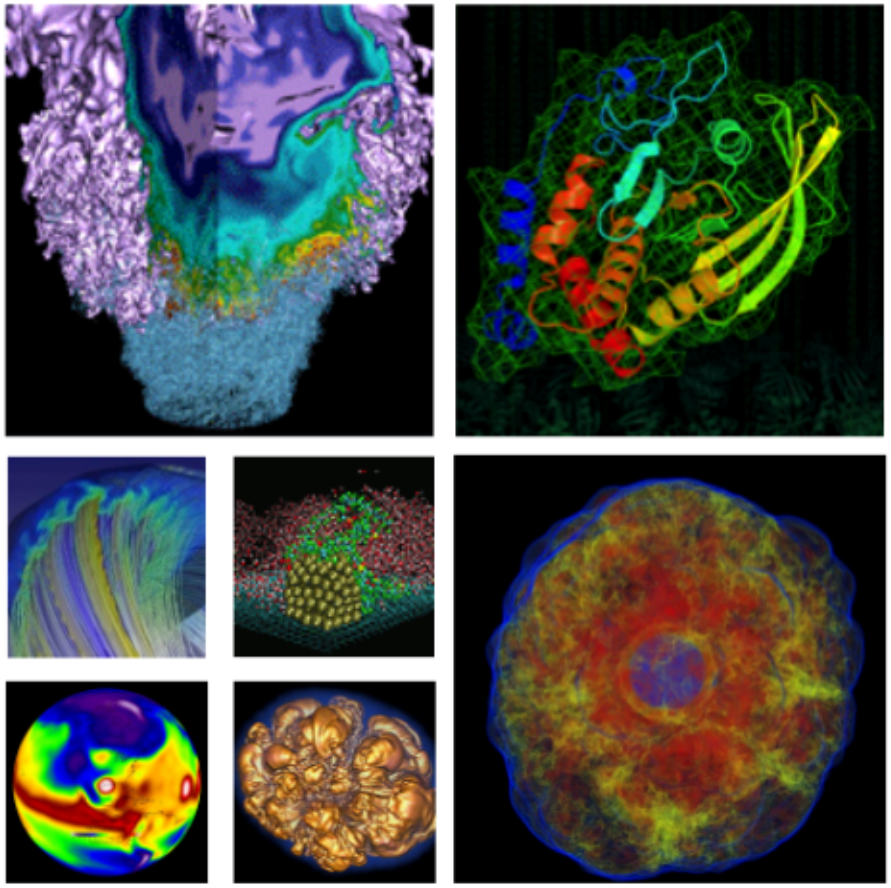
# Preparing Your Applications for Future NERSC Machines



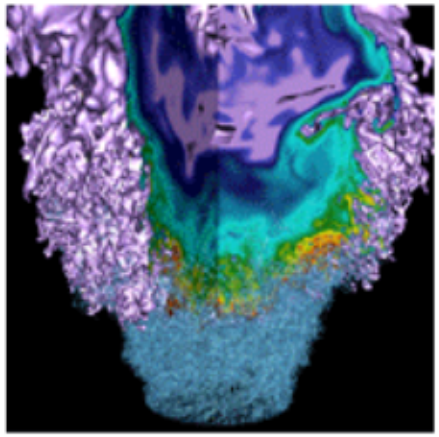
Jack Deslippe
NERSC User Services

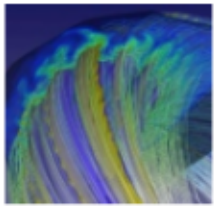
10/8/2013

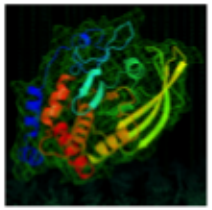
U.S. DEPARTMENT OF ENERGY | Office of Science

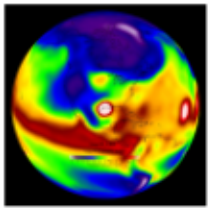
BERKELEY LAB

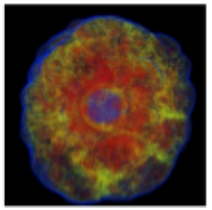
# What HPC Will Look Like at NERSC in 2017

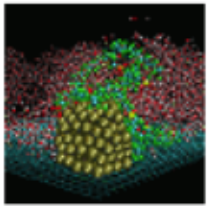












NERSC

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB
Lawrence Berkeley National Laboratory

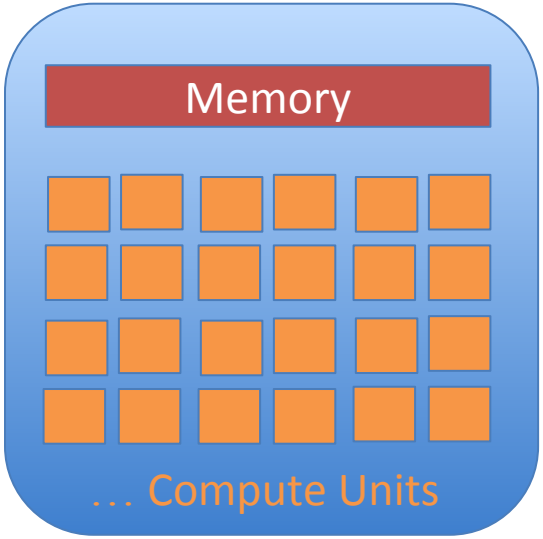
# Disruptive changes are coming!

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

# The Future Will Have Many-Cores

Due primarily to power constraints, chip vendors are moving to "many-core" architectures:

| | |
|---|---|
| Consumer/Server CPUs: | 10's of Threads per Socket |
| Intel Xeon-Phi: | 100's of Threads per Socket |
| NVIDIA GPUs: | 1000's of Threads per Socket |

