Performance Optimization of Scientific Codes with the Roofline Model

Charlene Yang
National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory
cjiang@lbl.gov

Zakhar Matveev
Intel Corporation
zakhar.a.matveev@intel.com

Aleksandar Ilic, Diogo Marques
INESC-ID, Instituto Superior Técnico
Universidade de Lisboa
{aleksandar.ilic, diogo.marques}@inesc-id.pt
Performance Optimization of Scientific Codes with the Roofline Model

- 2:00 - 2:40 Introduction to the Roofline Model - Charlene Yang
- 2:40 - 3:20 Roofline Analysis on NVIDIA GPUs - Charlene Yang
- 3:20 - 3:50 Cache-Aware Roofline Model (CARM) - Aleksandar Ilic
- 3:50 - 4:00 Installation of Intel Advisor - Zakhar Matveev
- 4:00 - 4:30 Break
- 4:30 - 4:45 Introduction to Intel Advisor - Zakhar Matveev
- 4:45 - 5:25 Roofline Analysis on Intel CPUs - Zakhar Matveev
- 5:25 - 5:55 Application Use Cases - Diogo Marques
- 5:55 - 6:00 Q&A - All

Please complete the survey afterwards!
Find this tutorial at https://2019.isc-program.com/ and click ‘Give Feedback’
Related Talks at ISC

Tutorial at Intel Developer Connect:

• **Roofline Model-Based Optimization Guidance and Tuning for Modern CPUs**
• Zakhar A. Matveev, Diogo Marques, Aleksandar Ilic
• Mon Jun 17 9am - 3:30pm, Matterhorn 1, Movenpick
Material Download

- **Google Drive:**
  - Slides
  - Intel Advisor remote viewers (Linux/Mac/Windows)
  - Stencil code snapshot


- **USB Sticks:**
  - Please pass them around after finishing copying
Introduction to the Roofline Model

Charlene Yang
Lawrence Berkeley National Laboratory
Jun 16 2019, Frankfurt
Performance Models

The Maze of Performance Optimization

The Map !!!
Performance Models

Modern architectures are complicated!


Intel Haswell CPU

NVIDIA Volta GPU
Performance Models

- Many components contribute to the kernel run time
- An interplay of application characteristics and machine characteristics

**Roofline Model**

<table>
<thead>
<tr>
<th>#FP operations</th>
<th>FLOP/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache data movement</td>
<td>Cache GB/s</td>
</tr>
<tr>
<td>DRAM data movement</td>
<td>DRAM GB/s</td>
</tr>
<tr>
<td>PCIe data movement</td>
<td>PCIe bandwidth</td>
</tr>
<tr>
<td>MPI Message Size</td>
<td>Network Bandwidth</td>
</tr>
<tr>
<td>MPI Send:Wait ratio</td>
<td>Network Gap</td>
</tr>
<tr>
<td>#MPI Wait’s</td>
<td>Network Latency</td>
</tr>
<tr>
<td>IO</td>
<td>File systems</td>
</tr>
</tbody>
</table>

Focus on one or two dominant components!
Sustainable performance is bound by

$$\text{GFLOP/s} = \min \left\{ \frac{\text{Peak GFLOP/s}}{\text{AI} \times \text{Peak GB/s}} \right\}$$

- **Arithmetic Intensity (AI)** = \(\frac{\text{FLOPs}}{\text{Bytes}}\)

- **How did this come about?**
  - A CPU DRAM example

**Transition @ AI ==**

$$\frac{\text{Peak GFLOP/s}}{\text{Peak GB/s}} = \text{‘Machine Balance’}$$
One could hope to always attain peak performance (FLOP/s)
However, finite locality (reuse) and bandwidth limit performance.
Assume:
- Idealized processor/caches
- Cold start (data in DRAM)

\[
\text{Time} = \max \left\{ \frac{\text{#FP ops}}{\text{Peak GFLOP/s}}, \frac{\text{#Bytes}}{\text{Peak GB/s}} \right\}
\]
One could hope to always attain peak performance (FLOP/s)
However, finite locality (reuse) and bandwidth limit performance.
Assume:
- Idealized processor/caches
- Cold start (data in DRAM)

\[
\frac{\text{Time}}{\#\text{FP ops}} = \max \left\{ \frac{1}{\text{Peak GFLOP/s}}, \frac{\#\text{Bytes}}{\#\text{FP ops}} \div \frac{\text{Peak GB/s}}{\text{DRAM Bandwidth}} \right\}
\]
One could hope to always attain peak performance (FLOP/s).

However, finite locality (reuse) and bandwidth limit performance.

