Quantitatively Assessing Performance Portability with Roofline

IDEAS Jan 23 2019

John Pennycook, Charlene Yang, Jack Deslippe
john.pennycook@intel.com, cjyang@lbl.gov, jrdeslippe@lbl.gov
NERSC: Mission HPC for DOE Office of Science

Largest funder of physical science research in U.S.

Bio Energy, Environment

Computing

Materials, Chemistry, Geophysics

Particle Physics, Astrophysics

Nuclear Physics

Fusion Energy, Plasma Physics

7,000 users, 750 projects, 700 codes, 48 states, 40 countries, universities & national labs
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 (KNL), Perlmutter (GPUs) and Beyond
What was different about Cori?

<table>
<thead>
<tr>
<th>Edison (“Ivy Bridge”)</th>
<th>Cori (“Knights Landing”)</th>
</tr>
</thead>
<tbody>
<tr>
<td>● 24 physical cores per node</td>
<td>● 68 physical 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 96 GB 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>● L1/L2/L3 Caches</td>
<td>● L1/L2 Cache, No L3 Cache</td>
</tr>
</tbody>
</table>
GPU-accelerated and CPU-only nodes meet the needs of large scale simulation and data analysis from experimental facilities

NERSC’s Goal is to provide a transition path from Cori to Perlmutter to NERSC-10
Science teams need a simple way to wrap their heads around performance and (performance portability) when main focus is scientific productivity:

1. Need a sense of absolute performance when optimizing applications.
   - How Do I know if My Performance is Good?
   - Why am I not getting peak performance advertised
   - How Do I know when to stop?

2. Many potential optimization directions:
   - How do I know which to apply?
   - What is the limiting factor in my app’s performance?
   - Again, how do I know when to stop?

3. How improve performance portably?
   - Users are scientists. Have accounts on many system. Don’t want yearly rewrite
Framing the Optimization Conversation

Energy-Efficient Processors Have Multiple Hardware Features to Optimize Against:
- Many (Heterogeneous) Cores
- Big WARPS/Vectors
- New ISA
- Multiple Memory Tiers

It is easy for users to get bogged down in the weeds:
- How do you know what KNL hardware feature to target?
- How do you know how your code performs in an absolute sense and when to stop?

Optimizing Code for Cori/Perlmutter is Like:
A Staircase?
B Labyrinth?
C Space Elevator?
OpenMP scales only to 4 Threads

MPI/OpenMP Scaling Issue

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

Utilize performant / portable libraries

large cache miss rate

Communication dominates beyond 100 nodes

Code shows no improvements when turning on vectorization

50% Walltime is IO

Compute intensive doesn’t vectorize

Memory bandwidth bound kernel

Can you use a library?

Increase Memory Locality

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.

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

The Ant Farm!

Communication dominates beyond 100 nodes

Code shows no improvements when turning on vectorization

50% Walltime is IO

Compute intensive doesn’t vectorize

Memory bandwidth bound kernel

Can you use a library?

Increase Memory Locality

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.

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

Ant Farm Model

- Utilize MCDRAM
- Data Reuse
- Improve OpenMP
- Improve Vectorization

large cache miss rate
Memory Bandwidth Bound

(b) KNL Roofline

GFLOP/s vs. Arithmetic Intensity (FLOP/byte)
Framing Performance Portability

Everyone knows “roughly” what performance portability is. But, in order to make progress, it pays to be precise and quantifiable.

DOE SC Facility Definition

An application is performance portable if it achieves a consistent ratio of the actual time to solution to either the best-known or the theoretical best time to solution on each platform with minimal platform specific code required.
Measuring Performance Portability

Bad Ways

1. Compare time-to-solution on one system vs another.
2. Compare ratio of actual app performance to peak system performance

Good Ways

1. Compare time-to-solution on each system against a well-known optimal implementation
2. Compare performance on each system against a relevant roofline-model ceiling on each system (We’ve included instructions for KNL and GPU)
Roofline Facilitates PP Analysis

**Focus:** Architectural Efficiency $e_i(a,p)$ and Roofline

- $F_i$ Peak GFLOP/s, $B_i$ Peak Bandwidth, $I_i(a,p)$ Arithmetic Intensity (AI)

$$
\Phi(a,p,H) = \begin{cases} 
\frac{|H|}{\sum_{i \in H} e_i(a,p)} & \text{if } i \text{ is supported, } \forall i \in H \\
0 & \text{otherwise}
\end{cases}
$$

$$
e_i(a,p) = \frac{P_i(a,p)}{\min(F_i, B_i \times I_i(a,p))}
$$

**Three Messages:**

- Use empirical Roofline ceilings
- Appropriately account for divides in FLOPs
- Roofline can capture nuances of performance analysis such as changes in AI, instruction mix, instruction issue/exec bandwidth, memory access pattern, etc

