Project Report: Optimizing the Spherical Geometry Code

a. Accuracy table for approximations

Original code

Test for USA:

Approximation Used	# of Distance Errors	# of Bearing Errors	# of Waypoint Name Errors
cos_32	283	136	1
cos_52	57	1	0
cos_73	31	0	0
cos_121	0	0	0

Test for non-polar regions:

Approximation Used	# of Distance Errors	# of Bearing Errors	# of Waypoint Name Errors
cos_32	1024	286	6
cos_52	94	2	0
cos_73	37	0	0
cos_121	0	0	0

Optimized code

Test for USA:

Approximation Used	# of Distance Errors	# of Bearing Errors	# of Waypoint Name Errors
cos_32	283	136	1
cos_52	56	1	0
cos_73	31	0	0
cos_121	0	0	0

Test for non-polar regions:

Approximation Used	# of Distance Errors	# of Bearing Errors	# of Waypoint Name Errors
cos_32	1024	286	6
cos_52	93	2	0
cos_73	37	0	0
cos_121	0	0	0

b. Run-time performance for original and optimized code

Original Code:

Cosine Function Used	Program Execution Time	Ex. Time for Single Call to Find_Nearest_Waypoint (prev. col. / N_TESTS)	Cycles per point (previous column*1.5GHz/ 163 waypoints)
library	1264807530 ns	12.648 us	116.393331 cycles
cos_32	1691019439 ns	16.910 us	155.615286 cycles
cos_52	1741692396 ns	17.417 us	160.278441 cycles
cos_73	1795512509 ns	17.955 us	165.231212 cycles
cos_121	1962357622 ns	19.624 us	180.585057 cycles

Optimized Code:

Cosine Function Used	Program Execution Time	Ex. Time for Single Call to Find_Nearest_Waypoint (prev. col. / N_TESTS)	Cycles per point (previous column*1.5GHz/ 163 waypoints)
library	698859404 ns	6.989 us	64.312215 cycles
cos_32	231742554 ns	2.317 us	21.326002 cycles
cos_52	255228628 ns	2.552 us	23.487297 cycles
cos_73	339166184 ns	3.392 us	31.211612 cycles
cos_121	403906109 ns	4.039 us	37.169274 cycles

c. Optimization of program using cos_52: A step-by-step explanation of your optimization process, with each step containing these sections:

i. Initial run time and profiles (function and instruction-level)

ii. Explanation of optimization applied, with excerpts of source code

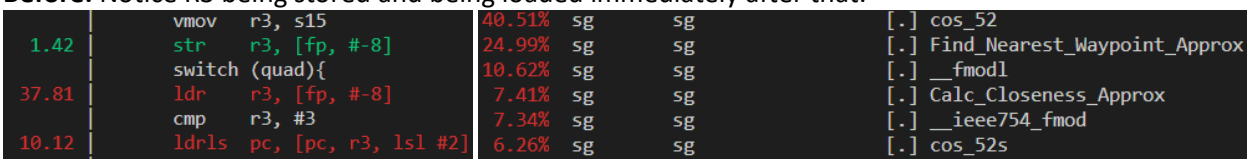
iii. Run time of optimized code

Step 1: *Added Compiler flag for optimization -Og in Makefile.*

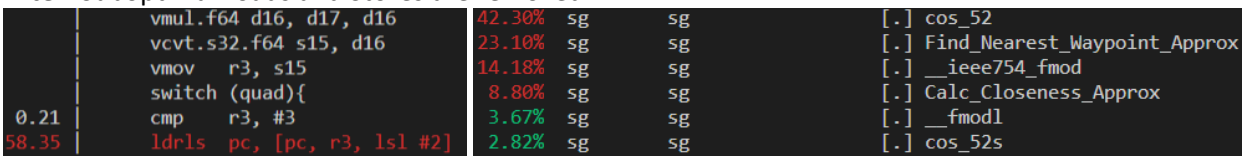
-Og Optimize for debugging experience rather than speed or size.

