Fachgebiet Scientific Computing *www.sc.tu-darmstadt.de*
Bischof, Mariano, Burger, Iwainsky, Berhe

Programmierung paralleler Rechnerarchitekturen, Winter 2014/15

Exercise 9

Task 9.1 (Stencil algorithm on Xeon Phi)**.** In this task you will work with a 9-point stencil algorithm as it is known from calculating partial differential equations. The algorithm runs over a 2d grid and computes a weighted sum of each entry with all its neighbors.

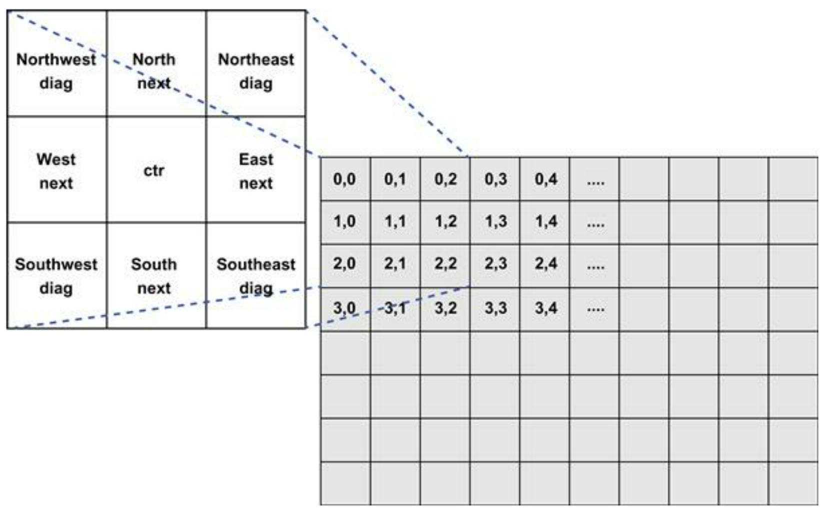

Figure 9.1: 9 point stencil

The results are written in a second array to avoid conflicts. After each iteration input- and output-arrays are swapped. The C-code of this algorithm can be found in the following.

```c
void
stencil9pt_omp(REAL *finp, REAL *foutp,
               int width, int height,
               REAL ctr, REAL next, REAL diag, int count)
{
  REAL *fin = finp;
  REAL *fout = foutp;
  int i,x,y;

  for (i=0; i<count; i++) {


    for (y=1; y < height-1; y++) {
      // starting center pt (avoid halo)
      int c = 1 + y*WIDTHP+1;
      // offsets from center pt.
      int n = c-WIDTHP;
      int s = c+WIDTHP;
      int e = c+1;
      int w = c-1;
      int nw = n-1;
      int ne = n+1;
      int sw = s-1;
      int se = s+1;


      for (x=1; x < width-1; x++) {
```

```
          fout[c] = diag * fin[nw] +
                    diag * fin[ne] +
                    diag * fin[sw] +
                    diag * fin[se] +
                    next * fin[w] +
                    next * fin[e] +
                    next * fin[n] +
                    next * fin[s] +
                    ctr  * fin[c];

          // increment to next location
          c++;n++;s++;e++;w++;nw++;ne++;sw++;se++;
        }
      }
      REAL *ftmp = fin;
      fin = fout;
      fout = ftmp;
    }
    return;
  }
```

The kernel should now be prepared for execution on the Xeon Phi.

(a) The compiler reports that it assumes a vector dependency when trying to vectorize the only loop candidate. What loop is this candidate? What can you do in order to resolve this issue?

(b) After the code vectorizes it needs to be parallelized. Which appropriate OpenMP pragmas must be placed (and where?) in order to do worksharing?

Solution:

```
void
stencil9pt_omp(REAL *finp, REAL *foutp,
                 int width, int height,
                 REAL ctr, REAL next, REAL diag, int count)
{
  REAL *fin = finp;
  REAL *fout = foutp;
  int i,x,y;

  for (i=0; i<count; i++) {
#pragma omp parallel for private(x)
    for (y=1; y < height-1; y++) {
      // starting center pt (avoid halo)
      int c = 1 + y*WIDTHP+1;
      // offsets from center pt.
      int n = c-WIDTHP;
      int s = c+WIDTHP;
      int e = c+1;
      int w = c-1;
      int nw = n-1;
      int ne = n+1;
      int sw = s-1;
      int se = s+1;
#pragma ivdep
      for (x=1; x < width-1; x++) {
        fout[c] = diag * fin[nw] +
                  diag * fin[ne] +
                  diag * fin[sw] +
                  diag * fin[se] +
                  next * fin[w] +
                  next * fin[e] +
                  next * fin[n] +
                  next * fin[s] +
```

```
                    ctr   * fin[c];

          // increment to next location
          c++;n++;s++;e++;w++;nw++;ne++;sw++;se++;
        }
      }
      REAL *ftmp = fin;
      fin = fout;
      fout = ftmp;
    }
    return;
  }
```

Task 9.2 (Ellpack Storage Format)**.** In this task the Ellpack storage format, known from the lecture slides, is investigated. As test case we are employing a matrix-vector-multiplication. A framework for this task is available in *05task_crs.zip* on moodle. It is compileable with *g++ 05task_ellpack.cpp -o ellpack -fopenmp* and can be executed. Of course the integrated verification of this file fails at the moment.

(a) First the input data has to be converted to the Ellpack format. To that end, complete the function *createELLPACKSystem* within the file *05task_ellpack.cpp*. Comments within the skeleton of the function indicate where you have to insert your code.

  Solution: See code from moodle.

(b) Now the actual method for the calculation must be implemented. A code skeleton for the function *multiplyELLPACK* is already included in *05task_crs.cpp*. You need not to implement an optimised or vectorized kernel.

  Solution: See code from moodle.

Discussion of this exercise on 10^{th} February 2015.