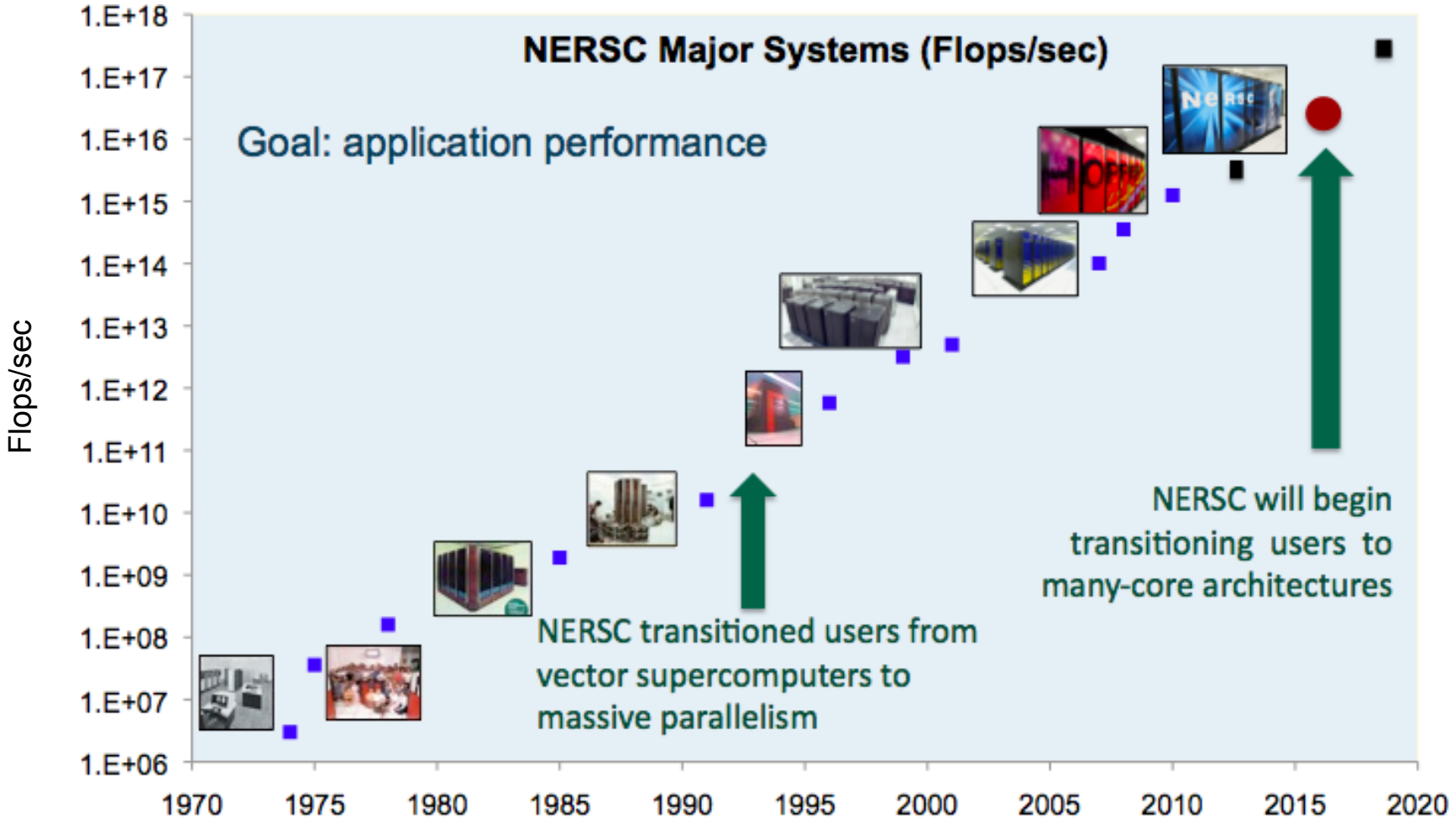


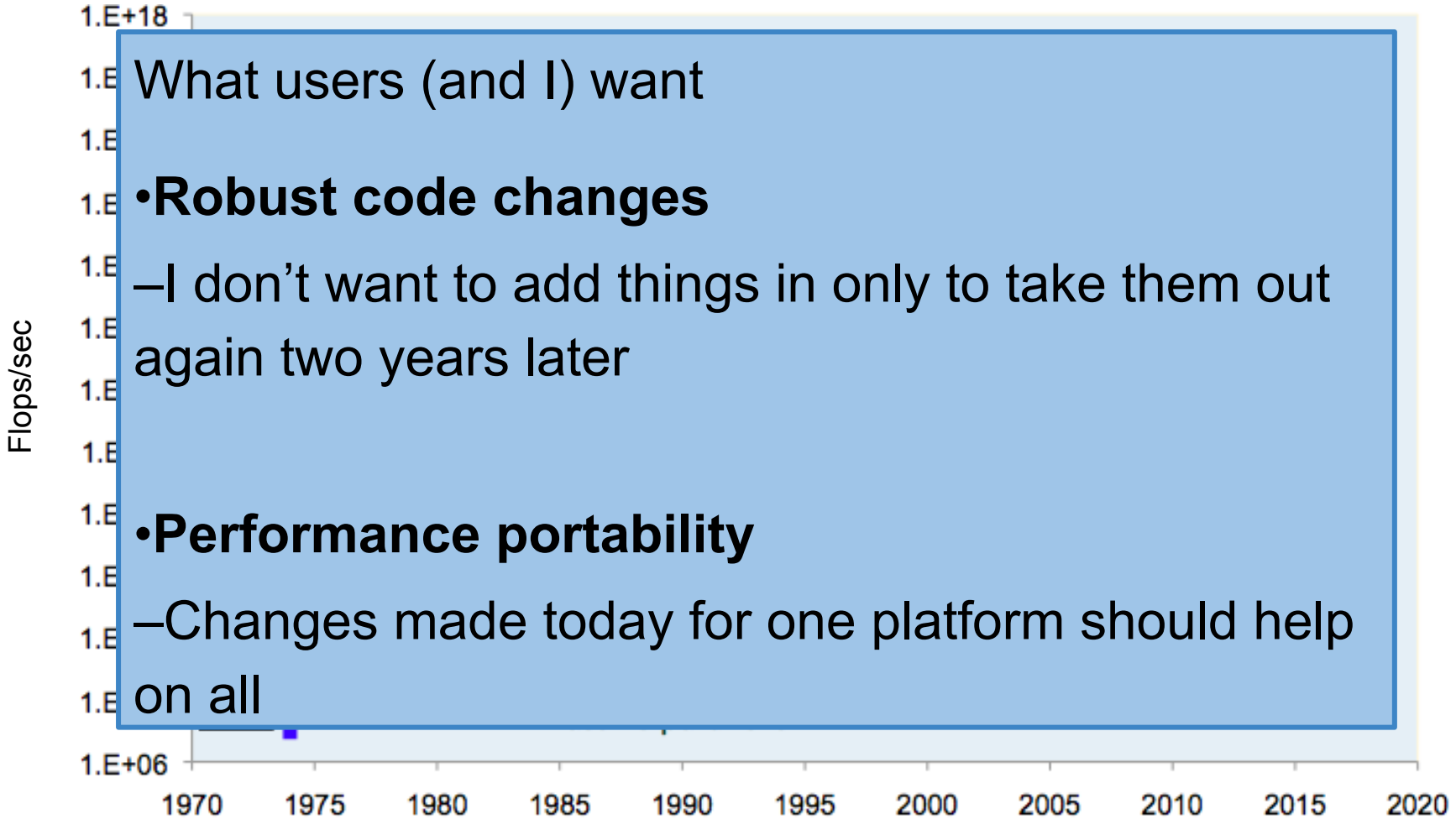

No matter what chip architecture is in NERSC's 2017 machines, **compute nodes will have many compute units with shared memory**.

