Comparing performace of different implementations of matrix multiplication

Last modified: October 20, 2015

The following function implements multiplication of two $n \times n$ matrices:

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

```
 * Naive implementation of matrix multiplication c += ab,
 * where a is m x n, b is n x p, and c is m x p, in column-major order.
 *
 * The physical sizes of a, b, and c are lda x n, ldb x p, and ldc x p,
 * but only the first m/n/m rows are used, respectively.
 *
 *   c_{ij} += \sum_{k=0}^{n-1} a_{ik} b_{kj}
 *
 *   a_{ij} <-> a[i + j*lda] <- column-major order
 */
void matmul_naive (const double *a, const double *b, double *c, const int m,
    const int n, const int p, const int lda, const int ldb, const int ldc)
{
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < p; j++)
        {
            double sum = 0.0;

            for (int k = 0; k < n; k++)
            {
                sum += a[i + lda * k] * b[k + ldb * j];
            }
            c[i + ldc * j] += sum;
        }
    }
}
```

The graph in Fig. 1 compares the CPU performance, in MFLOPS, vs. matrix size n achieved during matrix multiplication by two different implementations - the naive one shown above and the state of the art implementation in OpenBLAS library.

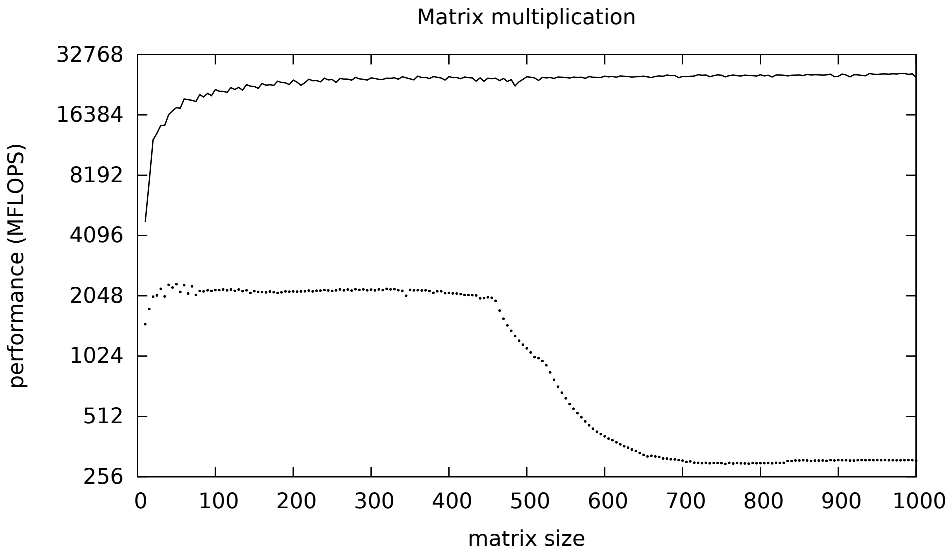

Figure 1: Perforamnce of two implementations of matrix multiplication: the code above (dotted line) and `dgemm` function from the OpenBLAS library (solid line). Notice the logarithmic vertical axis.

```
set term pdf mono lw .5
set o "performance.pdf"

unset key
set title "Matrix multiplication"
set xlabel "matrix size"
set ylabel "performance (MFLOPS)"
set logscale y 2

p [:] [:] "naive.res" u 1:2 w d, "dgemm.res" u 1:2 w l
```

Figure 2: Gnuplot script that produced Figure 1.