With default optimization enabled it gets difficult to analyze Perf Annotate file due to suboptimal loads and stores. Applying O3 helps remove loads and stores but at the same time hides a lot of useful debug information. -Og helps remove these loads and stores making the program run faster and at the same time makes debugging experience easier. -Og essentially performs mem2reg promotion pass.

Before: Notice R3 being stored and being loaded immediately after that.



After: Suboptimal Loads and Stores are removed.



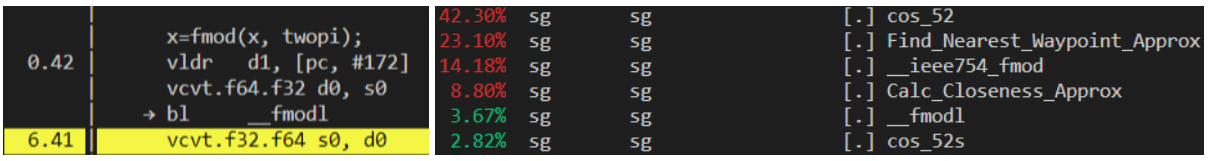
Program Execution Time

Before optimization: 17.208 us
After optimization: 8.639 us

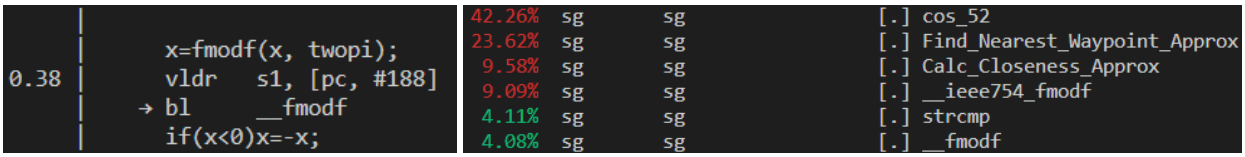
Step 2: Replace fmod with fmodf

Function fmod takes arguments of type double and returns a double. For this the arguments to fmod are converted to double precision and the result achieved gets converted to a single precision value afterwards. To eliminate this conversion use of fmodf is made.

Before:



After: vcvt instruction is removed



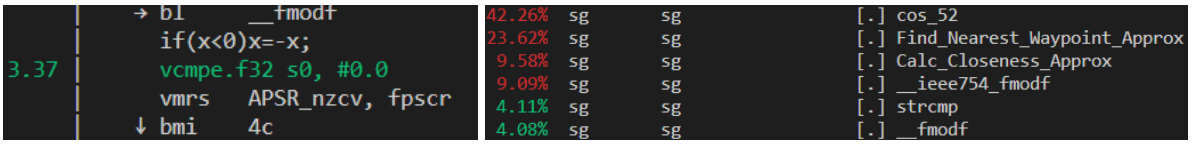
Program Execution Time

Before optimization: 8.639 us
After optimization: 7.698 us

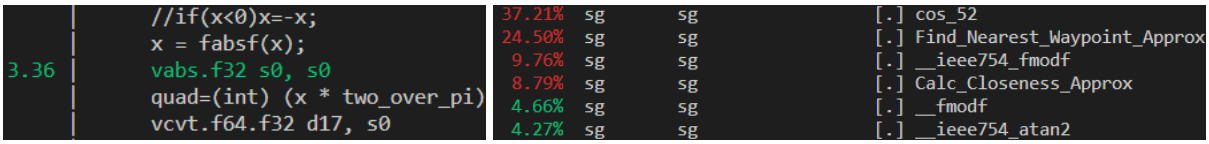
Step 3: Replace if statement with fabsf

fabsf returns absolute value of single precision. This processor has instruction for returning absolute value(vabs) as a result the no. of instructions needed to execute an if statement.

