Parallel implementations of matrix multiplication

Mandatory assignment No. 2 of INF3380

Each student should work independently and submit at Devilry her/his parallel programs (with ample comments) before the deadline.

Matrix multiplication

Multiplication between matrix A (of dimension $m \times \ell$) and matrix B (of dimension $\ell \times n$) will produce a result matrix C (of dimension $m \times n$). More specifically, each element of the result matrix is calculated as

$$c_{i,j} = a_{i,0}b_{0,j} + a_{i,1}b_{1,j} + \ldots + a_{i,\ell-1}b_{\ell-1,j} = \sum_{k=0}^{\ell-1} a_{i,k}b_{k,j}.$$

Task 1: MPI implementation

Note: Each student only needs to program one MPI implementation out of the following two alternatives.

Alternative one: row-wise 1D block partitioning

The starting point for the parallelization is that both the A and B matrices are partitioned by a row-wise 1D block partitioning among the MPI processes. As result of the parallel computation, the C matrix is also distributed among the MPI processes by a row-wise 1D block partitioning. It should be generally assumed that the number of rows in the three matrices, m or ℓ, may not be divisible by the number of MPI processes.

Alternative two: 2D block partitioning

As described in Chapter 8.2.2 of the textbook, *A. Grama, G. Karypis, V. Kumar and A. Gupta*, **Introduction to Parallel Computing**, you can parallelize the matrix multiplication by Cannon's algorithm.

Note that we now assume a general case where the matrices are rectangular (thus not necessarily square). You can, however, assume that the square root of the number of MPI processes is an integer, so that all the three matrices are to be partitioned into an equal number of blocks in the horizontal and vertical directions. Last but not least, the numbers of rows and columns of the A, B, C matrices may not be perfectly divisible by the number of blocks in both directions.

Common requirements for both alternatives

An MPI program should be implemented such that it can

- accept two file names at run-time,
- let process 0 read the A and B matrices from the two data files,
- let process 0 distribute the pieces of A and B, by either a 1D or a 2D partitioning, to all the other processes,
- calculate $C = A * B$ in parallel,
- let process 0 gather, from all the other processes, the different pieces of C,
- let process 0 write out the entire C matrix to an output data file.

Task 2: OpenMP-MPI implementation

The student should extend her/his MPI program from Task 1, so that OpenMP is used within each MPI process for the computation-intensive parts.

Input of matrix

For the sake of I/O efficiency, it is assumed that the A and B matrices are stored in binary formatted data files. More specifically, the following function can be used to read in a matrix stored in a binary file:

```
void read_matrix_binaryformat (char* filename, double*** matrix,
                               int* num_rows, int* num_cols)
{
  int i;
  FILE* fp = fopen (filename,"rb");
  fread (num_rows, sizeof(int), 1, fp);
  fread (num_cols, sizeof(int), 1, fp);

  /* storage allocation of the matrix */
  *matrix = (double**)malloc((*num_rows)*sizeof(double*));
  (*matrix)[0] = (double*)malloc((*num_rows)*(*num_cols)*sizeof(double));
  for (i=1; i<(*num_rows); i++)
    (*matrix)[i] = (*matrix)[i-1]+(*num_cols);

  /* read in the entire matrix */
  fread ((*matrix)[0], sizeof(double), (*num_rows)*(*num_cols), fp);
  fclose (fp);
}
```

For example, suppose the following three variables are declared:

```
double **matrix;
int num_rows;
int num_cols;
```

Then, a matrix stored in file `mat.bin` can be read in by calling `read_matrix_binaryformat` as follows:

```
read_matrix_binaryformat ("mat.bin", &matrix, &num_rows, &num_cols);
```

Output of matrix

Similarly, the multiplication result matrix C should be written to file in binary format by using the following function:

```
void write_matrix_binaryformat (char* filename, double** matrix,
                                int num_rows, int num_cols)
{
  FILE *fp = fopen (filename,"wb");
  fwrite (&num_rows, sizeof(int), 1, fp);
  fwrite (&num_cols, sizeof(int), 1, fp);
  fwrite (matrix[0], sizeof(double), num_rows*num_cols, fp);
  fclose (fp);
}
```

Examples of A and B matrices

From the website of INF3380, the following matrices (in binary format) can be downloaded for code debugging and testing:

`small_matrix_a.bin`	of dimension 100×50
`small_matrix_b.bin`	of dimension 50×100
`large_matrix_a.bin`	of dimension 1000×500
`large_matrix_b.bin`	of dimension 500×1000