cs6550

Brett Bonar

November 1, 2018

1 Implementation

The program splits the problem set into an equal number of rows for each process. Each process individually creates a two dimensional array of numbers representing the world and randomly populates its piece of the world. A 1 represents a living cell while a 0 represents a dead cell.

However, since cells at the edge of a process's boundaries will need to know about cells in the adjacent process to count the correct number of neighbors, each process sends its first and last rows to adjacent processes at the beginning of each iteration (with the exception of process 0 and the last process, which send only the first or last row).

After each process updates all the cells in its piece of the world, all of the results are gathered into the root process which writes an image to a PBM file as shown in Figure 1.

2 Results

Timing results including gathering and writing the entire state of the world to a PBM file at the end of each iteration:

- Average 10.15 seconds with 1 process
- Average 8.8 seconds with 2 processes
- Average 8.13 seconds with 4 processes
- Average 7.85 seconds with 8 processes

The time goes up when using more than 8 processes since it is now going over the number of available processors on the machine.

The performance improvement with additional processors is greatly mitigated by the requirement to transmit the results of each iteration to the root process for printing. This could be fixed by having each process write just its piece and concatenating files together at a later point or by disabling the final write step altogether.

The timing and improvement with additional processors is much better when omitting the final gather and write step:

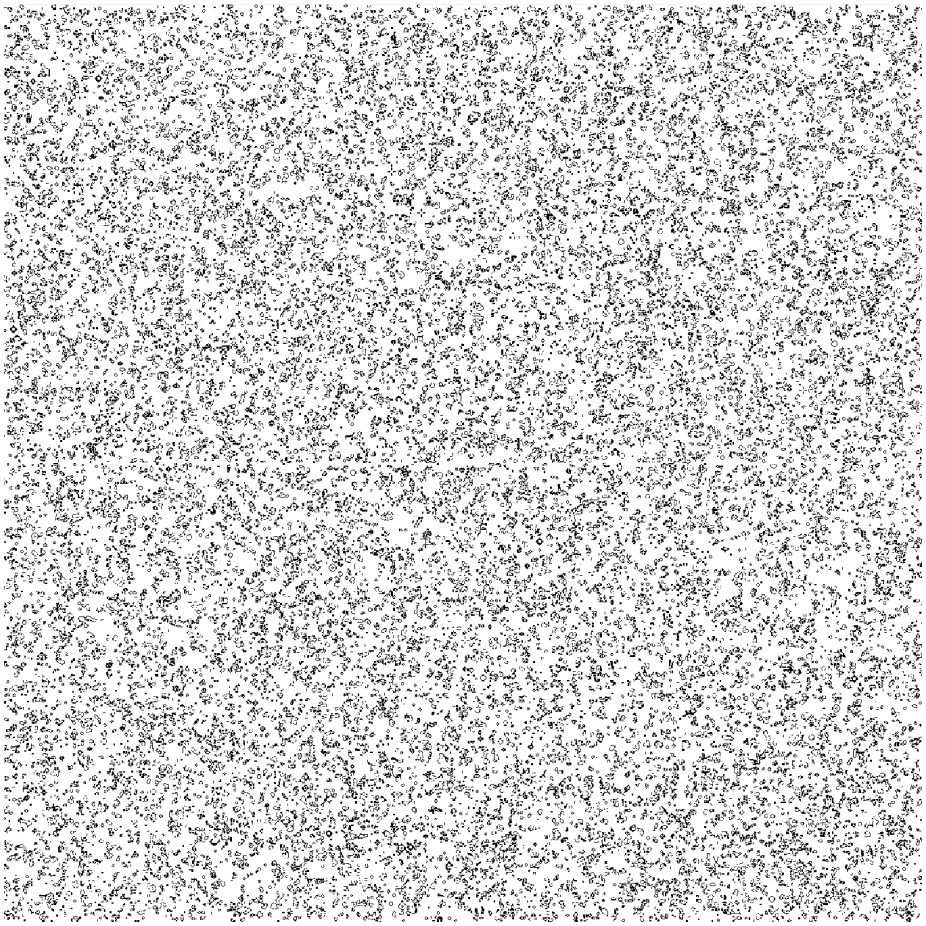

Figure 1: Game of Life Output

- Average 3 seconds with 1 process.
- Average 1.65 seconds with 2 processes.
- Average 0.96 seconds with 4 processes.
- Average 0.65 seconds with 8 processes.

