CS 280 ASSIGNMENT 2:

Accelerators

NAME: Xiaopeng Xu KAUST ID: 129052

Task 1:

- Annotate the rtm_kernel code (original, uploaded for the first assignment) with openMP directives to get a multithreaded code. Apply changes to the code if needed.
- Compile your openMP code using the Intel compiler on the MIC system for the Intel Xeon CPU (Sandy bridge)
- Run the code with 1, 2, 4, 8 and 16 threads on the CPU and report the corresponding speedups in a **graph**
- Compile the code for the coprocessor, run it using 60, 120, 180 and 240 threads natively on the coprocessor and report the corresponding speedups in a graph

CPU speedup report:

No. of threads (CPU)	1	2	4	8	16
Time (s)	373.6395	188.4551	94.8865	48.3044	36.7293
Speedup	1.00	1.98	3.94	7.74	10.17

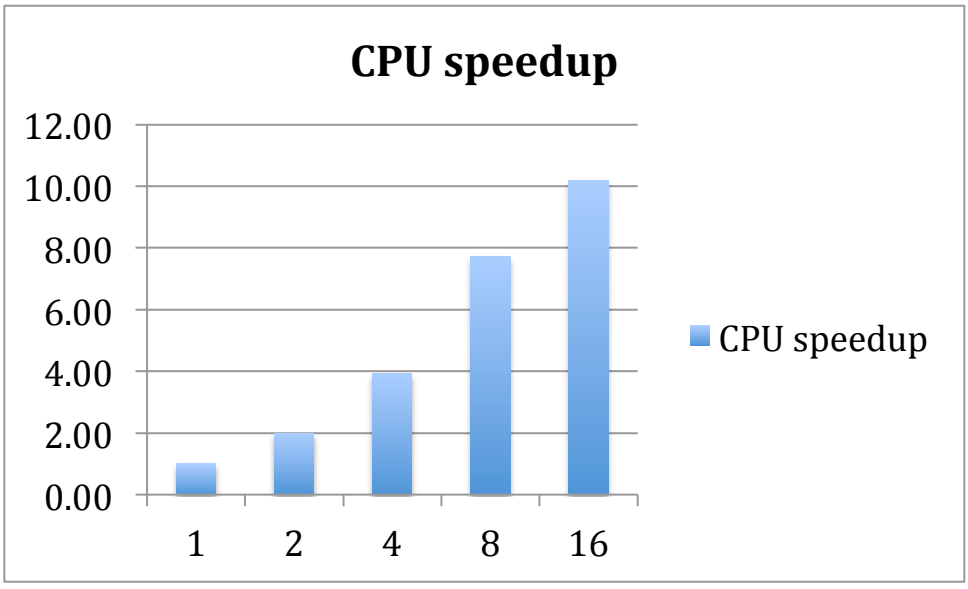

Figure 1. CPU speedup

Coprocessor speedup report:

No. of threads (mic0)	60	120	180	240
Time (s)	71.6174	58.2261	59.5833	56.6691
Speedup	5.22	6.42	6.27	6.59

