# A REPORT

# ON

# Implementation of parallel computing for Multi-objective optimization using differential evolution

**BY**

**Pavan Kumar Behara**

**5005 9584** **MS in Chemical Engg.**

**Done in partial fulfillment of the Subject CE620**

**High Performance Computing**

**under the guidance of**

**Dr. M. D. Jones**

**University at Buffalo, SUNY**

**December, 2012**

# Contents

## Acknowledgement

I express my deepest gratitude to my course instructor Dr. M. D. Jones, CCR, SUNY UB, for introducing all the programming tools and imparting knowledge regarding the usage in a scientific computation context. I would like to thank Dr. Ashish M. Gujarathi of Chemical Engg., BITS-Pilani, who introduced me to the world of evolutionary algorithms. Based on a previous work of mine in matlab (done under his guidance), I coded the C code of the DE implementation on my own.

## Abstract

Optimization of problems with conflicting objectives consists of a number of solutions which are better in one objective at the cost of the other. Most industrial processes can be optimized which results in greater utilization of the resources available at a minimal cost. In this project the optimization of the design of a welded beam is done using differential evolution technique and the code is parallelized to check whether there's an improvement in execution time due to parallel implementation. The observations are present for a welded beam optimization case study with cost and deflection as the objectives to be minimized.

# Multi Objective Optimization in Engineering Design

## 1. Introduction

Multi-Objective Optimization of engineering processes is being done using various algorithms viz., MODE, NSGA, etc., (Gujarathi & B.V.Babu, December 14-17, 2005) (Gujarathi & Purohit, 2010) which are non-traditional, population based search algorithms. An attempt has been made to check whether parallelizing the code results in a faster solution. The speedup is not so significant when there are too many threads may be because the problem is not so computationally intensive. Many real world multi-objective problems consume hours of computational time for a single run. Those type of complex problems need parallelization of code more.

## 2. Optimization Algorithm in brief

The following are the steps in the simple differential evolution algorithm, which forms a basis for the DE based optimization approaches. (detailed description can be found in references (Gujarathi & B.V.Babu).

1. Selecting the decision variables and the bounds on them.
2. Initializing the population.
3. Generating a trial vector from the population.
4. Evaluating and comparing the costs of trial vector and target vector.
5. Finding the best one with respect to objective function.
6. Replacing the population with the best one.
7. Again iterating from step 3 until the maximum number of generations specified is reached.

The pseudo code of DE used in the present study is taken from B.V.Babu & Jehan, 2003 is given below :

*· Choose a seed for the random number generator.*
*· Initialize the values of D, NP, CR, F and MAXGEN (maximumgeneration).*
*· Initialize all the vectors of the population randomly. The variable are normalized within the bounds. Hence generate a random number between 0 and 1 for all the design variables for initialization.*

*for i = 1 to NP{ for j = 1 to D*
$\mathbf{X}_{i,j}$ *=Lower bound+ random*
*number *( upper bound - lower bound)}*
· *All the vectors generated should satisfy the constraints. Penalty function approach, i.e., penalizing the vector by giving it a large value, is followed only for those vectors, which do not satisfy the constraints.*
· *Evaluate the cost of each vector. Profit here is the value of the objective function to be maximized calculated by a separate function defunct.profit()*
*for i = 1 to NP*
$\mathbf{C}_i$ *= defunct.profit()*
· *Find out the vector with the maximum profit i.e. the best vector so far.*
$\mathbf{C}_{max} = \mathbf{C}_1$ *and best =1*
*for i = 2 to NP*
*{ if (*$\mathbf{C}_i > \mathbf{C}_{max}$*)*
*then* $\mathbf{C}_{min} = \mathbf{C}_i$ *and best = i }*
· *Perform mutation, crossover, selection and evaluation of the objective function for a specified number of generations.*
*While (gen < MAXGEN)*
*{ for i = 1 to NP*
*{*
· *For each vector* $\mathbf{X}_i$ *(target vector), select three distinct vectors* $\mathbf{X}_a$*,* $\mathbf{X}_b$ *and* $\mathbf{X}_c$ *(select five, if two vector differences are to be used) randomly from the current population (primary array) other than the vector* $\mathbf{X}_i$
*do*
*{ r1 = random number * NP*
*r2 = random number * NP*
*r3 = random number * NP*
*} while*
*(r1=i)OR(r2=i)OR(r3=i)OR(r1=r2*
*)OR(r2=r3)OR(r1=r3)*
· *Perform crossover for each target vector* $\mathbf{X}_i$ *with its noisy vector* $\mathbf{X}_{n,i}$ *and create a trial vector,* $\mathbf{X}_{t,i}$*. The noisy vector is created by performing mutation.*

