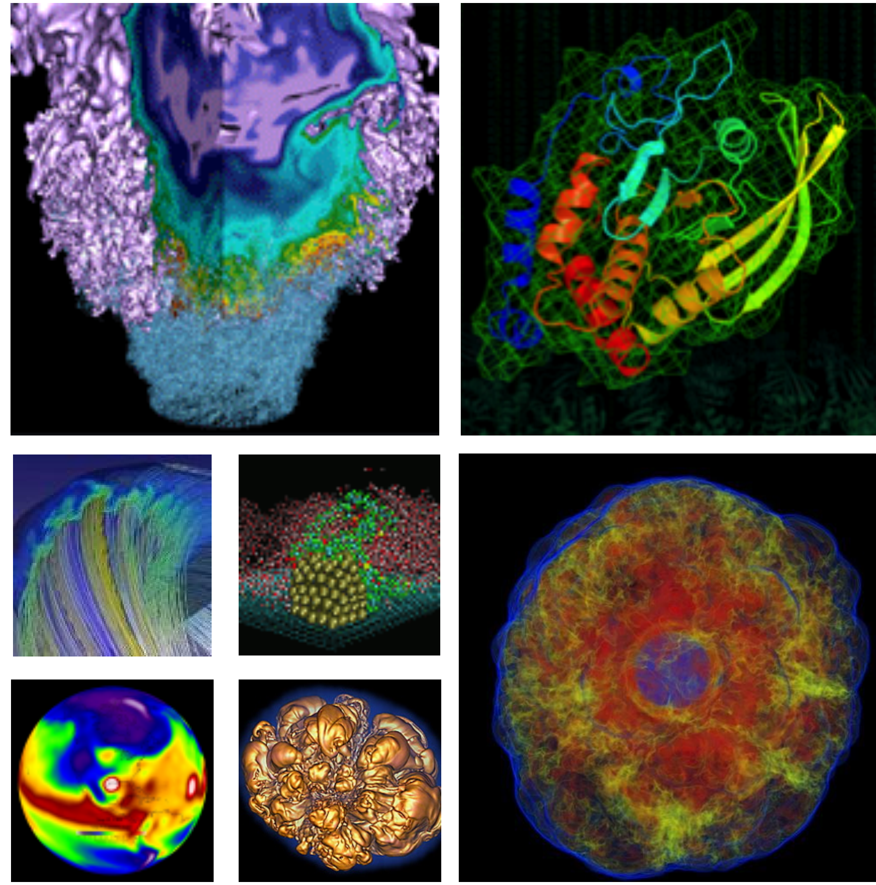
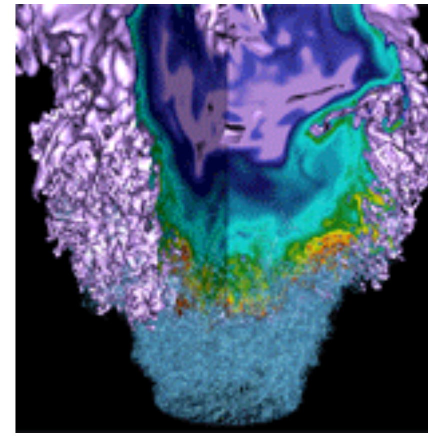
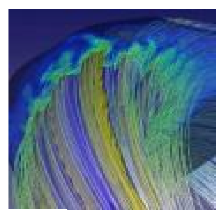
Preparing Applications for Future NERSC Architectures

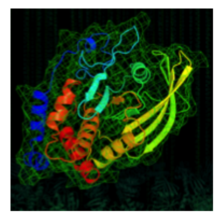


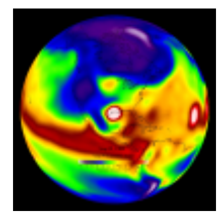
Jack Deslippe
NERSC User Services

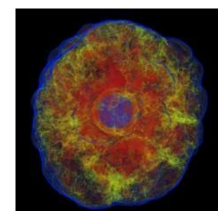
What We are Telling Users

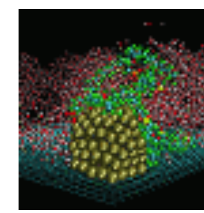












NERSC 40 YEARS at the FOREFRONT

Disruptive changes are coming!

- If you do nothing, your MPI-only code may run poorly on future machines
- Changes affect entire HPC community
- NERSC is here to help and here to lead

3 Important Areas of Change

- More cores (and/or hardware threads) per node
- Vectorization will become critical to performance.
- Hierarchical memory

3 Important Areas of Change

- More cores (and/or hardware threads) per node
- Vectorization will become critical to performance.

The App-readiness has been focused on these two changes in phase 1, since these affect all architectures.

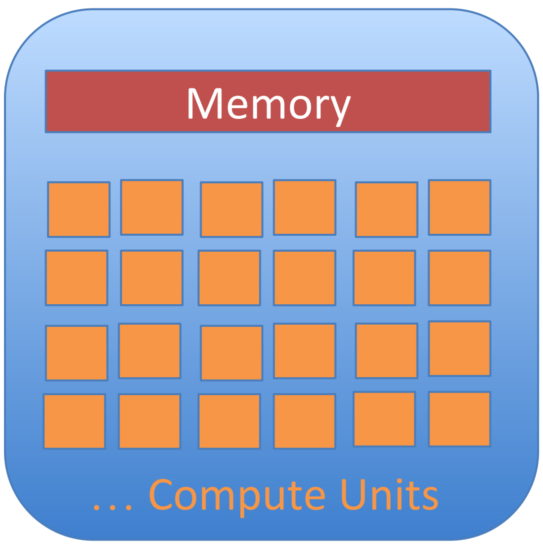
The nature of memory hierarchies is architecture dependent.

The Future Will Have Many-Cores

For the last decade: we've enjoyed massively parallel machines with MPI as the standard programming method

