Отчёт по курсу «Высокопроизводительные параллельные вычисления на кластерных системах».

Владислав Соврасов
аспирант гр. 2-о-051318

1 Постановка задачи

Требуется получить эффективную параллельную версию солвера MIDACO (http://www.midaco-solver.com), которая бы эффективно работала как на распределённых системах, так и на системах с общей паматью.

MIDACO (Mixed Integer Distributed Ant Colony Optimization) предназначен для решения глобальной оптимизации как с дискретными, так и с непрерывными параметрами.

Будем рассматривать задачу глобальной оптимизации в неперрывном многомерном пространстве:

$$\varphi(y^*) = \min\{\varphi(y) : y \in D\}, D = \{y \in \mathbf{R}^N : a_i \leqslant x_i \leqslant b_i, 1 \leqslant i \leqslant N\}$$

2 Реализация

3 Результаты

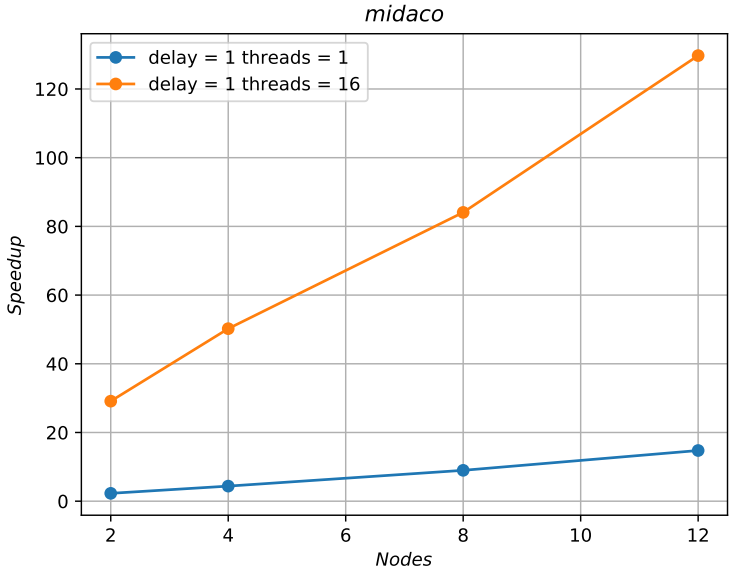

Рис. 1

4 Исходный код

```
#include <mpi.h>
#include <algorithm>
#include "midaco_mpi.hpp"

#include <midaco_core.h>
#include <omp.h>

MidacoSolution solve_midaco_mpi(const IGOProblem<double>* problem,
                                const MidacoMPIParameters& params,
                                std::function<bool(const double*)> external_stop)
{
  MidacoSolution solution;

  int proc, nprocs;
  MPI_Status status;
  MPI_Comm_rank( MPI_COMM_WORLD, &proc );
  MPI_Comm_size( MPI_COMM_WORLD, &nprocs );

  long int o,n,ni,m,me,maxeval,maxtime,printeval,save2file,iflag,istop;
```

```
20    long int liw,lrw,lpf,i,iw[5000],p=1; double rw[20000],pf[20000],param[13];
21    char key[] = "MIDACO_LIMITED_VERSION___[CREATIVE_COMMONS_BY-NC-ND_LICENSE]";
22
23    o  = 1; /* Number of objectives */
24    n  = problem->GetDimension(); /* Number of variables (in total) */
25    ni = 0; /* Number of integer variables (0 <= ni <= n)     */
26    m  = problem->GetConstraintsNumber(); /* Number of constraints (in total) */
27    me = 0; /* Number of equality constraints (0 <= me <= m) */
28
29    double* f = new double[o*params.numThreads];
30    double* g = new double[m*params.numThreads];
31    double* x = new double[n*params.numThreads];
32    double* xl = new double[n];
33    double* xu = new double[n];
34
35    problem->GetBounds(xl, xu);
36    std::copy_n(xl, n, x);
37
38    maxeval = params.maxEvals;
39    maxtime = 60*60*24;
40    printeval = 1000;
41    save2file = 0;
42
43    param[ 0] =  0.0;  /* ACCURACY  */
44    param[ 1] =  params.seed;  /* SEED      */
45    param[ 2] =  0.0;  /* FSTOP     */
46    param[ 3] =  0.0;  /* ALGOSTOP  */
47    param[ 4] =  0.0;  /* EVALSTOP  */
48    param[ 5] =  params.focus;  /* FOCUS     */
49    param[ 6] =  0.0;  /* ANTS      */
50    param[ 7] =  0.0;  /* KERNEL    */
51    param[ 8] =  0.0;  /* ORACLE    */
52    param[ 9] =  0.0;  /* PARETOMAX */
53    param[10] =  0.0;  /* EPSILON   */
54    param[11] =  0.0;  /* BALANCE   */
55    param[12] =  0.0;  /* CHARACTER */
56
57    long int num_points = params.numThreads * nprocs;
58    p = nprocs;
59
60    if (proc == 0)
61    {
62      double *xxx,*fff,*ggg;
63      /* Allocate arrays for parallelization */
64      xxx = new double[params.numThreads*p*n];
65      fff = new double[params.numThreads*p*o];
66      ggg = new double[params.numThreads*p*m];
67      /* Store starting point x in xxx array */
68      for(int c=0; c<p*params.numThreads; c++)
```

```
69      {
70        std::copy_n(x, n, xxx + c*n);
71      }
72      lrw=sizeof(rw)/sizeof(double);
73      lpf=sizeof(pf)/sizeof(double);
74      liw=sizeof(iw)/sizeof(long int);