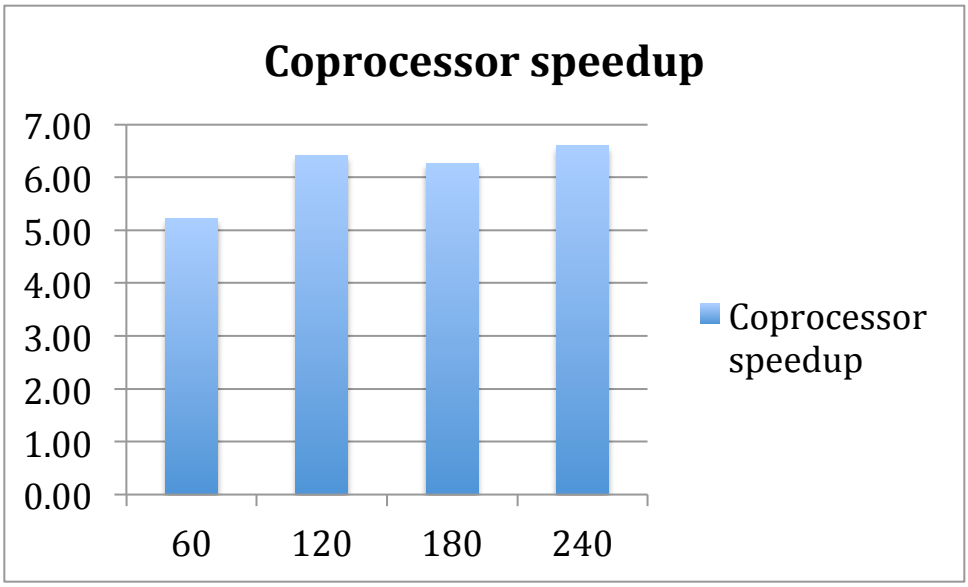

Figure 2. Coprocessor speedup

The directives I added are shown on figure 3. I added pragma for the two for loops in the time loop. Also I make i, x, y, z, lap private in order to make the threads independent, i is firstprivate in order to copy the value from outside.

CPU running shows that multiple threads give me a better performance. However, coprocessor speedup is not proportional to the number of threads. One possible reason is that there are race conditions among multiple threads. The CPU is 16 core with 2.60GHz frequency for each core. (Detailed information about the CPU is shown on attachment file cpuinfo.)

```
148     posix_memalign((void **)&source, p.alignment, sizeof(float)*nt);
149     csource( nt, fmax, dt, source);
150
151     ////////////////////////////////////////////////////////////////
152     ////////////////////////////////////////////////////////////////
153     // run the performance experiments of the target kernel
154     printf("\n**********************************************************\n");
155     printf("Performance results\n");
156     printf("**********************************************************\n");
157
158     gettimeofday(&tvStart, NULL);
159     for (i=0; i<nt/2; i+=2) { // time loop
160         //printf("Time step: %d\n", i);
161 #pragma omp parallel for firstprivate(i) private(x,y,z,lap)
162         for(x=4; x< p.ip-4; x++) {
163             for(y=4; y< p.jp-4; y++) {
164                 for(z=4; z< p.kp-4; z++) {
165                     lap=coef[0]*Vc(x,y,z)
166                         +coef[1]*(Vc(x+1,y  ,z  )+Vc(x-1,y  ,z  ))
167                         +coef[1]*(Vc(x  ,y+1,z  )+Vc(x  ,y-1,z  ))
168                         +coef[1]*(Vc(x  ,y  ,z+1)+Vc(x  ,y  ,z-1))
169                         +coef[2]*(Vc(x+2,y  ,z  )+Vc(x-2,y  ,z  ))
170                         +coef[2]*(Vc(x  ,y+2,z  )+Vc(x  ,y-2,z  ))
171                         +coef[2]*(Vc(x  ,y  ,z+2)+Vc(x  ,y  ,z-2))
172                         +coef[3]*(Vc(x+3,y  ,z  )+Vc(x-3,y  ,z  ))
173                         +coef[3]*(Vc(x  ,y+3,z  )+Vc(x  ,y-3,z  ))
174                         +coef[3]*(Vc(x  ,y  ,z+3)+Vc(x  ,y  ,z-3))
175                         +coef[4]*(Vc(x+4,y  ,z  )+Vc(x-4,y  ,z  ))
176                         +coef[4]*(Vc(x  ,y+4,z  )+Vc(x  ,y-4,z  ))
177                         +coef[4]*(Vc(x  ,y  ,z+4)+Vc(x  ,y  ,z-4));
178
179                     Uc(x,y,z) = 2.*Vc(x,y,z) - Uc(x,y,z) + ROC2(x,y,z)*lap;
180                     if( (x==ixs) && (y==iys) && (z==izs) ){
181                         Uc(ixs,iys,izs) =  Uc(ixs,iys,izs) + source[i];
182                     }
183                 }
184             }
185         }
186 #pragma omp parallel for firstprivate(i) private(x,y,z,lap)
187         for(x=4; x< p.ip-4; x++) {
188             for(y=4; y< p.jp-4; y++) {
189                 for(z=4; z< p.kp-4; z++) {
190                     lap=coef[0]*Uc(x,y,z)
191                         +coef[1]*(Uc(x+1,y  ,z  )+Uc(x-1,y  ,z  ))
192                         +coef[1]*(Uc(x  ,y+1,z  )+Uc(x  ,y-1,z  ))
193                         +coef[1]*(Uc(x  ,y  ,z+1)+Uc(x  ,y  ,z-1))
194                         +coef[2]*(Uc(x+2,y  ,z  )+Uc(x-2,y  ,z  ))
```

