CS 575: The Roofline Model

Wim Bohm

Colorado State University

Spring 2013

Roofline: An Insightful Visual Performance Model.. David Patterson.. U.C. Berkeley
submitted to CAM April 2008

Analyzing Program Performance

- In empirical Computer Science, we plot functions describing the run time (or the memory use) of a program:
  - This can be as a **function of the input size**. We have seen this in e.g. cs320 or cs420, where we studied polynomial and exponential (**monotonically growing**) complexity.
  - In this class we also study program performance as a function of the number of processors.
    - In this case the functions are positive and, hopefully decreasing.
    - Also we plot speedup curves, which are usually asymptotic
  - The roofline model plots GFlops/second as a **function of Operational Intensity** (GFlops/byte)

Straight Lines

CS475: When plotting data we get the most information from **straight lines**!

- We can easily recognize a straight line (y = ax+b)
  - The **slope (a)** and **y intercept (b)** tells us all.
- So we need to turn our data sets into straight lines.
- This is easiest done using log-s, because they turn a multiplicative factor into a shift up (y intercept) , and an exponential into a multiplicative factor (slope)

Exponential functions

$\log(2^n) = n \log 2$ linear in n

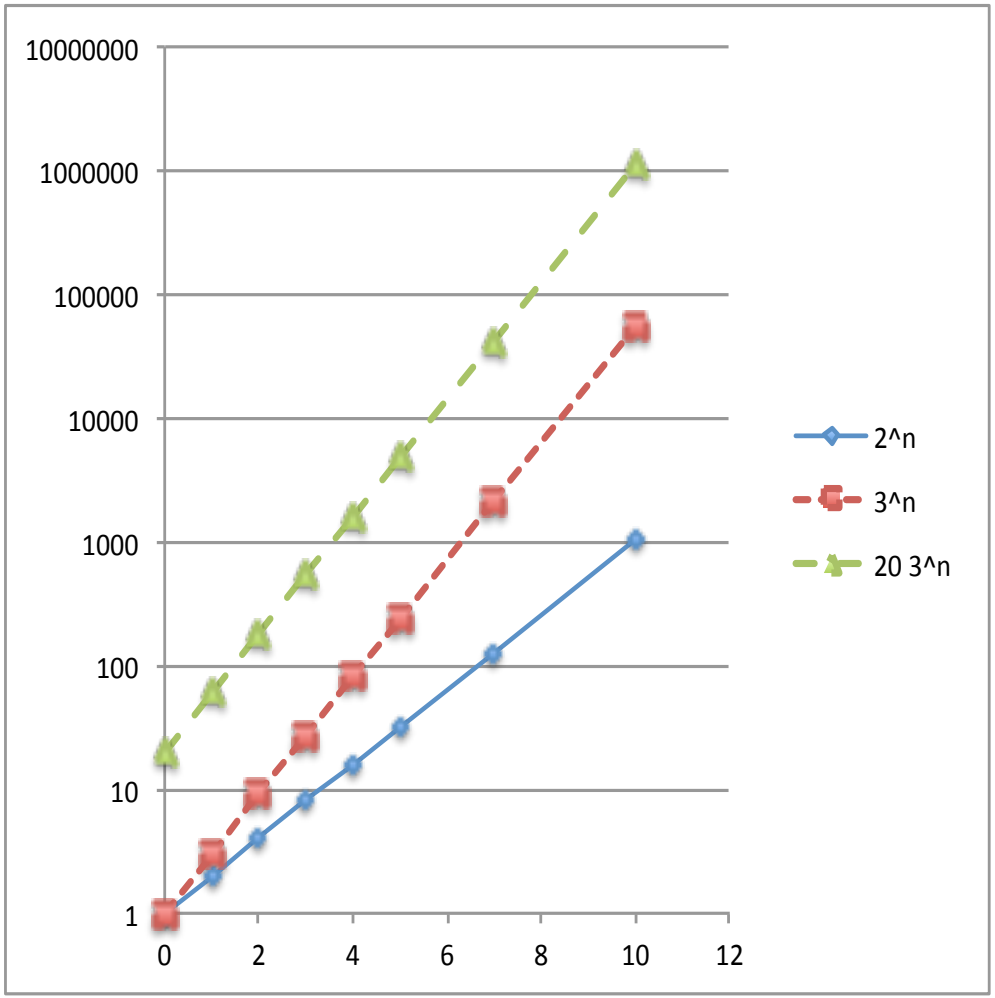
$\log(3^n) = n \log 3$ slope is base of log