Memory per compute-unit is not expected to rise.

The only way that NERSC can continue to provide compute speed improvements that meet user need is by moving to **"energy-efficient" architectures; tend to have lower clock-speeds, rely heavily on vectorization/SIMD.**

# NERSC Roadmap





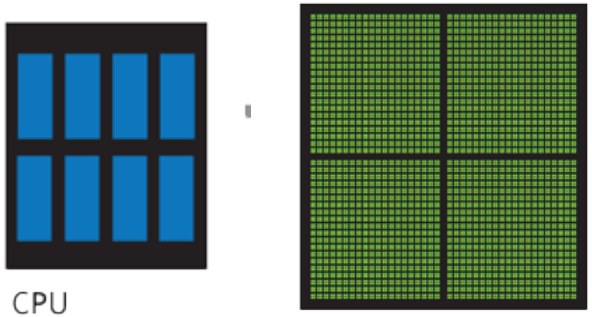

What users (and I) want

- **Robust code changes**
  - –I don't want to add things in only to take them out again two years later
- **Performance portability**
  - –Changes made today for one platform should help on all

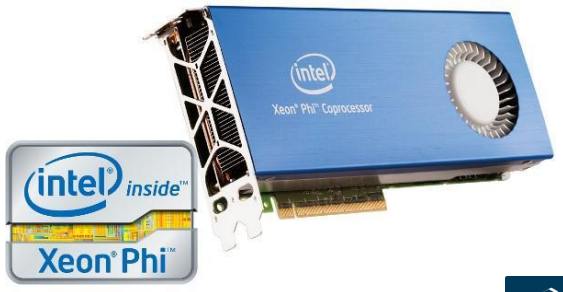
# The many-core challenge for application developers

For the last decade, we've enjoyed massively parallel machines with MPI as the standard programming method for exposing parallelism between nodes.


To study larger physical systems of interest, and get the most out of HPC resources, **we now need to exploit "on-node" parallelism and manage memory effectively.**




The recommended programming model for Edison is already MPI (between nodes) and OpenMP on node.

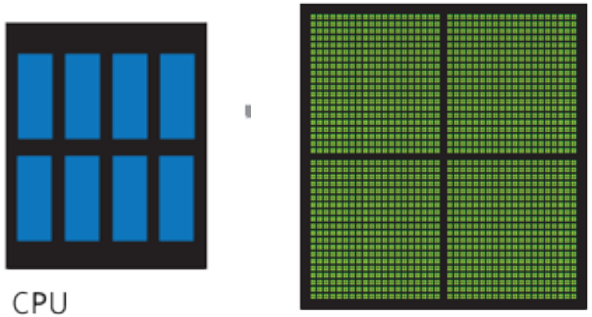



Xeon-Phi
100+ hardware threads

# The many-core challenge for application developers

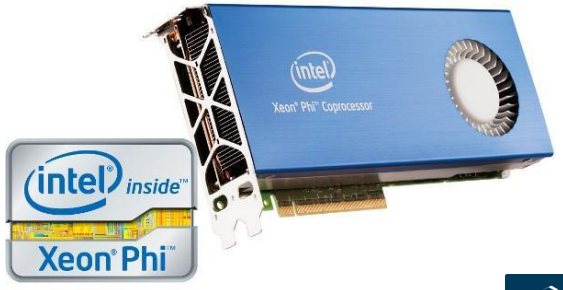
For the last decade, we've enjoyed massively parallel machines with MPI as the standard programming method for exposing parallelism between nodes.


To study larger physical systems of interest, and get the most out of HPC resources, **we now need to exploit "on-node" parallelism and manage memory effectively.**




The recommended programming model for Edison is already MPI (between nodes) and OpenMP on node.

**Porting to MPI+OpenMP on Edison, will position you for "many-core"**




Xeon-Phi
100+ hardware threads

# Vectorization

There is a another important form of on-node parallelism

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

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

Vectorization: CPU does identical operations on different data; e.g., multiple iterations of the above loop can be done concurrently.

# Vectorization

There is a another important form of on-node parallelism

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

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

Vectorization: CPU does identical operations on different data; e.g., multiple iterations of the above loop can be done concurrently.