*· If CR = 0 inherit all the parameters from the target vector* ***X****i, except one which should be from* ***X****n,i.*
*· for binomial crossover*
*{ p = random number*
*for n = 1 to D*
*{if ( p<CR )*
***X****n,i =* ***X****a,i + F (* ***X*** *b,i -* ***X*** *c,i )*
***X****t,i =* ***X****n,i*
*}else* ***X****t,i =* ***X****i,j*
*}*
*· Again, the NP noisy random vectors that are generated should satisfy the constraint and the penalty function approach is followed as mentioned above.*

*· Perform selection for each target vector,* ***X****i by comparing its profit with that of the trial vector,* ***X****t,i ; whichever has the maximum profit will survive for the next generation.*
***C****t,i = defunct.profit()*
*if (****C****t,i >* ***C****i )*
*new* ***X****i =* ***X****t,i*
*else new* ***X****i =* ***X****i }*
*/* for i=1 to NP */*
*}*
*· Print the results (after the stopping criteria is met).*

## 3. Case Study: Welded Beam Design

The problem is taken from Deb & Srinivasan, KanGAL Report Number 2006004. The resulting optimization problem has four design variables, **x** = (h, l, t, b), and four inequality constraints, involving different stress, buckling and logical limitations. The two objective functions the cost(f1) and the deflection(f2) of the beam are to be minimized. The constraint handling is by penalty approach where in those violating the constraints are penalized by a very high value in their objective function so that they doesn’t appear in the next generation.

$$
\begin{aligned}
\text{Minimize } & f_1(\boldsymbol{x}) = 1.10471h^2\ell + 0.04811tb(14.0+\ell),\\
\text{Minimize } & f_2(\boldsymbol{x}) = \tfrac{2.1952}{t^3b},\\
\text{Subject to } & g_1(\boldsymbol{x}) \equiv 13,600 - \tau(\boldsymbol{x}) \geq 0,\\
& g_2(\boldsymbol{x}) \equiv 30,000 - \sigma(\boldsymbol{x}) \geq 0,\\
& g_3(\boldsymbol{x}) \equiv b - h \geq 0,\\
& g_4(\boldsymbol{x}) \equiv P_c(\boldsymbol{x}) - 6,000 \geq 0,\\
& 0.125 \leq h, b \leq 5.0,\\
& 0.1 \leq \ell, t \leq 10.0.
\end{aligned}
$$

**Equations from Deb & Srinivasan, KanGAL Report Number 2006004**

$$
\begin{aligned}
\tau(\boldsymbol{x}) &= \sqrt{(\tau')^2 + (\tau'')^2 + (\ell\tau'\tau'')/\sqrt{0.25(\ell^2+(h+t)^2)}},\\
\tau' &= \frac{6,000}{\sqrt{2}h\ell},\\
\tau'' &= \frac{6,000(14+0.5\ell)\sqrt{0.25(\ell^2+(h+t)^2)}}{2\left\{0.707h\ell(\ell^2/12+0.25(h+t)^2)\right\}},\\
\sigma(\boldsymbol{x}) &= \frac{504,000}{t^2b},\\
P_c(\boldsymbol{x}) &= 64,746.022(1-0.0282346t)tb^3.
\end{aligned}
$$

**Constraint Equations from Deb & Srinivasan, KanGAL Report Number 2006004**

The parallel implementation is done using inserting a “#pragma omp for” before the generation loop. The C code is given below.

