ParadisEO – PEO : Lesson 1

Note : All the components are not presented in this lesson (binary, topology, asynchronous or synchronous ...). To know the completeness of components refer to API documentation of ParadisEO – EO and ParadisEO – PEO.

Introduction

For high computing-intensive evolutionary algorithms / particle swarm optimization where the fitness evaluation step is non-negligible, a parallel evaluation step of the population may and should significantly improve the execution time. As a general remark, the rapport between communication and fitness evaluation times has to be considered. If the fitness evaluation time and the communication time are comparable, evaluating the fitness in parallel brings less or no improvement - it may even increase the execution time. At the opposite extreme, if the fitness evaluation requires a long time as compared to communication, a significant speed-up should be achieved.

Requirements

Before to start this lesson 1, you should read **Technical Introduction**.

Of course, to execute the lesson 1, you should be in the directory of this lesson.

Problem

In the lesson 1 you can execute two different algorithms with a parallel evaluation function :

- Particle Swarm Optimization (PSO)
- Evolutionary Algorithm (EA)

The problem is the same in the two cases : minimizing the Rosenbrock

function.

$$f(x_1,x_2)=100*(x_2-x_1^2)^2+(1-x_1)^2$$

The optimal is :

$$f(x_{1,}x_2)=0$$
$$\text{with :}\quad X=(1,1)$$

PSO (mainPSO.cpp) :

```
#include <peo>
typedef eoRealParticle < double >Indi;
//Evaluation function
double f (const Indi & _indi)
{
  // Rosenbrock function f(x) = 100*(x[1]-x[0]^2)^2+(1-x[0])^2
    // => optimal : f* = 0 , with x* =(1,1)
    double sum;
    sum=_indi[1]-pow(_indi[0],2);
    sum=100*pow(sum,2);
    sum+=pow((1-_indi[0]),2);
    return (-sum);
}

int main (int __argc, char *__argv[])
{
// Initialization of the parallel environment : thanks this instruction,
   ParadisEO-PEO can initialize himself
    peo :: init( __argc, __argv );
//Parameters
    const unsigned int VEC_SIZE = 2; // Don't change this parameter when you are
                                     // resolving the Rosenbrock function
    const unsigned int POP_SIZE = 20; // As with a sequential algorithm, you change
                                      // the size of the population
    const unsigned int NEIGHBORHOOD_SIZE= 6; // This parameter define the
                                             //neighborhoods in the PSO's topology
    const unsigned int MAX_GEN = 150; // Define the number of maximal generation
    const double INIT_POSITION_MIN = -2.0; // For initialize x
    const double INIT_POSITION_MAX = 2.0;  // In the case of the Rosenbrock
                                           // function : -2 < x[i] < 2
    const double INIT_VELOCITY_MIN = -1.;  // For initialize PSO's velocity
    const double INIT_VELOCITY_MAX = 1.;   // ie Lesson 6 of ParadisEO-EO
    const double C1 = 0.5;  // For calculate the velocity
    const double C2 = 2.;   // ie Lesson 6 of ParadisEO-EO
    rng.reseed (time(0));
// Stopping
    eoGenContinue < Indi > genContPara (MAX_GEN);
    eoCombinedContinue <Indi> continuatorPara (genContPara);
    eoCheckPoint<Indi> checkpoint(continuatorPara);

 /* In this lesson, you should define a peoEvalFunc witch will allow to
```

```
initialize :
  *
  *    - peoSeqPopEval : using to the sequential evaluation
  *
  *   OR
  *
  *    - peoParaPopEval : using to the parallel evaluation
  *
  */

// For a parallel evaluation
    peoEvalFunc<Indi, double, const Indi& > plainEval(f);
    peoParaPopEval< Indi > eval(plainEval);

// Initialization
    eoUniformGenerator < double >uGen (INIT_POSITION_MIN, INIT_POSITION_MAX);
    eoInitFixedLength < Indi > random (VEC_SIZE, uGen);
// Velocity (ie Lesson 6 of ParadisEO-PEO)
    eoUniformGenerator < double >sGen (INIT_VELOCITY_MIN, INIT_VELOCITY_MAX);
    eoVelocityInitFixedLength < Indi > veloRandom (VEC_SIZE, sGen);
// Initializing the best (ie Lesson 6 of ParadisEO-PEO)
    eoFirstIsBestInit < Indi > localInit;
// Flight (ie Lesson 6 of ParadisEO-PEO)
    eoRealVectorBounds bndsFlight(VEC_SIZE,INIT_POSITION_MIN,INIT_POSITION_MAX);
    eoStandardFlight < Indi > flight(bndsFlight);
// Creation of the population
    eoPop < Indi > pop;
    pop.append (POP_SIZE, random);
// Initialization
    peoInitializer <Indi> init(eval,veloRandom,localInit,pop);
// Topology (ie Lesson 6 of ParadisEO-PEO)
    eoLinearTopology<Indi> topology(NEIGHBORHOOD_SIZE);
    eoRealVectorBounds bnds(VEC_SIZE,INIT_VELOCITY_MIN,INIT_VELOCITY_MAX);
    eoStandardVelocity < Indi > velocity (topology,C1,C2,bnds);
//Parallel algorithm
    peoPSO < Indi > psa(init,checkpoint, eval, velocity, flight);
    psa(pop);
    peo :: run();
    peo :: finalize();
    if(getNodeRank()==1)
            std::cout << "Final population :\n" << pop << std::endl;
}
```

