Michel Gonzalez

Mag9989

SDS 320E

Compiler Testing and optimization

I began by compiling rotate.c through each optimization level (-O0, -O1, -O2, -O3) through the icc and gcc compilers. The following code snippets show the time it took to run rotate.c at each optimization level for each compiler.

```
login1.frontera(1072)$ icc -O0 -o main rotate.c
login1.frontera(1073)$ main
Done after 2.857640e-01
```

```
login1.frontera(1100)$ gcc -O0 -o main rotate.c -lm
login1.frontera(1101)$ main
Done after 4.341660e-01
```

```
login1.frontera(1075)$ icc -O1 -o main rotate.c
login1.frontera(1076)$ main
Done after 2.698841e-01
```

```
login1.frontera(1102)$ gcc -O1 -o main rotate.c -lm
login1.frontera(1103)$ main
Done after 2.640519e-01
```

```
login1.frontera(1093)$ icc -O2 -o main rotate.c
login1.frontera(1094)$ main
Done after 2.145767e-06
```

```
login1.frontera(1106)$ gcc -O2 -o main rotate.c -lm
login1.frontera(1107)$ main
Done after 2.889199e-01
```

```
login1.frontera(1089)$ icc -O3 -o main rotate.c
login1.frontera(1090)$ main
Done after 9.536743e-07
```

```
login1.frontera(1111)$ gcc -O3 -o main rotate.c -lm
login1.frontera(1112)$ main
Done after 9.536743e-07
```

The results show that optimization levels -O2 and -O3 enhance the efficiency of the program much more than -O0 or -O1. However, the intel compiler tends to optimize/preform better than the gcc compiler, specially the -O2 optimization. We can investigate what each optimization level does and attempt to replicate those modifications to the program and see if we get similar output times.

The -O0 optimizer has no optimization. It generates the unoptimized code but has the fastest compilation time, which is the default when compiling. The -O1 gives a moderate optimization. It optimizes reasonably well but does not degrade compilation time significantly. The -O2 optimization performs nearly all supported optimizations that do not involve a space-speed tradeoff. The -O3 optimization is a full optimization similar to the -O2 optimization, however it uses more aggressive automatic inlining of subprograms within a unit and attempts to vectorize loops. It also turns on the -finline-functions, -funswitch-loops, -fpredictive-commoning, -fgcse-after-reload and -ftree-vectorize options. This may be worse than the -O2 optimization if the icc compiler is used due to the aggressive approach, but for the gcc compiler the -O3 optimization may be the best optimizer depending on the code.

```
login2.frontera(1029)$ icc -o main
rotate.c

login2.frontera(1030)$ main

Done after 9.536743e-07
```

```
login2.frontera(1021)$ gcc -o main
rotate.c -lm

login2.frontera(1022)$ main

Done after 4.429779e-01
```

When running the code through each compiler the intel compiler already performs at peak optimization while the gcc compiler runs the code at the -O0 optimization level. I will use the gcc compiler and optimize the code manually and see if we can get similar times to the optimization levels based on each of their transformations.

The first optimization was in the rotate function. Since cos(alpha) and sin(alpha) are constants and they are being recalculated every loop, this can be optimized by calculating those constant values and change the parameters of the rotate function to include those constants.

```
In main():

double t=cos(alpha);

double l=sin(alpha);
```

```
void rotate(double *x, double *y,
double *t, double *l, ) {

  double x0 = *x, y0 = *y;

  *x = *t * x0 - *l * y0;

  *y = *l * x0 + *t * y0;

  return;
```

This resulted in a better output time

```
login2.frontera(1066)$ gcc -o main rotate.c -lm
login2.frontera(1067)$ main
Done after 5.489707e-02
```

Next instead of having rotate be a separate function I inlined it into the main function

```
double x0 = x, y0 = y;
for (int i=0; i<NREPS; i++)
    x = t * x0 - l * y0;
    y = l * x0 + t * y0;
    x0 = x, y0 = y;
```

This gave a faster output since the loop is not having to refer to another location in memory and can calculate the arithmetic in the same space.

```
login2.frontera(1180)$ gcc -o main rotate.c -lm
login2.frontera(1181)$ main
Done after 1.499081e-02
```

Unfortunately, I was unable to lower the time anymore after trying various optimizing methods that applied to this code such as loop unrolling, inlining, subexpression elimination, and constant propagation.