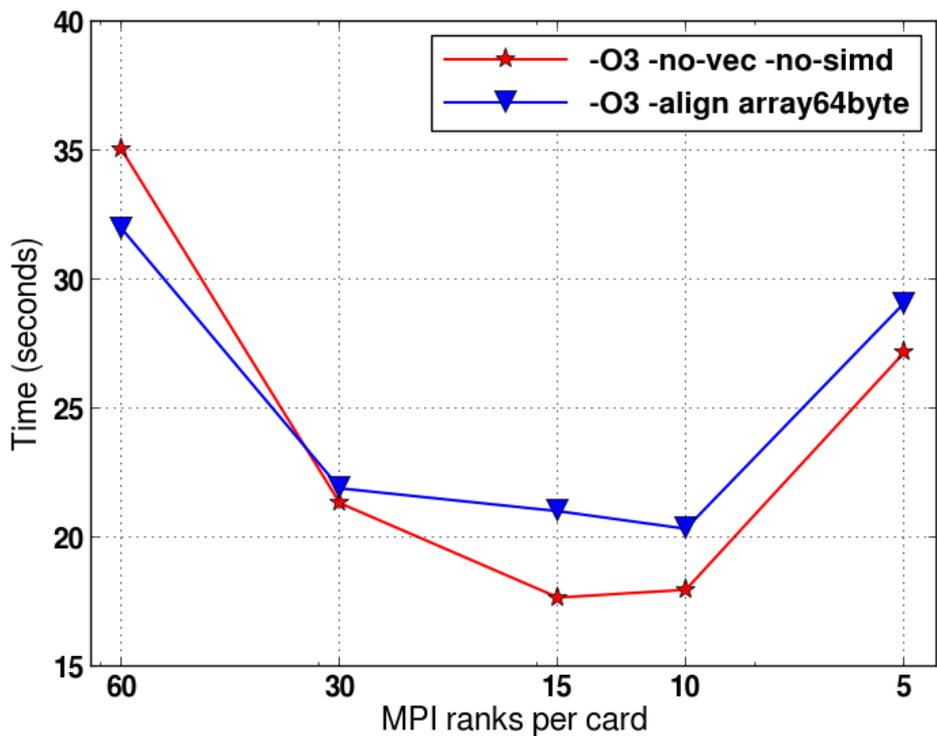
```
do i = 1, n
    a(i) = a(i-1) + b(i)
enddo
```

Task forking:

```
do i = 1, n
    if (a(i) < x) cycle
    ...
enddo
```

MIC performance study 2: vectorization

Lower is Better



No vectorization gain!

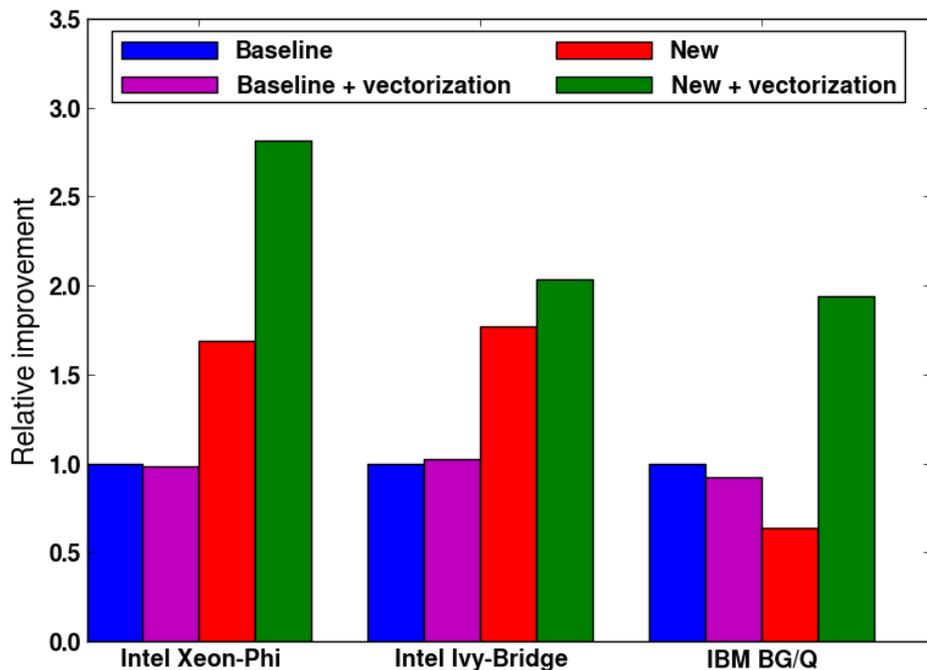
- We find that most time is spent in subroutines which update fluid state 1 grid point at a time