| | |
|---|---|
| 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
    if (a(i) > x) …
enddo
```

# NERSC is committed to helping BES

*Help transition the NERSC workload to future architectures by exploring and improving application performance on manycore architectures.*

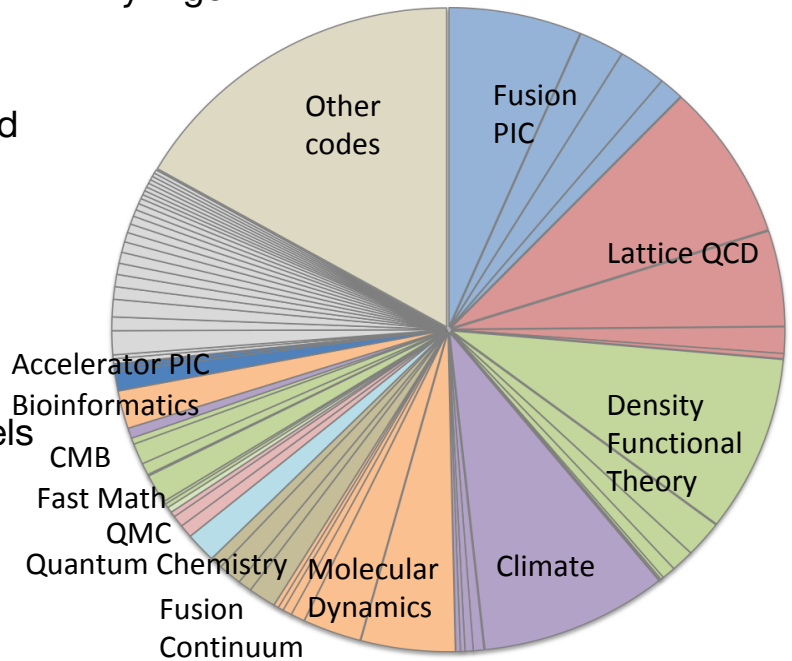
## Phase 1:

- ➔ Identify major algorithms in the NERSC workload. Assigned 14 codes to represent class.
  - ◆ BES Codes:
    - Quantum ESPRESSO/BGW (DFT Proxy)
    - NWCHEM (Quantum Chemistry Proxy)
    - Amber (MD Proxy)
    - Zori (QMC Proxy)
- ➔ Profile OpenMP/MPI scaling and vectorization in key kernels on GPU testbed (dirac) and Xeon-Phi testbed (babbage).

## Phase 2:

- ➔ Organize user training around OpenMP and vectorization.
- ➔ Meet with key application developers / workshops
- ➔ User accessible test-bed systems.

NERSC Workload
By Algorithm



# NERSC is Here to Help

NERSC is kicking off an “Application Readiness” effort. Devoting significant staff effort to help users and developers port their codes to many-core architectures

