Cori Application Readiness Strategy and Early Experiences
Code Coverage

Breakdown of Application Hours on Hopper and Edison 2013

- VASP
- CCSM
- MILC
- GROMACS
- LAMMPS
- NAMD
- M3D
- Espresso
- S3D
- NWCHEM
- GYRO
- chroma
- osiris
- WRF
- gtc
- viscoelasticDriver3d
- cp2k
- madam
- toast
- Gadget
- grow-bubbles
- Charmm
- parsec
- p3d
- gs2
- zori
- vorpal
- Compo
- CESAR
- NESAP Tier-1, 2 Code
- NESAP Proxy Code or Tier-3 Code
Resources for Code Teams

• Early access to hardware
  – Access to Babbage (KNC cluster) and early “white box” test systems expected in 2015
  – Early access and significant time on the full Cori system

• Technical deep dives
  – Access to Cray and Intel staff on-site staff for application optimization and performance analysis
  – Multi-day deep dive (‘dungeon’ session) with Intel staff at Oregon Campus to examine specific optimization issues

• User Training Sessions
  – From NERSC, Cray and Intel staff on OpenMP, vectorization, application profiling
  – Knights Landing architectural briefings from Intel

• NERSC Staff as Code Team Laisons (Hands on assistance)

• 8 Postdocs
NESAP Postdocs

Taylor Barnes
Quantum ESPRESSO

Brian Friesen
Boxlib

Andrey Ovsyannikov
Chombo-Crunch

Mathieu Lobet
WARP

Tuomas Koskela
XGC1

Tareq Malas
EMGeo

Target Application Team
Concept
(1 FTE Postdoc +)
0.2 FTE AR Staff

0.25 FTE COE
1.0 FTE User Dev.

1 Dungeon Ses. +
2 Week on site w/
Chip vendor staff
NERSC Staff associated with NESAP

Katie Antypas  Nick Wright  Richard Gerber  Brian Austin  Zhengji Zhao  Helen He  Ankit Bhagatwala  Stephen Leak

Woo-Sun Yang  Rebecca Hartman-Baker  Doug Doerfler  Jack Deslippe  Brandon Cook  Thorsten Kurth

Target Application Team
Concept
(1 FTE Postdoc +)
0.2 FTE AR Staff
0.25 FTE COE
1.0 FTE User Dev.
1 Dungeon Ses. +
2 Week on site w/ Chip vendor staff
Timeline

- **Prototype Code Teams (BerkeleyGW / Staff)**
  - Prototype good practices for dungeon sessions and use of onsite staff.

- **NERSC Led OpenMP and Vectorization Training (One Per Quarter)**

- **Vendor General Training**

- **Gather Early Experiences and Optimization Strategy**

- **Code Team Activity**

- **Post-Doc Program**

- **Center of Excellence**

- **Chip Vendor On-Site Personnel / Dungeon Sessions**

- **White Box Access**

- **Delivery**

- **NERSC User and 3rd Party Developer Conferences**
Timeline

Jan 2014  May 2014  Jan 2015  Jan 2016  Jan 2017

Requirements Evaluation
Gather Early Experiences and Optimization Strategy
Prototype Code Teams (BerkeleyGW / Staff)
-Prototype good practices for dungeon sessions and use of on site staff.

NERSC Led OpenMP and Vectorization Training (One Per Quarter)
Vendor General Training
Vendor General Training

Code Team Activity
Post-Doc Program
Center of Excellence
Chip Vendor On-Site Personnel / Dungeon Sessions
White Box Access  Delivery

NERSC User and 3rd Party Developer Conferences
The Ant Farm Flow Chart

1. Use IPM and Darshan to Measure and Remove Communication and IO Bottlenecks from Code
2. Run Example in "Half Packed" Mode
   - Is Performance affected by Half-Packing?
     - Yes: Your Code is at least Partially Memory Bandwidth Bound
       - Can You Increase Flops Per Byte Loaded From Memory in Your Algorithm?
         - Yes: Explore Using HBM on Cori For Key Arrays
         - No: Make Sure Your Code is Vectorized! Measure Cycles Per Instruction with VTune
     - No: Explore Using HBM on Cori For Key Arrays
3. Run Example at "Half Clock" Speed
   - Is Performance affected by Half-Clock Speed?
     - Yes: Likely Partially Memory Latency Bound (assuming not IO or Communication Bound)
       - Can You Reduce Memory Requests Per Flop in Algorithm?
         - Yes: Try Running With as Many Virtual Threads as Possible (> 240 Per Node on Cori)
         - No: Make Sure Your Code is Vectorized! Measure Cycles Per Instruction with VTune
     - No: Explore Using HBM on Cori For Key Arrays
4. Make Algorithm Changes
NERSC is uniquely positioned between HPC Vendors and HPC Users and Applications developers.