$\log(4.3^n) = n \log 3 + \log 4$ * shifts it up

$\log((3^n)/4) = n \log 3 - \log 4$ / shifts it down

Exponentials: semi-log plot

n	2^n	3^n	$20*3^n$
0	1	1	20
1	2	3	60
2	4	9	180
3	8	27	540
4	16	81	1620
5	32	243	4860
7	128	2087	41740
10	1024	56349	1126980



semi-log plot:
    y–axis on log scale
    x-axis linear
angle: base
shift: multiplicative factor

Polynomials

- What if we take the log of a polynomial?

  e.g. $f(n) = 5n^3$

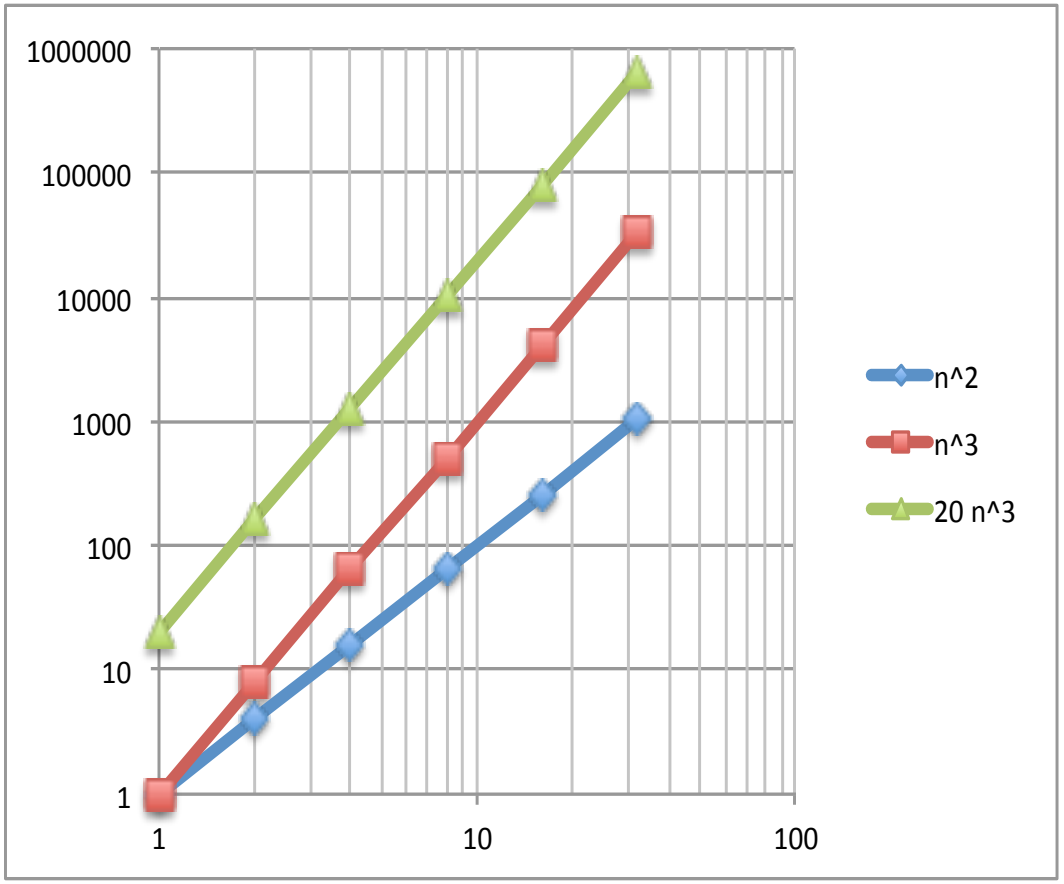
  $\log(\mathbf{f(n)}) = \log(5n^3) = \log 5 + 3 \log(n)$

- while the **log of an exponential is linear in n**, the **log of a polynomial is linear in log(n)**
- therefore we need to plot polynomials on a log-log scale (both x and y axis logarithmic)

Polynomials: log-log plot

n	n^2	n^3	$20*n^3$
1	1	1	20
2	4	8	160
4	16	64	1280
8	64	512	10240
16	256	4096	81820
32	1024	32768	655360

angle: degree
shift: multiplicative factor



logs of sums

- Often we don't have a single factor in our function:
  - $3^n + 2^n$
  - $n^3 + n^2$
  - **Watch it:** log of sum is not sum of logs (what is?)
- Straight lines not completely straight anymore but **asymptotically straight:**