Before:



After:



Program Execution Time

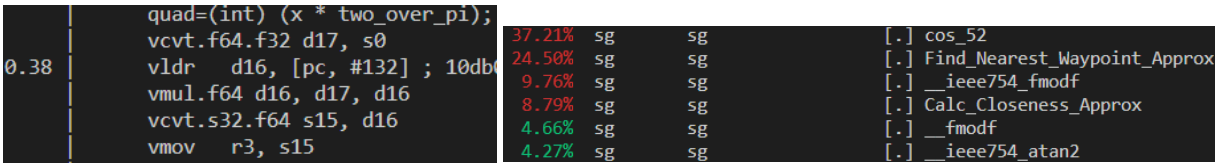
Before optimization: 7.698 us
After optimization: 7.082 us

Step 4: Replaced Double precision math constants with Single precision math constants

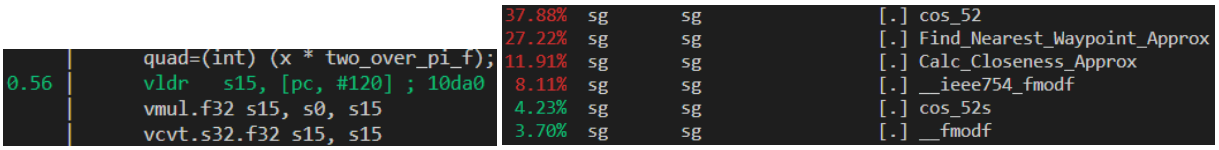
```
// Math constants we'll use
#define DP_PI (3.14159265358979323846264433) // pi
double const twopi=2.0*DP_PI;      // pi times 2
double const two_over_pi= 2.0/DP_PI;     // 2/pi
double const halfpi=DP_PI/2.0;       // pi divided by 2

// Single precision math consatnts
#define SP_PI (3.14159265358979323846264433f)  // pi
float const twopi_f=2.0*SP_PI;       // pi times 2
float const two_over_pi_f= 2.0/SP_PI;    // 2/pi
float const halfpi_f=SP_PI/2.0;      // pi divided by 2
```

Helps to remove conversion required of variables from f32 to f64 before performing any operation on them. Also replaces f64 operations like multiply with f32 versions.

Before: vcvt is required to convert x from f32 to f64 before multiply operation and multiply operation required is of f64 type. An instruction is needed to convert result of multiply f64 to f32.



After: vcvt required to convert x from f32 to f64 before multiply operation is removed. And multiply operation required is converted from f64 to f32 type. Instruction needed to convert multiply to f32 is removed.

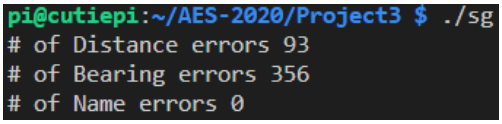


Program Execution Time

Before optimization: 7.082 us
After optimization: 7.050 us

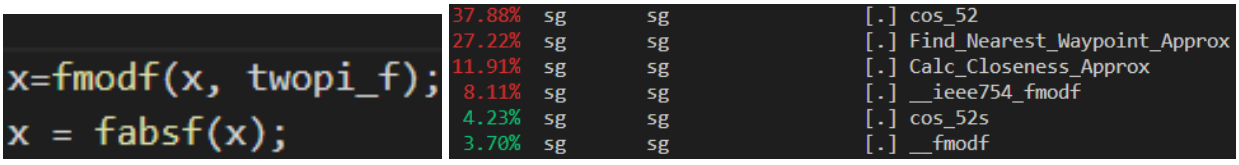
Step 5: Applying fmodf only if x is greater than 2 Pi

On removing fmodf from cos_52 function the number of distance errors don't change. This can lead to conclusion that majority of the values of x are smaller than 2Pi. A check using if statement can be added to verify if x is greater than 2 Pi or less than -2 Pi. The no. of checks can be reduced by changing the order of x = fabsf(x) and x = fmodf(x, twopi_f). Now it is only needed to verify if x is greater than 2 Pi.