Katerina Antypas
(Co-Lead)

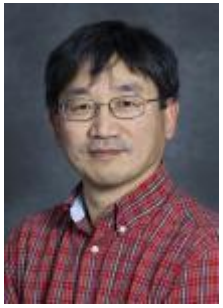
Nick Wright (Co-Lead)
Amber (Proxy for NAMD, LAMMPS)

Harvey Wasserman
SNAP ($S_N$ transport proxy)

Brian Austin
Zori (Proxy for QWalk etc.)

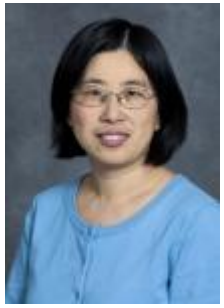
Hongzhang Shan
NWChem (Proxy for qchem, GAMESS)

Aaron Collier
Madam-Toast / Gyro



Woo-Sun Yang
CAM (Proxy for CESM)

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



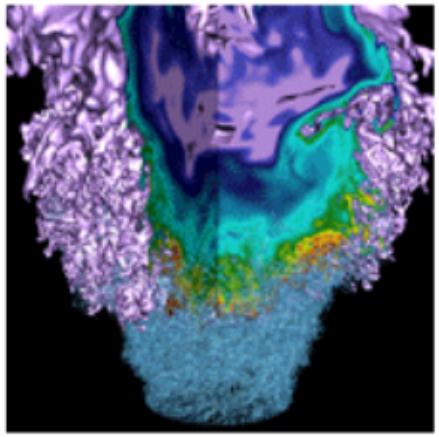
Helen He
WRF

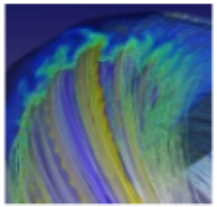
Matt Cordery
MPAS

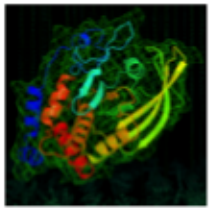
Kirsten Fagnan
Bio-Informatics

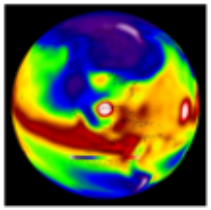
Hari Krishnan
Vis.

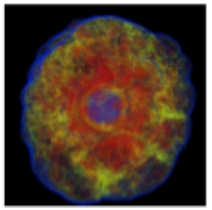
# Case Study

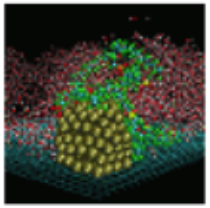












# Case Study: BerkeleyGW

BerkeleyGW

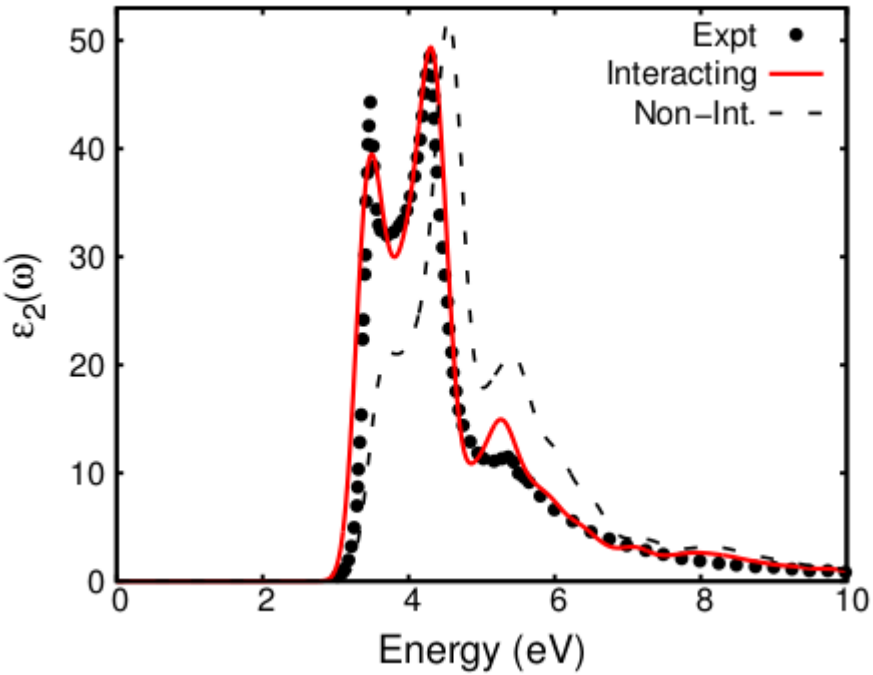
**Description:**

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

**Algorithms:**

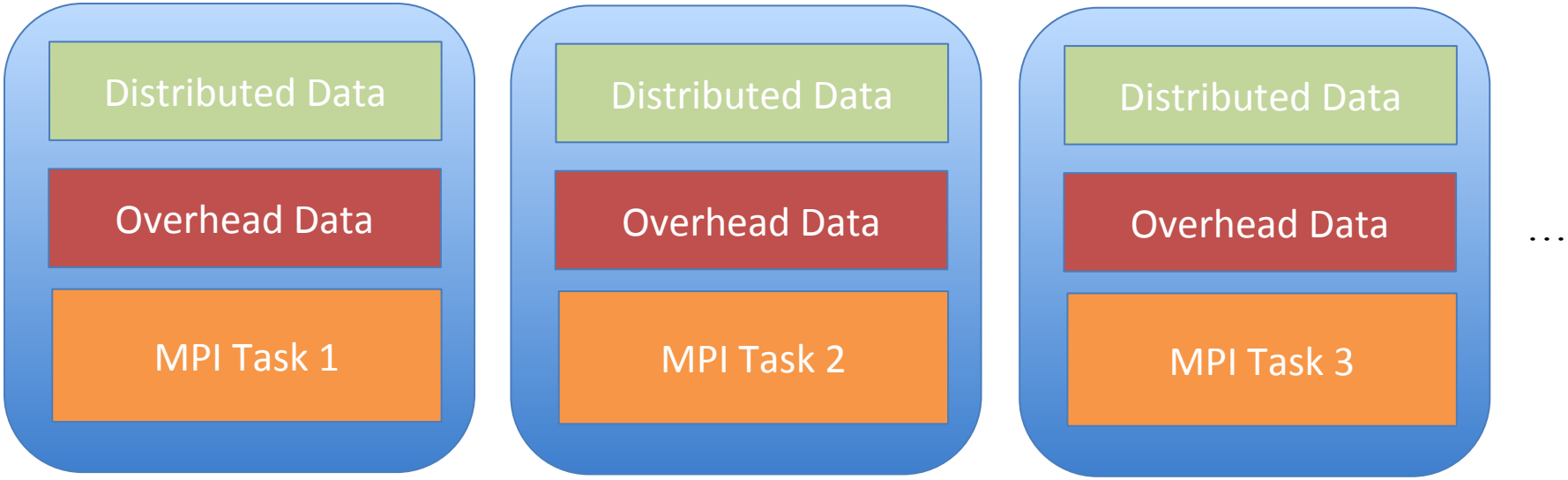