- The data for 1 grid point is laid out as a structure of fluid fields, e.g. density, pressure, ..., temperature next to each other: A(HY_DENS:HY_TEMP)
- Vectorization can only happen when the same operation is performed on multiple fluid fields of 1 grid point!

Enabling vectorization

- **Must restructure the code**
 - The fluid fields should no longer be next to each other in memory
 - A(HY_DENS:HY_TEMP) should become A_dens(1:N), ..., A_temp(1:N)
 - The 1:N indicates the kernels now operate on N grid points at a time
- **We tested these changes on part of a data reconstruction kernel**

Higher is Better



- **The new code compiled with vectorization options gives the best performance on 3 different platforms**

Summary

Good Parallel Efficiency AND Vectorization = Good MIC Performance

- **FLASH on MIC**
 - MPI+OpenMP parallel efficiency – OK
 - Vectorization – **zero / negative gain** ...must restructure!
 - Compiler auto-vectorization / vectorization directives do not help the current code
- **Changes needed to enable vectorization**
 - Make the kernel subroutines operate on multiple grid points at a time
 - Change the data layout by using a separate array for each fluid field
 - Effectively a change from array of structures (AoS) to structure of arrays (SoA)
- **Tested these proof-of-concept changes on a reduced hydro kernel**
 - Demonstrated improved performance on Ivy-Bridge, BG/Q and Xeon-Phi platforms

Summary



- **Change is Coming!**

- **NERSC is Here to Help Our Users**

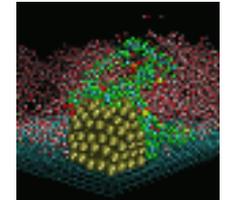
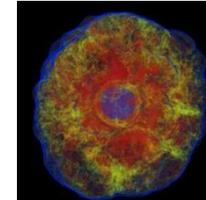
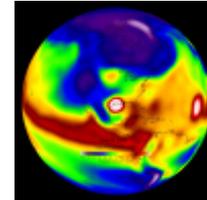
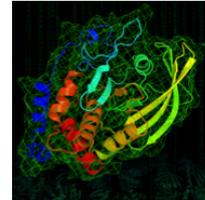
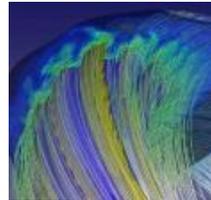
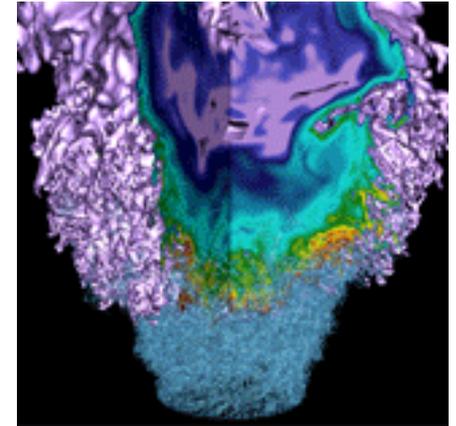
- **Good performance will require code changes**
 - ◆ Identify more on-node parallelism
 - ◆ Ensure vectorization for critical loops

