How to Write Fast Numerical Code

Spring 2016

*Lecture:* Cost analysis and performance

**Instructor:** Markus Püschel
**TA:** Gagandeep Singh, Daniele Spampinato & Alen Stojanov

**Technicalities**

- **Research project:** Let us know ([fastcode@lists.inf.ethz.ch](mailto: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 7th*

- **If you need partner:** [fastcode-forum@lists.inf.ethz.ch](mailto:fastcode-forum@lists.inf.ethz.ch)

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

- Compiler doesn’t do the job
- Doing by hand: 

```
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, mults, ...)
- **Examples:**
  - Cooley-Tukey fast Fourier transform (FFT)
  - A description of MMM by definition
  - JPEG encoding
  - Mergesort
  - $y = x << 2 + x$

**Reminder: Do You Know The O?**

- $O(f(n))$ is a ... ?  
  set
- How are these related?  
  - $O(f(n))$
  - $\Theta(f(n))$
  - $\Omega(f(n))$
- $O(2^n) = O(3^n)$?  
  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 & Problems

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

- **Asymptotic runtime of an algorithm:**
  - Count “elementary” steps
    - *numerical algorithms*: usually floating point operations
  - Result in \( O \)-notation
  - Example MMM (square and rectangular): \( C = A*B + C \)

- **Runtime complexity of a problem** =
  - Minimum of the runtimes of all possible algorithms
  - Result also stated in asymptotic \( O \)-notation

*Complexity is a property of a problem, not of an algorithm*
Valid?

- Is asymptotic analysis still valid given this?

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

- Constants matter
  - Not clear when “eventual” starts
  - $n^2$ is likely better than $1000n^2$
  - $10000000000n$ 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: focusing on floating point operations

---

Example

```c
/* Multiply n x 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?**
  - $C(n) = (\text{fladds}(n), \text{flmults}(n)) = (n^3, n^3)$
  - $C(n) = \text{flops}(n) = 2n^3$
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

Matrix-Matrix Multiplication (MMM) on 2 x Core 2 Duo 3 GHz

Performance [Gflop/s]

2n^3 flops
Why Cost Analysis?

- Enables performance analysis:
  \[ \text{performance} = \frac{\text{cost}}{\text{runtime}} \text{ [flops/cycle] or [flops/sec]} \]

- Upper bound through machine’s peak performance

![Peak performance graph](image)

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 \textit{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