- FFTs (FFTW)
- Dense Linear Algebra (BLAS / LAPACK / SCALAPACK / ELPA)
- Large Reduction Loops.

Silicon Light Absorption vs. Photon Energy as Computed in BerkeleyGW

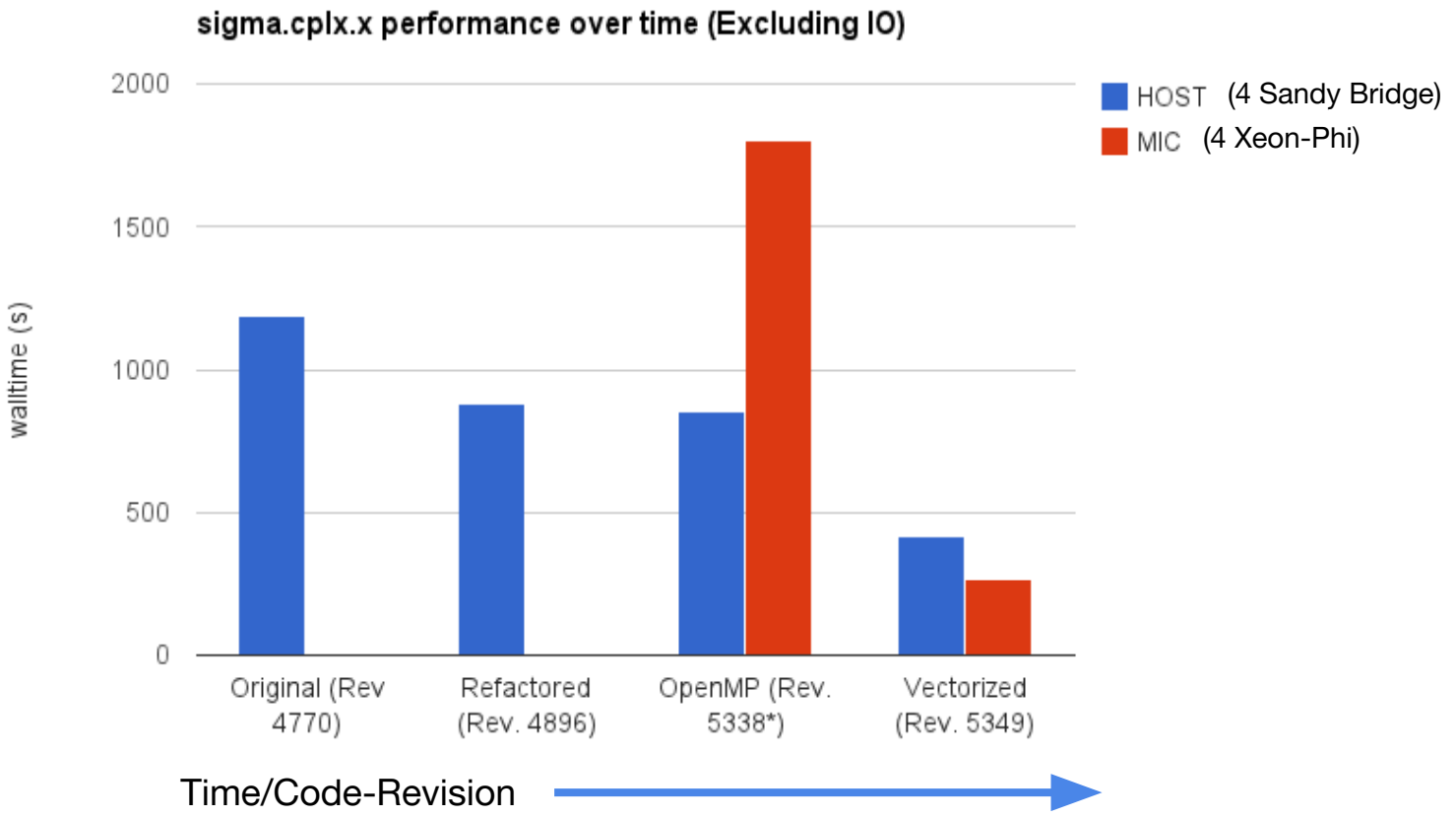


# Failure of the MPI-Only Programming Model in BerkeleyGW

- ★ 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**.

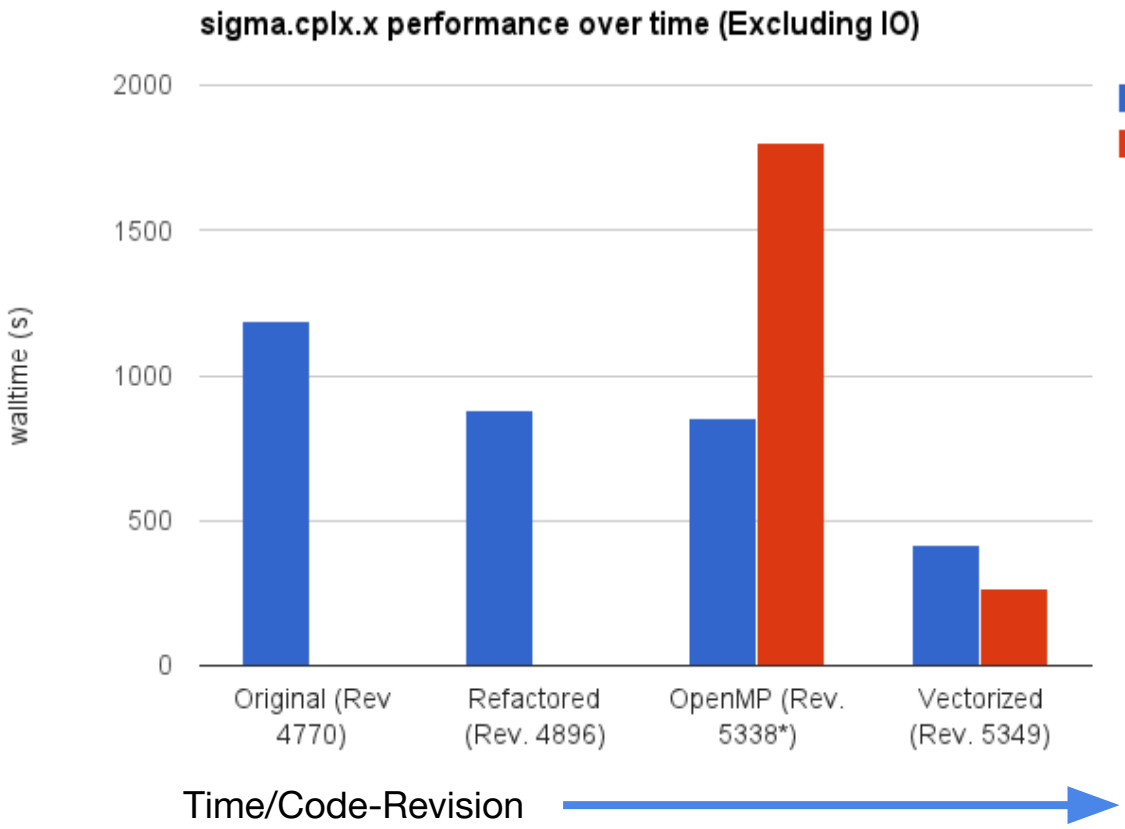


# Steps to Optimize BerkeleyGW on Xeon-Phi Testbed




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.

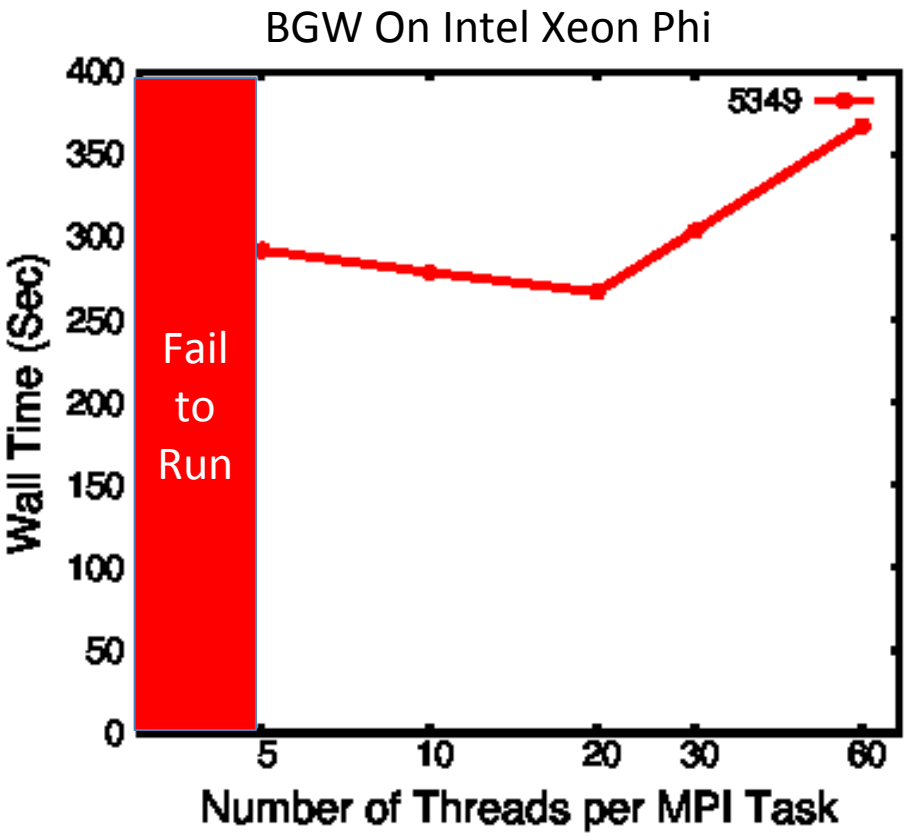
# Steps to Optimize BerkeleyGW on Xeon-Phi Testbed




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.

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

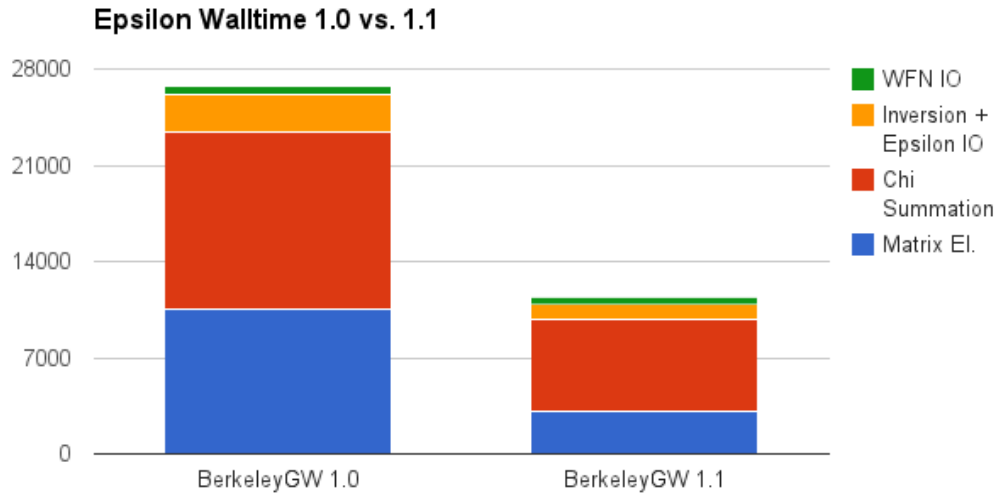



★ Example problem cannot fit into memory when using less than 5 OpenMP threads per MPI task.

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

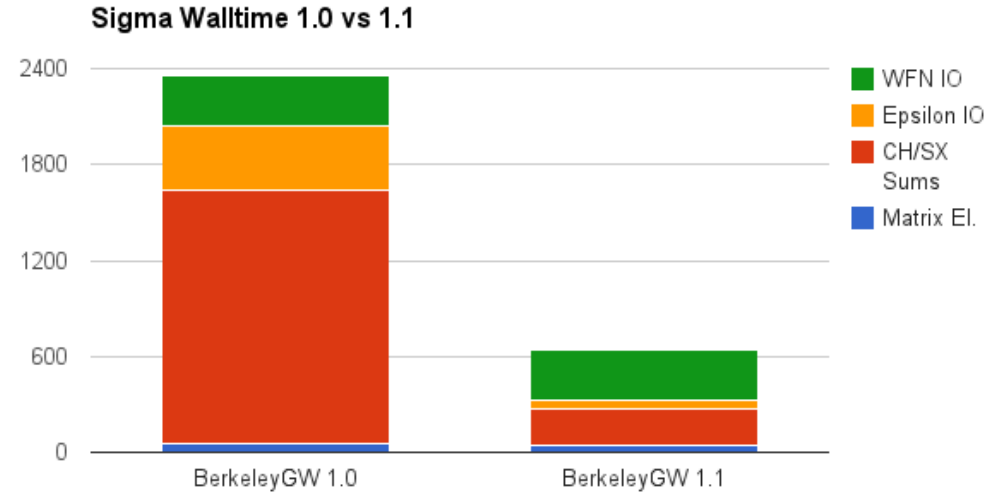
# Improvements for Many-Core improve your Code on Hopper.

Hopper Runtimes: BGW 1.0 vs 1.1