- **Need to leverage community. Other centers, NERSC users, 3rd Party Developers**

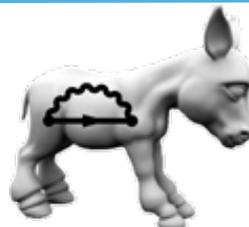
- **The code changes you make for many-core architectures will improve performance on all architectures.**

BerkeleyGW Case Study

Jack Deslippe



Case Study: BerkeleyGW



BerkeleyGW

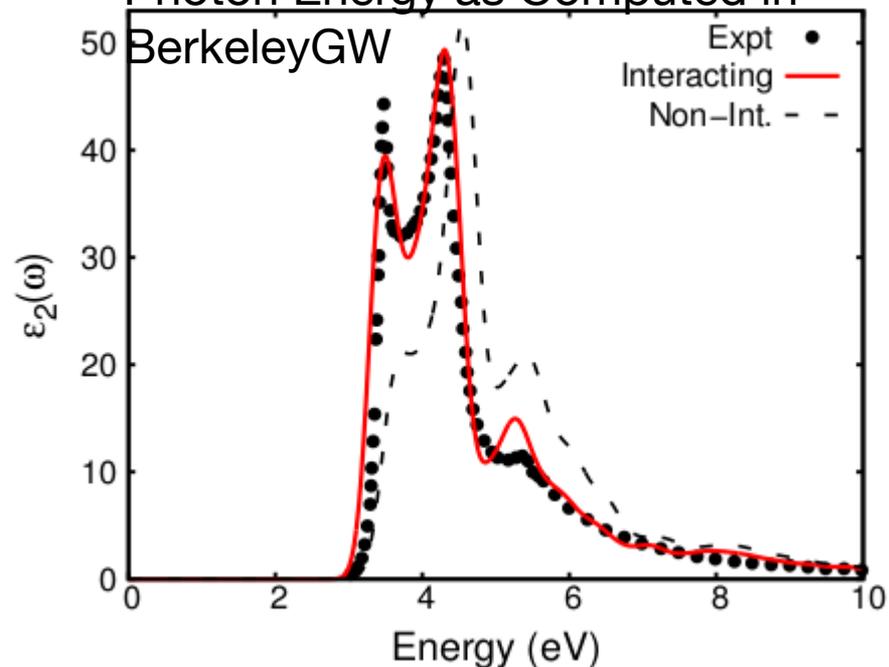
Description:

A material science code to compute excited state properties of materials. Works with many common DFT packages.

Algorithms:

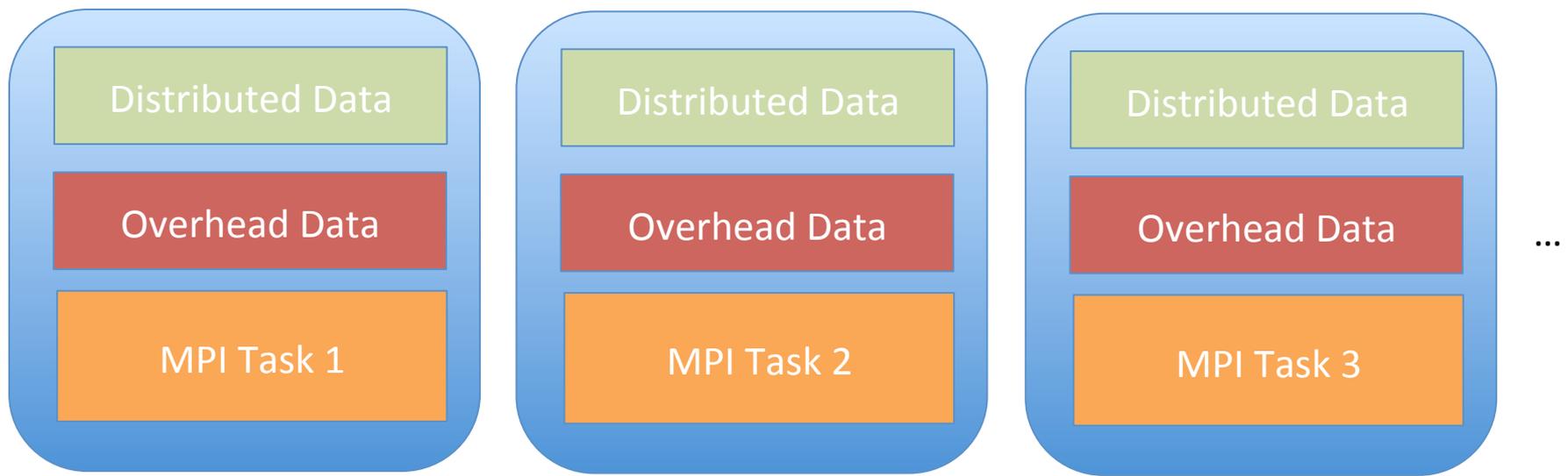
- FFTs (FFTW)
- Dense Linear Algebra (BLAS / LAPACK / SCALAPACK / ELPA)
- Large Reduction Loops.