3 Code

```
#include <iostream>
#include <unistd.h>
#include <cmath>
#include <mpi.h>
#include <string>
#include <vector>
#include <ctime>
#include <iostream>
#include <fstream>
#define MCW MPI_COMM_WORLD

const int WORLD_SIZE = 1024;

int printWorld(int world[], int it)
{
  std::ofstream file;
  file.open(std::to_string(it) + ".pbm");
  file << "P1" << std::endl;
  file << WORLD_SIZE << " " << WORLD_SIZE << std::endl;

  for(int i = 0; i < WORLD_SIZE; ++i)
  {
    for(int j = 0; j < WORLD_SIZE; ++j)
    {
      file << world[i * WORLD_SIZE + j] << " ";
    }
    file << std::endl;
  }
}

int countNeighbors(int world[][WORLD_SIZE], int x, int y, int
    localSize,
  int front[], int back[])
{
  int count = 0;
  if (x > 0)
  {
    count += world[x - 1][y];
    if (y > 0)
    {
      count += world[x - 1][y - 1];
    }
    else if (front)
    {
      count += front[x - 1];
```

```
45      }
46
47      if (y < WORLD_SIZE - 1)
48      {
49        count += world[x - 1][y + 1];
50      }
51      else if (back)
52      {
53        count += back[x - 1];
54      }
55    }
56    if (x < localSize - 1)
57    {
58      count += world[x + 1][y];
59
60      if (y > 0)
61      {
62        count += world[x + 1][y - 1];
63      }
64      else if (front)
65      {
66        count += front[x + 1];
67      }
68
69      if (y < WORLD_SIZE - 1)
70      {
71        count += world[x + 1][y + 1];
72      }
73      else if (back)
74      {
75        count += back[x + 1];
76      }
77    }
78
79    if (y > 0)
80    {
81      count += world[x][y - 1];
82    }
83    else if (front)
84    {
85      count += front[x];
86    }
87
88    if (y < WORLD_SIZE - 1)
89    {
90      count += world[x][y + 1];
91    }
92    else if (back)
93    {
94      count += back[x];
95    }
96
97    return count;
98  }
99
100 int updateCell(int world[][WORLD_SIZE], int x, int y, int localSize
      ,
```

```
101   int front[], int back[])
102 {
103   int neighbors = countNeighbors(world, x, y, localSize, front,
          back);
104   // if (world[x][y])
105   // {
106   //   std::cerr << neighbors << std::endl;
107   // }
108
109   if (world[x][y] == 1)
110   {
111     if (neighbors <= 1 || neighbors >= 4)
112     {
113       return 0;
114     }
115     return 1;
116   }
117   else
118   {
119     if (neighbors == 3)
120     {
121       return 1;
122     }
123     return 0;
124   }
125
126   return 1;
127 }
128
129 void updateWorld(int world[][WORLD_SIZE], int targetWorld[][
        WORLD_SIZE], int localSize,
130   int front[], int back[])
131 {
132   for (int x = 0; x < localSize; x++)
133   {
134     for (int y = 0; y < WORLD_SIZE; y++)
135     {
136       targetWorld[x][y] = updateCell(world, x, y, localSize, front,
              back);
137     }
138   }
139 }
140
141 int main(int argc, char **argv){
142   int rank, size;
143   int data;
144   auto world = new int[WORLD_SIZE * WORLD_SIZE]();
145   int iterations = 100;
146 -
147   MPI_Init(&argc, &argv);
148   MPI_Comm_rank(MCW, &rank);
149   MPI_Comm_size(MCW, &size);
150
151   int localSize = WORLD_SIZE / size;
152   int frontTag = 0;
153   int backTag = 1;
154
```

```
155   auto sourceWorld = new int[localSize][WORLD_SIZE]();
156   auto targetWorld = new int[localSize][WORLD_SIZE]();
157   std::cerr << "Rank: " << rank << ", Local Size: " << localSize <<
          std::endl;
158
159   srand(rank * time(NULL));
160   for (int x = 0; x < localSize; x++)
161   {
162     for (int y = 0; y < WORLD_SIZE; y++)
163     {
164       if (rand() % 5 == 0)
165       {
166         sourceWorld[x][y] = 1;
167       }
168     }
169   }
170
171   int* front = nullptr;
172   int* back = nullptr;
173
174   if (rank > 0)
175   {
176     front = new int[WORLD_SIZE];
177   }
178   if (rank < size - 1)
179   {
180     back = new int[WORLD_SIZE];
181   }
182
183   int displacements[size];
184   int recvCounts[size];
185   for (int i = 0; i < size; i++)
186   {
187     recvCounts[i] = WORLD_SIZE;
188   }
189
190   for (int i = 0; i < localSize; i++)
191   {
192     for (int p = 0; p < size; p++)
193     {
194       displacements[p] = p * localSize * WORLD_SIZE + i *
              WORLD_SIZE;
195     }
196
197     MPI_Gatherv(sourceWorld[i], WORLD_SIZE, MPI_INT, world,
            recvCounts,
198       displacements, MPI_INT, 0, MCW);
199   }
200   if (rank == 0)
201   {
202     printWorld(world, 0);
203   }
204
205   for (int i = 1; i < iterations; i++)
206   {
207     if (rank == 0)
208     {
```

```
      std::cerr << "Start iteration " << i << std::endl;
    }
    if (rank < size - 1)
    {
      MPI_Send(sourceWorld[localSize - 1], WORLD_SIZE , MPI_INT,
          rank + 1, frontTag, MCW);
    }
    if (rank > 0)
    {
      MPI_Recv(front, WORLD_SIZE, MPI_INT, rank - 1, frontTag, MCW,
           MPI_STATUS_IGNORE);
    }

    if (rank > 0)
    {
      MPI_Send(&sourceWorld[0], WORLD_SIZE, MPI_INT, rank - 1,
          backTag, MCW);
    }
    if (rank < size - 1)
    {
      MPI_Recv(back, WORLD_SIZE, MPI_INT, rank + 1, backTag, MCW,
          MPI_STATUS_IGNORE);
    }

    updateWorld(sourceWorld, targetWorld, localSize, front, back);
    auto temp = sourceWorld;
    sourceWorld = targetWorld;
    targetWorld = temp;

    for (int i = 0; i < localSize; i++)
    {
      for (int p = 0; p < size; p++)
      {
        displacements[p] = p * localSize * WORLD_SIZE + i *
            WORLD_SIZE;
      }

      MPI_Gatherv(sourceWorld[i], WORLD_SIZE, MPI_INT, world,
          recvCounts,
        displacements, MPI_INT, 0, MCW);
    }
    if (rank == 0)
    {
      printWorld(world, i);
      std::cerr << "End iteration " << i << std::endl;
    }
  }

  MPI_Finalize();

  return 0;
}
```

4 Compile and Run Commands

```
mpic++ Assignment8/Assignment8.cpp -o Assignment8/run.out
time mpirun -np # Assignment8/run.out
```