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.**

Vectorization

There is a another important form of on-node parallelism

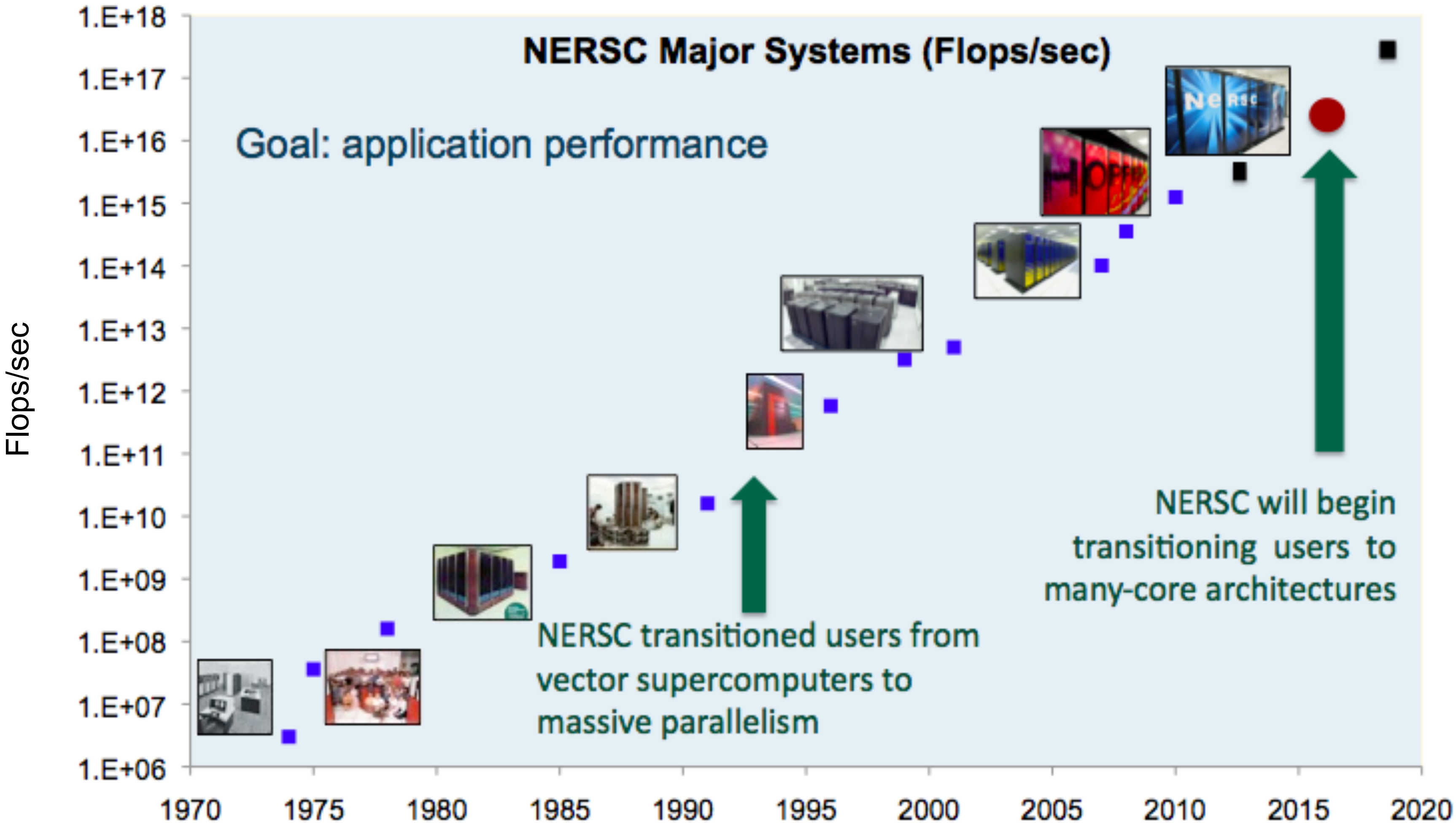
```
do i = 1, n
     a(i) = b(i) + c(i)
enddo
```

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

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

NERSC Roadmap



U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

NERSC is committed to helping our users

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.
  - 1 team member per code

➔ Code status discovery
  - What has been done at other centers
  - How are various code teams preparing

➔ Profile OpenMP/MPI scaling and vectorization in key kernels on GPU testbed (dirac) and Xeon-Phi testbed (babbage).

Phase 2:

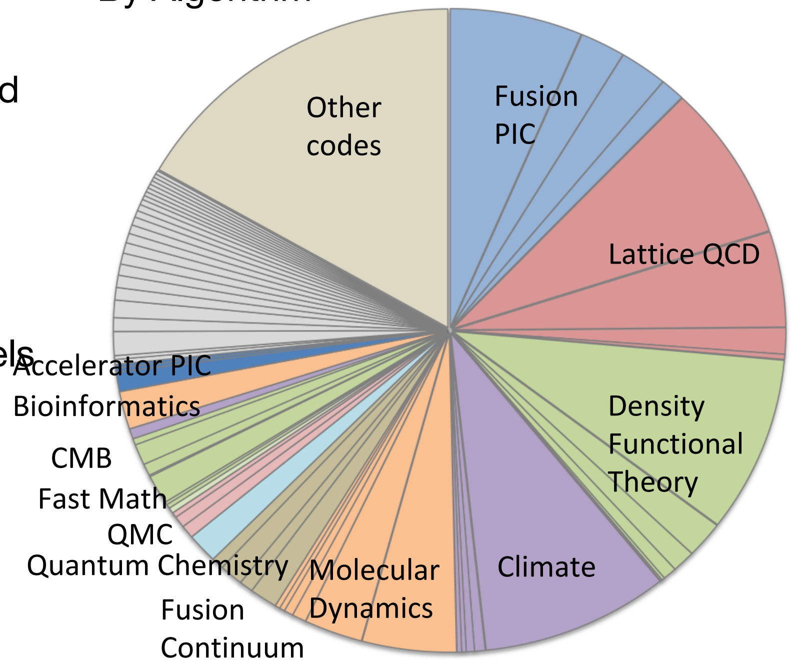
➔ Organize user training around node-parallelism, vectorization and other architecture specific details.

➔ Meet with key application developers / workshops at NERSC. Leverage/lead community efforts.

➔ Application deep dives.

➔ User accessible test-bed systems.

NERSC Workload By Algorithm



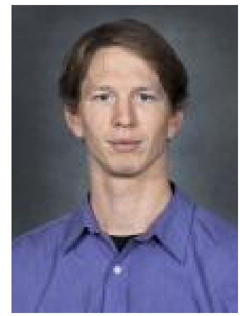
NERSC App Readiness Team

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)

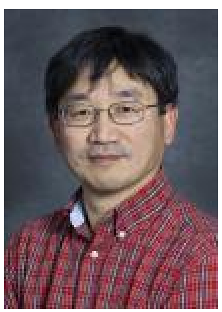
Harvey Wasserman
SNAP (S_N transport proxy)



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)

