Príloha 1 - Triedenie zlučovaním

Utriedenie časti poľa počnúc iteráciou "iter":

```
void mergesort (TYPE *data1, TYPE *data2, int size, int iter, int slicesize=1)
{
  register int psize;
  register TYPE *p1, *p2, *p, *pmarg, *pmarg2;
  TYPE *pp;   int idx, i, j;
  for (i = 0; i < iter; i++) {
    pp = &data2[size];
    psize = 2 * slicesize;
    idx = 0;
    do {
      p1 = &data1[idx];
      p2 = &data1[idx + slicesize];
      pmarg = p2;   pmarg2 = p2 + slicesize;
      p = &data2[idx];
      for (j = 0; j < psize; j++) {
        if ((p2 >= pmarg2) || ((*p1 < *p2) && (p1 < pmarg)))
          *p = *(p1++);
        else *p = *(p2++);
        ++p;
      }
      idx += psize;
    } while (p != pp);
    //switch pointers
    pp = data1; data1 = data2; data2 = pp;
    //update slice-size
    slicesize *= 2;
  }
}
```

Časť hlavného kódu - jeden procesor:

```
    //refresh array to be random again
    memcpy (data1, origdata, DATASIZE );
    for (int p = 0; p < DATASIZE ; p += slicesize2)
      mergesort (&data1[p], &data2[p], slicesize2, iter1);
    //ensure that data1 is the partially sorted array
    if (iter1 % 2 == 1) {
      TYPE *pom = data1;
      data1 = data2; data2 = pom;
    }
    mergesort (data1, data2, DATASIZE , ITER - iter1, slicesize2);
    //ensure that sorted data is now in array data1
    if ((ITER - iter1) % 2 == 1) {
      TYPE *pom = data1;
      data1 = data2; data2 = pom;
    }
```

Časť hlavného kódu - OpenMP:

```
    memcpy (data1, origdata, DATASIZE );
    for (int p = 0; p < DATASIZE ; p += slicesize2)
#pragma omp parallel shared (data1, data2)
    {
      int num = omp_get_num_threads();
      if (num != 2) return;  //require 2 threads
      int tid = omp_get_thread_num();
      int offset = tid * slicesize2 / 2;  //0 or half-of-array
      if (!problem)
        mergesort (&data1[p]+offset, &data2[p]+offset, slicesize2 / 2, ite1);
    }
    //ensure that data1 is the partially sorted array
    if (ite1 % 2 == 1)
    {
      TYPE *pom = data1;
      data1 = data2; data2 = pom;
    }
    mergesort (data1, data2, DATASIZE , ITER - ite1, slicesize2 / 2);
```

```
    //ensure that sorted data is now in array data1
    if ((ITER - ite1) % 2 == 1)
    {
      TYPE *pom = data1;
      data1 = data2; data2 = pom;
    }
```

Časť hlavného kódu – MPI:

```
  //initialize MPI
  MPI_Init (&argc, &argv);
  int rank, size;
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);
  MPI_Comm_size (MPI_COMM_WORLD, &size);
  int SLAVES = size - 1;

  if (size != 4)
    return;
[časť kódu vynechaná]
  if (master)
  {
    //generate random data
    preparedata ();
    //give data to slaves
    for (int i = 1; i <= SLAVES; i++) {
      TYPE *datastart = &data1[PROC_DATASIZE * i];
      MPI_Send ((char *) datastart, PROC_DATASIZE * sizeof (TYPE), MPI_CHAR,
          i, 0, MPI_COMM_WORLD);
    }
  }
  else
    //accept data from parent
    MPI_Recv ((char *) data1, PROC_DATASIZE * sizeof (TYPE), MPI_CHAR, 0, 0,
        MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  for (int p = 0; p < PROC_DATASIZE; p += SLICESIZE)
    mergesort (&data1[p], &data2[p], SLICESIZE, SLICEITER);
  //ensure that data1 is the partially sorted array
  if (SLICEITER % 2 == 1) {
    TYPE *pom = data1;
    data1 = data2; data2 = pom;
  }

  mergesort (data1, data2, PROC_DATASIZE, ITER - MASTERITER - SLICEITER, SLICESIZE);
  //ensure that sorted data is now in array data1
  if ((ITER - MASTERITER - SLICEITER) % 2 == 1) {
    TYPE *pom = data1;
    data1 = data2; data2 = pom;
  }
  // merge parts from all processors
  if (master) {
    //master waits for data from all slaves, then performs the final sort
    for (int i = 1; i <= SLAVES; i++) {
      TYPE *datastart = &data1[PROC_DATASIZE * i];
      MPI_Recv ((char *) datastart, PROC_DATASIZE * sizeof (TYPE), MPI_CHAR,
          i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }
    //final sort...
    mergesort (data1, data2, DATASIZE, MASTERITER, PROC_DATASIZE);
    //ensure that sorted data is now in array data1
    if (MASTERITER % 2 == 1) {
      TYPE *pom = data1;
      data1 = data2; data2 = pom;
    }
  }
  else {
    //slave sends data to the master and quits
    MPI_Send ((char *) data1, PROC_DATASIZE * sizeof (TYPE), MPI_CHAR,
        0, 0, MPI_COMM_WORLD);
    MPI_Finalize ();
    return 0;
  }
```