#include <stdio.h>

double powern (double d, unsigned n)

{

double x = 1.0;

unsigned j;

for (j = 1; j <= n; j++) x \*= d;

return x;

}

int main (void)

{

double sum = 0.0;

unsigned i;

for (i = 1; i <= 100000000; i++) sum += powern (i, i % 5);

printf ("sum = %g\n", sum);

return 0;

}

$ gcc -Wall -O0 test.c -lm

$ time ./a.out

real 0m13.388s

user 0m13.370s

sys 0m0.010s

$ gcc -Wall -O1 test.c -lm

$ time ./a.out

real 0m10.030s

user 0m10.030s

sys 0m0.000s

$ gcc -Wall -O2 test.c -lm

$ time ./a.out

real 0m8.388s

user 0m8.380s

sys 0m0.000s

$ gcc -Wall -O3 test.c -lm

$ time ./a.out

real 0m6.742s

user 0m6.730s

sys 0m0.000s

$ gcc -Wall -O3 -funroll-loops test.c -lm

$ time ./a.out

real 0m5.412s

user 0m5.390s

sys 0m0.000s

The main program contains a loop calling the powern function. This function computes the n-th power of a floating point number by repeated multiplication--it has been chosen because it is suitable for both inlining and loop-unrolling. The run-time of the program can be measured using the time command in the GNU Bash shell.

The relevant entry in the output for comparing the speed of the resulting executables is the 'user' time, which gives the actual CPU time spent running the process. The other rows, 'real' and 'sys', record the total real time for the process to run (including times where other processes were using the CPU) and the time spent waiting for operating system calls. Although only one run is shown for each case above, the benchmarks were executed several times to confirm the results.

From the results it can be seen in this case that increasing the optimization level with -O1, -O2 and -O3 produces an increasing speedup, relative to the unoptimized code compiled with -O0. The additional option -funroll-loops produces a further speedup. The speed of the program is more than doubled overall, when going from unoptimized code to the highest level of optimization.

Note that for a small program such as this there can be considerable variation between systems and compiler versions. For example, on a Mobile 2.0GHz Intel Pentium 4M system the trend of the results using the same version of GCC is similar except that the performance with -O2 is slightly worse than with -O1. This illustrates an important point: optimizations may not necessarily make a program faster in every case.