Roofline Model
Markus Püschel

Computer Science
ETH Zürich

Operational Intensity

Definition: Given a program P, assume cold (empty) cache

\[ \text{Operational intensity: } I(n) = \frac{W(n)}{Q(n)} \]

- \#flops (input size n)
- \#bytes transferred cache ↔ memory (for input size n)

Examples: Determine asymptotic bounds on \( I(n) \)

- Vector sum: \( y = x + y \) \( O(1) \)
- Matrix-vector product: \( y = Ax \) \( O(1) \)
- Fast Fourier transform \( O(\log(n)) \)
- Matrix-matrix product: \( C = AB + C \) \( O(n) \)
Compute/Memory Bound

A function/piece of code is:

- **Compute bound** if it has high operational intensity
- **Memory bound** if it has low operational intensity

The roofline model makes this more precise

Blackboard

---

**Roofline model** (Williams et al. 2008)

Resources in a processor that bound performance:

- peak performance \([\text{flops/cycle}]\)
- memory bandwidth \([\text{bytes/cycle}]\)
- others

**Platform model**

- \(P_\text{e}ak\) performance \(\pi\) [ops/cycle]
- Raw bandwidth from manual is unattainable (maybe 60% is)
- Stream benchmark may be conservative

**Algorithm model** (n is the input size)

- Operational intensity \(I(n) = \frac{W(n)}{Q(n)}\)
  - number of flops (cost) [ops/byte]
  - number of bytes transferred between memory and cache
- \(Q(n)\): assumes empty cache; best measured with performance counters

**Notes**

- In general, \(Q\) and hence \(W/Q\) depend on the cache size \(m\) [bytes].
- For some functions the optimal achievable \(W/Q\) is known:
  - FFT/sorting: \(O(\log(m))\)
  - Matrix multiplication: \(O(\sqrt{m})\)

**Bound based on \(\beta\)**

- Assume program as operational intensity of \(x\) ops/byte
- It can get only \(\beta\) bytes/cycle
- Hence: performance \(= y \leq \beta x\)
- In log scale: \(\log_2(y) \leq \log_2(\beta) + \log_2(x)\)
- Line with slope 1: \(y = \beta x\) for \(x = 1\)

**Variations**

- Vector instructions: peak bound goes up (e.g., 4 times for AVX)
- Multiple cores: peak bound goes up \(p\) times for \(p\) cores
- Program has uneven mix adds/mults: peak bound comes down (note: now this bound is program specific)
- Accesses with little spatial locality: operational intensity decreases (because entire cache blocks are loaded)
Roofline Measurements

Tool developed in our group
[G. Ofenbeck, R. Steinmann, V. Caparros-Cabezas, D. Spampinato]
http://www.spiral.net/software/roofline.html

Example plots follow

**Blackboard:** Get (non-asymptotic) bounds on I(n) for
\[
\begin{align*}
\text{daxpy:} & \quad y = ax + y \\
\text{dgemv:} & \quad y = Ax + y \\
\text{dgemm:} & \quad C = AB + C
\end{align*}
\]

**Blackboard:** for which sizes n should the last one be tight?

---

Roofline Measurements

Core i7 Sandy Bridge, 6 cores
Code: Intel MKL, **sequential**
Cold cache

What happens when we go to parallel code?
Roofline Measurements

Core i7 Sandy Bridge, 6 cores
Code: Intel MKL, parallel
Cold cache

What happens when we go to warm cache?

Roofline Measurements

Core i7 Sandy Bridge, 6 cores
Code: Intel MKL, sequential
Warm cache
**Summary**

Roofline plots distinguish between memory and compute bound

Can be used on paper

Measurements difficult (performance counters) but doable

Can yield interesting insights
Extended Roofline Model

Victoria Caparrós Cabezas
Markus Püschel

Computer Science
ETH Zürich

Roofline Plot

Performance [Flops/Cycle]
Bound based on memory bandwidth
Bound based on compute throughput
The roofline model can only handle "throughput" bottlenecks
What if the bottleneck is
1. Latency of flops
2. Cache size
3. Memory latency
4. Size of 000 buffers
5. Overlap
6. ...

Bound based on memory bandwidth
2800 dgemv

Bound based on compute throughput
100... 2800 dgemm

Operational Intensity [Flops/Byte]
Goal
Include additional hardware-related bottlenecks as performance bounds in the roofline plot

The performance bottlenecks are:
1. Latency of flops
2. Latency of L1 accesses

Applications
- Assess the impact of hardware upgrades
- Code optimization


**Simple example for latency**

\[ P = I\beta \]

**Performance**

\[ [\text{Flops/Cycle}] \]

\[ P = \pi \]

**Instruction Mix**

\[ P' = \pi / 2 \]

**Latency**

\[ P'' = P'/\lambda \]

---

**Question 1:** What if the latency only affects some of the operations?

---

**Simple example for overlap**

**Perfect overlap**

\[ T = \max(T_{\text{comp}}, T_{\text{mem}}) \]

\[ P = \min(\beta I, \pi) \]

---

**Performance**

\[ [\text{Flops/Cycle}] \]
Simple example for overlap

Perfect overlap

\[ T = \max(T_{\text{comp}}, T_{\text{mem}}) \]
\[ P = \min(\beta I, \pi) \]

Overlap \(0 \leq \alpha \leq 1\)

\[ T = T_{\text{comp}} + T_{\text{mem}} - \alpha \min(T_{\text{comp}}, T_{\text{mem}}) \]
\[ P = \frac{I}{\frac{T}{\pi} + \frac{1}{\beta} - \alpha \min(\frac{T}{\pi}, \frac{1}{\beta})} \]

Question 2: How to get overlap information?