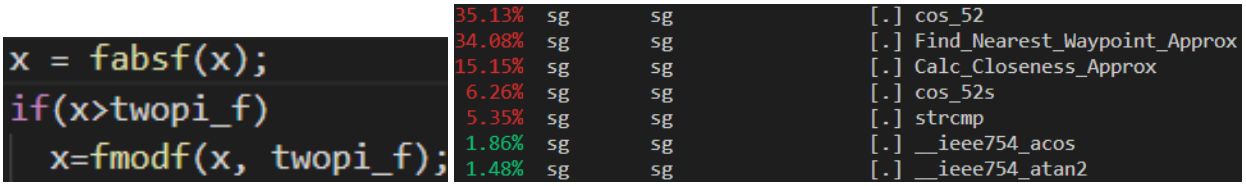


Test for non-polar regions

Before: __fmodf takes 3.7% of execution time.



After: __fmodf is removed from percentage chart.



Program Execution Time

Before optimization: 7.082 us
After optimization: 5.152 us

Step 6: Replacing Switch with if statements

With switch statement the address of case to be executed is loaded using ldrls instruction whereas in if statements the code to be executed is reached using a series of branch instructions. A load instruction has an execution latency of 4 whereas as a branch has latency of 1. Also loading PC leads to creation of branch with added latency of 2 cycles. For control flow whereas no of paths is lower in number using if statement is more beneficial.

Before: cos_52 takes 35%.

```
switch (quad){
case 0: return  cos_52s(x);
case 1: return -cos_52s(SP_PI-x);
case 2: return -cos_52s(x-SP_PI);
case 3: return  cos_52s(twopi_f-x);
}
```

```
35.13%  sg      sg              [.] cos_52
34.08%  sg      sg              [.] Find_Nearest_Waypoint_Approx
15.15%  sg      sg              [.] Calc_Closeness_Approx
 6.26%  sg      sg              [.] cos_52s
 5.35%  sg      sg              [.] strcmp
 1.86%  sg      sg              [.] __ieee754_acos
 1.48%  sg      sg              [.] __ieee754_atan2
```

```
        |       switch (quad){
        |       vmov   r3, s15
  2.96  |       cmp    r3, #3
 55.19  |       ldrls  pc, [pc, r3, lsl #2]
        |     ↓ b      9c
        |       .word  0x00010d04
        |       .word  0x00010d14
        |       .word  0x00010d28
        |       .word  0x00010d3c
```

After: cos_52 is no longer at top of use percentage it takes 25% now.

```
if(quad == 0)
  return  cos_52s(x);
else if(quad == 1)
  return -cos_52s(SP_PI-x);
else if(quad == 2)
  return -cos_52s(x-SP_PI);
else if(quad == 3)
  return  cos_52s(twopi_f-x);
```

```
44.59%  sg      sg              [.] Find_Nearest_Waypoint_Approx
25.02%  sg      sg              [.] cos_52
16.72%  sg      sg              [.] Calc_Closeness_Approx
 5.11%  sg      sg              [.] strcmp
 4.28%  sg      sg              [.] cos_52s
 2.11%  sg      sg              [.] __ieee754_acos
```

```
Percent |       if(quad == 0)
  8.32  |       cmp    r3, #0
        |     ↓ beq    60
        |         return  cos_52s(x);
        |       else if(quad == 1)
  3.09  |       cmp    r3, #1
        |     ↓ beq    70
```

Program Execution Time

Before optimization: 5.152 us
After optimization: 4.982 us

Step 7: Simplify ref.lat and ref.lon calculation

Use of vcvt and a vdiv instruction can be removed.