Silicon Light Absorption vs. Photon Energy as Computed in BerkeleyGW



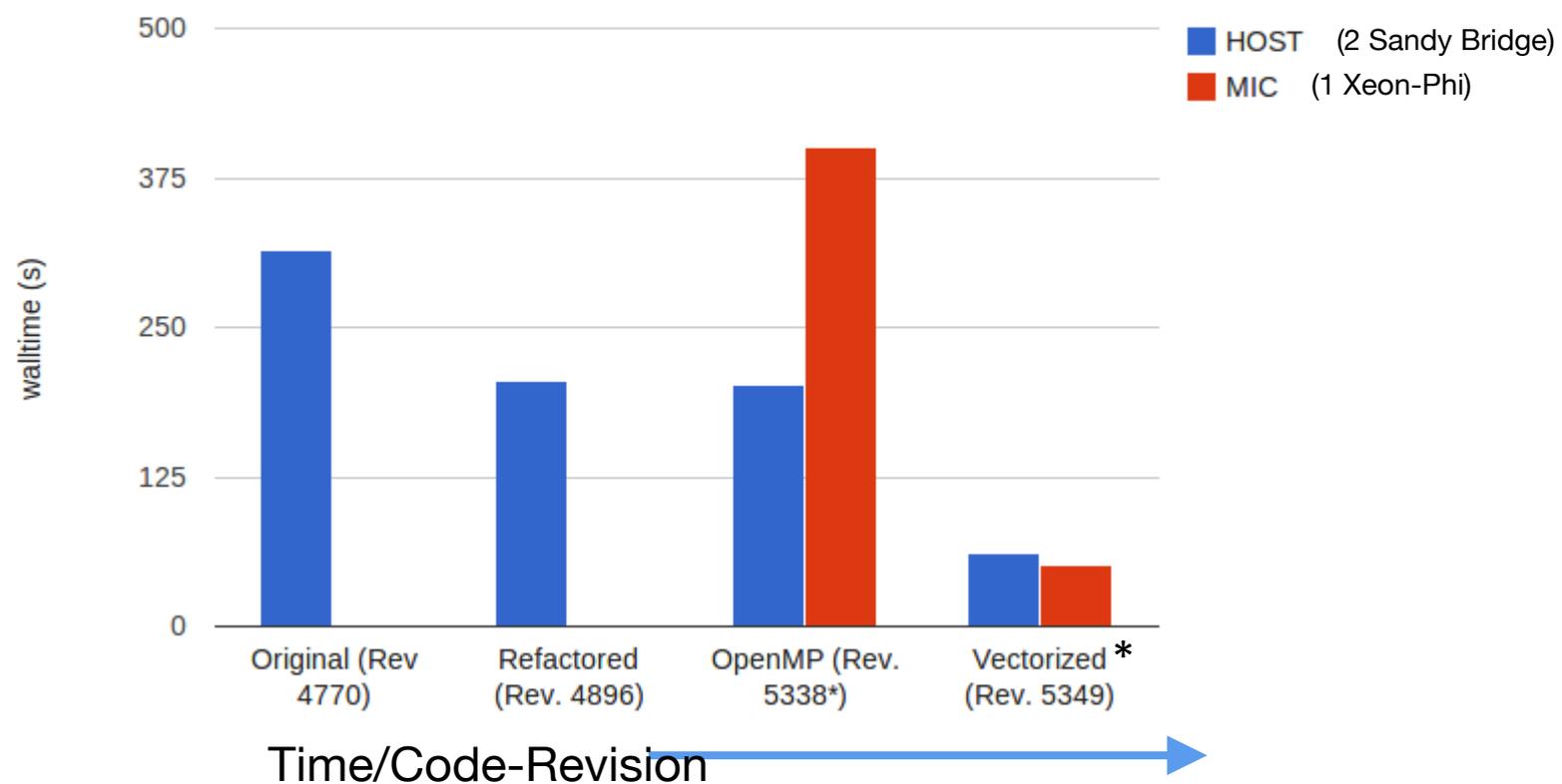
Failure of the MPI-Only Programming Model in BerkeleyGW