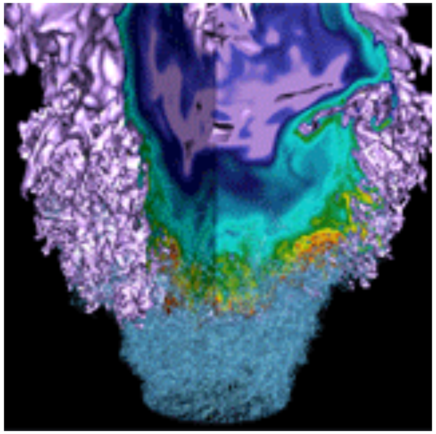
Helen He
WRF

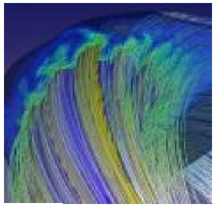
Matt Cordery
MPAS

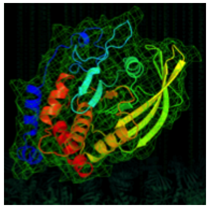
Kirsten Fagnan
Bio-Informatics

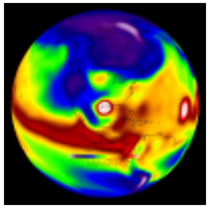
Christopher Daley
FLASH

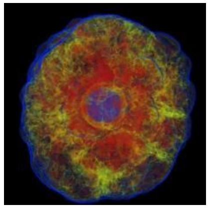
BerkeleyGW Case Study

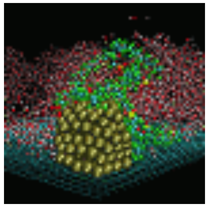












NERSC 40 YEARS at the FOREFRONT

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Case Study: BerkeleyGW

BerkeleyGW

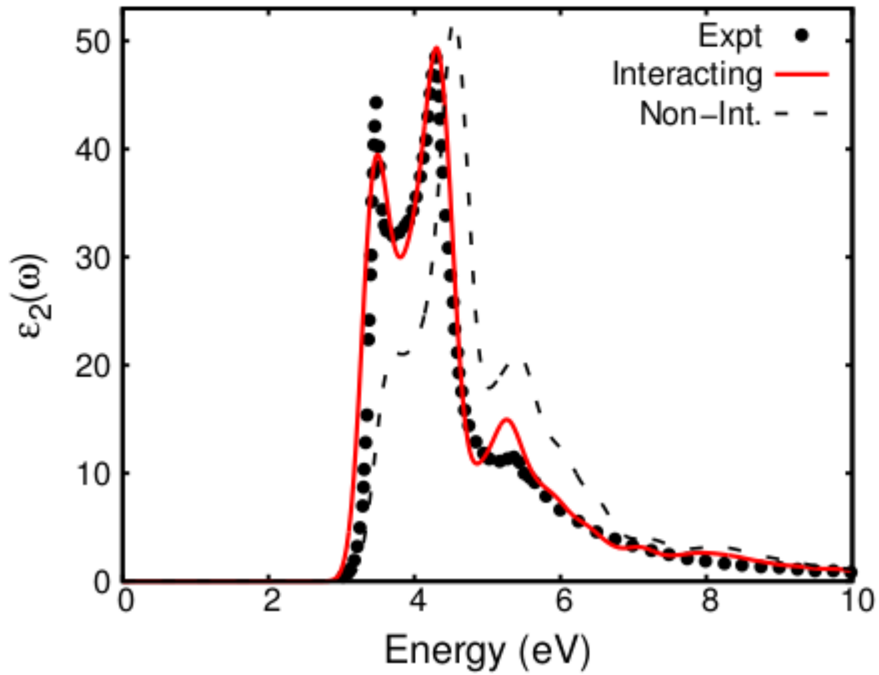
Description:

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

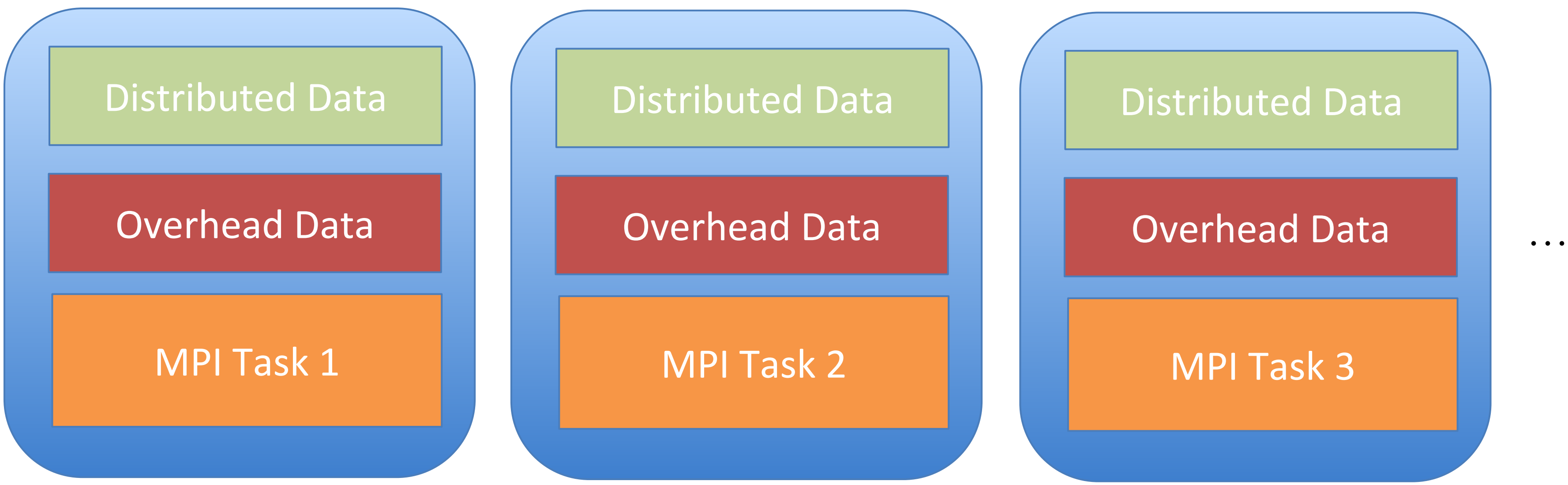


Failure of the MPI-Only Programming Model in BerkeleyGW

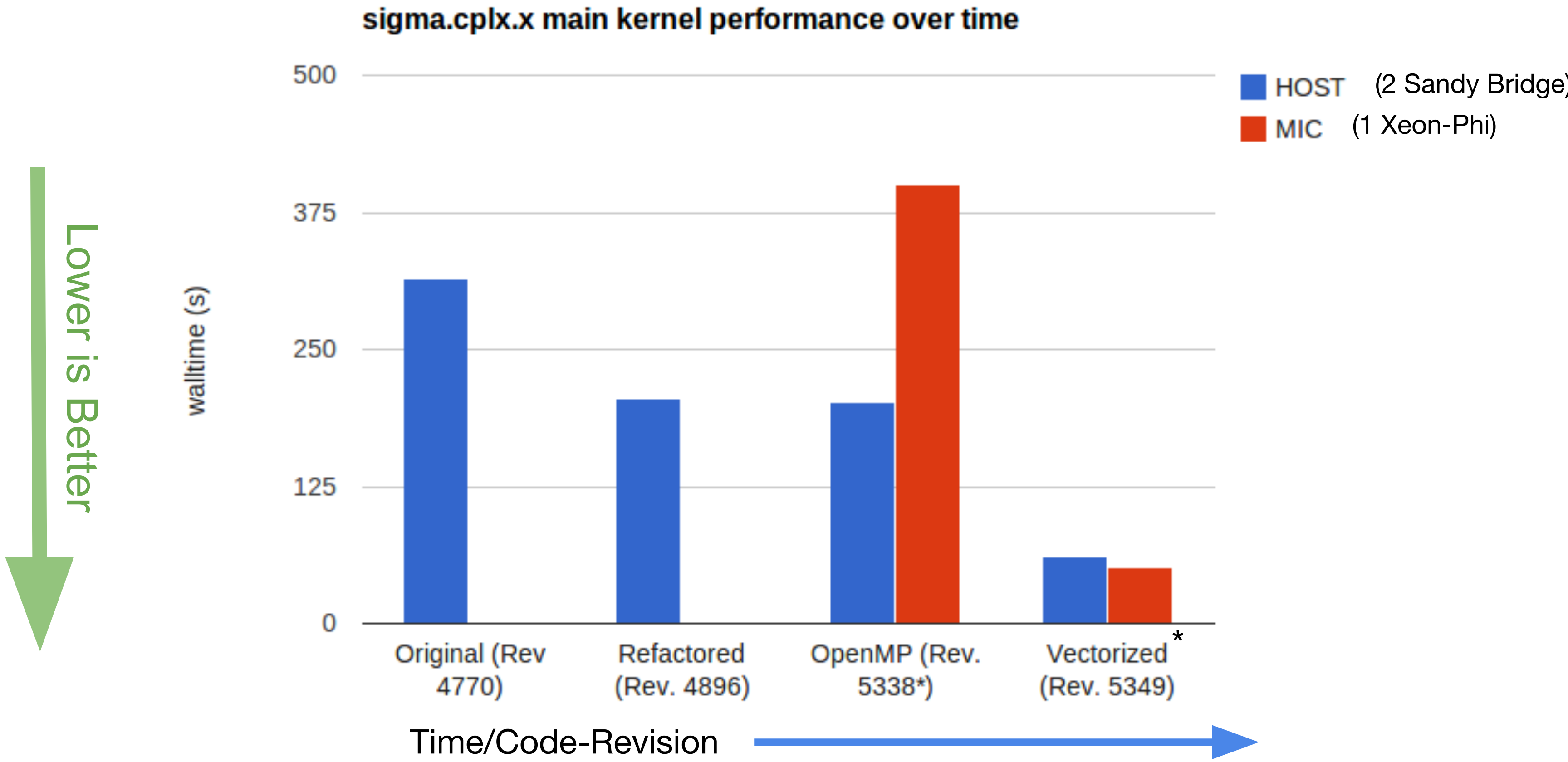
★ Big systems require more memory. Cost scales as N_{atm}^2 to store the data.

★ 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**.



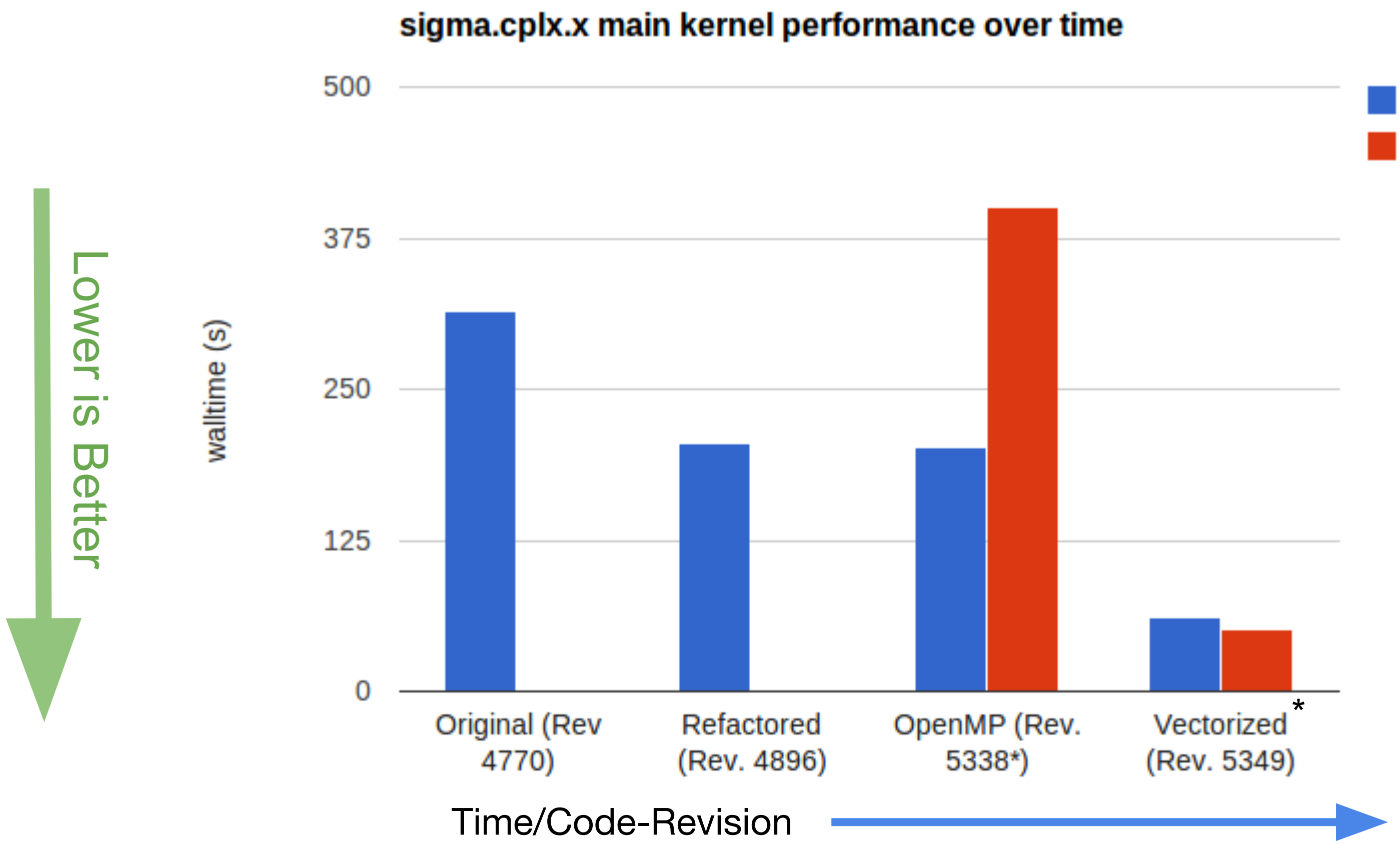
Steps to Optimize BerkeleyGW on Xeon-Phi Testbed



