ICS 632 Final Project Report

--- GMT Optimization

Hualiang Li

Outcome:

With OPENMP and 20 cores, I manage to get a speedup of **3.1** for spheredist.sh, and speedup of **15** for spharm.sh.

1. Introduction.

   This is a report for the final project of ICS 632, High Performance Computing. The remaining report will be organized as following: Section 2 will introduce the process of buiding the GMT code on CRAY. In Section 3, I will talk about what I modified and added to the current GMT code. The optimized performance will be shown in Secton 4. In the same section, I will also introduce some issue of my project and possible future work.

2. Build code

Before we build the code, we have to make sure we are on product node, not login node. The following is the complete steps to achieve my ourcome

Step 1. Build GMT code on Cray.

In order to run GMT on Cray, I have to configure the install path and specify the path of dependencies. To do that, I made a copy of "ConfiguserTemplate.cmake" in cmake folder, and name it "ConfigUser.cmake". In which, I add three line to specify my build folder and two libraries: gshhg, and dcw. If we do not do this, we have to type every time I "cmake" it.

```
41 # Installation path (usually defaults to /usr/local) [auto]:
42 #set (CMAKE_INSTALL_PREFIX "prefix_path")
43 set (CMAKE_INSTALL_PREFIX /home/hualiang/apps/local/gmt5-dev)
44 set (GSHHG_ROOT /home/hualiang/apps/local/gshhg-gmt-2.3.5)
45 set (DCW_ROOT /home/hualiang/apps/local/dcw-gmt-1.1.2)
46 # Set install name suffix used for directories and gmt executables
47 # [undefined]:
48 #set (GMT_INSTALL_NAME_SUFFIX "suffix")
49
```

After that, in order to compile the code, we have to load intel and cmake, to make life easier, I use aliases in .bashrc file:

```
 8 # User specific aliases and functions
 9 alias mysandbox='srun -p sb.q -N 1 -c 1 --mem=20480 --pty -t 0-01:00 /bin/bash'
10 alias myics="module load intel/ics"
11 alias myimpi="module load intel/impi"
12 alias mycmake="module load tools/cmake/3.3.2"
13 alias myintel="module load lib/netcdf/intel/4.x.x"
```

Then, I made a folder called “build” in the main directory of GMT. Lastly, change directory to the “build” folder. Run the command:

>mysandbox

>mycamke

>myintel

Then we start to build the code by run:

>cmake .

>make

>make install

Finally, we need to add our GMT path to the environment by adding the following to “.bash_profile”:

```
 8 # User specific environment and startup programs
 9
10 PATH=$PATH:$HOME/bin
11
12 export PATH
13
14 PATH=$PATH:$HOME/apps/local/gmt5-dev/bin
15 export PATH
```

3. Optimize code

I. “spheredist.sh”

Before I change any code, I use gprof and Mac instrument software to identify the bottleneck or critical part of this project, here is a screenshot of the running result for “spheredist.sh”. As we can see that two function takes 90% of the running time:

a. gmt_inonout

b. gmt_distance

According to Ahmdal’s Law, it is a good idea to optimize these parts of code to benefit most.

<u>Optimization 1, switching the order of if-else condition to break early.</u>

I found that there are lots of if-else branch in gmt_inonout function, I try to reorder them and make sure most frequently branch come before the others. Here is the screenshot of part of my code:

```
if (P->pole) {  /* Case 1 of an enclosed polar cap */
        if (P->pole == +1) {    /* N polar cap */
                if (plat < P->min[GMT_Y]) return (GMT_OUTSIDE); /* South of a N polar cap */
                if (plat > P->lat_limit) return (GMT_INSIDE);   /* Clearly inside of a N polar cap */
        }
        if (P->pole == -1) {    /* S polar cap */
                if (plat > P->max[GMT_Y]) return (GMT_OUTSIDE); /* North of a S polar cap */
                if (plat < P->lat_limit) return (GMT_INSIDE);   /* Clearly inside of a S polar cap */
        }

        /* Tally up number of intersections between polygon and meridian through P */

        if (support_inonout_sphpol_count (plon, plat, P, count)) return (GMT_ONEDGE);   /* Found P is on S */

        if (P->pole == +1 && count[0] % 2 == 0) return (GMT_INSIDE);
        if (P->pole == -1 && count[1] % 2 == 0) return (GMT_INSIDE);

        return (GMT_OUTSIDE);
}

/* Here is Case 2.  First check latitude range */

if (plat < P->min[GMT_Y] || plat > P->max[GMT_Y]) return (GMT_OUTSIDE);

/* Longitudes are tricker and are tested with the tallying of intersections */

if (support_inonout_sphpol_count (plon, plat, P, count)) return (GMT_ONEDGE);   /* Found P is on S */

if (count[0] % 2) return (GMT_INSIDE);

return (GMT_OUTSIDE);   /* Nothing triggered the tests; we are outside */
```

Optimization 2, inline function and use macro

In the critical part, there are lots of small functions, which may cause overhead to call function. To overcome this issue, I tried to use inline function and macro.

