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?

\[ C = A \times B \]

the “obvious” C code:

```c
#include <stdio.h>

#define ADDS log2(n) + 1
#define MULTS log2(p) + 1

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;
        }
}
```

just three loops, how complicated can it get?

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

\[2mnp\text{ flops}
\]

(adds+mults)
flops/time is not constant!

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

(2.66GHz processor? why < 1 gigaflops?)
Not all “noise” is random
All flops are not created equal

same \#operations
same *abstract* algorithm
factor of 10 in speed

nearly peak theoretical flop rate
(2 flops/cycle via SSE2 instructions)
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)