12 MPI Tasks/Node on Hopper (Requires > 2GB per MPI Task).

**Speed-Up on Hopper Largely Due to Addition of OpenMP Support**

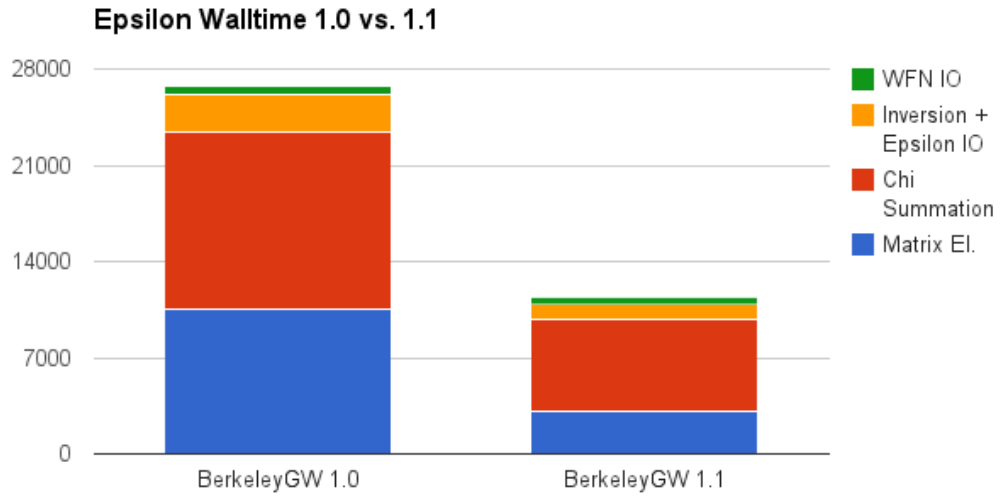



24 MPI Tasks Per Node.

**Speed-Up on Hopper Largely Due to Vectorization and addition of Parallel-IO**

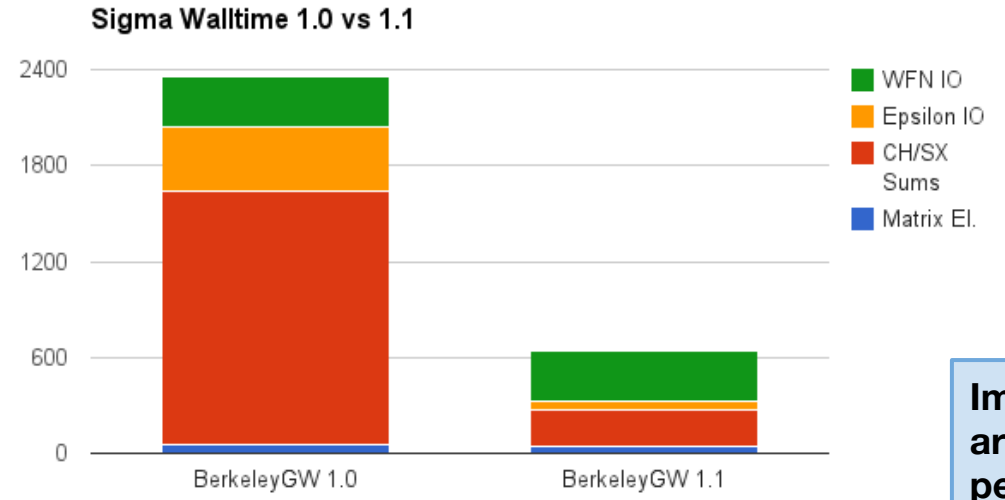
# Improvements for Many-Core improve your Code on Hopper.

Hopper Runtimes: BGW 1.0 vs 1.1




12 MPI Tasks/Node on Hopper (Requires > 2GB per MPI Task).

**Speed-Up on Hopper Largely Due to Addition of OpenMP Support**

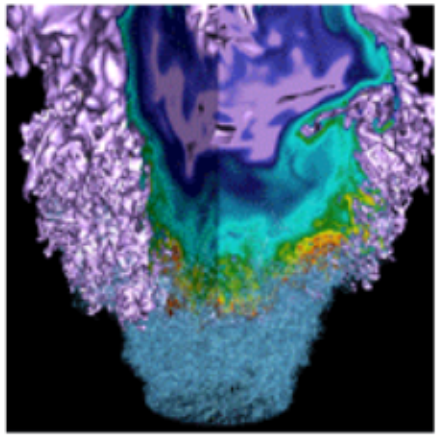


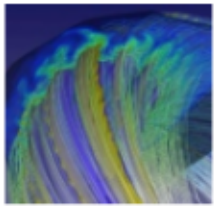

24 MPI Tasks Per Node.

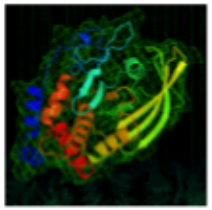
**Speed-Up on Hopper Largely Due to Vectorization and addition of Parallel-IO**

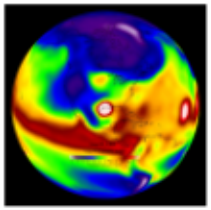
**Improving your code for advanced architectures can result in performance improvements on traditional architectures.**

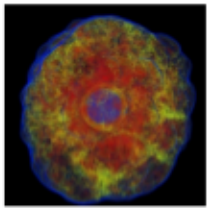
# Conclusions and Lessons Learned

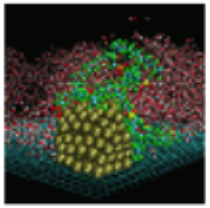