$\log(3^n+2^n) = \log((1+(2/3)^n)3^n)=\log(1+(2/3)^n) + n\log(3)$

$\log(n^3+n^2) = \log((1+(1/n))n^3)=\log(1+(1/n)) + 3\log(n)$

$\log(1+(2/3)^n)$ and $\log(1+(1/n))$ go to zero for large n

Decreasing functions

- This second class of functions can be used to represent running times of programs as a function of the number of processors.
- **Amdahl's Law:** programs have inherently sequential parts, that do not speed up with more processors:

$$T(p) = a + c/p$$

a: the sequential part

c: the parallelizable part

Plotting hyperbolic functions

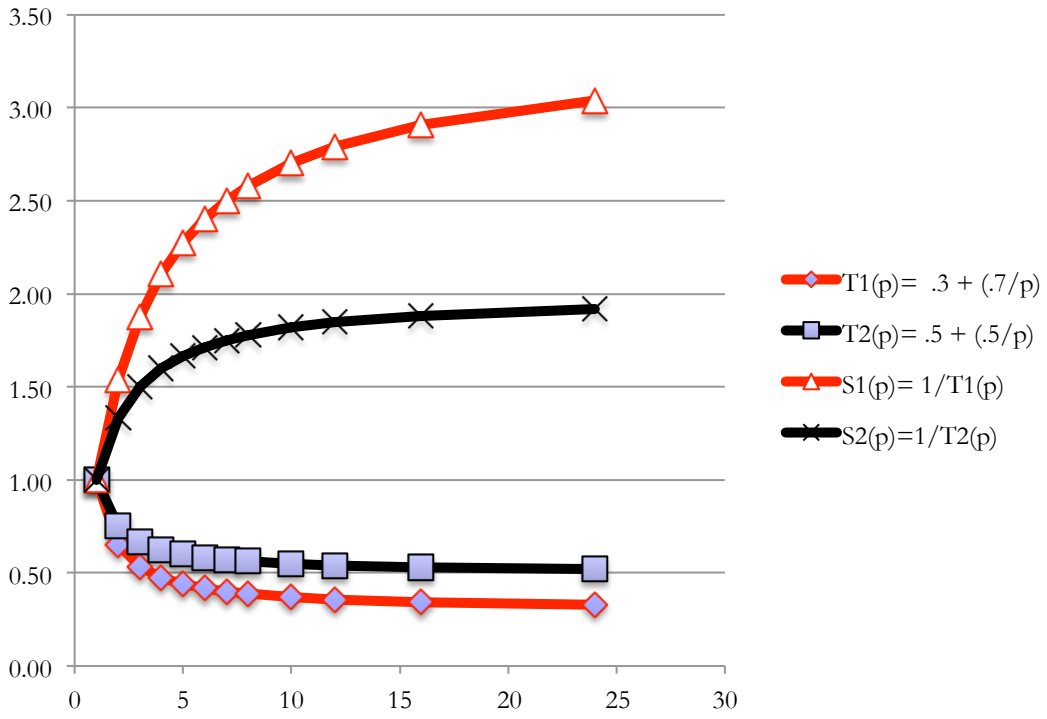
- A simple way to turn $T(p) = c/p$ into a straight line is to plot its reciprocal
- In the case of $T(p) = a + c/p$, the speedup $S(p)$ is

$$S(p)=T(1)/T(p)= (a+c)/(a+c/p)$$

For $a>0$ this is not a straight, but a curve that grows and then flattens out to a constant $(a+c)/a$

Time T(p) and Speedup S(p)

p	T1(p)= .3 + (.7/p)	T2(p)= .5 + (.5/p)	S1(p)= 1/T1(p)	S2(p)= 1/T2(p)
1	1.00	1.00	1.00	1.00
2	0.65	0.75	1.54	1.33
3	0.53	0.67	1.88	1.50
4	0.48	0.63	2.11	1.60
5	0.44	0.60	2.27	1.67
6	0.42	0.58	2.40	1.71
7	0.40	0.57	2.50	1.75
8	0.39	0.56	2.58	1.78
10	0.37	0.55	2.70	1.82
12	0.36	0.54	2.79	1.85
16	0.34	0.53	2.91	1.88
24	0.33	0.52	3.04	1.92



