Performance Modeling
Study Case: The Polynomial Computation

Erik Saule
esaule@uncc.edu

ITCS 4182/5182: Intro to High Performance Computing

Outline

1 Generalities

2 Polynomial Expansion

3 Further

Purpose

Gain Understanding

- How do you know performance is good?
- How do you know performance is bad?
- How do you know what is the bottleneck of the computation?
- Can I get more performance by rewritting my code?
- Should I buy a different machine?

Understand the machine

How many flops can it do?
How fast is the disk?

Understand the problem

Will it stress the cores?
Will stress the memory?
How much memory moves?
Are there constraints on memory movements?

How does they relate?

Provided what I know about the problem, how do is the machine going to be stressed by the calculation?

The performance of the machine

Core level effects

- maximum flops
- dependency between instructions
- register capacity
- clockspeed varies with number of cores
- performance will depend on instruction mix
- power limitation of embedded systems

Memory level effects

- latency
- bandwidth
- access patterns
- how many cores to saturate bandwidth
- false sharing
- numa

Network level issues

Filesystem level issues

Accelerator level issues

(later)

The needs of the problem

Basic analysis

- How much data does it need?
- Where can the data be stored?
- What kind of access pattern will the problem generate?
- How much computation is there?

Choosing a metric

Time is a terrible metric. It is very hard to interpret.
Metrics that look more like throughput are typically prefered:

- flop/s
- GB/s
- edges traversed/s
- query/s

The Model-Code-Benchmark-Analyze loop

Model

You know the important operations.
You know how fast the different components go.
Make a runtime prediction and extract performance predictions.

Code

You know what the bottleneck is going to be, write Code with that in mind.
You can probably even ignore what the Model says is not important.

Benchmark

The Model tells you what properties (size, structure) of the problem are important. Make multiple instances of the problem that maps to the important properties.
Run the Code (or all Codes, parameters) on ALL instances.

Analyze

Compare the results of the benchmark to the Model.
Does the performance match expectation?
Why doesn't it match? Are there strange effects?

And remember

You are going to be wrong!
You are always forgetting something.
The analysis should highlight what is wrong.
Use the Analysis to start the loop again.
Did you learn something about the machine you did not know?
Did you learn something about the problem you did not think was important?

Outline

1 Generalities

2 Polynomial Expansion

3 Further

Problem

Compute a polynomial function F of degree d:

$$F(x) = \sum_{i=0}^{d} a_i x^i$$

for an array of n elements.

Characteristics

Instructions

Mostly arithmetic instructions. n values to compute. $d+1$ additions and $d+1$ multiplications twice (one to compute powers of x, one to apply coefficient).

$$flops = n * 3 * (d+1)$$

Memory

One array to store the $d+1$ coefficients of the polynomial to read
One array to store the n values to compute the function of to read and to write to store the results.

$$read = (d+1+n) * datatype$$

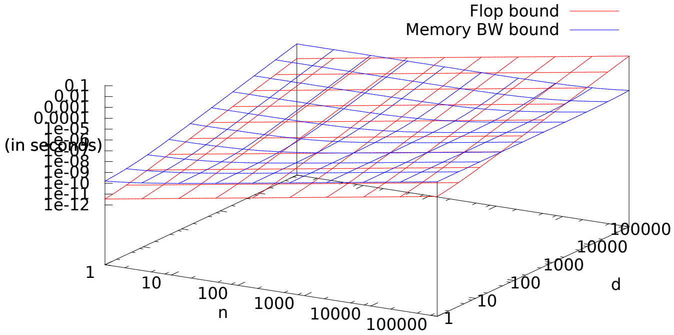
$$write = (n) * datatype$$

Time prediction

Assuming Single Precision and mamba (1.7Tflop/s and 100GB/s).

$$floptime = \frac{flops}{peakflop} = \frac{flops}{1.7 * 10^{12}}$$

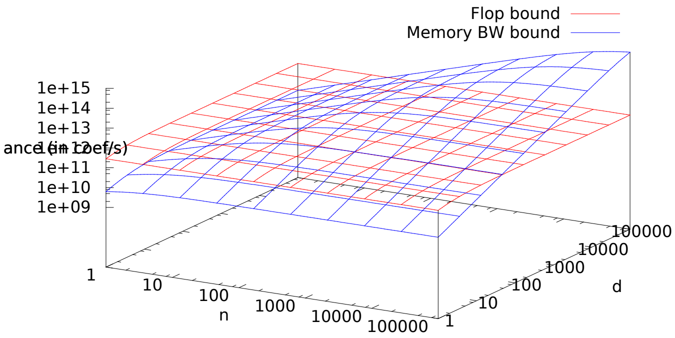
$$memorytime = \frac{read + write}{peakBW} = \frac{read + write}{100 * 10^{9}}$$