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

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

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

* - 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
```

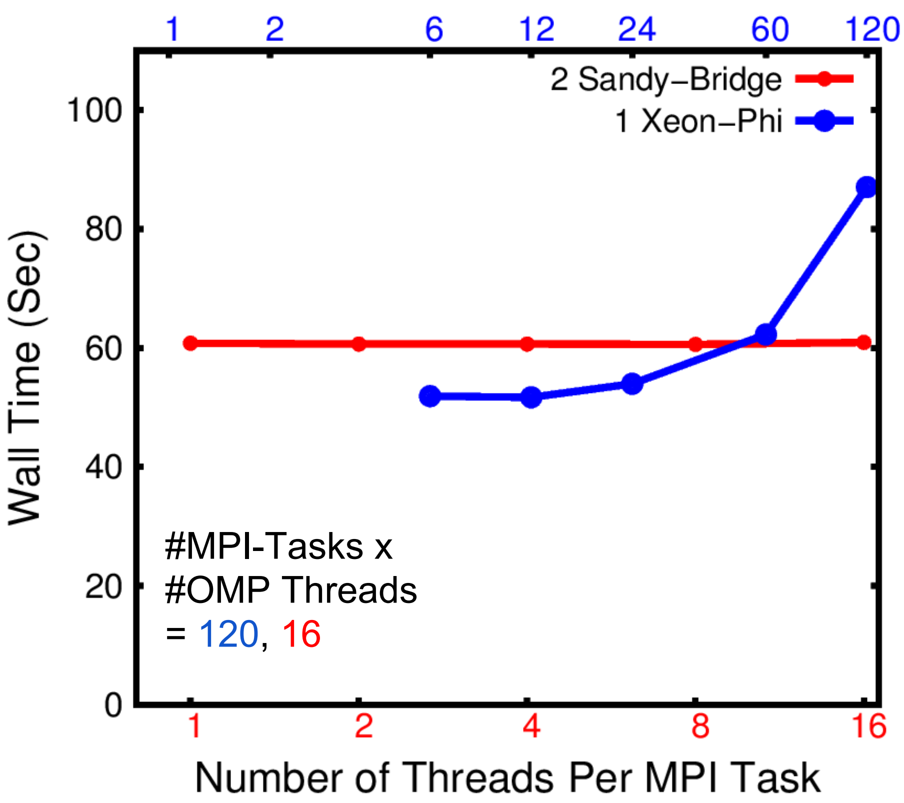
ngpown typically in 100's to 1000s. Good for many threads.

Original inner loop. Too small to vectorize!

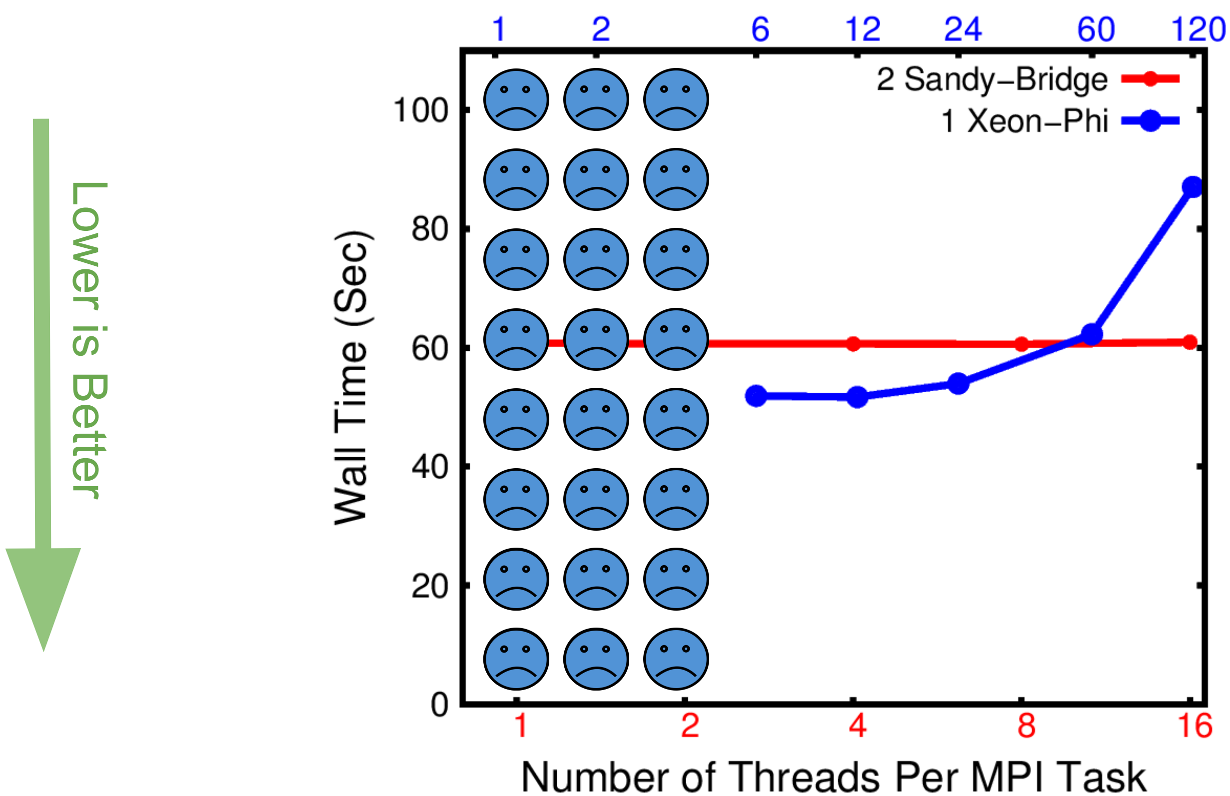
ncouls typically in 1000s - 10,000s. Good for vectorization. Don't have to worry much about memory. alignment.

Attempt to save work breaks vectorization and makes code slower.

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



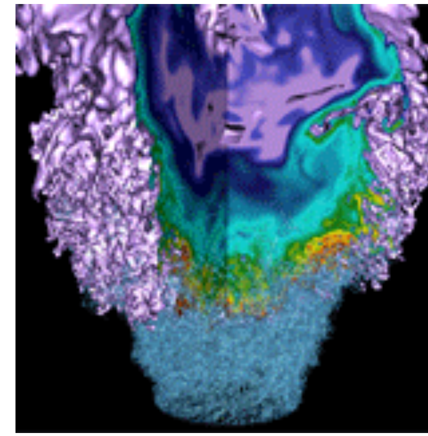
Running on Many-Core Xeon-Phi Requires OpenMP Simply To Fit Problem in Memory

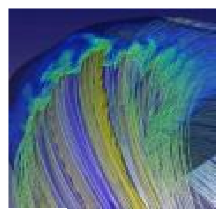


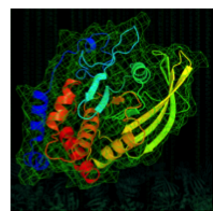
- Example problem cannot fit into memory when using less than 5 OpenMP threads per MPI task.