Plotting Data: Summary

- Visually, a straight line conveys the most information.
  - If your data is not linear, massage it so that is linear, then deduce the original function.
  - If y=f(x) is polynomial: **log y is linear with log x**

$$y = f(x) = a_0 + a_1 x + \cdots + a_n x^n \approx a_n x^n \quad \text{(asymptotically)}$$

$$\log y = \log a_n + n \log x$$

  - If y=f(x) is exponential: **log y is linear with x**

$$y = f(x) = ba^x$$

$$\log y = \log b + x(\log a)$$

- In the case of T(p) = a + c/p, the speedup is

$$S(p)=T(1)/T(p)= (a+c)/(a+c/p)$$

growing asymptotically to (a+c)/a

Performance Models

- Single CPU /core systems all behave pretty similarly (cache, ILP)
- Multicore Systems are more diverse than single core (Cell, GPU, multi CPU), so we need to get a new intuition for multicore performance
- We like simple performance models, expressing the essence of program behavior. They may be not perfect, but they help our intuition

Example Models

- 3C model (CS475) for caches
- Amdahl for parallel computing
- Roofline for interaction between processors and memory

Cache misses: the 3C model

- Compulsory: On the first access to a block; the block must be brought into the cache.
- Conflict: several memory locations are mapped to the same cache location.

- Capacity: blocks are discarded from cache because cache cannot contain all blocks needed for program execution (program working set is much larger than cache capacity). This is the one we can improve through **data locality**.

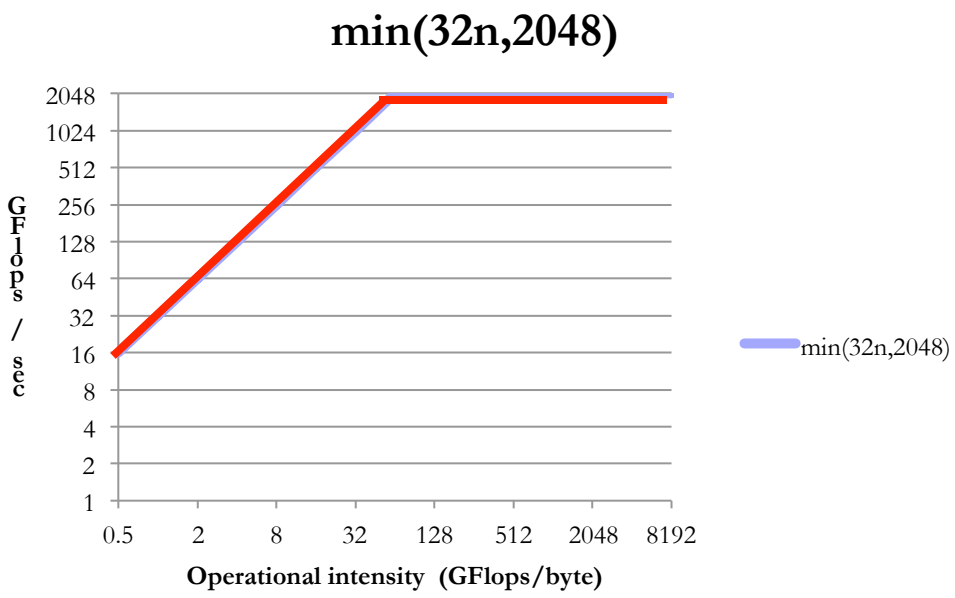
Improving cache performance

- Merging Arrays: Improve spatial locality by single array of structs vs. parallel arrays.
- Loop Interchange: Change nesting of loops to access data in the order stored in memory.
- Loop Fusion: Combine 2 or more independent loops that have the same looping and some variables overlap.
- Blocking or "tiling" : Improve temporal locality by accessing "blocks" of data repeatedly vs. going down whole columns or rows.

Roofline Model

- Architectural model, based on intuition that **off-chip memory bandwidth** is the constraining resource.
- **Operational Intensity**: flops per byte of memory traffic, i.e. bytes exchanged between cache(s) and memory.
- Roofline plots Gflops/sec as a function of Gflops/byte on a log log scale
  - Polynomia become straight lines
    - y intersect: multiplicative factor
    - slope: exponent → linear: 45^0 slope