Before:

```
ref.Lat = cur_pos_lat*PI/180;
ref.Lon = cur_pos_lon*PI/180;
ref.SinLat = SIN(ref.Lat);
ref.CosLat = COS(ref.Lat);
```

```
44.59%  sg      sg              [.] Find_Nearest_Waypoint_Approx
25.02%  sg      sg              [.] cos_52
16.72%  sg      sg              [.] Calc_Closeness_Approx
 5.11%  sg      sg              [.] strcmp
 4.28%  sg      sg              [.] cos_52s
 2.11%  sg      sg              [.] __ieee754_acos
```

```
ref.Lat = cur_pos_lat*PI/180;
vcvt.f64.f32 d18, s0
vldr   d20, [pc, #272] ; 10b70
vmul.f64 d18, d18, d20
vldr   d19, [pc, #272] ; 10b78
vdiv.f64 d16, d18, d19
vcvt.f32.f64 s0, d16
vstr   s0, [sp]
ref.Lon = cur_pos_lon*PI/180;
vcvt.f64.f32 d17, s1
vmul.f64 d17, d17, d20
vdiv.f64 d16, d17, d19
vcvt.f32.f64 s15, d16
vstr   s15, [sp, #12]
```

After: No. of instructions is reduced.

```
ref.Lat = cur_pos_lat*PI_OVER_180_f;
ref.Lon = cur_pos_lon*PI_OVER_180_f;
ref.SinLat = SIN(ref.Lat);
ref.CosLat = COS(ref.Lat);
```

```
ref.Lat = cur_pos_lat*PI_OVER_180_f;
vldr   s15, [pc, #264] ; 10b64 <Find
vmul.f32 s0, s0, s15
vstr   s0, [sp]
ref.Lon = cur_pos_lon*PI_OVER_180_f;
vmul.f32 s1, s1, s15
vstr   s1, [sp, #12]
```

```
#define PI_f 3.14159265f
#define PI_OVER_180_f (0.017453293f)
```

```
44.65%  sg      sg              [.] Find_Nearest_Waypoint_Approx
23.84%  sg      sg              [.] cos_52
15.01%  sg      sg              [.] Calc_Closeness_Approx
 5.35%  sg      sg              [.] cos_52s
 4.81%  sg      sg              [.] strcmp
 2.97%  sg      sg              [.] __ieee754_atan2
```

Program Execution Time

Before optimization: 4.982 us
After optimization: 4.970 us

Step 8: Modify while loop condition

Since the size of waypoint data array is fixed the condition statement can be changed to check for length rather than a strcmp function.

Before: Strcmp takes 4.81% of execution time.

```
while (strcmp(waypoints[i].Name, "END")) {
  c = Calc_Closeness_Approx(&ref, &(waypoints[i]) );
  if (c>max_c) {
    max_c = c;
    closest_i = i;
  }
  i++;
}
```

```
44.65%  sg      sg              [.] Find_Nearest_Waypoint_Approx
23.84%  sg      sg              [.] cos_52
15.01%  sg      sg              [.] Calc_Closeness_Approx
 5.35%  sg      sg              [.] cos_52s
 4.81%  sg      sg              [.] strcmp
 2.97%  sg      sg              [.] __ieee754_atan2
```

After: Call to strcmp is removed and Find_Nearest_Waypoint_Approx is reduced by 1% points.

```
while (i<=WAYPOINTS){
  c = Calc_Closeness_Approx(&ref, &(waypoints[i]) );
  if (c>max_c) {
    max_c = c;
    closest_i = i;
  }
  i++;
}
```

```
#define WAYPOINTS 163
```

```
43.39%  sg      sg              [.] Find_Nearest_Waypoint_Approx
26.57%  sg      sg              [.] cos_52
16.64%  sg      sg              [.] Calc_Closeness_Approx
 6.94%  sg      sg              [.] cos_52s
 3.71%  sg      sg              [.] __ieee754_atan2
 2.33%  sg      sg              [.] __ieee754_acos
```