75      /* Print midaco headline and basic information */
76      midaco_print(1,printeval,save2file,&iflag,&istop,&*f,&*g,&*x,&*xl,&*xu,
77                   o,n,ni,m,me,&*rw,&*pf,maxeval,maxtime,&*param,num_points,&*key);
78      int n_evals = 0;
79      while(istop==0) /*~~~ Start of the reverse communication loop ~~~*/
80      {
81          for (int c=2; c<=p; c++) /* Send iterates X for evaluation */
82          {
83            for(int i=0; i<params.numThreads; i++)
84              if (external_stop(xxx + (c-1)*n*params.numThreads + i*n))
85                istop = 1;
86          }
87          MPI_Scatter(xxx, n*params.numThreads, MPI_DOUBLE, x,
88                      n*params.numThreads, MPI_DOUBLE, 0, MPI_COMM_WORLD);
89
90          #pragma omp parallel for num_threads(params.numThreads)
91          for (unsigned t = 0; t < params.numThreads; t++)  {
92            for (int i = 0; i < m; i++)
93              g[t*m + i] = problem->Calculate(xxx + t*n, i);
94            f[t*o] = problem->Calculate(xxx + t*n, m);
95            if (external_stop(xxx + t*n))
96            #pragma omp atomic write
97              istop = 1;
98          }
99
100         /* Collect results F & G */
101         MPI_Gather(f, o*params.numThreads, MPI_DOUBLE, fff, o*params.numThreads,
102                    MPI_DOUBLE, 0, MPI_COMM_WORLD);
103         MPI_Gather(g, m*params.numThreads, MPI_DOUBLE, ggg, m*params.numThreads,
104                    MPI_DOUBLE, 0, MPI_COMM_WORLD);
105
106         n_evals += p*params.numThreads;
107         /* Call MIDACO */
108         midaco(&num_points,&o,&n,&ni,&m,&me,&*xxx,&*fff,&*ggg,&*xl,&*xu,&iflag,
109                &istop,&*param,&*rw,&lrw,&*iw,&liw,&*pf,&lpf,&*key);
110         /* Call MIDACO printing routine */
111         midaco_print(2,printeval,save2file,&iflag,&istop,&*fff,&*ggg,&*xxx,&*xl,&*xu,
112                      o,n,ni,m,me,&*rw,&*pf,maxeval,maxtime,&*param,num_points,&*key);
113         /* Send istop to slave */
114         for (int c=2; c<=p; c++)
115         {
116           MPI_Send( &istop,1, MPI_INTEGER, c-1,4, MPI_COMM_WORLD);
117         }
```

```
118       }
119
120       solution.optValues = std::vector<double>(ggg, ggg + m);
121       solution.optValues.push_back(*fff);
122       solution.optPoint = std::vector<double>(xxx, xxx + n);
123       solution.calcCounters = std::vector<int>(m + 1, n_evals);
124       delete[] xxx;
125       delete[] fff;
126       delete[] ggg;
127     }
128     else
129     {
130      istop = 0;
131      while(istop<=0)
132      {
133        MPI_Scatter(nullptr, n*params.numThreads, MPI_DOUBLE, x,
134          n*params.numThreads, MPI_DOUBLE, 0, MPI_COMM_WORLD);
135
136        #pragma omp parallel for num_threads(params.numThreads)
137        for (unsigned t = 0; t < params.numThreads; t++)  {
138          for (int i = 0; i < m; i++)
139            g[t*m + i] = problem->Calculate(x + t*n, i);
140          f[t*o] = problem->Calculate(x + t*n, m);
141        }
142
143        MPI_Gather(f, o*params.numThreads, MPI_DOUBLE, nullptr, 0,
144                   MPI_DOUBLE, 0, MPI_COMM_WORLD);
145        MPI_Gather(g, m*params.numThreads, MPI_DOUBLE, nullptr, 0,
146                   MPI_DOUBLE, 0, MPI_COMM_WORLD);
147        MPI_Recv( &istop,1,MPI_INTEGER,0,4,MPI_COMM_WORLD, &status );
148      }
149     }
150
151     delete[] f;
152     delete[] g;
153     delete[] x;
154     delete[] xl;
155     delete[] xu;
156
157     return solution;
158   }
```