Optimization 3, Implement “sqrt” function used by “sphdistance”

The regular square root function takes O(log n) time to achieve high accuracy. However, this accuracy is possible not necessary. One of a faster square root algorithms I used in this project is called “fast inverse square root”. This algorithm runs in constant time while it can still achieve relatively high accuracy (15 digits). Here is the algorithm that I found in Wikipedia:

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;                       // evil floating point bit level hacking
    i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
//  y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed

    return y;
}
```

Then I traced down the code and find the function to calculate the distance in gmt_map.c:

```
case GMT_DIST_M+GMT_FLATEARTH:  /* 2-D lon, lat data, but scale to Cartesian flat earth in meter */
        GMT->current.map.dist[type].func = &map_flatearth_dist_meter;
        GMT->current.map.azimuth_func  = &map_az_backaz_flatearth;
        GMT_Report (GMT->parent, GMT_MSG_LONG_VERBOSE, "%s distance calculation will be Flat Earth
_name);
        break;
  case GMT_DIST_M+GMT_GREATCIRCLE:        /* 2-D lon, lat data, use spherical distances in meter */
        GMT->current.map.dist[type].func = &gmt_great_circle_dist_meter;
        GMT->current.map.azimuth_func = &map_az_backaz_sphere;
        GMT_Report (GMT->parent, GMT_MSG_LONG_VERBOSE, "%s distance calculation will be using great
uxiliary latitudes and %s radius = %.4f m, in %s.\n",
                type_name[type], aux[choice], rad[GMT->current.setting.proj_mean_radius], GMT->curr
);
        break;
  case GMT_DIST_M+GMT_GEODESIC:   /* 2-D lon, lat data, use geodesic distances in meter */
        GMT->current.map.dist[type].func = GMT->current.map.geodesic_meter;
        GMT->current.map.azimuth_func = GMT->current.map.geodesic_az_backaz;
        GMT_Report (GMT->parent, GMT_MSG_LONG_VERBOSE, "%s distance calculation will be using %s ge
e], GEOD_TEXT[GMT->current.setting.proj_geodesic], unit_name);
        break;
```

Optimization 4, Use OPENMP to parallelize computing

For spheredist.sh, there is a triple for-loop in sphdistance.c, the structure is something like this:

```
for (node=0; node<n; node++) {
    for (row=0; row<n_row; row++) {
        for (col=0; col<n_col; col++) {
            index = compute_index;
            data = compute_data;
            Grid[index] = data;
        }
    }
}
```

I sue OPENMP to parallelize the most inner for loop, following is my screenshot:

```
567      /* So here, any polygon will have a positive (or 0) west_col with an east_col >= west_col */
568      for (s_row = north_row; s_row <= south_row; s_row++) {   /* For each scanline intersecting this polygon */
569         if (s_row < 0) continue;       /* North of region */
570         row = s_row; if (row >= Grid->header->n_rows) continue;       /* South of region */
571 #pragma omp parallel for private(col, side, ij, f_val, n_set) //num_threads(6)
572         for (p_col = west_col; p_col <= east_col; p_col++) {   /* March along the scanline using col >= 0 */
573            //int id = omp_get_thread_num();
574            //if (id == 0) printf("Running %d threads \n", omp_get_num_threads());
575            if (p_col >= Grid->header->n_columns) {     /* Off the east end of the grid */
576               if (periodic)    /* Just shuffle to the corresponding point inside the global grid */
577                  col = p_col - nx1;
578               else             /* Sorry, really outside the region */
579                  continue;
580            }
581            else
582               col = p_col;
```

For spharm.sh, the screenshot is following:

```
#ifdef _OPENMP
#pragma omp parallel for private(col,node,sum,kk,L,M) shared(Grid,row,n_columns,L_min,L_max,P_lm,C,Cosm,S,Sinm)
#endif
                for (col = 0; col < n_columns; col++) { /* For each longitude along this parallel */
                        sum = 0.0;      /* Initialize sum to zero for new output node */
                        kk = (L_min) ? LM_index (L_min, 0) : 0; /* Set start index for P_lm packed array */
                        for (L = L_min; L <= L_max; L++) {      /* For all degrees */
                                for (M = 0; M <= L; M++, kk++) {        /* For all orders <= L */
                                        sum += P_lm[kk] * (C[L][M] * Cosm[col][M] + S[L][M] * Sinm[col][M]);
                                }
                        }
                        node = gmt_M_ijp (Grid->header, row, col);
                        Grid->data[node] = (float)sum;  /* Assign total to the grid, cast as float */
                }
```

Optimization 5, Use MPI to run the code on different nodes.

The idea is that, for each non-master process, it computes someparts of the grid, then send those parts of grid to master. After the master collects all the data, it then writes the grid data to the file. The sending and receiving part is shown in the figure below.

```
#ifdef MPI
            int data[1];
            if(myrank == 0) {    // master
               MPI_Recv(data,1,MPI_INT, node/num_procs+1,0,MPI_COMM_WORLD, &status);
               side = data[0];
            } else {
               side = gmt_inonout (GMT, grid_lon[col], grid_lat[row], P);
               data[0] = side;
COMM_WORLD, &s MPI_Send(data,1,MPI_INT, 0,0,MPI_COMM_WORLD, &status);
            }
#endif
```

4. Result performance and Future work

The original code runs about 197 seconds on Cray with single core.

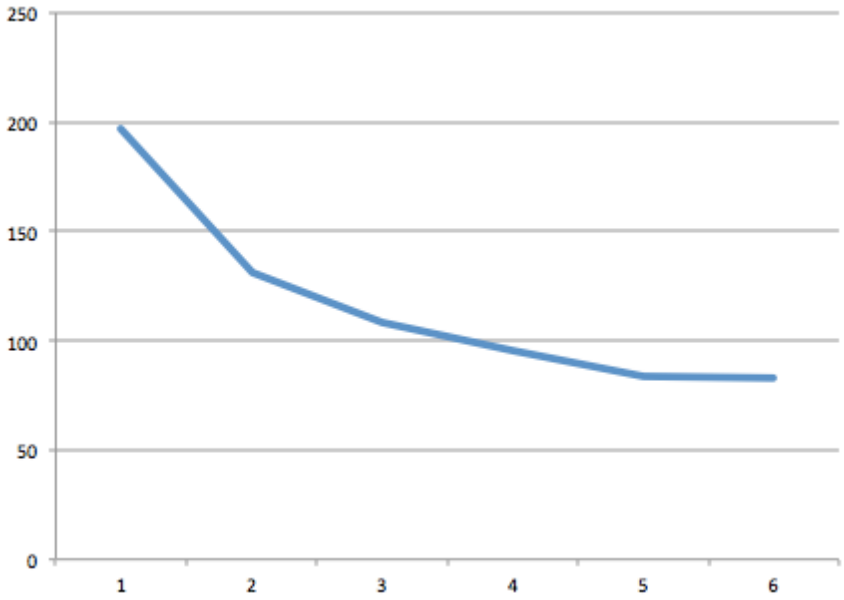
a. For switch the order of if-else condition, the optimized code reduces 1.2 seconds less running time on 10 trials.

b. For inline or Macro, the optimized code has 0.2 seconds more running time. This indicates that inline or Macro is not always going to reduce the running time, sometimes it might even make code runs slower. I guess it's because the compiler use default flag with –O3 optimization, it may automatically inline the small functions. If we specify it manually, it may cause some overhead.

c. By using Fast Inverse Square Root method, the program only reduce 1 second running time, which surprises me a lot. I doubt if I modify the correct distance calculation function, because there are lots of different definition of distance, and using different formula, thus using different functions. Since I do not have any knowledge about geography distance, I could not identify if it is the right function, which is used by spheredist.sh. The reason I doubt the result is because spheredist.sh is heavy distance computation program, using fast inverse square root should improve the computation time a lot due to the O(log n) VS O(1). In the future work, we should test if the function I

modified is the right function that computes the distance.

d. By using OPENMP, for sphdistance.sh, it has a speedup of 2.5 with 5 cores and up to 3.1 if using 20 cores. For spharm.sh, it has a speedup of 15 with 20 cores. The following is the plot, which shows the elapsed time VS number of cores for sphdistance.sh.



Running time (s) VS number of cores

e. For MPI implementation, I did not produce a successful graph, which may be caused by race issue. The following picture is the comparison between the correct picture (left) and the result picture with MPI (right). One potential fix-up is to use an extra variable to store a copy of grid data, have the worker set the region not belonging to itself to zero, then use MPI_reduce to reduce all copy of grid data to master.