Guiding Optimization on KNL with the Roofline Model

February, 2018
What is different about Cori?

<table>
<thead>
<tr>
<th>Edison (&quot;Ivy Bridge&quot;)</th>
<th>Cori (&quot;Knights Landing&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>● 5576 nodes</td>
<td>● 9304 nodes</td>
</tr>
<tr>
<td>● 24 physical cores per node</td>
<td>● 68 physical cores per node</td>
</tr>
<tr>
<td>● 48 virtual cores per node</td>
<td>● 272 virtual cores per node</td>
</tr>
<tr>
<td>● 2.4 - 3.2 GHz</td>
<td>● 1.4 - 1.6 GHz</td>
</tr>
<tr>
<td>● 8 double precision ops/cycle</td>
<td>● 32 double precision ops/cycle</td>
</tr>
<tr>
<td>● 64 GB of DDR3 memory (2.5 GB per physical core)</td>
<td>● 16 GB of fast memory 96GB of DDR4 memory</td>
</tr>
<tr>
<td>● ~100 GB/s Memory Bandwidth</td>
<td>● Fast memory has 400 - 500 GB/s</td>
</tr>
<tr>
<td></td>
<td>● No L3 Cache</td>
</tr>
</tbody>
</table>
NERSC’s Challenge

How to Enable NERSC’s diverse community of 7,000 users, 750 projects, and 700 codes to run on advanced architectures like Cori and beyond?
1. Need a sense of absolute performance when optimizing applications.
   - How Do I know if My Performance is Good?
   - How Do I know when to stop?
   - Why am I not getting peak performance? Or the predicted ceiling for my application?

2. Many potential optimization directions:
   - How do I know which to apply?
   - What is the limiting factor in my app’s performance?
   - How do I know when to stop?
Optimizing Code For Cori is like:

A. A Staircase?
A. A Labyrinth?
A. A Space Elevator?
OpenMP scales only to 4 Threads

Communication dominates beyond 100 nodes

large cache miss rate

Code shows no improvements when turning on vectorization

50% Walltime is IO

IO bottlenecks

The Dungeon:
Simulate kernels on KNL.
Plan use of on-package memory, vector instructions.

The Ant Farm!
Communication dominates beyond 100 nodes

Compute intensive doesn’t vectorize

MPI/OpenMP Scaling Issue

Use Edison to Test/Add OpenMP
Improve Scalability.
Help from NERSC/Cray COE
Available.

Utilize performant / portable libraries

Can you use a library?

Increase Memory Localibty

Create micro-kernels or examples to examine thread level performance, vectorization, cache use, locality.

Utilize High-Level IO-Libraries.
Consult with NERSC about use of Burst Buffer.

Memory bandwidth bound kernel

50% Walltime is IO
Are you memory or compute bound? Or both?

Run Example in “Half Packed” Mode

If you run on only half of the cores on a node, each core you do use has access to more bandwidth

```
aprun -n 24 -N 12 - S 6 ...
```

VS

```
aprun -n 24 -N 24 - S 12 ...
```

```
srun -N 2 -n 24 -c 2 - S 6 ...
```

VS

```
srun -N 1 -n 24 -c 1 ...
```

If your performance changes, you are at least partially memory bandwidth bound
Are you memory or compute bound? Or both?

Reducing the CPU speed slows down computation, but doesn’t reduce memory bandwidth available.

**Run Example at “Half Clock” Speed**

```
aprun --p-state=2400000 ...  VS  aprun --p-state=1900000 ...
srun --cpu-freq=2400000 ...  VS  srun --cpu-freq=1900000 ...
```

If your performance changes, you are at least partially compute bound.
Roofline helps visualize this information! Guides optimizations

WARP Optimizations:
1. Add tiling over grid targeting L2 cache on both Xeon-Phi Systems
2. Add particle sorting to further improve locality and memory access pattern
3. Apply vectorization over particles

(a) Haswell Roofline  
(b) KNL Roofline
NESAP Example
Optimization process for Kernel-C (Sigma code):

1. Refactor (3 Loops for MPI, OpenMP, Vectors)
2. Add OpenMP
3. Initial Vectorization (loop reordering, conditional removal)
4. Cache-Blocking
5. Improved Vectorization (Divides)
6. Hyper-threading
Vectorization

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.

```c
!$OMP DO reduction(+:achtemp)
   do my_igp = 1, ngpown
     ...
     do iw=1,nfreq ! nfreq is 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 * aqsnltemp(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
```
Change in Roofline

The loss of L3 on MIC makes locality more important.
Why KNC worse than Haswell for GPP Kernel?

!$OMP DO
  do my_igp = 1, ngpown
    do iw = 1 , 3
      do ig = 1, igmax
        load wtilde_array(ig,my_igp) 819 MB, 512KB per row
        load aqsntemp(ig,n1) 256 MB, 512KB per row
        load l_eps_array(ig,my_igp) 819 MB, 512KB per row
        do work (including divide)
      end do
    end do
  end do
end do

Required Cache size to reuse 3 times:

1536 KB

L2 on KNL is 512 KB per core
L2 on Has. is 256 KB per core
L3 on Has. is 3800 KB per core

Without blocking we spill out of L2 on KNL and Haswell. But, Haswell has L3 to catch us.
Why KNC worse than Haswell for GPP Kernel?

