ACM/CS 114
Parallel algorithms for scientific applications

Michael A. G. Aïvázis

California Institute of Technology

Winter 2012

The implementation of `Jacobi::solve`, part 1

```
1  void Jacobi::solve(Problem & problem) {
2     // initialize the problem
3     problem.initialize();
4
5     // get a reference to the solution grid
6     Grid & current = problem.solution();
7     // build temporary storage for the next iterant
8     Grid next(current.size());
9
10    // put an upper limit on the number of iterations
11    size_t maxIterations = (size_t) 1e4;
12    for (size_t iteration = 0; iteration < maxIterations; iteration++) {
13       // print out a progress repot
14       if ((problem.rank() == 0) && (iteration % 100 == 0)) {
15          std::cout
16             << "jacobi: iteration " << iteration
17             << std::endl;
18       }
19       // enforce the boundary conditions
20       problem.applyBoundaryConditions();
21       // reset the local maximum change
22       double localMax = 0.0;
23       // update the interior of the grid
24       for (size_t j=1; j < next.size()-1; j++) {
25          for (size_t i=1; i < next.size()-1; i++) {
26             // the cell update
27             next(i,j) = .25*(current(i+1,j)+current(i-1,j)+current(i,j+1)+current(i,j-1));
28             // compute the change from the current cell value
29             double dev = std::abs(next(i,j) - current(i,j));
30             // and update the local maximum
31             if (dev > localMax) {
32                localMax = dev;
33             }
34          }
35       } // done with the grid update
```

The implementation of `Jacobi::solve`, part 2

```
36          // swap the blocks of the two grids, leaving the solution in current
37          Grid::swapBlocks(current, next);
38          // compute global maximum deviation
39          double globalMax;
40          MPI_Allreduce(&localMax, &globalMax, 1, MPI_DOUBLE, MPI_MAX, problem.communicator());
41          // convergence check
42          if (globalMax < _tolerance) {
43             if (problem.rank() == 0) {
44                std::cout
45                   << "jacobi: convergence in " << iteration << " iterations"
46                   << std::endl;
47             }
48             break;
49          }
50          // otherwise
51       }
52       // when we get here, either we have converged or ran out of iterations
53       // update the fringe of the current grid
54       problem.applyBoundaryConditions();
55       // all done
56       return;
57    }
```

Boundary conditions and data exchanges, part 1

```
1  void Example::applyBoundaryConditions() {
2     // a reference to my grid
3     Grid & g = _solution;
4     // my rank;
5     int rank = _rank;
6     // the ranks of my four neighbors
7     int top, right, bottom, left;
8     // get them
9     MPI_Cart_shift(_cartesian, 1, 1, &rank, &top);
10    MPI_Cart_shift(_cartesian, 0, 1, &rank, &right);
11    MPI_Cart_shift(_cartesian, 1, -1, &rank, &bottom);
12    MPI_Cart_shift(_cartesian, 0, -1, &rank, &left);
13
14    // allocate send and receive buffers
15    double * sendbuf = new double[g.size()];
16    double * recvbuf = new double[g.size()];
```

Boundary conditions and data exchanges, part 2

```
17    // shift to the right
18    // fill my sendbuf with my RIGHT DATA BORDER
19    for (size_t cell=0; cell < g.size(); cell++) {
20       sendbuf[cell] = g(g.size()-2, cell);
21    }
22    // do the shift
23    MPI_Sendrecv(
24              sendbuf, g.size(), MPI_DOUBLE, right, 17,
25              recvbuf, g.size(), MPI_DOUBLE, left, 17,
26              _cartesian, MPI_STATUS_IGNORE
27              );
28    if (left == MPI_PROC_NULL) {
29       // if i am on the boundary, paint the dirichlet conditions
30       for (size_t cell=0; cell < g.size(); cell++) {
31          g(0, cell) = 0;
32       }
33    } else {
34       // fill my LEFT FRINGE with the received data
35       for (size_t cell=0; cell < g.size(); cell++) {
36          g(0, cell) = recvbuf[cell];
37       }
38    }
```

Boundary conditions and data exchanges, part 4

```
39    // shift to the left
40    // fill my sendbuf with my LEFT DATA BORDER
41    for (size_t cell=0; cell < g.size(); cell++) {
42       sendbuf[cell] = g(1, cell);
43    }
44    // do the shift
45    MPI_Sendrecv(
46              sendbuf, g.size(), MPI_DOUBLE, left, 17,
47              recvbuf, g.size(), MPI_DOUBLE, right, 17,
48              _cartesian, MPI_STATUS_IGNORE
49              );
50    if (right == MPI_PROC_NULL) {
51       // if i am on the boundary, paint the dirichlet conditions
52       for (size_t cell=0; cell < g.size(); cell++) {
53          g(g.size()-1, cell) = 0;
54       }
55    } else {
56       // fill my RIGHT FRINGE with the received data
57       for (size_t cell=0; cell < g.size(); cell++) {
58          g(g.size()-1, cell) = recvbuf[cell];
59       }
60    }
```

Boundary conditions and data exchanges, part 5

```
62    // shift up
63    // fill my sendbuf with my TOP DATA BORDER
64    for (size_t cell=0; cell < g.size(); cell++) {
65       sendbuf[cell] = g(cell, g.size()-2);
66    }
67    // do the shift
68    MPI_Sendrecv(
69              sendbuf, g.size(), MPI_DOUBLE, top, 17,
70              recvbuf, g.size(), MPI_DOUBLE, bottom, 17,
71              _cartesian, MPI_STATUS_IGNORE
72              );
73    if (bottom == MPI_PROC_NULL) {
74       // if i am on the boundary, paint the dirichlet conditions
75       for (size_t cell=0; cell < g.size(); cell++) {
76          g(cell, 0) = std::sin((_x0 + cell*_delta)*pi);
77       }
78    } else {
79       // fill my BOTTOM FRINGE with the received data
80       for (size_t cell=0; cell < g.size(); cell++) {
81          g(cell, 0) = recvbuf[cell];
82       }
83    }
```

Boundary conditions and data exchanges, part 6

```
84    // shift down
85    // fill my sendbuf with my BOTTOM DATA BORDER
86    for (size_t cell=0; cell < g.size(); cell++) {
87       sendbuf[cell] = g(cell, 1);
88    }
89    // do the shift
90    MPI_Sendrecv(
91              sendbuf, g.size(), MPI_DOUBLE, bottom, 17,
92              recvbuf, g.size(), MPI_DOUBLE, top, 17,
93              _cartesian, MPI_STATUS_IGNORE
94              );
95    if (top == MPI_PROC_NULL) {
96       // if i am on the boundary, paint the dirichlet conditions
97       for (size_t cell=0; cell < g.size(); cell++) {
98          g(cell, g.size()-1) =
99             std::sin((_x0 + cell*_delta)*pi) * std::exp(-pi);
100      }
101   } else {
102      // fill my TOP FRINGE with the received data
103      for (size_t cell=0; cell < g.size(); cell++) {
104         g(cell, g.size()-1) = recvbuf[cell];
105      }
106   }
107
108   return;
109}
```