- ★ Big systems require more memory. Cost scales as N_{atm}^2 to store the data.
- ★ In an MPI GW implementation, in practice, to avoid communication, data is duplicated and **each MPI task has a memory overhead.**
- ★ On Hopper, users often forced to use 1 of 24 available cores, in order to provide MPI tasks with enough memory. **90% of the computing capability is lost.**



Steps to Optimize BerkeleyGW on Xeon-Phi Testbed

sigma.cplx.x main kernel performance over time



Lower is Better

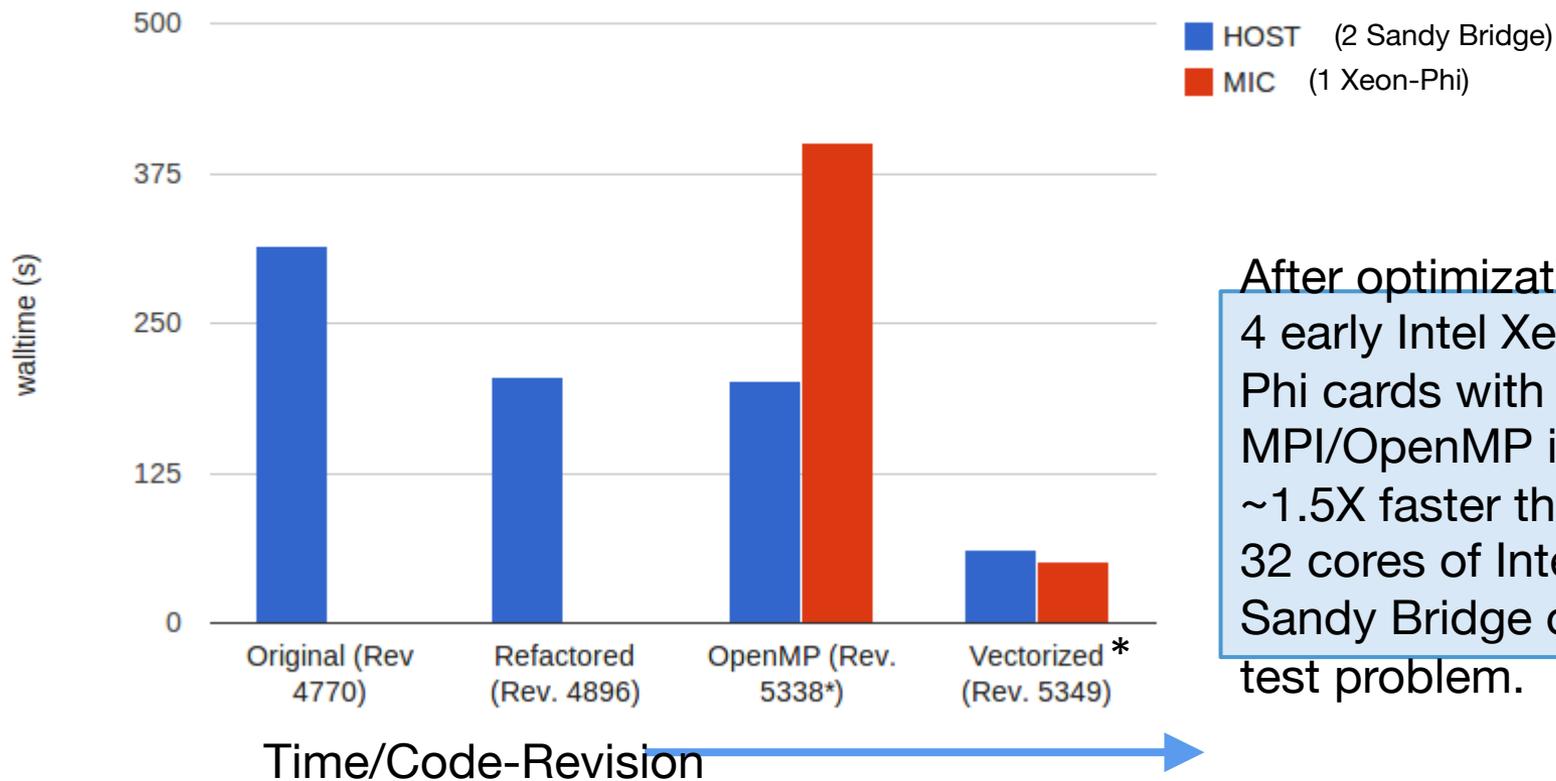
Time/Code-Revision →

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

Steps to Optimize BerkeleyGW on Xeon-Phi Testbed

sigma.cplx.x main kernel performance over time



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



Simplified Final Loop Structure

```
!$OMP DO reduction(+:achtemp)
  do my_igp = 1, ngpown

  ...

  do iw=1,3

    scht=0D0
    wxt = wx_array(iw)

    do ig = 1, ncouls