Typical Roofline Plot

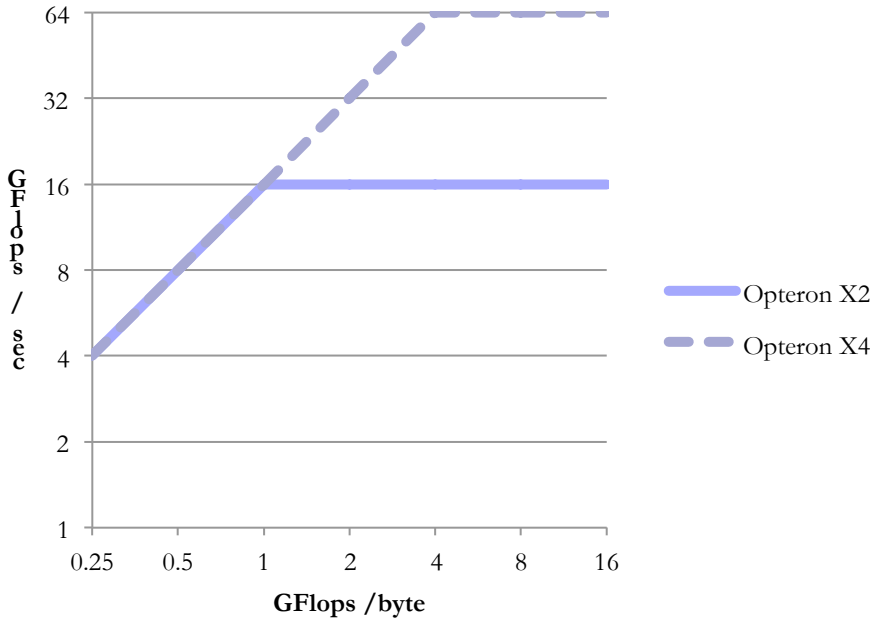


Low Operational Intensity:
- very few Flops per byte
- memory bandwidth is limiting factor
- linear slope behavior

High Operational Intensity:
- many Flops per byte
- machine peak Flop rate is limiting factor
- constant performance

ridge point, where slope meets horizontal:
minimum operational intensity to get maximal performance

Example: Opteron X2 vs. Opteron X4



- Both in same socket, so same memory behavior
- X4: 4x higher Gflops rate

  double # cores

  double peak performance / core
- 4X higher roofline, but only advantageous when there is enough work per byte accessed. Low operational intensity programs do not benefit.

Adding ceilings to roofline

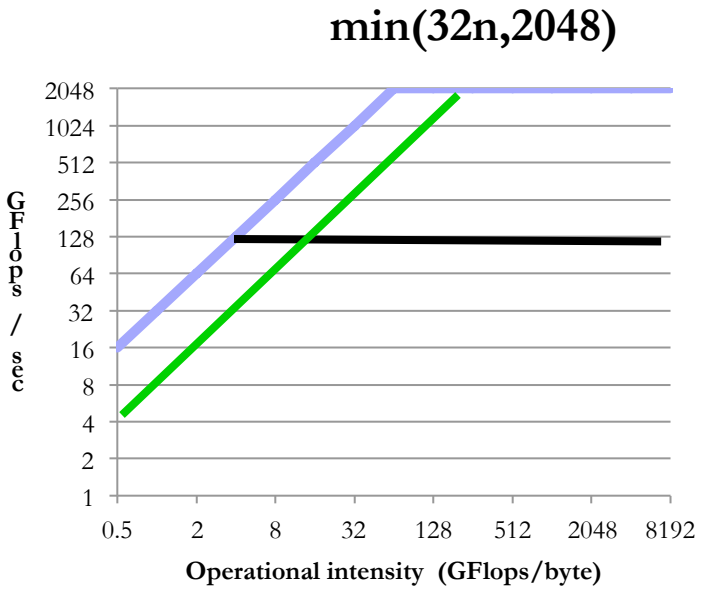
- Roofline gives upper bound on performance, achieved only if the program you run can exploit all architectural phenomena.
- Without some optimizations, only a lower ceiling can be reached

Reducing computational bottlenecks

- Improve ILP
  - Better ILP covers the functional units of the machine better.
  - Can e.g. be achieved by loop unrolling, or applying SIMD (e.g. SSE instructions on Intel machines)
- Balance Flop mix (add, multiply)
  - many machine have multiply-add units (inner product)
  - or equal number of add and multiply units

Reducing memory bottlenecks

- Restructure loops for unit stride access (cache, hardware prefetching)
- Ensure **memory affinity**
  - some memory banks are closer to one core, some are closer to another cored, so allocate threads and their data to a core / memory pair
- Software prefetching can outperform hardware prefetching, e.g., in case of irregular memory access patterns



balance

ILP

With perfect flop balance you can reach this line

Without good ILP, you cannot get above this line

Similar lower slope ceilings for memory
e.g.unit stride

Roofline and cache

- Operational intensity can vary with problem size (e.g. matrix multiply, FFT) because of data reuse and hence better cache behavior, providing a shift right on the roofline. **By doing flops better you go faster**
- Also, we can exchange computation, and thus operational intensity, for memory access (table lookup) and shift left on the roofline

  By doing fewer flops you can go faster

- Paper now discusses 4 kernels on 4 architectures.