# Summary

- ➔ Disruptive Change is Coming!
- ➔ NERSC is Here to Help
- ➔ Good performance will require code changes
  - ◆ Identify more on-node parallelism
  - ◆ Ensure vectorization for critical loops
- ➔ The code changes you make for many-core architectures will improve performance on all architectures.

# NERSC is Here to Help

But We Need Your Help Too….

Let us know the preparation status of the codes you use. Let us know which developers to get in touch with and what features are important to you.

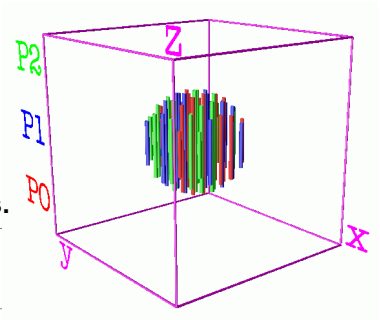

# National Energy Research Scientific Computing Center

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB
Lawrence Berkeley National Laboratory

# Hybrid OpenMP + MPI For Distributed FFTs in DFT Minimizes All-to-All Communication Costs

**Hybrid (OpenMP/MPI) reduces MPI communication costs over pure MPI implementation.**



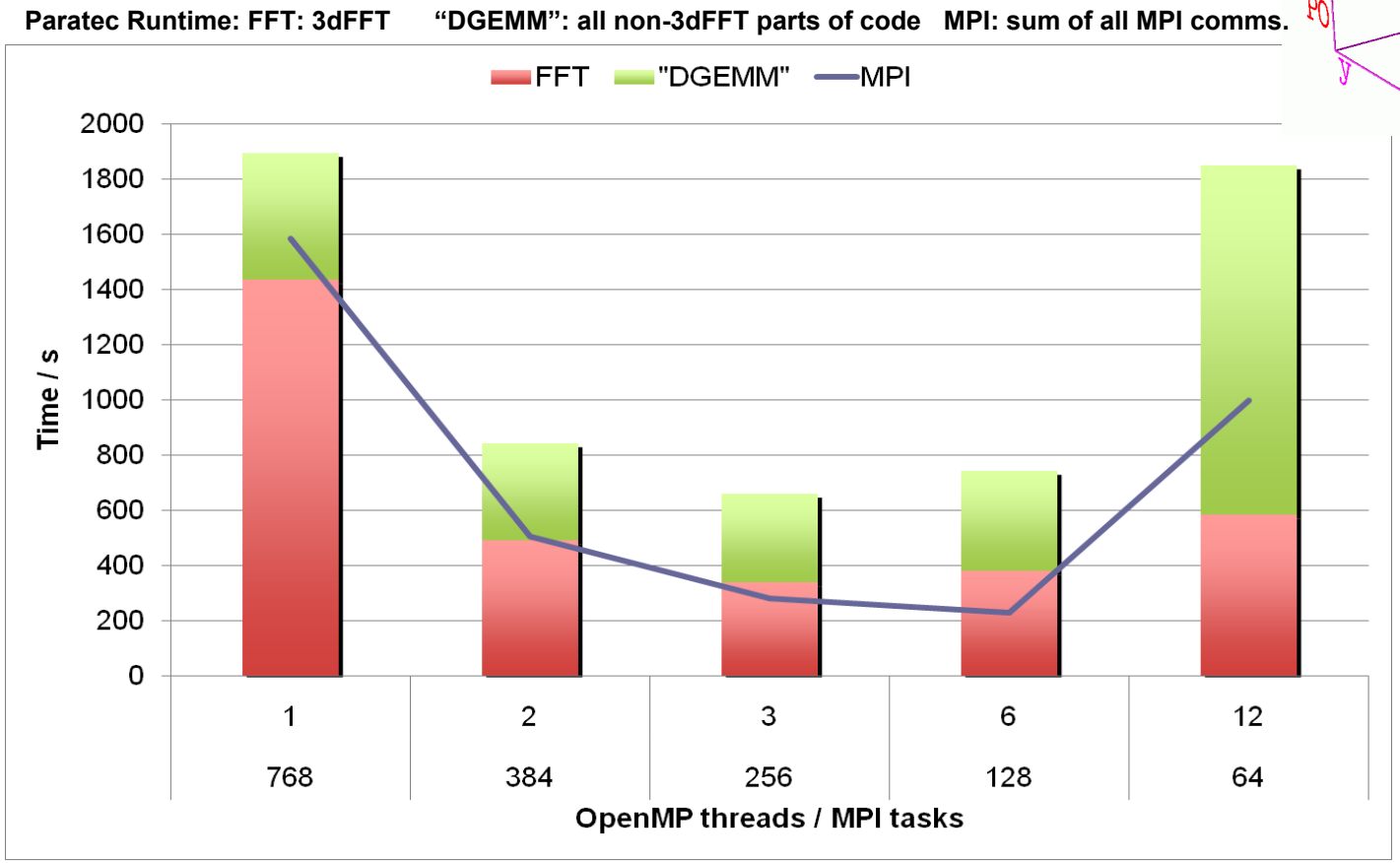




Figure courtesy of Andrew Canning. Test on Jaguar, OLCF.

# Blank Slide

Blank Content