

18.335 Fall 2008  
Performance Experiments  
with Matrix Multiplication

Steven G. Johnson

Hardware: 2.66GHz Intel Core 2 Duo  
64-bit mode, double precision, gcc 4.1.2

optimized BLAS dgemm: ATLAS 3.6.0

<http://math-atlas.sourceforge.net>

# A trivial problem?

$$\begin{matrix} C & = & A & B \\ m \times p & & m \times n & n \times p \end{matrix}$$

the “obvious” C code:

```
/* C = A B, where A is m x n, B is n x p,
   and C is m x p, in row-major order */
void matmul(const double *A, const double *B,
            double *C, int m, int n, int p)
{
    int i, j, k;
    for (i = 0; i < m; ++i)
        for (j = 0; j < p; ++j) {
            double sum = 0;
            for (k = 0; k < n; ++k)
                sum += A[i*n + k] * B[k*p + j];
            C[i*p + j] = sum;
        }
}
```

**for  $i = 1$  to  $m$**

**for  $j = 1$  to  $p$**

$$C_{ij} = \sum_{k=1}^n A_{ik} B_{kj}$$

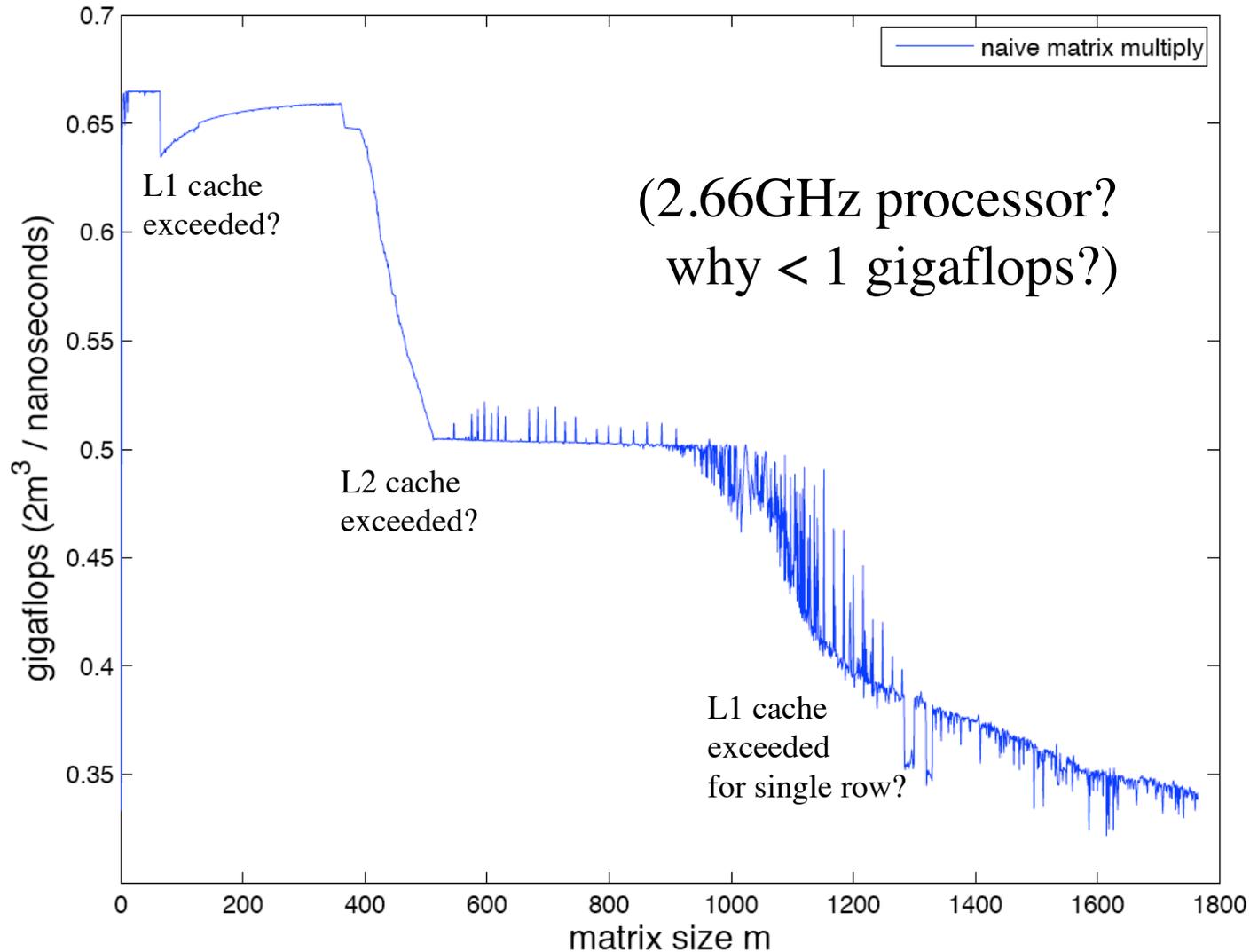
**$2mnp$  flops**

**(adds+mults)**

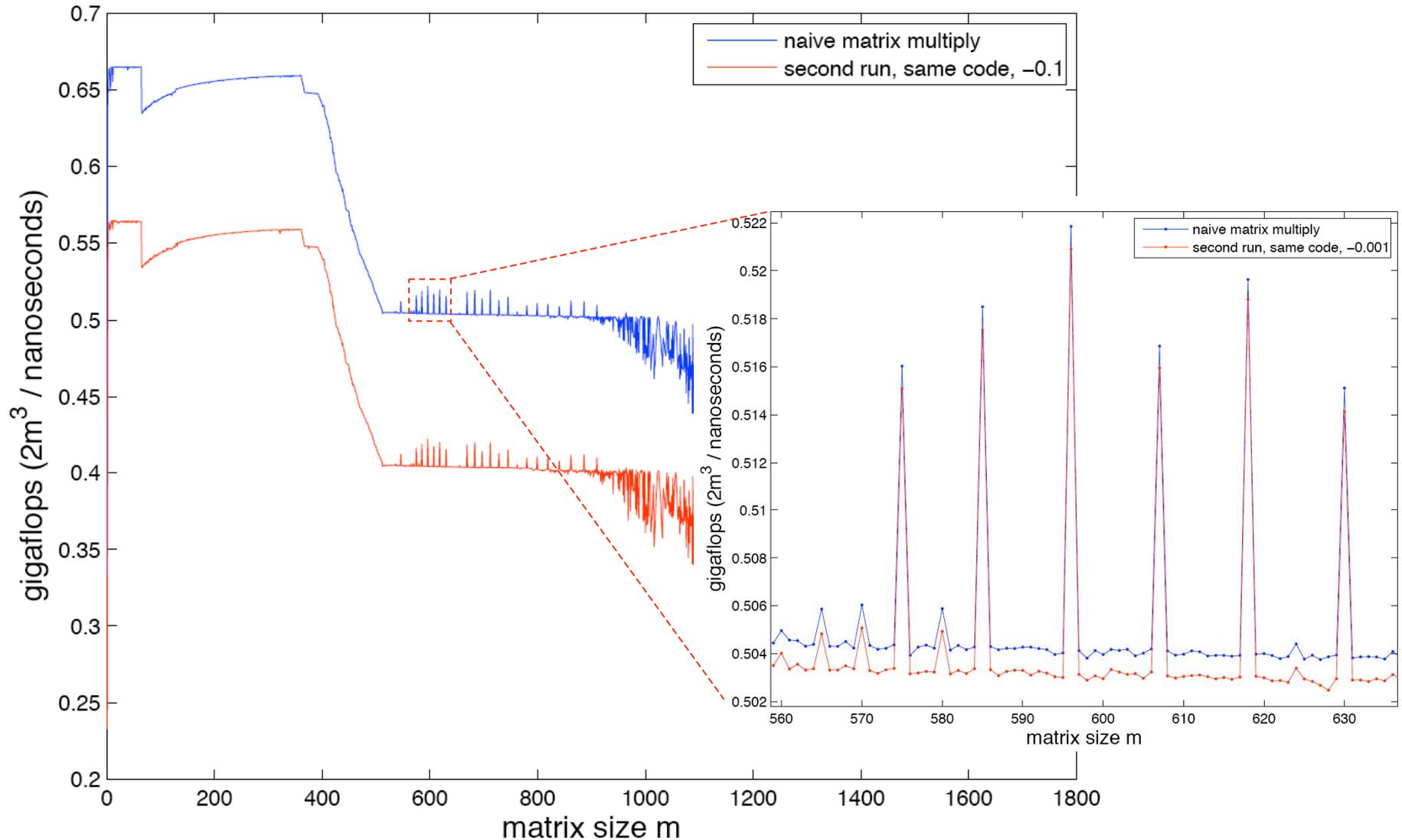
just three loops, how complicated can it get?

