Application Readiness for NERSC Cori

Zhengji Zhao[1], Scott French[1], Jack Deslippe[1], Mathias Jacquelin[2], Brian Friesen[1], and Helen He[1]

[1)] National Energy Scientific Computing Center at Lawrence Berkeley National Laboratory
[2)] Lawrence Berkeley National Laboratory

NERSC Exascale Science Application Program (NESAP)

Preparing NERSC science applications for Cori

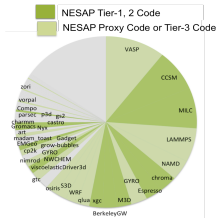
NERSC's next supercomputer, Cori, will begin to transition our workload to more energy efficient architectures

- Cray XC system with over 9300 Intel Knights Landing compute nodes
- Focus: increased on-node parallelism (72cores/node); Larger vector units (512 bits)
- On-package high-bandwidth memory

NERSC Exascale Science Application Program

- 20 application code teams were selected to work with Cray, Intel and NERSC staff (August, 2014)
- Resources available to application teams: Access to vendor resources and staff including "dungeon sessions" with Intel and Cray Center of Excellence, early access to KNL "whitebox" systems, early access and time on Cori
- Funding available for 8 NESAP Post-doctoral Fellows to work directly with application teams
- Many KNL-focused optimization strategies have been explored, and here we will highlight our efforts and achievements with a few selected NESAP codes

System named after Gerty Cori, Biochemist and the first American woman to receive the Nobel prize in science.



Above: Breakdown of the NERSC workload (2013 MPP hours), highlighting the NESAP codes and code proxies

Additional NESAP readiness and support efforts

- NERSC hosts and/or organizes various training events for users and developers, e.g., OpenMP, Vectorization, MPI-3, Intel and Cray profiling and optimization tools, Intel and Cray compiler optimizations
- Hosting code optimizations hackathons for NERSC users and application developers
- Intensive participation in the Intel Xeon Phi User Group (IXPUG)
- Involved in the OpenMP committee

BerkeleyGW: Materials Science Applications

Developed at LBNL/UCB. Used as "prototype" for App Readiness at NERSC.
NESAP PI: Jack Deslippe (NERSC)

Significant Bottleneck is large matrix reduction like operations.

$$\langle n\mathbf{k}|\Sigma_{\mathrm{CH}}(E)|n'\mathbf{k}\rangle = \frac{1}{2}\sum_{n''}\sum_{\mathbf{q}\mathbf{G}\mathbf{G}'} M^*_{n''n}(\mathbf{k},-\mathbf{q},-\mathbf{G})M_{n''n'}(\mathbf{k},-\mathbf{q},-\mathbf{G}') \times \frac{\Omega^2_{\mathbf{G}\mathbf{G}'}(\mathbf{q})\left(1-i\tan\phi_{\mathbf{G}\mathbf{G}'}(\mathbf{q})\right)}{\omega_{\mathbf{G}\mathbf{G}'}(\mathbf{q})\left(E-E_{n''\mathbf{k}-\mathbf{q}}-\omega_{\mathbf{G}\mathbf{G}'}(\mathbf{q})\right)} v(\mathbf{q}+\mathbf{G}')$$

Optimization steps:

1. Target more on-node parallelism. (MPI model already failing users)
2. Ensure key loops/kernels can be vectorized.

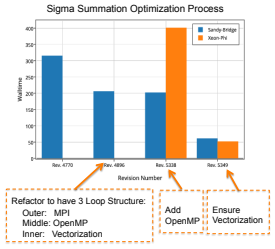
```
!$OMP DO reduction(+:achtemp)
  do my_igp = 1, ngpown
  ...
    do iw=1,3

      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.

Attempt to save work breaks vectorization and makes code slower.



EMGeo: Geophysical Imaging Applications

Developed by researchers at LBL Earth Science Division
Project PI: Gregory Newman (ESD); NESAP Liaison: Scott French (NERSC)

EMGeo (EM imaging and Seismic imaging) dominated by Krylov solver (> 90% of wallclock)

- Different solver methods (QMR vs. IDR), but similar motifs

Optimization Steps:

- **Adding OpenMP to solvers (starting with IDR)**
  - Overall thread scaling is good on Knights Corner Xeon Phi (Fig. 1)
- **Understanding memory bandwidth saturation**
  - Solvers dominated by SpMV, STREAM-like triads, etc.
- **Focusing initial architectural optimization experiments on complex*16 ELLPACK SpMV**
  - Many potential optimizations techniques for KNL
  - Alignment tweaks; Loop reordering, unrolling; Memory layout optimizations; Fortran "SIMD-ization"
- **Developed SpMV kernel variants that span the space of likely optimizations** (Fig. 2)
  - Ready for profiling when we have KNL access
- **Identify the candidate for HBM**
  - Using NUMA affinity to simulate HBM on a dual socket system

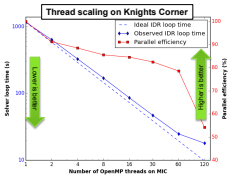


Fig. 1: Preliminary thread scaling on KNC (NERSC Babbage)

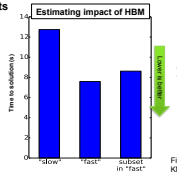


Fig. 3 (Right): Estimating the impact of allocation in "fast" and "slow" memory using a recent two-socket Haswell system.

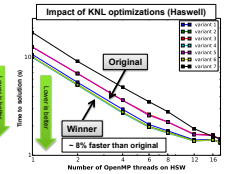


Fig 2: Time to solution / thread scaling to assess the impact of KNL focused optimizations on other architectures, also Haswell.

NWChem: Computational Chemistry Codes

Project PI: Wibe De Jong (LBNL), and Eric Bylasca (PNNL) NERSC Liaison: Zhengji Zhao

Plane wave Lagrange multiplier

- Many matrix multiplication of complex numbers
- Smaller matrix products
  - Typical ne = 100, ngrid = 10,000

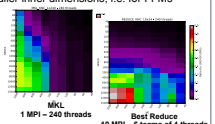
Product	Type	Size	Flops
FFM	Inner	Ne x Ng x Ne	200 MFlop
MMM	Square	Ne x Ne x Ne	2 MFlop
FMF	Tall-skinny	Ng x Ne x Ne	200 MFlop

- Threading only with MKL not satisfactory
  - FFM does not scale with threads

# CPU	18	1
# Threads	1	18
Total Lagrange Multiplier	5.7 ms	9.5 ms
FFM	4.3 ms	8.7 ms
FMF	0.5 ms	0.6 ms

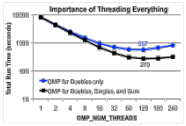
Xeon E7-8890v3 (Haswell,18 cores)

- "Reduce" algorithm
  - Distribute work on A and B along the k dimension
  - A thread puts its contribution in a buffer of size m x n
  - Buffers reduced to produce C
  - OMP teams of threads
- Better for smaller inner dimensions, i.e. for FFMs



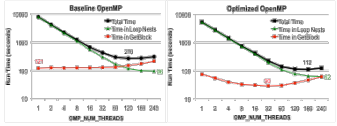
- Multiple FFMs can be done in concurrently in different thread pools

Coupled cluster Triples algorithm



- Double terms usually dominate in (T) term
- Other terms become new performance bottleneck on many-core architectures - Amdahl's Law

- Optimizations lead to significant performance improvements
  - Threading enables us to use all 240 hardware threads
  - Optimized code performs 2.5X better than baseline
  - Up to 65X better compared to 1 MPI rank



BoxLib: Block-structured AMR Framework

Developed by Center for Computational Sciences and Engineering at LBNL
NESAP PI: Ann Almgren (LBNL) NERSC Postdoc: Brian Friesen

- Designed for numerical solution of PDEs on distributed, structured grids
- Hybrid MPI+OpenMP parallelism

Traditional threading model

- Domain decomposed into *N* boxes, distributed among *M* MPI tasks, each with *m* threads
- Each thread operates on ~(*N* / (*M***m*)) boxes
- Leads to **load imbalance** with large #s of threads (>20), esp. in AMR where box distribution among MPI tasks is uneven

Tiling threading model

- Iteration space within each box is divided into smaller "tiles," which are distributed among threads
- Tile size specified by user; contiguous in unit-stride dimension for optimal caching
- Tile-level threading reduces "surface area-to-volume" ratio of memory halo to FP data for distributed workloads

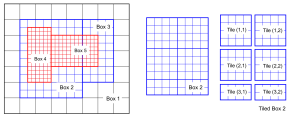
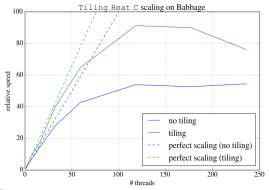


Fig. 1: Box 1 is at level 0, Box2 and 3 are at level 1, and Box 4 and 5 are at level 2. A number of boxes are assigned to a NUMA node, then each box is subdivide into tiles by TiDA.



Preliminary performance results

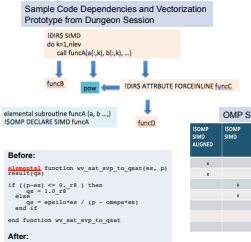
- Tiling implementation strong scales efficiently up to ~120 threads on Babbage (KNC testbed at NERSC)
- Saturates memory bandwidth at ~2 HW threads/core
- Strong scales more efficiently than non-tiled version up to memory BW saturation point
- Even with only 1 thread, performance with tiling is still faster due to improved data locality within tiled loop iteration space

CESM: Community Earth System Model

NESAP PI: John Dennis (NCAR). NERSC Liaison: Helen He

MG2: CESM kernel for radiation transfer workload

- Typically takes 10% of CESM time
- Compute bound
- Very little vectorization: pipeline dependencies, heavy use of math intrinsics



Optimization Steps

Version 1

- Simplify expressions to minimize #operations
- Use internal GAMMA function

Version 2

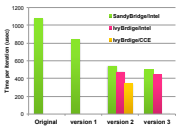
- Remove "elemental" attribute, move loop inside.
- Inline subroutines. Divide, fuse, exchange loops.
- Replace assumed shaped arrays with loops
- Replace division with inversion of multiplication
- Remove initialization of loops to be overwritten later
- Use more aggressive compiler flags
- Use profile-guided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

Version 3 (Intel compiler only)

- Use !$OMP SIMD ALIGNED to force vectorization

OMP SIMD with Intel Compiler

!$OMP SIMD ALIGNED	!$OMP SIMD	Idrt VECTOR ALIGNED	-align array64byte	-openmp	Time per iteration (usec) on Edison
x			x	x	444
x				x	446
	x		x	x	484
	x			x	483
		x	x		452
		x			456
					473



Summary

- Directives and flags are helpful. Not a replacement for code modifications.
- Break up loops and push loops into functions where vectorization can be dealt with directly. Try different compilers.
- Incremental improvements not necessary a BIG win from any one thing. Accumulative results matter.
- Performance and portability: use !$OMP SIMD is beneficial but very hard to use: need to provide the aligned list manually.
- Requested optional alignment declaration in Fortran language standard.

IXPUG
THE INTEL® XEON PHI™ USERS GROUP
September 28 - October 2, 2015
Berkeley, CA

Acknowledgements

Intel, Inc., NESAP partnership
Cray, Inc., Collaboration via Center of Excellence