- ★ **Conclusion: you need OpenMP to perform well on Xeon-Phi in practice**

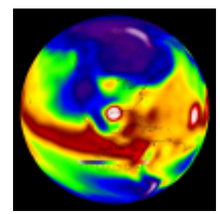
FLASH Case Study

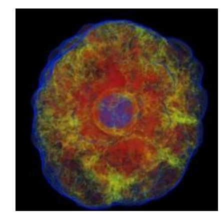
Christopher Daley

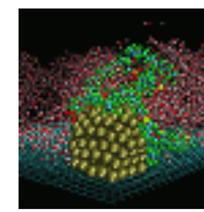








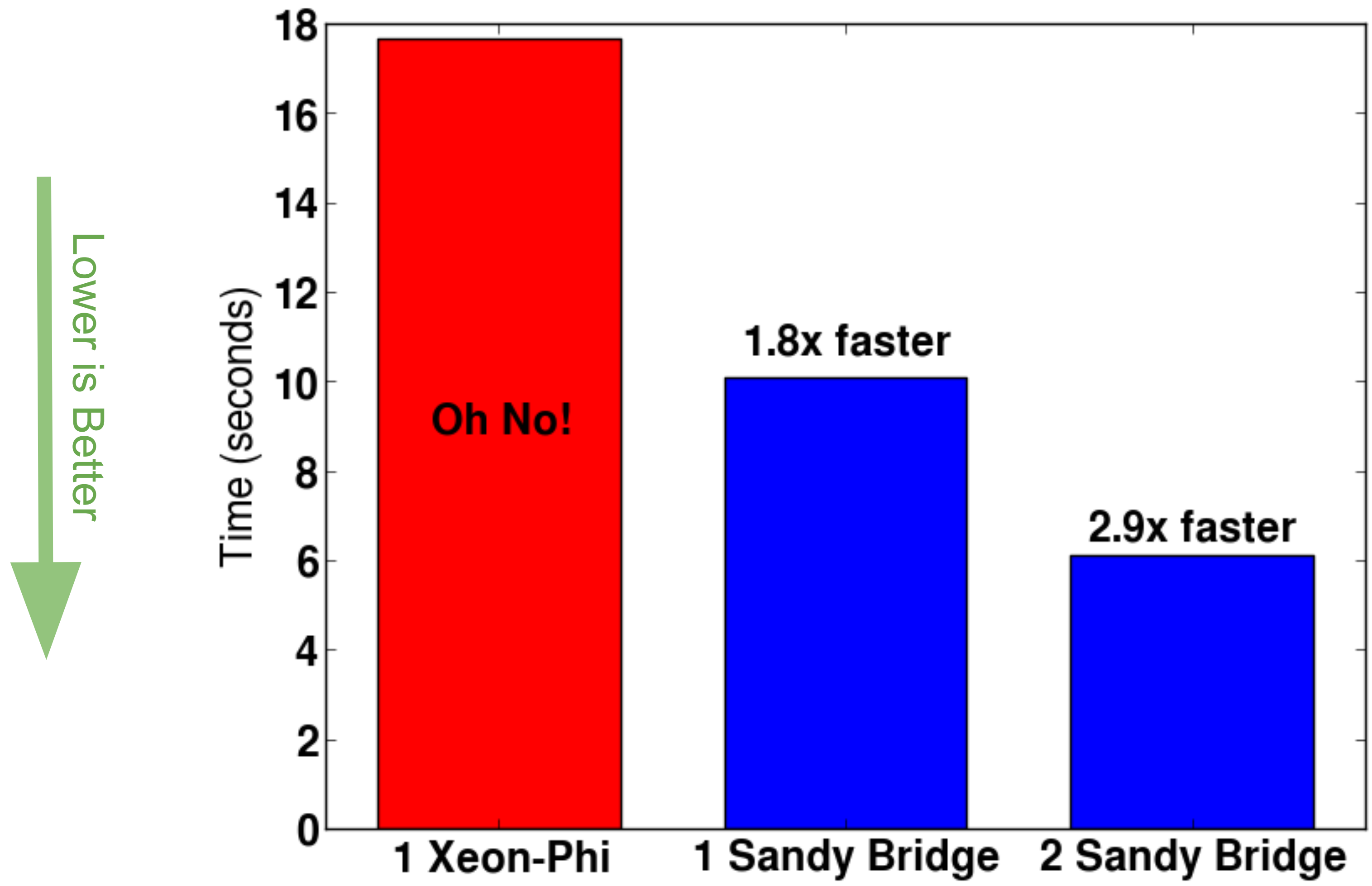




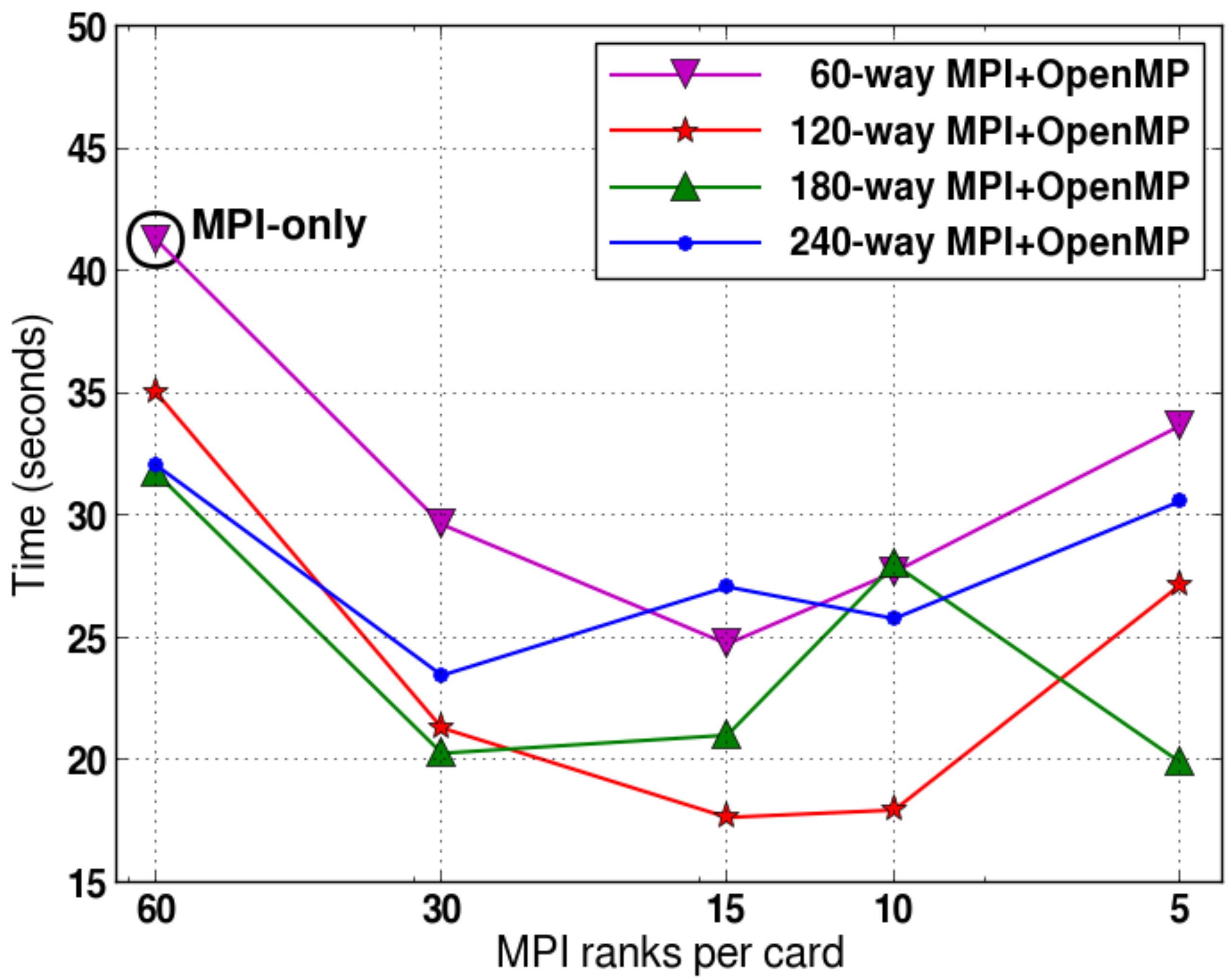
FLASH application readiness

- **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

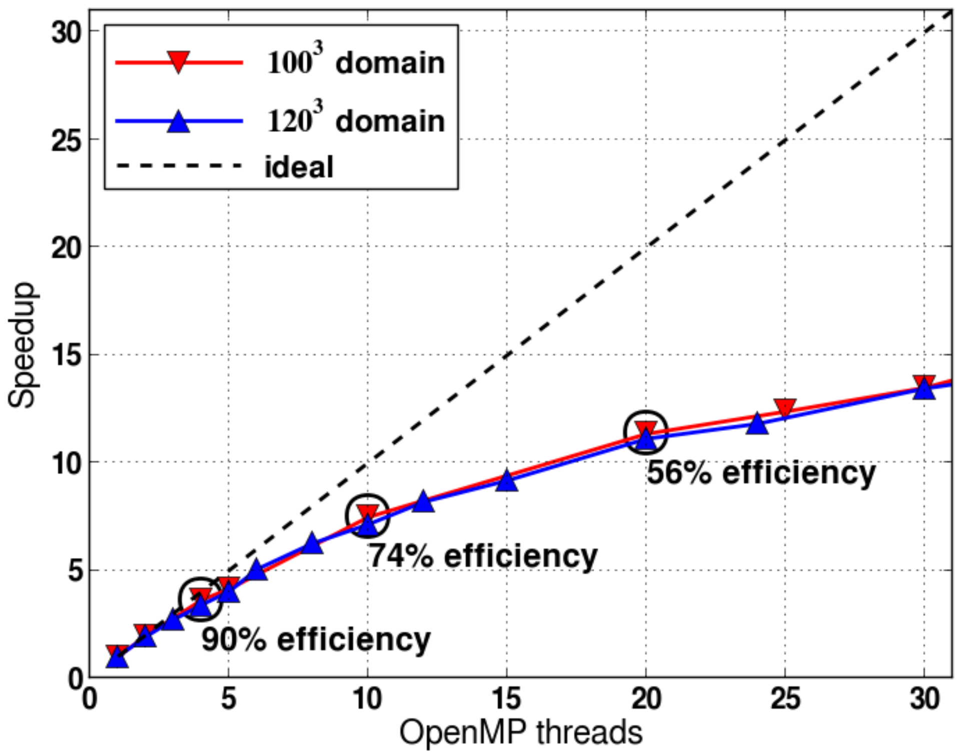


Best configuration on 1 MIC card



MIC performance study 1: thread speedup

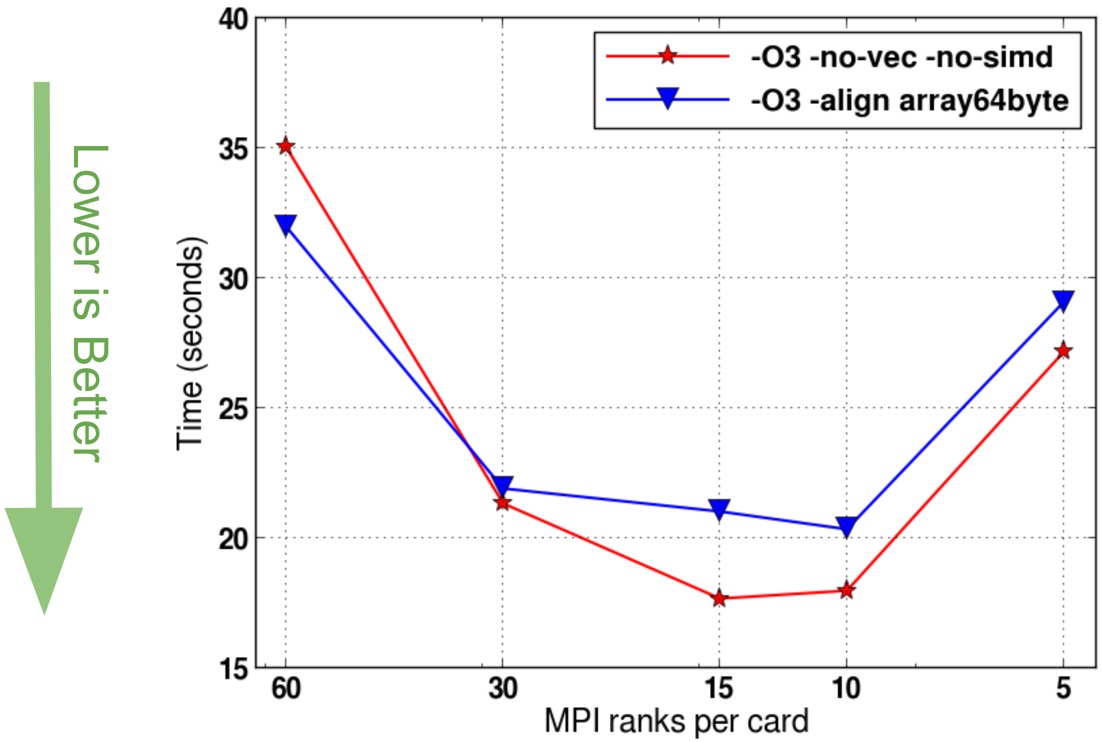
Higher is Better



- 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)

MIC performance study 2: vectorization



No vectorization gain!

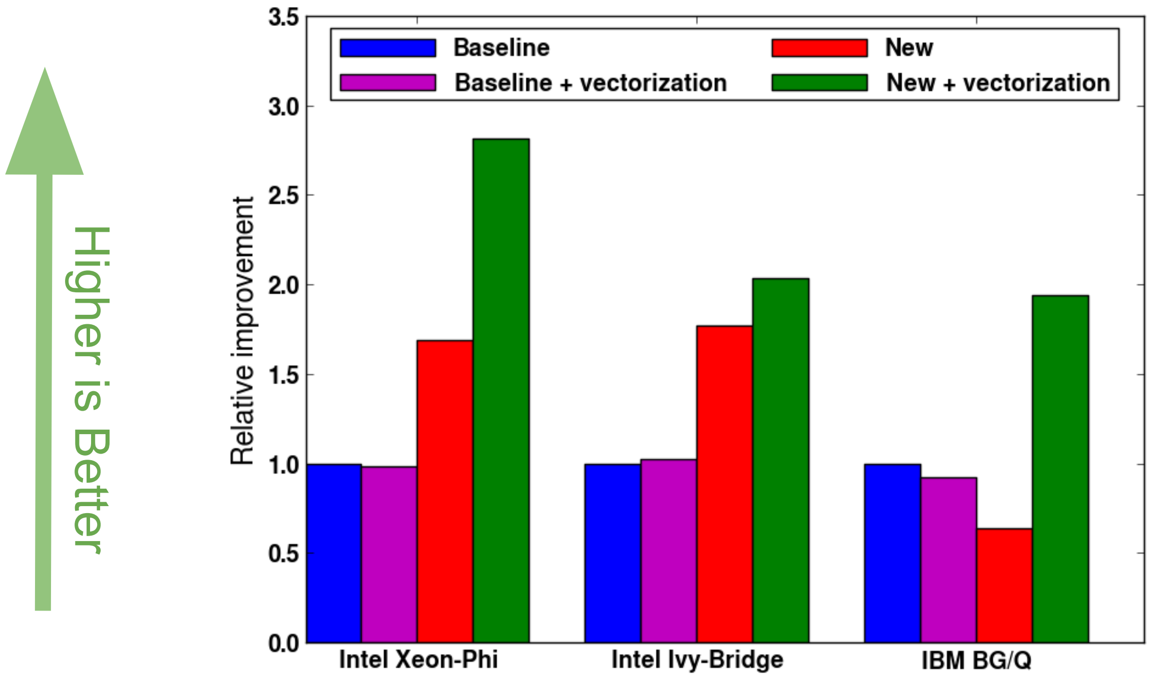
- We find that most time is spent in subroutines which update fluid state 1 grid point at a time

  – 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!