!$OMP DO
do my_igp = 1, ngpown
   do igbeg = 1, igmax, igblk
      do iw = 1, 3
         do ig = igbeg, min(igbeg + igblk, igmax)
            load wtilde_array(ig,my_igp) 819 MB, 512KB per row
            load aqsntemp(ig,n1) 256 MB, 512KB per row
            load I_eps_array(ig,my_igp) 819 MB, 512KB per row
            do work (including divide)

Required Cache size to reuse 3 times:

1536 KB

L2 on KNL is 512 KB per core
L2 on Has. is 256 KB per core
L3 on Has. is 3800 KB per core

Without blocking we spill out of L2 on KNL and Haswell. But, Haswell has L3 to catch us.
Cache Blocking Optimization (Hierarchical Roofline)

Original Code

Cache-Blocking Code
Why Complex Divides so Slow?

Found significant x87 instructions from 1/complex_number instead of AVX/AVX-512

Can significantly speed up by using -fp-model fast=2
Additional Speedups from Hyperthreading

Haswell Roofline Optimization Path

- Peak
- ILP
- AVX
- BGW

Arithmetic Intensity

GFLOP/s

1 2 5 10 20 50 100

GFLOP/s

1 2 5 10 20 50 100

KNL Roofline Optimization Path

- Peak (HBM)
- Peak (DDR)
- ILP (HBM)
- ILP (DDR)
- AVX (HBM)
- AVX (DDR)
- BGW (DDR)
- BGW (HBM)
Big systems require more memory. Cost scales as $N_{\text{atoms}}^2$ 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.

Users sometimes forced to use 1 of 24 available cores, in order to provide MPI tasks with enough memory. 90% of the computing capability is lost.
In house code (I’m one of main developers). Use as “prototype” for App Readiness.

Significant Bottleneck is large matrix reduction like operations. Turning arrays into numbers.

\[
\langle n\mathbf{k} | \Sigma_{CH}(E) | n'\mathbf{k} \rangle = \frac{1}{2} \sum_{n''} \sum_{\mathbf{qGG'}} M_{n''n}(\mathbf{k}, -\mathbf{q}, -\mathbf{G'}) M_{n''n'}(\mathbf{k}, -\mathbf{q}, -\mathbf{G'}) \times \frac{\Omega_{GG'}^2(\mathbf{q}) (1 - i \tan \phi_{GG'}(\mathbf{q}))}{\tilde{\omega}_{GG'}(\mathbf{q}) (E - E_{n''\mathbf{k} - \mathbf{q}} - \tilde{\omega}_{GG'}(\mathbf{q}))} v(\mathbf{q} + \mathbf{G'})
\]
Make Algorithm Changes

Run Example in “Half Packed” Mode

Is Performance affected by Half-Packing?

Yes

Your Code is at least Partially Memory Bandwidth Bound

No

Run Example at “Half Clock” Speed

Is Performance affected by Half-Clock Speed?

Yes

You are at least Partially CPU Bound

No

Likely Partially Memory Latency Bound (assuming not IO or Communication Bound)

Make Algorithm Changes

Use IPM and Darshan to Measure and Remove Communication and IO Bottlenecks from Code
So, you are Memory Bandwidth Bound?

What to do?

1. Try to improve memory locality, cache reuse

1. Identify the key arrays leading to high memory bandwidth usage and make sure they are/will-be allocated in HBM on Knights Landing.

Profit by getting ~ 4-5x more bandwidth GB/s.
So, you are Compute Bound?

What to do?

1. Make sure you have good OpenMP scalability. Look at VTune to see thread activity for major OpenMP regions.

![Graph showing thread activity over time](image)

1. Make sure your code is vectorizing. Look at Cycles per Instruction (CPI) and VPU utilization in vtune.

See whether intel compiler vectorized loop using compiler flag: `-qopt-report=5`
Are you latency bound?

You may be memory latency bound (or you may be spending all your time in IO and Communication).

If running with hyper-threading improves performance, you *might* be latency bound:

\[
\text{aprun -j 2 -n 48 .... vs aprun -n 24 .....}
\]

If you can, try to reduce the number of memory requests per flop by accessing contiguous and predictable segments of memory and reusing variables in cache as much as possible.

On Knights-Landing, each core will support up to 4 threads. Use them all.
If your performance changes, you are at least partially memory bandwidth bound.

Are you memory or compute bound? Or both?

Run Example in “Half Packed” Mode

If you run on only half of the cores on a node, each core you do run has access to more bandwidth.

**aprun**
- **-n 24** -N 12
- **-S 6** 2 ...

**srun**
- **-N 2 -n 24** -c 2
- **-S 6** 2 ...

If your performance improves when you run in “Half Packed” mode, you are memory bandwidth bound.
Cori KNL System

Cray XC40 system with 9,600+ Intel Knights Landing (KNL) nodes:

- 68 cores, 272 Hardware Threads
- Up to 32 FLOPs per Cycle, 1.2-1.4 GHz Clock Rate
- Wide (512 Bit) vector Units
- Multiple Memory Tiers: 96 GB DRAM / 16 GB HBM
- NVRAM Burst Buffer 1.5 PB, 1.5 TB/sec