How to Write Fast Numerical Code
Spring 2019

Lecture: Cost analysis and performance

Instructor: Markus Püschel
TA: Tyler Smith, Gagandeep Singh, Alen Stojanov

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zürich

Technicalities

- Research project: Let us know (fastcode@lists.inf.ethz.ch)
  - if you know with whom you will work
  - if you have already a project idea
  - current status: on the web
  - Deadline: March 4th

- If you need partner: fastcode-forum@lists.inf.ethz.ch
- If you need partner and project: fastcode-forum@lists.inf.ethz.ch
Matrix-Matrix Multiplication (MMM) on 2 x Core 2 Duo 3 GHz
Performance [Gflop/s]

- Multiple threads: 4x
- Vector instructions: 4x
- Memory hierarchy: 20x

- Compiler doesn’t do the job
- Doing by hand: \textit{nightmare}

Performance is different than other software quality features
Today

- Problem and Algorithm
- Asymptotic analysis
- Cost analysis


Problem

- Problem: Specification of the relationship between a given input and a desired output
- Numerical problem (this course): In- and output are numbers (or lists, vectors, arrays, ... of numbers)
- Examples
  - Compute the discrete Fourier transform of a given vector x of length n
  - Matrix-matrix multiplication (MMM)
  - Compress an n x n image with a ratio ...
  - Sort a given list of integers
  - Multiply by 5, y = 5x, using only additions and shifts
Algorithm

- **Algorithm**: A precise description of a sequence of steps to solve a given problem
- **Numerical algorithm**: Dominated by arithmetic (adds, multis, ...)
- **Examples**:
  - Cooley-Tukey fast Fourier transform (FFT)
  - A description of MMM by definition
  - JPEG encoding
  - Mergesort
  - $y = x \ll 2 + x$

Reminder: Do You Know The O?

- $O(f(n))$ is a ... ?
  - set
- How are these related? $\Theta(f(n)) = \Omega(f(n)) \cap O(f(n))$
  - $O(f(n))$
  - $\Theta(f(n))$
  - $\Omega(f(f(n)))$
- $O(2^n) = O(3^n)$?
  - no
- $O(\log_2(n)) = O(\log_3(n))$
  - yes
- $O(n^2 + m) = O(n^2)$?
  - no
Always Use Canonical Expressions

- Example:
  - *not* $O(2n + \log(n))$, *but* $O(n)$

- Canonical? If not replace:
  - $O(100)$
  - $O(\log_2(n))$
  - $\Theta(n^{1.1} + n \log(n))$
  - $2n + O(\log(n))$
  - $O(2n) + \log(n)$
  - $\Omega(n \log(m) + m \log(n))$

Asymptotic Analysis of Algorithms

- Analysis for
  - Runtime
  - Space (= memory footprint)
  - Data movement (e.g., between cache and memory)

- Example MMM: $C = A*B + C$, $A,B,C$ are all $n \times n$
  - Runtime: $O(n^3)$
  - Space: $O(n^2)$
Valid?

- Is asymptotic analysis still valid given this?

![Matrix-Matrix Multiplication (MMM) on 2 x Core 2 Duo 3 GHz](image)

All algorithms are $O(n^3)$ when counting flops.

**What happens to asymptotics if I take memory accesses into account?**
No problem: $O(f(n))$ flops means at most $O(f(n))$ memory accesses

**What happens if I take vectorization/parallelization into account?**
More parameters needed: E.g., $O(n^3/p)$ on $p$ processors

Asymptotic Analysis: Limitations

- $\Theta(f(n))$ describes only the *eventual trend* of the runtime

![Graphical representation](image)

- Constants matter
  - Not clear when “eventual” starts
  - $n^2$ is likely better than $1000n^2$
  - $100000000000n$ is likely worse than $n^2$
Cost Analysis for Numerical Problems

- **Goal:** determine exact “cost” of an algorithm
- **Cost:** number of relevant operations
- **Formally:** define *cost measure* \( C(n) \). Examples:
  - Counting adds and mults separately: \( C(n) = (\text{adds}(n), \text{mults}(n)) \)
  - Counting adds, mults, divs separately: \( C(n) = (\text{adds}(n), \text{mults}(n), \text{divs}(n)) \)
  - Counting all flops together: \( C(n) = \text{flops}(n) \)
- **This course:** focus on floating point operations

The cost measure should count only the operations that constitute the mathematical algorithm (e.g., as written on paper) and not operations that arise due to its mapping on a computer (e.g., index computations, data movement)

Example

/* Multiply \( n \times n \) matrices \( a \) and \( b \) */
void mmm(double *a, double *b, double *c, int n) {
  int i, j, k;
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      for (k = 0; k < n; k++)
        c[i*n+j] += a[i*n + k]*b[k*n + j];
}

- **Asymptotic runtime**
  - \( O(n^3) \)
- **Cost measure?**
  - \( C(n) = (\text{fladds}(n), \text{fimults}(n)) = (n^3, n^3) \)
  - \( C(n) = \text{flops}(n) = 2n^3 \)
Cost Analysis: How To Do

- Define suitable cost measure
- Count in algorithm or code
  - Recursive function: solve recurrence
- Instrument code
- Use performance counters (maybe in a later lecture)
  - Intel PCM
  - Intel Vtune
  - Perfmon (open source)
  - Counters for floating points are recently less and less available

Remember: Even Exact Cost ≠ Runtime
Why Cost Analysis?

- Enables performance analysis:

\[
\text{performance} = \frac{\text{cost}}{\text{runtime}} \quad \text{[flops/cycle] or [flops/sec]}
\]

- Upper bound through machine’s peak performance

![Peak performance of this computer](image)

90% of peak performance

Example

```c
/* Matrix-vector multiplication y = Ax + y */
void mmm(double *A, double *x, double *y, int n) {
    int i, j, k;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            y[i] += A[i*n + j]*x[j];
}
```

- Flops? For \( n = 10 \)
  - \( 2n^2 \), 200

- Performance for \( n = 10 \) if runs in 400 cycles
  - 0.5 flops/cycle

- Assume peak performance: 2 flops/cycle percentage peak?
  - 25%
Summary

- Asymptotic runtime gives only an idea of the runtime trend
- Exact number of operations (cost):
  - Also no good indicator of runtime
  - But enables performance analysis
- Always measure performance (if possible)
  - Gives idea of efficiency
  - Gives percentage of peak