Assume:

- Idealized processor/caches
- Cold start (data in DRAM)

\[
\frac{\text{#FP ops}}{\text{Time}} = \min \left\{ \frac{\text{Peak GFLOP/s}}{\left( \frac{\text{#FP ops}}{\text{#Bytes}} \right) \times \text{Peak GB/s}} \right\}
\]
One could hope to always attain peak performance (FLOP/s) however, finite locality (reuse) and bandwidth limit performance.

Assume:
- Idealized processor/caches
- Cold start (data in DRAM)

$$\text{Peak GFLOP/s} = \min \left\{ \text{Arithmetic Intensity (AI)} \times \text{Peak GB/s} \right\}$$

Arithmetic Intensity (AI) = FLOPs / Bytes (as presented to DRAM)
Thus we obtain the model as

\[ \text{GFLOP/s} = \min \begin{cases} \text{Peak GFLOP/s} \\ \text{AI} \times \text{Peak GB/s} \end{cases} \]

where Arithmetic Intensity (AI) is \( \frac{\text{FLOPs}}{\text{Bytes}} \)

- Machine Balance (FLOPs/Byte) = 8.9 (V100, DP, HBM) or 5.1 (KNL, DP, HBM)

\[ \text{Transition @ AI} = \frac{\text{Peak GFLOP/s}}{\text{Peak GB/s}} = \text{‘Machine Balance’} \]
Roofline Performance Model

• A throughput-oriented model
  – tracks rates not times, i.e. GFLOP/s, GB/s, not seconds

• An abstraction over
  – architectures, ISA (CPU, GPU, Haswell, KNL, Pascal, Volta)
  – programming models, programming languages
  – numerical algorithms, problem sizes

• In log-log scale to easily extrapolate performance along Moore’s Law
More Advanced on Roofline
Roofline Performance Model

- This is a single Roofline
- What about the memory hierarchy, different execution configurations, and instruction mixes?
  - Hierarchical Roofline
  - Multiple compute ceilings
Hierarchical Roofline

- Superposition of multiple Rooflines
  - Incorporate full memory hierarchy
  - Arithmetic Intensity = \( \frac{\text{FLOPs}}{\text{Bytes}_{L1/L2/HBM/SysMem}} \)

- Each kernel will have multiple AI’s but one observed GFLOP/s performance

- Hierarchical Roofline tells you about cache locality
Multiple Compute Ceilings

- Impact of **execution configuration**

- Concurrency affects your peak
  - OpenMP thread concurrency
  - SM occupancy
  - load balance
  - threadblock/thread configuration

- Performance is bound by the **actual concurrency** ceiling
Multiple Compute Ceilings

- Impact of **instruction mix**

- Applications are usually a mix of FMA.f64, ADD.f64, MUL.f64...

- Performance is a **weighted** average ... bound by a **partial FMA ceiling**
The Roofline Model

- helps you identify the bottlenecks
- guides you through optimization
- tells you when to stop

An example:
- NESAP for Cori - BerkeleyGW
Roofline Example: BerkeleyGW

Optimization Path for Kernel-C (Sigma):

1. Add OpenMP
2. Initial Vectorization
   - loop reordering
   - conditional removal
3. Cache-Blocking
4. Improved Vectorization
   - divides
5. Hyper-threading
General Optimization Strategy

- Broadly speaking, three approaches to improving performance:

  - Peak FLOP/s
  - No FMA
  - No vectorization

Arithmetic Intensity (FLOP:Byte) vs. Attainable FLOP/s
Broadly speaking, three approaches to improving performance:

- Maximize compute performance
  - multithreading
  - vectorization
  - increase SM occupancy
  - utilize FMA instructions
  - minimize thread divergence
General Optimization Strategy

- Broadly speaking, three approaches to improving performance:
  - Maximize compute performance
  - Maximize memory bandwidth
    - utilize higher-level caches
    - NUMA-aware allocation
    - avoid H-D transfers
    - avoid uncoalesced memory access

![Diagram showing the relationship between Attainable FLOP/s and Arithmetic Intensity (FLOP:Byte). There are curves for HBM GB/s and NUMA, with a point indicating the No FMA scenario.](image)
General Optimization Strategy

- Broadly speaking, three approaches to improving performance:
  - Maximize compute performance
  - Maximize memory bandwidth
  - Improve AI
    - minimize data movement
    - exploit cache reuse
Roofline Data Collection
Example #1: STREAM Triad

- 2 FLOPs per iteration
- Transfer 24 bytes per iteration
  - read $X[i]$, $Y[i]$, and write $Z[i]$
- $AI = 0.083$ FLOPs per byte
- Memory bound

```c
for(i=0;i<N;i++)
    Z[i] = X[i] + alpha*Y[i];
```

Arithmetic Intensity (Flop:Byte)