      !if (abs(wtilde_array(ig,my_igp) * eps(ig,my_igp)) .lt. TOL) cycle

      wdiff = wxt - wtilde_array(ig,my_igp)
      delw = wtilde_array(ig,my_igp) / wdiff

      ...

      scha(ig) = mygpvar1 * aqsntemp(ig) * delw * eps(ig,my_igp)

      scht = scht + scha(ig)

    enddo ! loop over g

    sch_array(iw) = sch_array(iw) + 0.5D0*scht

  enddo

  achtemp(:) = achtemp(:) + sch_array(:) * vcoul(my_igp)

enddo
```



Simplified Final Loop Structure

```
!$OMP DO reduction(+:achtemp)
do my_igp = 1, ngpown
...
do iw=1,3
  scht=0D0
  wxt = wx_array(iw)
  do ig = 1, ncouls
    !if (abs(wtilde_array(ig,my_igp) * eps(ig,my_igp)) .lt. TOL) cycle
    wdiff = wxt - wtilde_array(ig,my_igp)
    delw = wtilde_array(ig,my_igp) / wdiff
    ...
    scha(ig) = mygpvar1 * aqsntemp(ig) * delw * eps(ig,my_igp)
    scht = scht + scha(ig)
  enddo ! loop over g
  sch_array(iw) = sch_array(iw) + 0.5D0*scht
enddo
achtemp(:) = achtemp(:) + sch_array(:) * vcoul(my_igp)
enddo
```