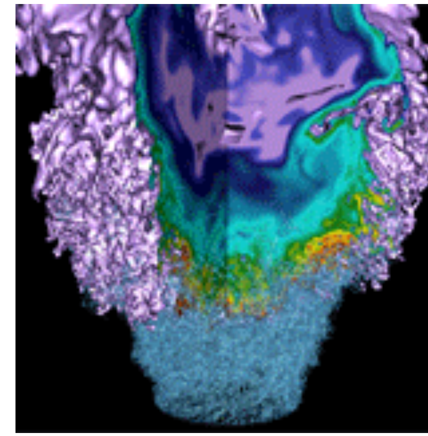
Enabling vectorization

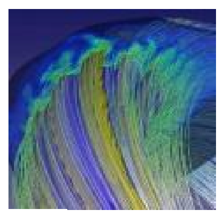
- **Must restructure the code**
  - The fluid fields should no longer be next to each other in memory
  - A(HY_DENS:HY_TEMP) should become A_dens(1:N), …, A_temp(1:N)
    - The 1:N indicates the kernels now operate on N grid points at a time
- **We tested these changes on part of a data reconstruction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

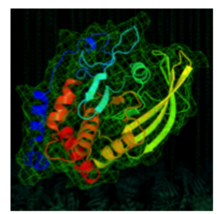


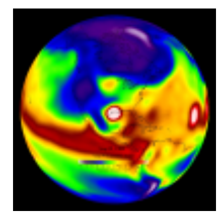
- **The new code compiled with vectorization options gives the best performance on 3 different platforms**

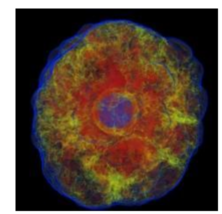
Conclusions and Lessons Learned

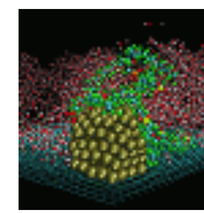












NERSC 40 YEARS at the FOREFRONT

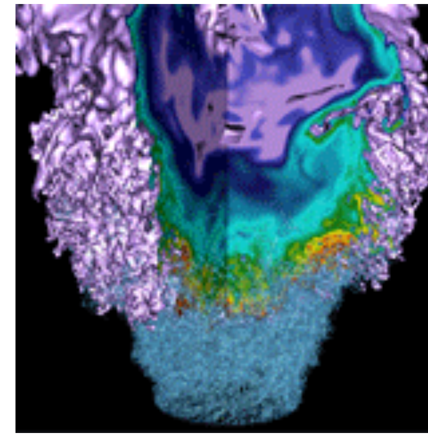
U.S. DEPARTMENT OF ENERGY | Office of Science

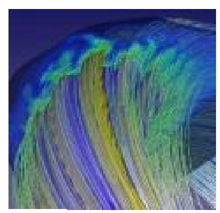