# flops/time is not constant!

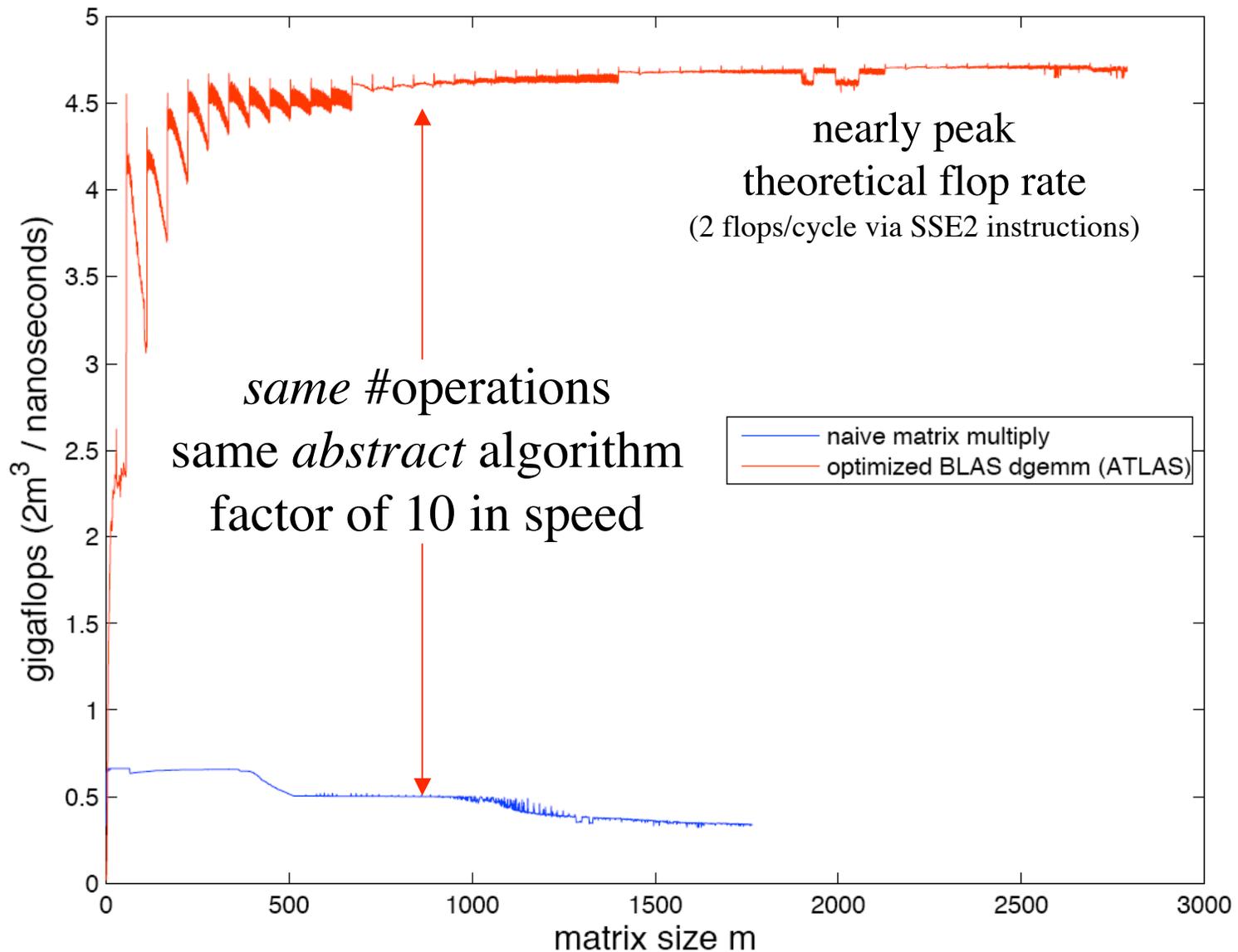
(square matrices,  $m=n=p$ )



# Not all “noise” is random



# All flops are not created equal



# Things to remember

- We **cannot understand performance without understanding memory** efficiency (caches).
  - ~10 times more important than arithmetic count
- Computers are **more complicated than you think**.
- Even a trivial algorithm is nontrivial to implement *well*.
  - matrix multiplication: 10 lines of code → **130,000+** (ATLAS)

MIT OpenCourseWare  
<https://ocw.mit.edu>

18.335J Introduction to Numerical Methods  
Spring 2019

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>.