Performance

Since the complexity is $\theta(nd)$. We can measure the performance in coefficient computed per second.

$$flopperf = \frac{n(d+1)}{floptime}; memperf = \frac{n(d+1)}{memorytime}$$



What should bound the computation?

$$\textit{floptime} < \qquad\qquad \textit{memorytime}$$

What should bound the computation?

$$\begin{aligned}
floptime &< memorytime \\
\frac{flops}{peakF} &< \frac{read + write}{peakBW} \\
peakBW * flops &< peakF(read + write) \\
n * 3 * (d+1) peakBW &< peakF(d + 1 + 2n)DT \\
n * 3 * (d+1) peakBW - 2 * n * DT * peakF &< (d+1) peakF * DT \\
n * (3 * (d+1) peakBW - 2 * DT * peakF) &< (d+1) peakF * DT
\end{aligned}$$

What should bound the computation?

$$\begin{aligned}
floptime &< memorytime \\
\frac{flops}{peakF} &< \frac{read + write}{peakBW} \\
peakBW * flops &< peakF(read + write) \\
n * 3 * (d+1)peakBW &< peakF(d + 1 + 2n)DT \\
n * 3 * (d+1)peakBW - 2 * n * DT * peakF &< (d+1)peakF * DT \\
n * (3 * (d+1)peakBW - 2 * DT * peakF) &< (d+1)peakF * DT
\end{aligned}$$

Case 1

True when $3 * (d+1)peakBW - 2 * DT * peakF < 0$.
On mamba, $3 * (d+1)10^{11} - 4 * 2 * 1.7 * 10^{12} < 0$
$3 * (d+1) - 4 * 2 * 1.7 * 10 < 0$
i.e., $d < 45.3$, so $d \leq 45$

What should bound the computation? (continued)

Case 2

True when $3 * (d+1) peakBW - 2 * DT * peakF > 0$, and

$$n * (3 * (d+1) peakBW - 2 * DT * peakF) < \quad (d+1) peakF * DT$$

$$\begin{aligned}
n &< \frac{(d+1) peakF * DT}{3 * (d+1) peakBW - 2 * DT * peakF} \\
n &< \frac{(d+1) 4 * 1.7 * 10^{12}}{3 * (d+1) 100 * 10^{9} - 2 * 4 * 1.7 * 10^{12}} \\
n &< \frac{(d+1) 6.8 * 10^{12}}{(d+1) 3 * 10^{11} - 13.6 * 10^{12}} \\
n &< \frac{(d+1) 68}{3(d+1) - 136}
\end{aligned}$$

What should bound the computation? (Summary)

On mamba, the computation is memory bound:

- when $d \leq 45$
- and when $d \geq 46$ and $n < \frac{68(d+1)}{3(d+1)-136}$ (i.e., for tiny n)

Summary

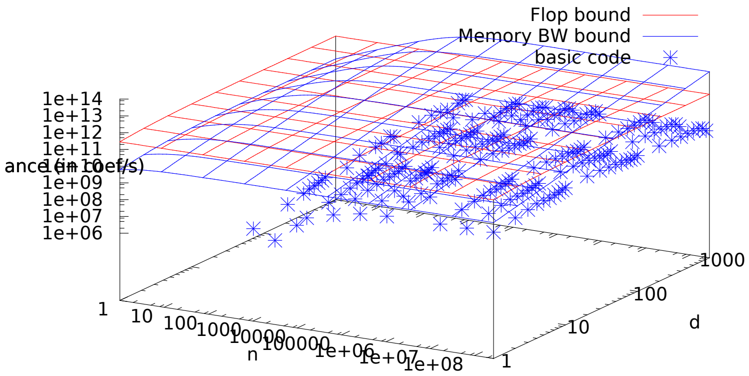
Analyzing performance is useful for multiple reasons:

- gives you an off-the-bat idea of where the problems are in the application
  - similar to complexity analysis
- enables a prediction of runtime before writing a single line of code
  - While preparing this talk, I made a mistake in the code resulting in the polynomial not being computed. I caught it because of the discrepancy in runtime
- provide a expectation of performance (here coefficient per second) that is easy to remember and scale independent
- enable to predict if the computation is compute-bound or memory-bound
  - which indicates on what to spend time when writing code
  - which indicates cases they are compute- or memory- bound
  - which provides interesting region of the parameter space to benchmark