BERKELEY LAB

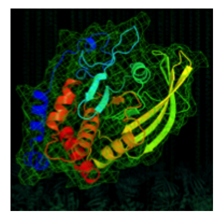
Summary

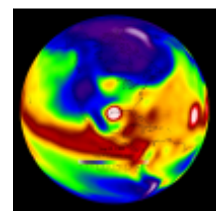
- ➔ Disruptive Change is Coming!
- ➔ NERSC is Here to Help Our Users
- ➔ Good performance will require code changes
  - ◆ Identify more on-node parallelism
  - ◆ Ensure vectorization for critical loops
- ➔ Need to leverage community. Other centers, NERSC users, 3rd Part Developers
- ➔ The code changes you make for many-core architectures will improve performance on all architectures.

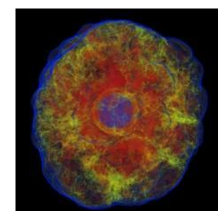
Extra Slides

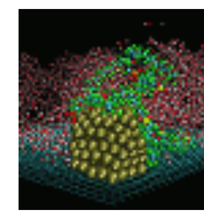












NERSC 40 YEARS at the FOREFRONT

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

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 (AofS) to structure of arrays (SofA)
- **Tested these proof-of-concept changes on a reduced hydro kernel**
  - Demonstrated improved performance on Ivy-Bridge, BG/Q and Xeon-Phi platforms