EA (mainEA.cpp) :

```
#include <peo>
#include <es.h>
typedef eoReal<double> Indi;
//Evaluation function
double f (const Indi & _indi)
{
    // Rosenbrock function f(x) = 100*(x[1]-x[0]^2)^2+(1-x[0])^2
    // => optimal : f* = 0 , with x* =(1,1)
    double sum;
    sum=_indi[1]-pow(_indi[0],2);
    sum=100*pow(sum,2);
    sum+=pow((1-_indi[0]),2);
    return (-sum);
}
```

```
int main (int __argc, char *__argv[])
{
// Initialization of the parallel environment : thanks this instruction,
// ParadisEO-PEO can initialize himself
    peo :: init( __argc, __argv );

//Parameters
    const unsigned int VEC_SIZE = 2;
    const unsigned int POP_SIZE = 20;
    const unsigned int MAX_GEN = 300;
    const double INIT_POSITION_MIN = -2.0;
    const double INIT_POSITION_MAX = 2.0;
    const float CROSS_RATE = 0.8; // Crossover rate
    const double EPSILON = 0.01; // Range for real uniform mutation
    const float MUT_RATE = 0.3; // Mutation rate
    rng.reseed (time(0));
// Stopping
    eoGenContinue < Indi > genContPara (MAX_GEN);
    eoCombinedContinue <Indi> continuatorPara (genContPara);
    eoCheckPoint<Indi> checkpoint(continuatorPara);

 /* In this lesson, you should define a peoEvalFunc witch will allow to
initialize :
  *
  *    - peoSeqPopEval : using to the sequential evaluation
  *
  *   OR
  *
  *    - peoParaPopEval : using to the parallel evaluation
  *
  */

// For a parallel evaluation
      peoEvalFunc<Indi> plainEval(f);
      peoParaPopEval< Indi > eval(plainEval);


// Initialization
    eoUniformGenerator < double >uGen (INIT_POSITION_MIN, INIT_POSITION_MAX);
    eoInitFixedLength < Indi > random (VEC_SIZE, uGen);
// Selection
    eoRankingSelect<Indi> selectionStrategy;
    eoSelectNumber<Indi> select(selectionStrategy,POP_SIZE);
// Transformation
    eoSegmentCrossover<Indi> crossover; // Crossover
    eoUniformMutation<Indi> mutation(EPSILON); // Mutation
    eoSGATransform<Indi> transform(crossover,CROSS_RATE,mutation,MUT_RATE);
    peoSeqTransform<Indi> eaTransform(transform);
// Replacement
    eoPlusReplacement<Indi> replace;
// Creation of the population
    eoPop < Indi > pop;
    pop.append (POP_SIZE, random);
//Parallel algorithm
    peoEA<Indi> Algo(checkpoint,eval,select,eaTransform,replace);
    Algo(pop);

    peo :: run();
    peo :: finalize();
```

```
    if(getNodeRank()==1)
            std::cout << "Final population :\n" << pop << std::endl;
}
```

Launching the program

Your file should be called mainPSO.cpp or mainEA.cpp - please make sure you do not rename the file (we will be using a pre-built makefile, thus you are required not to change the file names). Please make sure you are in the paradiseo-peo/tutorial/build/Lesson1 directory - you should open a console and you should change your current directory to the one of Lesson1.

Compilation :

- make
- make install

Execution (ie Technical Introduction):

mpiexec -n 4 ./pso @param

or

mpiexec -n 4 ./ea @param