AN MPI IMPLEMENTATION OF A SELF-SUBMITTING PARALLEL JOB QUEUE

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

Factoring example: `factor.cpp`

- line 1: We include the MPQueue header file. This also includes all the header files needed to use MPI and the required parts of the Boost Serialization Library.
- line 5: Every job type has a positive integer identifier. In this simple example we only have one type of job.

MPQswitch:

- line 8: This function is called every time a new job needs to be processed. In this example, the function is only executed by worker nodes. The *job.type* variable determines the type of the job, although in this case there is only one type. Typically a program has several kinds of jobs and so this function contains a switch statement. The *job.data* variable contains the serialized input of the job at the time the function is called. This variable is replaced by the serialized output of the job. The output is the same as the input if the input is a prime, otherwise the output is empty.
- lines 9–10: The input is deserialized into the variable x.
- line 11: The worker calculates the square root of x. We hope to split x into the product of two integers that are as large as possible. Thus, the search for these factors starts at the square root of x.
- line 12: The loop tries to find a factor y of x.
- line 13: Check if we have found a factor.
- lines 14–15: The worker has found a factor y. The worker submits two new jobs to the current job queue to split the two factors y and y/x.
- line 16: The split produces no output since the further splitting of the found factors is going to be done by other workers. Hence the output is set to empty.
- line 17: The job is done.
- line 19: We did not find any divisors so x is a prime. Thus the unchanged *job.data* containing x is returned to the boss node as the result of the job.

main:

- line 21: The main function is executed by every node.
- line 22: The MPI is initialized.
- line 23: The nodes are split into one boss and several workers. Only the boss returns from this function call. The workers are ready to accept jobs.
- line 24: The boss creates two job queues, one for storing jobs to do, and another for storing results.
- line 25: The goal is to find the prime factorization of this number.
- line 26: The boss places one splitting job into the job queue.
- line 27: The boss starts the supervision of the workers. The workers split numbers into factors and submit these factors to *inqueue* for further splitting. The workers return the prime factors, which the boss collects in the *outqueue*. The boss node spends the vast majority of its running time in this function.

- line 29: The boss retrieves all the prime factors from the output queue.
- line 30–33: One of the prime factors is retrieved and printed.
- line 35: The boss halts all the workers.

```
 1 #include "MPQueue.h"
 2 #include "math.h"
 3 using namespace std;
 4
 5 const int SPLIT = 1;                     // only one job type, so no
 6                                          // cases in MPQswitch
 7
 8 void MPQswitch (Tjob & job) {
 9   int x;
10   from_string (x, job.data);
11   int sqt = int (sqrt (double (x)));
12   for (int y = sqt; y > 1; y--)          // search for divisors
13     if (0 == x % y) {                    // found a divisor
14       MPQsubmit (Tjob(SPLIT, x/y));      // submit two new jobs
15       MPQsubmit (Tjob(SPLIT, y));
16       job.data = "";                     // no output to return
17       return;
18     }
19 }
20
21 int main (int argc, char *argv[]) {
22   MPQinit (argc, argv);
23   MPQstart ();                           // only the boss returns
24   Tjobqueue inqueue, outqueue;
25   int x = 1120581000;                    // factor this number
26   inqueue.push(Tjob(SPLIT,x));           // add one job to the job queue
27   MPQrunjobs (inqueue, outqueue);        // supervise queue processing
28   cout << x << "␣factors␣as:\n";
29   while (!outqueue.empty ()) {           // get the results
30     int factor;                          // one factor
31     from_string(factor, outqueue.front().data);
32     cout << factor << "␣";
33     outqueue.pop();
34   }
35   MPQstop ();
36 }
```

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