Massively Parallel Computer Architecture: Probelems for Term Report (AY 2017)

- Submit a paper on Q1 *or* Q2 shown in the following pages referring to the *basic code* '`diff.c`' included in '`kobe-paper-17.zip`' toghether with related files distributed by from the department office.
- The paper must be written in *English*, compiled in a '`.pdf`' or '`.doc[x]`' file, and submitted by an e-mail, to which the file is attached and in which your name and student-ID number are shown, to `h.nakashima@media.kyoto-u.ac.jp` by 23:59, Friday, September 29, 2017.

Q1 The temporal tiling, shown in p.49 of "High-Performance Microprocessor Architecture and Programming" (slide title is "Performance with Cache & TLB (4/7)"), is implemented by replacing the loop in the *basic code*;

```
for(t=0; t<NT; t++) { ... }
```

with the following, where `TX` and `TY` are the width and height of a tile and `TT` is the number of time steps of computation on a series of shifted tiles,

```
for (t=0; t<NT; t+=TT) {
  const int nt = t+TT>NT ? NT-t : TT;
  const int ny = NY + nt,  nx = NX + nt;
  for (y=0; y<ny; y+=TY) {
    const int t0y = (a)       ,  t1y = (b)       ;
    for (x=0; x<nx; x+=TX) {
      double *ut = u,  *unt = un;
      int t0 = 0, t1 = nt;
      const int t0x = (c)       ,  t1x = (d)       ;
      if (t0<t0y)  t0 = t0y;  if (t0<t0x)  t0 = t0x;
      if (t1>t1y)  t1 = t1y;  if (t1>t1x)  t1 = t1x;
      for (tt=t0; tt<t1; tt++) {
        double (*restrict uu)[NX+2]=ut, (*restrict uun)[NX+2]=unt;
        int y0 = (e)       ,  y1 = (f)       ;
        if (y0<0)  y0 = 0;  if (y1>NY)  y1 = NY;
        int x0 = (g)       ,  x1 = (h)       ;
        if (x0<0)  x0 = 0;  if (x1>NX)  x1 = NX;
        for (yy=y0;  yy<y1;  yy++)  for (xx=x0; xx<x1; xx++) {
            uun[yy][xx] = uu[yy][xx] +
              dthh*(uu[yy][xx-1]+uu[yy][xx+1]+
                    uu[yy-1][xx]+uu[yy+1][xx]-4.0*uu[yy][xx]);
        }
        tmp = ut;  ut = unt;  unt = tmp;
      }
    }
  }
  if (nt&1) {
    tmp = u;  u = un;  un = tmp;
  }
}
```

Answer the following questions about the code shown above.

(1) Complete the code by filling the vacancies (a) to (h).

(2) Show the reason why you assign particular values to the variables `t0`, `t1`, `y0`, `y1`, `x0` and `x1` related to the vacancies.

(3) Discuss the implementation issues about appropriate values of `TX`, `TY` and `TT`.

Q2 In the *basic code*, a pair of two-dimensional arrays of `(NY+2)*(NX+2)` is used to have *old* and *new* values for the grid of the two-dimensional domain. These two arrays, however, can be replaced with an array of `(NY+3)*(NX+2)` as follows, so that the required memory space is almost halved.

- Replace the declarations of `uold[NY+2][NX+2]` and `unew[NY+2][NX+2]` with;

  ```
  double uoldnew[NY+3][NX+2];
  ```

- Initialize pointer variables `u` and `un` appropriately.

- Replace the initialization of the arrays and the loop

  ```
  for(t=0; t<NT; t++) { ... }
  ```

  with the following.

```
for (y=0; y<=NY; y++) {
  for (x=0; x<=NX; x++)  u[y][x] = 0.0;
  u[y][-1] = 0.5;
}
for (x=-1; x<NX; x++)  u[-1][x] = un[-1][x] = 1.0;

gettimeofday(tv, NULL);
for (t=0; t<NT; t+=2) {
  for ((a)       ) {
    un[y][-1] = u[y][-1];  un[y][NX] = u[y][NX];
    for (x=0; x<NX; x++)
      un[y][x] = u[y][x] +
        dthh*(u[y][x-1]+u[y][x+1]+u[y-1][x]+u[y+1][x]-4.0*u[y][x]);
  }
  for (x=0; x<NX; x++)  un[NY][x] = u[NY][x];
  tmp = u;  u = un;  un = tmp;
  if (t+1>=NT)  break;
  for ((b)       ) {
    un[y][-1] = u[y][-1];  un[y][NX] = u[y][NX];
    for (x=0; x<NX; x++)
      un[y][x] = u[y][x] +
        dthh*(u[y][x-1]+u[y][x+1]+u[y-1][x]+u[y+1][x]-4.0*u[y][x]);
  }
  for (x=0; x<NX; x++)  un[-1][x] = u[-1][x];
  tmp = u;  u = un;  un = tmp;
}
```

Answer the following questions about the code shown above.

(1) Show the initial values of the pointer variables `u` and `un`.

(2) Complete the code by filling the vacancies (a) and (b).

(3) Discuss the way to modify the code in order to SIMD-vectorize the innermost loop `for(x=0; x<NX; x++) ...`

(4) Discuss the method to paralleize the code by OpenMP.