Figure 3. Directives added for OpenMP

Task 2:

Annotate the rtm_kernel code (original, uploaded for the first assignment) with openACC directives

- Compile it using the PGI compiler on the SMC system
- Run it using a fermi GPU and report the corresponding speedup
- Run it using a K20 GPU and report the corresponding speedup

GPU speedup report:

GPUs	Fermi	K20
Time (s)	24.3965	17.8206
Speedup	15.32	20.97

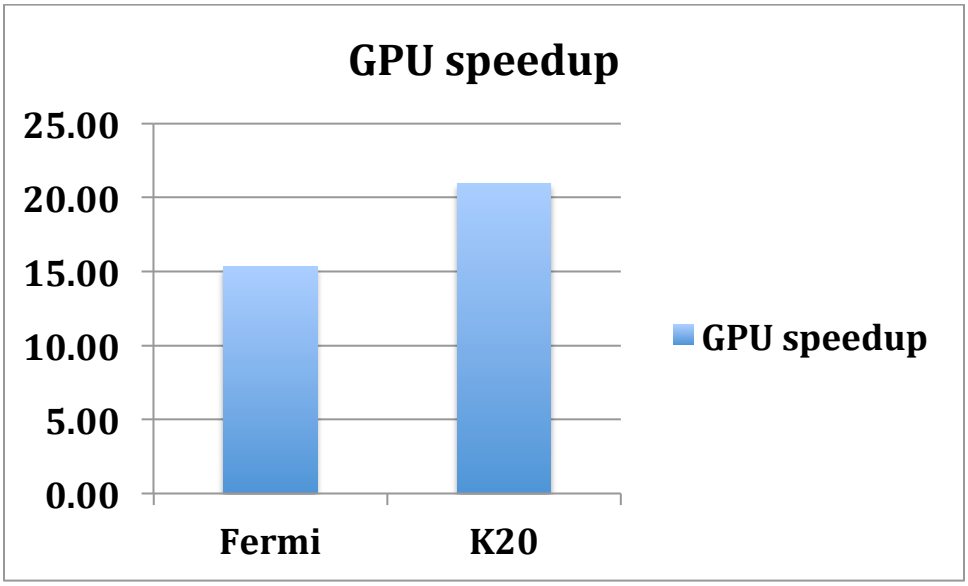

Figure 4. GPU speedup

The directives I added for OpenACC is shown on Figure 5. I copy the data outside of time loop, copy coeff, U3, source, and create/copy tempU1 and tempU2 (At first I just create tempU1, and tempU2. Then, in order to check the result I copied tempU1 and tempU2). I add two pragmas for each loop inside the time loop in order to achieve parallelism. Here collapse is used to make the 3 for loops collapse in serial codes, thus can easily be parallelized.