ngpown typically
in 100's to 1000s.
Good for many

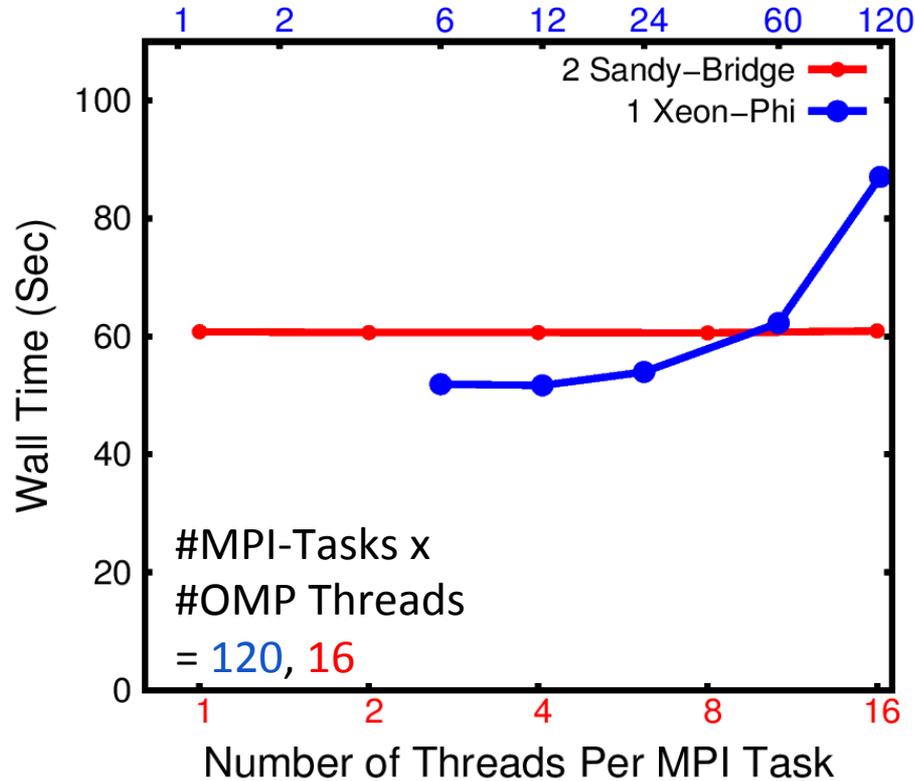
Original inner loop.
Too small to
vectorize!

ncouls typically in
1000s - 10,000s.
Good for
vectorization.
Don't have to
worry much about
memory.

Attempt to save
work breaks
vectorization and
makes code
slower.

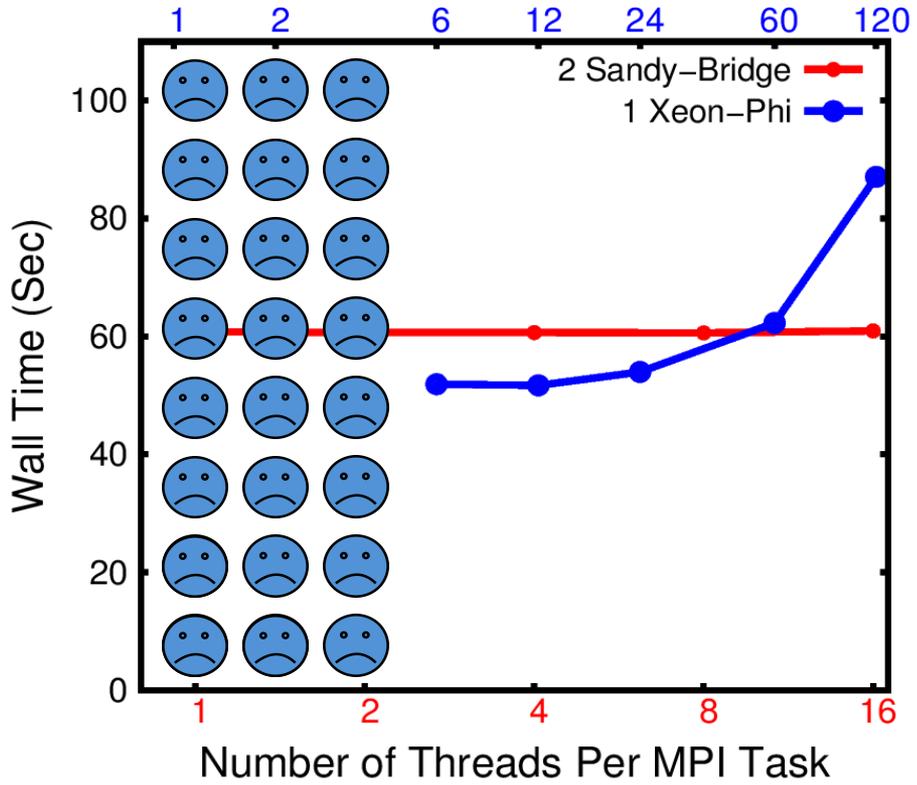
Running on Many-Core Xeon-Phi Requires OpenMP Simply To Fit Problem in Memory