These all affect your PP score!
A Primer on Roofline

• An application’s maximum attainable performance on a machine is:

\[ P_{\text{attainable}} = \min(F, B \times I) \]

• \( F \): peak FLOP/s
• \( B \): peak bandwidth
• \( I \): arithmetic intensity (AI) = FLOPs / Bytes

• Hierarchical Roofline
  - DRAM/HBM/L2/L1 bandwidths
  - vector/scalar/etc compute peaks
• Log-Log scale, easy to extrapolate
How to Collect Roofline Data

- Methodology to build a Roofline for an application
  - Measure empirical compute and bandwidth ceilings:
    - Empirical Roofline Toolkit (ERT)
    - https://bitbucket.org/berkeleylab/cs-roofline-toolkit/
  - Measure application performance:
    - SDE and LIKWID on KNL
    - NVPROF on V100

Arithmetic Intensity = \( \frac{\text{SDE or nvprof FLOPs}}{\text{LIKWID or nvprof Data Movement}} \)

(X coordinate: FLOPs/Byte)

Application Performance = \( \frac{\text{SDE or nvprof FLOPs}}{\text{Runtime}} \)

(Y coordinate: GFLOP/s)
How to Plot Roofline Data

- Use Python, gnuplot, or other tools to plot Roofline
  - Example: `plot_roofline.py data.txt`
  - https://github.com/cyanguwa/nersc-roofline/tree/master/Plotting

---

```
data.txt
# all data is space delimited
memroofs  828.758
mem_roof_names 'HBM'
comproofs 7068.86 3535.79
comp_roof_names 'FMA' 'No-FMA'

# omit the following if only plotting roofs
AI 2.584785579
GFLOPs 2085.756683
labels 'FMA, nw=1'
```
Message 1: Empirical vs. Theoretical

- Discrepancy between empirically measured peaks and arch specs
- You may be closer to the ‘realistic’ performance bounds than you think you are!

![Graph showing performance vs. arithmetic intensity](image-url)
Message 2: Account for Divides

- Operations such as div, exp, log and trigonometric functions usually take more than one instructions

- Gap between canonical and empirical FLOPs:
  - Empirical: each divide counts as multiple FLOPs
  - Canonical: each counts as 1 FLOP
Message 2: Account for Divides

- Operations such as div, exp, log and trigonometric functions usually take more than one instructions

- GPP (General Plasmon Pole) kernel from BerkeleyGW (Material Science)
  - Tensor-contraction, abundant parallelism, large reductions
  - Low FMA counts, divides, complex double data type

```plaintext
do band = 1, nbands #threadblocks
  do igp = 1, ngpown
    do ig = 1, ncouls #threads
      do iw = 1, nw #unrolled
        compute; reductions
    end do
  end do
end do
```

Highly parameterizable:

- Varying $nw$ from 1 to 6 to increase arithmetic intensity
  - increasing FLOPs, same HBM data movement

```c
do band = 1, nbands  #threadblocks
  do igp = 1, ngpown
    do ig = 1, ncouls  #threads
      do iw = 1, nw    #unrolled
        compute; reductions
```
Message 2: Account for Divides

Highly parameterizable:
- Varying \(nw\) from 1 to 6 to increase arithmetic intensity
  - increasing FLOPs, same HBM data movement
- Striding \(ig\) loop to analyze impact of strided memory access
  - Split \(ig\) loop to two loops and place the 'blocking' loop outside

```plaintext
do band = 1, nbands      #threadblocks
  do  igp = 1, ngpown
      do  igs = 0, stride - 1  #threads
          do  ig = 1, ncouls/stride
              do  iw = 1, nw        #unrolled
                  compute; reductions
```
Message 2: Account for Divides

- Gap between canonical and empirical FLOPs:
  - Empirical: each divide counts as multiple FLOPs
  - Canonical: each counts as 1 FLOP

<table>
<thead>
<tr>
<th>Count (GFLOPs)</th>
<th>KNL</th>
<th>V100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( nw = 1 )</td>
<td>( nw = 3 )</td>
</tr>
<tr>
<td>Canonical</td>
<td>921.4</td>
<td>2354.7</td>
</tr>
<tr>
<td>Empirical</td>
<td>1055.8</td>
<td>2834.5</td>
</tr>
<tr>
<td>Difference</td>
<td>15%</td>
<td>20%</td>
</tr>
</tbody>
</table>

- Kernel performance will move diagonally up!
  - Increased GFLOP/s and arithmetic intensity (FLOPs/Byte)
Message 2: Account for Divides

- Your code may be in a different regime or closer to the ceiling than you realize!
Message 3: Roofline Capabilities

Again, test with different variants of the GPP kernel:

• Vary AI by varying $nw$ from 1 to 6
• Enable/Disable FMA by compiling with `-fmad=true/false`
• Change memory access pattern by striding the $ig$ loop

Platforms: Intel KNL and NVIDIA V100

Architectural Efficiency $\rightarrow$ Performance Portability Score

$$e_i(a, p) = \frac{P_i(a, p)}{\min(F_i, B_i \times I_i(a, p))}$$

$$\Phi(a, p, H) = \begin{cases} 
\frac{|H|}{\sum_{i \in H} e_i(a, p)} & \text{if } i \text{ is supported, } \forall i \in H \\
0 & \text{otherwise}
\end{cases}$$
• Varying AI: bottleneck shifts at $nw = 2$ from KNL to V100
• Easier to achieve no-FMA ceiling on V100 than KNL
  – KNL issues 2 instr./cycle and executes 2 instr./cycle
  – V100 issues 4 warps/cycle and executes 1 warp/cycle (32 FP64 cores)
Message 3: Roofline Capabilities

• With increasing $n_W$ (and AI):
  – No-FMA performance portability score is consistently > 80%
  – FMA benefit is far less than 2x at high $n_W$’s. Architectural efficiency suffers and so does performance portability.

• At high $n_W$’s, increasing FMA instruction percentage is key on both platforms!

<table>
<thead>
<tr>
<th>Architectural Efficiency</th>
<th>$n_W = 1$</th>
<th>$n_W = 2$</th>
<th>$n_W = 3$</th>
<th>$n_W = 4$</th>
<th>$n_W = 5$</th>
<th>$n_W = 6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>No-FMA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KNL</td>
<td>82.06%</td>
<td>72.95%</td>
<td>73.74%</td>
<td>78.72%</td>
<td>81.28%</td>
<td>82.81%</td>
</tr>
<tr>
<td>V100</td>
<td>92.88%</td>
<td>92.88%</td>
<td>97.43%</td>
<td>98.91%</td>
<td>1</td>
<td>99.73%</td>
</tr>
<tr>
<td>Performance Portability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KNL</td>
<td>87.14%</td>
<td>81.72%</td>
<td>83.95%</td>
<td>87.67%</td>
<td>89.93%</td>
<td>90.49%</td>
</tr>
<tr>
<td>V100</td>
<td>84.98%</td>
<td>77.50%</td>
<td>66.77%</td>
<td>55.28%</td>
<td>46.56%</td>
<td>39.65%</td>
</tr>
<tr>
<td>FMA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KNL</td>
<td>97.36%</td>
<td>91.50%</td>
<td>76.70%</td>
<td>65.44%</td>
<td>65.07%</td>
<td>66.38%</td>
</tr>
<tr>
<td>V100</td>
<td>90.76%</td>
<td>83.92%</td>
<td>71.39%</td>
<td>59.93%</td>
<td>54.28%</td>
<td>49.65%</td>
</tr>
<tr>
<td>Performance Portability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Message 3: Roofline Capabilities

- Strided memory access pattern
  - Transaction size: 64B on KNL vs. 32B on V100
  - Data: 16B per complex number
Message 3: Roofline Capabilities

• With increasing stride size
  – GPP becomes more and more bandwidth bound on both architectures, eventually all saturating HBM
• Even though performance in GFLOP/s drops, architecture efficiency grows and so does performance portability score.
• Stride-\( n \) performance is bound by a lower ceiling than stride-1 performance.

<table>
<thead>
<tr>
<th>Architectural Efficiency</th>
<th>Original</th>
<th>Stride 2</th>
<th>Stride 4</th>
<th>Stride 8</th>
<th>Stride 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNL</td>
<td>38.40%</td>
<td>75.24%</td>
<td>98.39%</td>
<td>99.20%</td>
<td>98.00%</td>
</tr>
<tr>
<td>V100</td>
<td>65.64%</td>
<td>85.43%</td>
<td>98.81%</td>
<td>99.89%</td>
<td>-</td>
</tr>
<tr>
<td>Performance Portability</td>
<td>48.46%</td>
<td>80.01%</td>
<td>98.60%</td>
<td>99.55%</td>
<td>-</td>
</tr>
</tbody>
</table>
Summary and Conclusions

• Why performance portability is important and past attempts to define it and quantify it → PP Metric proposed by Pennycook et al.

• Methodology to collect Roofline data for performance port analysis

• Roofline is very powerful in capturing nuances of performance analysis such as changes in AI, instruction mix, instruction issue/exec bandwidth and memory access pattern.

• It is imperative to use empirical Roofline ceilings, account for complex instructions such as divides appropriately, and select relevant ceilings to compare performance with, in order to assess architectural efficiency more accurately and also perform performance portability analysis more accurately.


Thank You!