AN MPI IMPLEMENTATION OF A SELF-SUBMITTING PARALLEL JOB QUEUE

JOHN M. NEUBERGER, NÁNDOR SIEBEN, AND JAMES W. SWIFT

Matrix square example: `square.cpp`

- line 4: There are three types of jobs in this example. The job types need to be positive integers, so *NONE* is added to take the unused value of zero.
- line 6: The variable *matrix* contains the input matrix. The variable *square* contains the square of the input matrix, which is the result of the program.
- line 7–10: This data type is used to send back one row of the result matrix, together with the position of this row.
- line 12–16: The BSL requires a simple template function for the serialization of each structure.

MPQswitch:

- line 18: This function is executed by the workers with the *job.type* variable set to *DATA* or *MULTIPLY*. It is also executed by the boss with *job.type* set to *RESULT*.
- line 19: The variable *data* contains one row of the result produced by a worker.
- line 20: Decide the type of the current job.
- lines 21–23: The input matrix is shared by all the workers. Each worker receives the matrix as a *DATA* job. The workers deserialize it and store it locally in the variable *matrix*.
- line 24–29: A worker calculates one row of the goal matrix.
- line 25: The input is deserialized into *data.pos*. It tells the worker which row to compute.
- line 26: The calculated row is stored in *data.result*.
- lines 27–29: The row is calculated using standard matrix multiplication.
- lines 30–31: The worker sends the calculated row to the boss.
- line 33–36: The boss receives a row and puts in into the result matrix *square*.
- line 34: The row is deserialized.
- line 35: The row is placed at the appropriate location.

main:

- line 41: The MPI is initialized.
- line 42: The nodes are split into one boss and several workers.
- line 43: The example input matrix contains rows of length 10 containing a 1 at each position.
- line 44: The input variable *matrix* is initialized.
- line 45: The output variable *square* is resized to the correct dimensions.
- line 46: The input matrix is sent to all the workers
- lines 48–49: The job queue is filled with *MULTIPLY* jobs, each requesting the calculation of one row of the goal matrix.
- line 50: The boss starts the supervision of the workers. At the end of this work, the goal matrix *square* will have been calculated. No other result is created, so *outqueue* is empty.
- lines 51–55: The goal matrix is printed.
- line 56: The boss halts all the workers.

Department of Mathematics and Statistics, Northern Arizona University, Flagstaff, AZ 86011-5717, USA
E-mail address: `john.neuberger@nau.edu, nandor.sieben@nau.edu, jim.swift@nau.edu`

```
#include "MPQueue.h"
using namespace std;

enum { NONE, DATA, MULTIPLY, RESULT };
typedef vector < int > Trow;
vector < Trow > matrix, square;
typedef struct {
  int pos;                                       // row index
  Trow result;                                   // output row
} Tdata;

template < class Archive > void                  // needed to serialize Tdata
serialize (Archive & ar, Tdata & data, const unsigned int version) {
  ar & data.pos;
  ar & data.result;
}

void MPQswitch (Tjob & job) {
  Tdata data;
  switch (job.type) {
  case DATA:                                     // receive the input matrix
    from_string (matrix, job.data);
    break;
  case MULTIPLY:
    from_string (data.pos, job.data);            // get the row position
    data.result = Trow (matrix.size (), 0);
    for (int j = 0; j < matrix.size (); j++)     // calutate one row
      for (int k = 0; k < matrix.size (); k++)
        data.result[j] += matrix[data.pos][k] * matrix[k][j];
    job = Tjob(RESULT, data);                    // prepare the result
    MPQtask (job);                               // send result to boss
    break;
  case RESULT:                                   // receive one output row
    from_string (data, job.data);
    square[data.pos] = data.result;
    job.data = "";                               // nothing to return
  }
}

int main (int argc, char *argv[]) {
  MPQinit (argc, argv);
  MPQstart ();
  Trow row (10, 1);                          // input matrix containing
  vector < Trow > mat (row.size (), row);    // 1 at every entry
  square.resize (row.size ());               // container for the output
  MPQsharedata (Tjob(DATA, mat));            // input matrix sent to workers
  Tjobqueue inqueue, outqueue;
  for (int i = 0; i < mat.size (); i++)      // every row is a separate job
    inqueue.push (Tjob(MULTIPLY, i));
  MPQrunjobs (inqueue, outqueue);            // run the jobs
  for (int i = 0; i < row.size (); i++) {    // print the output matrix
    for (int j = 0; j < row.size (); j++)
      cout << square[i][j] << "␣";
    cout << "\n";
  }
  MPQstop ();
}
```