UNIVERSITÄT SIEGEN

Fakultät IV
Betriebssysteme und verteilte Systeme
Prof. Dr. rer. nat. Roland Wismüller

Exercise Sheet 2
Solution

Parallel Processing
Winter Term 2023/24

Exercise 1: Dependence analysis

The correct answers are:

a) There is a **true** dependence between **different** loop iterations from S1 to S2, e.g.:
S1 (i=7): `a[7] = (b[6] + b[7]) / 2;`
S2 (i=8): `b[8] = a[7] * c[8];`

b) There is an **anti** dependence within a **single** loop iteration from S1 to S2, e.g.:
S1 (i=3): `a[3] = (b[2] + b[3]) / 2;`
S2 (i=3): `b[3] = a[7] * c[3];`

d) There is a **true** dependence between **different** loop iterations from S2 to S1, e.g.:
S2 (i=3): `b[3] = a[7] * c[3];`
S1 (i=4): `a[4] = (b[3] + b[4]) / 2;`

e) There is an **anti** dependence between **different** loop iterations from S2 to S1, e.g.:
S2 (i=6): `b[6] = a[7] * c[6];`
S1 (i=7): `a[7] = (b[6] + b[7]) / 2;`

g) There is an **anti** dependence between **different** loop iterations from S2 to S3, e.g.:
S2 (i=1): `b[1] = a[7] * c[1];`
s3 (i=2): `c[1] = 2 * b[2];`

j) The loop does not contain any **output** dependencies, since each iteration writes to a different element of arrays `a`, `b`, and `c`.

Exercise 6: Parallelization of the Gauss/Seidel method using OpenMP

```
int solver(double **a, int n)
{
  int kmax = (int)(0.35 / eps);
  int i,j;
  int k;

  for (k=0; k<kmax; k++) {
    #pragma omp parallel private(i, j) firstprivate(a, n)
    {
      int ij, ja, je;

      for (ij=1; ij<2*n-4; ij++) {
        ja = (ij <= n-2) ? 1 : ij-(n-3);
        je = (ij <= n-2) ? ij : n-2;

        #pragma omp for
        for (j=ja; j<=je; j++) {
          i = ij-j+1;

          a[i][j] = 0.25 * (a[i][j-1] + a[i-1][j] +
          a[i+1][j] + a[i][j+1]);
```

```
        }
      }
    }
  }
  return kmax;
}
```

Exercise 7 (For Motivated Students): Pipelined parallelization of the Gauss/Seidel method using OpenMP

```
int solver(double **a, int n)
{
  int kmax = (int)(0.35 / eps);

  #pragma omp parallel firstprivate(a, n, kmax)
  {
    int i, j;
    int k;

    #pragma omp for ordered(2) schedule(static, 1)
    for (k=0; k<kmax; k++) {
      for (i=1; i<n-1; i++) {
        #pragma omp ordered depend(sink: k, i-1) depend(sink: k-1, i+1)
        for (j=1; j<n-1; j++) {
          a[i][j] = 0.25 * (a[i][j-1] + a[i-1][j] +
          a[i+1][j] + a[i][j+1]);
        }
        #pragma omp ordered depend(source)
      }
    }
  }
  return kmax;
}
```