Lower is Better



Running on Many-Core Xeon-Phi Requires OpenMP Simply To Fit Problem in Memory

Lower is Better



☹️ Example problem cannot fit into memory when using less than 5 OpenMP threads per MPI task.

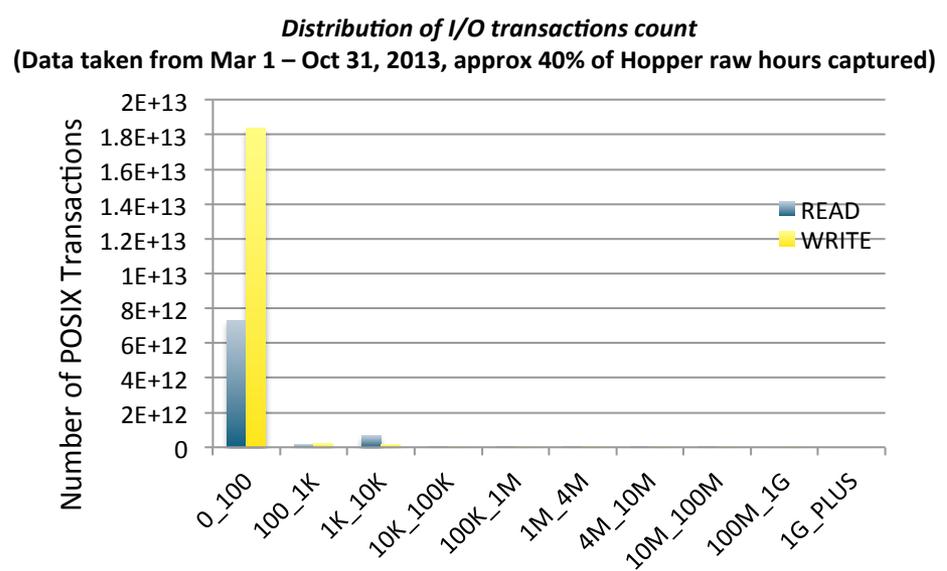
★ **Conclusion: you need OpenMP to perform well on Xeon-Phi in practice**

Burst Buffer cont.

- **NERSC I/O is not well structured**

- Even after all file system attempts to optimize I/O operations, unaligned, or small I/O patterns result in **50%** of all I/Os logged by LMT at NERSC being less than the filesystem block size.
- Given that Lustre is combining sequential small I/O ops into larger ones, this implies that these are not in the middle of a larger sequential I/O stream, and require head movement in order to be serviced, and would benefit from flash's better small block, and random I/O characteristics vs. disk.

Application(client) I/O request sizes



Lustre(server) I/O sizes