Answers to questions 1 and 2 require cycle-by-cycle information

Our approach: DAG-based performance model
main()
{
    int n = 1000;
    A=(double*)_mm_malloc(...);
    ...
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            for(int k=0; k<n; k++)
                C[i][j]=A[i][k]*B[k][j];
}

DAG scheduling with uarch constraints

Execution through the LLVM Interpreter

Extension to the LLVM Interpreter to schedule the DAG (Greedy schedule based on Tomasulo’s algorithm)
DAG scheduling with uarch constraints

Parameters to model an Intel Xeon E5-2680 with a Sandy Bridge microarchitecture

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Unit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
<td>flops/cycle</td>
<td>1.3/2.4</td>
</tr>
<tr>
<td>L2 latency</td>
<td>cycle</td>
<td>3.5</td>
</tr>
<tr>
<td>Capacity</td>
<td>byte</td>
<td>512K, 256K, 20M</td>
</tr>
</tbody>
</table>

Cycles

Access to memory hierarchy: reuse distance analysis

Quantifying Properties of the DAG

Node properties

Number of nodes

\[ W_A = 2 \quad T_A = 6 \]

\[ W_M = 2 \quad T_M = 7 \]

\[ Q_{L1} = 3 \quad T_{L1} = 5 \]

\[ Q_{L2} = 2 \quad T_{L2} = 14 \]

DAG properties

Execution time

\[ T = 26 \text{ cycles} \]

Operational Intensity

\[ I_{L1} = \frac{W}{Q_{L1}} = 0.17 \text{ flops/byte} \]

Performance

\[ P = \frac{W}{T} = 0.15 \text{ flops/cycle} \]

How good is this performance estimation in real code?
Performance Estimation

Utilization-based Bottleneck Modeling

Where are execution cycles spent so that peak performance is not reached?

Roofline Model: Basic and Extended
Math/CS/HPC Workshop, St. Germain au Mont D’Or, May 2016
© Markus Püschel, Computer Science, ETH Zürich
Utilization-based Bottleneck Modeling

$$T_{\text{issue}} = \text{Number of cycles in which nodes are issued}$$

$$T_{\text{issue}} = 4$$
Utilization-based Bottleneck Modeling

\[ T^{\text{issue}} = \text{Number of cycles in which nodes are issued} \]
\[ T^{\text{lat}} = \text{Number of cycles in which nodes are executed, but not issued} \]
\[ T^{\text{stall}} = \text{Number of cycles in which OOO buffers are full} \]

\[ T^{\text{issue}} = 4 \]
\[ T^{\text{lat}} = 8 \]
\[ T^{\text{stall}} = 5 \]
Scheduled DAG — Roofline Plot

Bottleneck lines are an indicator of where cycles are spent, and their relative importance to performance

Bottleneck Properties and Limitations

There is no guarantee that changing the parameters associated to a bottleneck, performance will improve

Generalized roofline plots become specific to program and input

Does not consider multiple cores

Our analysis enables the handling of code with different phases
Some Results

Vector Sum Reduction (cold cache)

Performance [Flops/Cycle]

- Peak \( \pi \) (2 f/s)
- Issue
- Latency/RS
- \( a_{mem-comp} \)

Operational Intensity [Flops/Byte]
Vector Sum Reduction (cold cache)

Increase mem latency to 200 cycles

Reduce RS to 20

FFT NR (warm cache)—Increasing Size

Size $2^0$

Size $2^{20}$
MVM Double Loop — Cold vs. Warm Cache

Cold cache

Warm cache

Ongoing Work — Shrinking Platform
Algorithm to remove overprovisioned resources based on utilization information from bottleneck analysis
Conclusion and Discussion

Generalization of the roofline plot that integrates many hardware-related bottlenecks as performance bounds into one viewgraph

Structure bottlenecks and show their relative importance

New insights into the interplay between different microarchitectural components and the associated bottlenecks


References

Samuel Williams, Andrew Waterman, David Patterson
*Roofline: an insightful visual performance model for multicore architectures*
Communications ACM 55(6): 121-130 (2012)

Georg Ofenbeck, Ruedi Steinmann, Victoria Caparros, Daniele G. Spampinato and Markus Püschel
*Applying the Roofline Model*