NESAP provides a power venue for these two groups to interact.
What Has Gone Well

1. Setting requirements for Dungeon Session motivates teams to get started early and improves quality of dungeon session.
2. Engagement with IXPUG and user communities (Exascale Workshops at CRT)
3. Large number of NERSC and Vendor Training (Vectorization, OpenMP, Tools/Compilers) Well Received
4. Learned a Massive Amount about Tools and Architecture (VTune, SDE, HBM etc.)
5. Vendor staff helpful to work with. Very pro-active.
6. Pipelining Code Work Via Cray and Intel resources

Warp Vectorization Improvements at The Dungeon - Directly enabled by tiling work with Cray COE in Pre-dungeon

<table>
<thead>
<tr>
<th>Application</th>
<th>All memory on far memory</th>
<th>All memory on near memory</th>
<th>Key arrays on near memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>BerkeleyGW</td>
<td>baseline</td>
<td>52% faster</td>
<td>52.4% faster</td>
</tr>
<tr>
<td>EmGeo</td>
<td>baseline</td>
<td>40% faster</td>
<td>32% faster</td>
</tr>
<tr>
<td>XGC1</td>
<td>baseline</td>
<td>24% faster</td>
<td></td>
</tr>
</tbody>
</table>
What Has Gone Well (Cont)

7. Bandwidth sensitive applications that live in HBM expected to perform very well.

8. A lot of Lessons Learned: techniques to place key-arrays in fast-memory, improve prefetching effectiveness, coping without L3 cache etc...

9. CPU Intensive tasks (BGW GPP Kernel) expected to perform well (> Haswell) on KNL.

Version 1
• Simplify expressions to minimize #operations
• Use internal GAMMA function

Version 2
• Remove “elemental” attribute, move loop inside.
• Inline subroutines. Divide, fuse, exchange loops.
• Replace assumed shape arrays with loops
• Replace division with inversion of multiplication
• Remove initialization of loops to be overwritten later
• Use more aggressive compiler flags
• Use profile-guided optimization (PGO)

Version 3 (Intel compiler only)
• Use !$OMP SIMD ALIGNED to force vectorization
Example From Cray COE Work on XGC1

~40% speed up for kernel
Example From Cray COE Work on XGC1

~40% speed up for kernel
subroutine ell_spmv(mat, ind, x, z, m, n, ndiag)
 implicit none
 ! --
 integer :: m, n, ndiag
 integer, dimension(ndiag, m) :: ind
 complex*16, dimension(n) :: x
 complex*16, dimension(m) :: z
 complex*16, dimension(ndiag, m) :: mat
 ! --
 integer :: i, j
 complex*16 :: ztmp
 !$omp parallel do private(ztmp)
 do i = 1, m
 ztmp = (0.0d0, 0.0d0)
 do j = 1, ndiag
 ztmp = ztmp + mat(j,i) * x(ind(j,i))
 end do
 z(i) = ztmp
 end do
end subroutine ell_spmv

Vector loads when vectorized in \( i \)
Estimating the performance impact of HBW memory to VASP code via FASTMEM compiler directive and the memkind library on Edison

Test case: benchPdO2

VASP is a material science code that consumes the most computing cycles at NERSC.

This test used a development version of the VASP code.

Adding the FASTMEM directives to the code was done by Martijn Marsman at Vienna University
Boxlib (NERSC LEAD Brian Friesen)

Block AMR Framework.

Added “tiling” to improve data locality and improve OMP scaling on Xeon-Phi.

### SAMPLE WALL TIME MEASUREMENTS (SEC)

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>Outer-loop-level threading</th>
<th>Loop tiling</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.76</td>
<td>0.56</td>
</tr>
<tr>
<td>1</td>
<td>3.67</td>
<td>0.69</td>
</tr>
<tr>
<td>2</td>
<td>3.66</td>
<td>0.70</td>
</tr>
<tr>
<td>3</td>
<td>3.66</td>
<td>0.69</td>
</tr>
<tr>
<td>4</td>
<td>3.64</td>
<td>0.66</td>
</tr>
</tbody>
</table>

Now exploring in transit analysis using specialized analysis ranks and burst buffer.
How improve a code where most FLOPs occur in libraries?

Targeting Exact Exchange Problems. Characterized by many parallel FFTs.

Strategy:

Improve on-node performance by increase the on-node FLOP density and reducing inter-node communication. Moving individual FFTs to a single node shared memory model and exploiting new band-pair parallelism with MPI.
How improve a code where most FLOPs occur in libraries?

Targeting Exact Exchange Problems. Characterize many parallel FFTs.

Strategy:

Improve on-node performance by increase the on-node FLOP density and reducing inter-node communication. Moving individual FFTs to a single node shared memory model and exploiting new band-pair parallelism with MPI.

Exploit parallelism not used by default in app.