The running results shows that K20 gives me a better performance over Fermi.

```
#pragma acc data copy(coef[0:5],U3[0:p.domain_size],source[0:nt]) copy(tempU1[0:p.domain_size],tempU2
    [0:p.domain_size])
    for (i=0; i<nt/2; i+=2) { // time loop
        //printf("Time step: %d\n", i);
#pragma acc kernels
#pragma acc loop independent gang(256) collapse(3)// private(x,y,z,lap)
        for(x=4; x< p.ip-4; x++) {
            for(y=4; y< p.jp-4; y++) {
                for(z=4; z< p.kp-4; z++) {
                    lap=coef[0]*Vc(x,y,z)
                        +coef[1]*(Vc(x+1,y  ,z  )+Vc(x-1,y  ,z  ))
                        +coef[1]*(Vc(x  ,y+1,z  )+Vc(x  ,y-1,z  ))
                        +coef[1]*(Vc(x  ,y  ,z+1)+Vc(x  ,y  ,z-1))
                        +coef[2]*(Vc(x+2,y  ,z  )+Vc(x-2,y  ,z  ))
                        +coef[2]*(Vc(x  ,y+2,z  )+Vc(x  ,y-2,z  ))
                        +coef[2]*(Vc(x  ,y  ,z+2)+Vc(x  ,y  ,z-2))
                        +coef[3]*(Vc(x+3,y  ,z  )+Vc(x-3,y  ,z  ))
                        +coef[3]*(Vc(x  ,y+3,z  )+Vc(x  ,y-3,z  ))
                        +coef[3]*(Vc(x  ,y  ,z+3)+Vc(x  ,y  ,z-3))
                        +coef[4]*(Vc(x+4,y  ,z  )+Vc(x-4,y  ,z  ))
                        +coef[4]*(Vc(x  ,y+4,z  )+Vc(x  ,y-4,z  ))
                        +coef[4]*(Vc(x  ,y  ,z+4)+Vc(x  ,y  ,z-4));

                    Uc(x,y,z) = 2.*Vc(x,y,z) - Uc(x,y,z) + ROC2(x,y,z)*lap;
                    if( (x==ixs) && (y==iys) && (z==izs) ){
                        Uc(ixs,iys,izs) =  Uc(ixs,iys,izs) + source[i];
                    }
                }
            }
        }
#pragma acc kernels
#pragma acc loop  independent collapse(3) gang(256)// private(x,y,z,lap)
        for(x=4; x< p.ip-4; x++) {
            for(y=4; y< p.jp-4; y++) {
                for(z=4; z< p.kp-4; z++) {
                    lap=coef[0]*Uc(x,y,z)
                        +coef[1]*(Uc(x+1,y  ,z  )+Uc(x-1,y  ,z  ))
                        +coef[1]*(Uc(x  ,y+1,z  )+Uc(x  ,y-1,z  ))
                        +coef[1]*(Uc(x  ,y  ,z+1)+Uc(x  ,y  ,z-1))
                        +coef[2]*(Uc(x+2,y  ,z  )+Uc(x-2,y  ,z  ))
                        +coef[2]*(Uc(x  ,y+2,z  )+Uc(x  ,y-2,z  ))
                        +coef[2]*(Uc(x  ,y  ,z+2)+Uc(x  ,y  ,z-2))
                        +coef[3]*(Uc(x+3,y  ,z  )+Uc(x-3,y  ,z  ))
                        +coef[3]*(Uc(x  ,y+3,z  )+Uc(x  ,y-3,z  ))
                        +coef[3]*(Uc(x  ,y  ,z+3)+Uc(x  ,y  ,z-3))
                        +coef[4]*(Uc(x+4,y  ,z  )+Uc(x-4,y  ,z  ))
                        +coef[4]*(Uc(x  ,y+4,z  )+Uc(x  ,y-4,z  ))
                        +coef[4]*(Uc(x  ,y  ,z+4)+Uc(x  ,y  ,z-4));

                    Vc(x,y,z) = 2.*Uc(x,y,z) - Vc(x,y,z) + ROC2(x,y,z)*lap;
                    if( (x==ixs) && (y==iys) && (z==izs) && ((i+1) <nt)){
                        Vc(ixs,iys,izs) =  Vc(ixs,iys,izs) + source[i+1];
                    }
                }
```

Figure 5. Directives added for OpenACC