Program Execution Time

Before optimization: 4.970 us
After optimization: 4.116 us

Step 9: Replace acos function with acosf

Before: acos takes 2.33% of execution time.

```
// Finish calcuations for the closest point
d = acos(max_c)*6371; // finish distance calcuation
```

```
43.39%  sg      sg              [.] Find_Nearest_Waypoint_Approx
26.57%  sg      sg              [.] cos_52
16.64%  sg      sg              [.] Calc_Closeness_Approx
 6.94%  sg      sg              [.] cos_52s
 3.71%  sg      sg              [.] __ieee754_atan2
 2.33%  sg      sg              [.] __ieee754_acos
```

After:

```
// Finish calcuations for the closest point
d = acosf(max_c)*6371; // finish distance calcuation
```

```
43.53%  sg       sg                [.] Find_Nearest_Waypoint_Approx
25.78%  sg       sg                [.] cos_52
15.56%  sg       sg                [.] Calc_Closeness_Approx
 6.87%  sg       sg                [.] __ieee754_atan2
 6.44%  sg       sg                [.] cos_52s
 0.49%  sg       sg                [.] Calc_Bearing_Approx
 0.49%  sg       sg                [.] __ieee754_acosf
```

Program Execution Time

Before optimization: 4.116 us
After optimization: 4.053 us

Step 10: Replace atan2 with atan2f in Calc_Bearing_Approx

Removes the need for a convert instruction.

Before: __ieee754_atan2 takes 6.87%

```
term1 = sin(p2->Lon - p1->Lon)*p2->CosLat;
term2 = p1->CosLat*p2->SinLat -
  p1->SinLat*p2->CosLat*cos(p2->Lon - p1->Lon);
angle = atan2(term1, term2) * (180/PI);
```

```
43.53%  sg       sg                [.] Find_Nearest_Waypoint_Approx
25.78%  sg       sg                [.] cos_52
15.56%  sg       sg                [.] Calc_Closeness_Approx
 6.87%  sg       sg                [.] __ieee754_atan2
 6.44%  sg       sg                [.] cos_52s
 0.49%  sg       sg                [.] Calc_Bearing_Approx
 0.49%  sg       sg                [.] __ieee754_acosf
```

After: __ieee754_atan2f takes 0.86%

```
term1 = SIN(p2->Lon - p1->Lon)*p2->CosLat;
term2 = p1->CosLat*p2->SinLat -
  p1->SinLat*p2->CosLat*COS(p2->Lon - p1->Lon);
angle = atan2f(term1, term2) / (PI_OVER_180_f);
```

```
46.30%  sg       sg                [.] Find_Nearest_Waypoint_Approx
26.57%  sg       sg                [.] cos_52
16.13%  sg       sg                [.] Calc_Closeness_Approx
 8.00%  sg       sg                [.] cos_52s
 0.86%  sg       sg                [.] __ieee754_atan2f
 0.67%  sg       sg                [.] __atan2f
 0.61%  sg       sg                [.] __atanf
```

Program Execution Time

Before optimization: 4.053 us
After optimization: 4.036 us

Step 11: quadrant calculation optimization in cos_52

Conversion of quad to int isn't required we can change it to a float and modify if statements accordingly.

Before:

```
quad=(int) (x * two_over_pi_f);
```

After:

```
float quad;           // what

x = fabsf(x);           // cos
if(x>twopi_f)
  x=fmodf(x, twopi_f);
quad= (x * two_over_pi_f);
if(quad <1)
  return  cos_52s(x);
else if(quad <2)
  return -cos_52s(SP_PI-x);
else if(quad <3)
  return -cos_52s(x-SP_PI);
else if(quad <4)
  return  cos_52s(twopi_f-x);
```

Program Execution Time

Before optimization: 4.036 us
After optimization: 4.007 us

Step 12: Makefile changes

Add -O3 optimization flag and remove -ggdb and -Og flags.

Program Execution Time

Before optimization: 4.036 us
After optimization: 2.552 us