AN MPI IMPLEMENTATION OF A SELF-SUBMITTING PARALLEL JOB QUEUE

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

NON-ATTACKING QUEENS EXAMPLE: `queens.cpp`

- line 6: A placement of queens is stored in a vector. In a valid placement, every column contains exactly one queen. The i-th entry of the vector contains the location of the queen in the i-th column.
- line 7: The total number of solutions is stored in this global variable.
- line 9: We work on a 20×20 board.

fits:

- lines 11–17: This function checks if the queen in the last column of a placement interferes with the other queens.

MPQswitch:

- line 21: Each worker keeps a local job queue of partial placements.
- line 22: Solutions found by a worker are counted locally.
- line 24: A worker receives a partial placement and tries to extend it.
- line 25: The input is deserialized.
- line 26: The initial job received from the boss is stored as the first job in the local job queue.
- line 27: The worker finishes all the jobs in the local job queue.
- lines 28–29: The worker receives the first job in the local job queue.
- line 30: The job contains a partial placement of queens. The worker tries to add a queen in every possible position of the next column.
- line 31: If this particular placement of the new queen does not interfere with the other queens, then this new placement is a possible extension of the original placement.
- lines 32–33: If the extended placement is a full placement, then this is a new solution.
- line 35: The extended placement is not yet a full placement, so it is stored as a new job in the local job queue.
- lines 36–39: If the local job queue size is large enough, then it is time to submit one of the local jobs to the global job queue.
- lines 42–43: When the local job queue is empty, the worker sends the number of solutions it found to the boss node. This method is more efficient than the use of the output queue would be, since there is a large number of results and we are only interested in the sum of them.
- lines 45–48: The boss receives a result from a worker and updates the total number of solutions.

main:

- line 56: The job queue originally contains an empty placement containing no queens.

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"
#include <deque>
using namespace std;

enum { NONE, PLACE, RESULT };
typedef vector < int > Trow;
unsigned long int allsolutions = 0;                 // total number solutions
int overflow = 70;                                  // controlls local queue size
const int size = 20;                                // size of the board

bool inline fits (const Trow & row) {
  int j = row.size () - 1;
  for (int i = 0; i < j; i++)
    if ( ( row[i] == row.back () ) || ( abs (row[i] - row[j]) == j - i ) )
      return false;                                 // queens interfere
  return true;                                      // last queen fits
}

void MPQswitch (Tjob & job) {
  Trow row;                                         // a partial placement
  deque < Trow > rows;                              // local job queue
  unsigned int solutions = 0;                       // local number of solutions
  switch (job.type) {
  case PLACE:                                       // add queen in next column
    from_string (row, job.data);
    rows.push_back (row);                           // populate a local job queue
    while (!rows.empty ()) {                        // still local jobs to do
      row = rows.back ();
      rows.pop_back ();
      for (row.push_back (0); row.back () < size; row.back ()++)
        if (fits (row))                             // does the new queen fit?
          if (row.size () == size)                  // all queens added
            solutions++;                            // found a new solution
          else {
            rows.push_back (row);                   // add to local job queue
            if (rows.size () > overflow) {          // if too many local jobs
              MPQsubmit (Tjob(PLACE, rows.front ())); // send one to the boss
              rows.pop_front ();
            }
          }
    }
    job = Tjob(RESULT, solutions);                  // prepare the result
    MPQtask (job);                                  // send result to the boss
    break;
  case RESULT:
    from_string (solutions, job.data);              // update total solutions
    allsolutions += solutions;                      // with new result
    job.data = "";                                  // no result to return
  }
}

int main (int argc, char *argv[]) {
  MPQinit (argc, argv);
  MPQstart ();
  Tjobqueue inqueue, outqueue;
  inqueue.push (Tjob(PLACE, Trow ()));              // initial empty placement
  MPQrunjobs (inqueue, outqueue);
  cout << allsolutions << " solutions\n";
  MPQstop ();
}
```