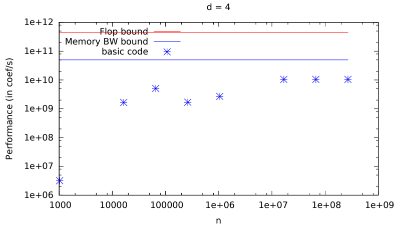
Code

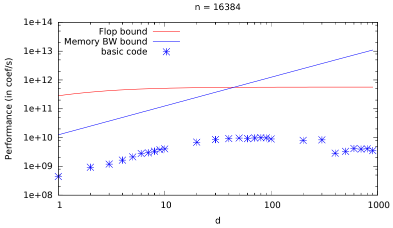
```
float polynomial (float x, float* poly, int degree) {
  float out = 0.;
  float xtothepowerof = 1.;
  for (int i=0; i<=degree; ++i) {
    out += xtothepowerof*poly[i];
    xtothepowerof *= x;
  }
  return out;
}

void polynomial_expansion (float* poly, int degree,
                           int n, float* array) {

#pragma omp parallel for schedule(runtime)
  for (int i=0; i< n; ++i) {
    array[i] = polynomial (array[i], poly, degree);
  }
}
```

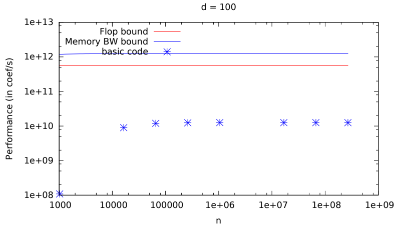
Performance measured

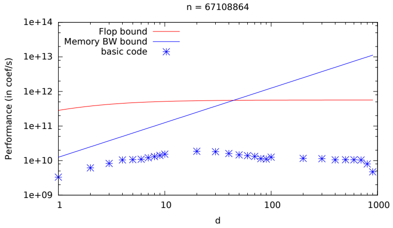


Some planes

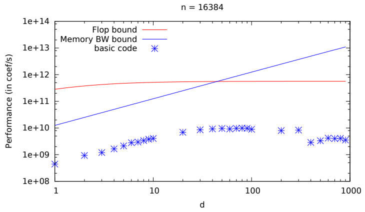

d = 4
Flop bound
Memory BW bound
basic code
Performance (in coef/s)
1e+06
1e+07
1e+08
1e+09
1e+10
1e+11
1e+12
1000
10000
100000
1e+06
1e+07
1e+08
1e+09
n


n = 16384
Flop bound
Memory BW bound
basic code
Performance (in coef/s)
1e+08
1e+09
1e+10
1e+11
1e+12
1e+13
1e+14
1
10
100
1000
d


d = 100
Flop bound
Memory BW bound
basic code
Performance (in coef/s)
1e+08
1e+09
1e+10
1e+11
1e+12
1e+13
1000
10000
100000
1e+06
1e+07
1e+08
1e+09
n


n = 67108864
Flop bound
Memory BW bound
basic code
Performance (in coef/s)
1e+09
1e+10
1e+11
1e+12
1e+13
1e+14
1
10
100
1000
d

Analysis



What can we learn?

The chart has the right shape, but 50 times lower than expected.
Is the model wrong? Or is the code wrong?
The model assumed 1.7Tflop/s. To get there we need to do 2 FMA per clock cycle.
The problem has twice more multiplication than addition. The 1.7Tflop/s is unrealistic. At most one mul and one FMA per cycle.
To get there, we need vector processing. The code does not.
At low *d*, we need to saturate the memory subsystem, AVX will help.

Outline

1. Generalities
2. Polynomial Expansion
3. Further

Resources

General:

- Jain. The art of computer systems performance analysis. 1991.
- Roofline model (talk) `https://crd.lbl.gov/assets/pubs_presos/hotchips08-roofline-talk.pdf`
- Roofline model (wikipedia) `https://en.wikipedia.org/wiki/Roofline_model`

Some Use Case:

- G. Erlebacher, E. Saule, N. Flyer, and E. Bollig. Acceleration of derivative calculations with application to radial basis function - finite-differences on the Intel MIC architecture. In ICS, 2014.
- P. Guo, L. Wang, and P. Chen. A performance modeling and optimization analysis tool for sparse matrix-vector multiplication on gpus. IEEE Transactions on Parallel and Distributed Systems, 25(5):1112-1123, 2014.
- K. Li, W. Yang, and K. Li. Performance analysis and optimization for spmv on gpu using probabilistic modeling. IEEE Transactions on Parallel and Distributed Systems, 26(1):196-205, Jan 2015.
- S. Beamer, K. Asanovic, and D. Patterson. Locality exists in graph processing: Workload characterization on an ivy bridge server. In Proceedings of the 2015 IEEE International Symposium on Workload Characterization, IISWC 15, pages 56-65, Washington, DC, USA, 2015. IEEE Computer Society.
- D. Culler, R. Karp, D. Patterson, A Sahay, K. Schauser, E. Santos, R. Subramonian, R. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. SIGPLAN 1993.
- Erik Saule and Ümit V. Çatalyürek. An early evaluation of the scalability of graph algorithms on the Intel MIC architecture. In 26th International Symposium on Parallel and Distributed Processing, Workshops and PhD Forum (IPDPSW), Workshop on Multithreaded Architectures and Applications (MTAAP), 2012.