$\text{FLOP/s} \leq AI \times \text{DRAM GB/s}$
Example #2: 7-pt stencil

- 7 FLOPs; 8 memory references (7 reads, 1 store) per pt
- Cache can filter all but 1 read and 1 write per pt
- \( AI = 0.44 \) FLOPs per byte
- Memory bound, but 5x the GFLOP/s rate

```
for(k=1;k<dim+1;k++){
    for(j=1;j<dim+1;j++){
        for(i=1;i<dim+1;i++){
            new[k][j][i] = -6.0*old[k][j][i] + old[k][j][i-1] + old[k][j][i+1] + old[k][j-1][i] + old[k][j+1][i] + old[k-1][j][i] + old[k+1][j][i];
        }
    }
}
```
Pen and Paper

- Not scalable for real-life applications
- Millions of lines of code; mix of different languages
- Complicated modern architecture
  - memory hierarchy, caching effects
  - ISA
- Different problem sizes

We need tools!
We Need Tools!

- Roofline ceilings
  - vendor specifications
  - empirical measurements
    - ERT
    - https://bitbucket.org/berkeleylab/cs-roofline-toolkit
We Need Tools!

Where to put these dots?
We Need Tools!

Require three raw measurements:
- Runtime
- FLOPs
- Bytes (on each cache level)

In order to calculate AI and GFLOP/s:

\[
\text{Arithmetic Intensity} = \frac{\text{FLOPs}}{\text{Data Movement}} \\
\text{Performance} = \frac{\text{FLOPs}}{\text{Runtime}}
\]

Where to put these dots?
Methodology to Construct Roofline

1. Collect Roofline ceilings
   - **compute** (FMA/no FMA) and **bandwidth** (DRAM, L2, …)
   - ERT: https://bitbucket.org/berkeleylab/cs-roofline-toolkit

2. Collect application performance
   - **FLOPs**, **bytes** (DRAM, L2, …), **runtime**
   - SDE, VTune, LIKWID, Advisor, nvprof, …

3. Plot Roofline with Python Matplotlib (or other tools of your preference)
   - **arithmetic intensity**, **GFLOP/s** performance, **ceilings**
   - example scripts: https://github.com/cyanguwa/nersc-roofline
Automated Data Collection
Data Collection on Intel CPUs

The not-so-automated way 1:

- **Intel SDE** for FLOPs (emulation)
- **Intel VTune** for DRAM bytes (HW counters)
- Runtime

- **DRAM Roofline** only

- Used by NESAP for Cori
  - NERSC Exascale Science Application Program
Data Collection on Intel CPUs

MFDn

EMGeo

PICSAR

DRAM Rooflines of NESAP Codes
Data Collection on Intel CPUs

The not-so-automated way 2:

- **LIWID** for FLOPs and bytes
  - Both are based on HW counters
- Runtime
- Hierarchical Roofline
- Limited by quality of HW counters
- High-level characterization, no callstack

https://github.com/RRZE-HPC/likwid
Data Collection on Intel CPUs

The fully automated way:

- Intel Advisor, Roofline feature
- Instrument applications **automatically**
  - one dot per loop nest/function
- FLOPs, bytes and runtime
- Hierarchical Roofline
- Integrates with other Advisor capabilities
- Benchmarks target system
Data Collection on NVIDIA GPUs

• Still very manual at this stage, but...

• Runtime:
  – Internal timers or `nvprof --print-gpu-trace`

• FLOPs:
  – DP/SP/HP counters and metrics, `nvprof --metrics flop_count_dp/sp/hp` or `tensor_precision_fu_utilization`

• Bytes for different cache levels:
  – Bytes = (read transactions + write transactions) x transaction size
  – `nvprof --metrics 'metric_name'` e.g. `gld/gst_transactions`

• Hierarchical Roofline
Summary

• The Roofline Model formulizes the interaction between machine characteristics and application characteristics, and guides optimization
  - Peak computational throughput and bandwidth
  - Arithmetic intensity, cache locality, instruction mix...

• Automate Roofline data collection
  - Intel CPUs
    • Intel SDE + Intel VTune, Intel Advisor
  - NVIDIA GPUs
    • nvprof, Nsight Compute

More in the next few talks!
• S. Williams, A. Waterman and D. Patterson, “Roofline: An Insightful Visual Performance Model for Multicore Architectures,” *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009

• LBNL CRD Roofline Research:
  
  https://crd.lbl.gov/departments/computer-science/PAR/research/roofline

• Empirical Roofline Toolkit (ERT):
  
  https://bitbucket.org/berkeleylab/cs-roofline-toolkit

• Python scripts for plotting manually-collected Roofline:
  
  https://github.com/cyanguwa/nersc-roofline/tree/master/Plotting
Acknowledgement

- This material is based upon work supported by the Advanced Scientific Computing Research Program in the U.S. Department of Energy, Office of Science, under Award Number DE-AC02-05CH11231.

- This material is based upon work supported by the DOE RAPIDS SciDAC Institute.

- This research used resources of the National Energy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02- 05CH11231.