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<time.h>
#include<omp.h>

void main()
{
double evalfunc(double h,double l,double b,double t);
double evalfunc2(double ba,double ta);


int D=4;                            /*D = no. of varibales on which
constriants are placed, x ={h, l, t, b}*/
```

```
      double x_lower[4] = {0.125, 0.1, 0.125, 0.1};   /*lower limits of the
varibales x ={h, l, t, b} */
      double x_upper[4] = {5, 10, 5, 10};       /*upper   limits   of   the
varibales x ={h, l, t, b} */
      double CR=0.8;                            /*Crossover constant*/
      double F=0.4;                             /*Scaling factor*/
      int Np=200;                         /*Population size*/
      double pop1[Np][D], f1[Np], f2[Np],*Xt, *Xa, *Xb, *Xc, Xtr[D], f1Xtr,
f2Xtr, f1Xt,f2Xt, Xc1[D], taud,taudd, tau, sigma, pc, Diff[D], Wdiff[D];
      int flag, i, j, gen, k, c, r1, r2, r3, n[D],temp=0;
      srand(time(NULL));
      double start=omp_get_wtime();
      for (i=0; i<Np; i++)
      {
          for (j=0; j<D; j++)
          {
          pop1[i][j]=x_lower[j]+(x_upper[j]-
x_lower[j])*((rand()%32767+1)/32767.0f);      /*Generating population*/
            }
      }

      for (i=0; i<Np; i++)
      {
          f1[i]=evalfunc(pop1[i][0],pop1[i][1],pop1[i][2],pop1[i][3]);
/*Evaluating function value*/
          f2[i]=evalfunc2(pop1[i][2],pop1[i][3]);
      }


      #pragma omp for
      for (gen=1; gen<=1000; gen++)
      {     for (i=0; i<Np; i++)
      {     Xt=(pop1[i]);
            for(int kt=0; kt<D; kt++)
            {n[kt] = floor(((rand()%32767+1)/32767.0f)*D);}
            flag=0;
          for(k=0; k<D; k++)
      {
              c=((rand()%32767+1)/32767.0f);

          if ((c<CR)||(k==D))
      {
          if(flag==0)
      {
              flag=1;
              r1=floor(((rand()%32767+1)/32767.0f)*Np);
              while(r1==i)
              {r1=floor(((rand()%32767+1)/32767.0f)*Np);}
              r2=floor(((rand()%32767+1)/32767.0f)*Np);
```

```
            while(r2==i||r2==r1)
            {r2=floor(((rand()%32767+1)/32767.0f)*Np);}
            r3=floor(((rand()%32767+1)/32767.0f)*Np);
            while(r3==i||r3==r1||r3==r2)
            {r3=floor(((rand()%32767+1)/32767.0f)*Np);}

            Xa=pop1[r1];
            Xb=pop1[r2];
            Xc=pop1[r3];
        for(int z=0;z<D;z++)
          Diff[z]=*(Xa+z)-*(Xb+z);
          F=((rand()%32767+1)/32767.0f);  /*scaling factor*/
        for(int jt=0;jt<D;jt++)
          Wdiff[jt]=F*Diff[jt];
        for(int g=0;g<D;g++)
          Xc1[g]=*(Xc+g)+Wdiff[g];
    }
    temp=n[k];
    Xtr[temp]=Xc1[temp];
    }
        else
        Xtr[temp]=*(Xt+temp);   /*Trial vector*/
    }
    }

    for (j=0;j<D;j++)  /*if outside the bounds  then map*/
       if((Xtr[j]<x_lower[j]) || (Xtr[j]>x_upper[j]))
         Xtr[j]=x_lower[j]                 +                (x_upper[j]-
x_lower[j])*((rand()%32767+1)/32767.0f);


          f1Xt=f1[i];
          f2Xt=f2[i];

     /*Constraint handling by penalty method*/
     taud  =  6000/(sqrt(2)*(Xtr[0])*(Xtr[1]));                 /*  first
differential of tau*/
     taudd                                                               =
6000*(14+0.5*(Xtr[1]))*sqrt(0.25*(pow(Xtr[1],2.)+pow(Xtr[0]+Xtr[2],2.)))/(2
*(0.707*(Xtr[0])*(Xtr[1])*((pow(Xtr[1],2.)/12)+0.25*pow(Xtr[0]+Xtr[2],2.)))
); /* second differential of tau*/
     tau                                                                 =
sqrt(pow(taud,2.)+pow(taudd,2.)+((Xtr[1]*taud*taudd)/sqrt(0.25*(pow(Xtr[1],
2.)+pow(Xtr[0]+Xtr[2],2.)))));
    sigma = 504000/(pow(Xtr[2],2.)*(Xtr[3]));
    pc = 64746.022*(1-0.0282346*(Xtr[2]))*(Xtr[2])*pow(Xtr[3],3.);
```

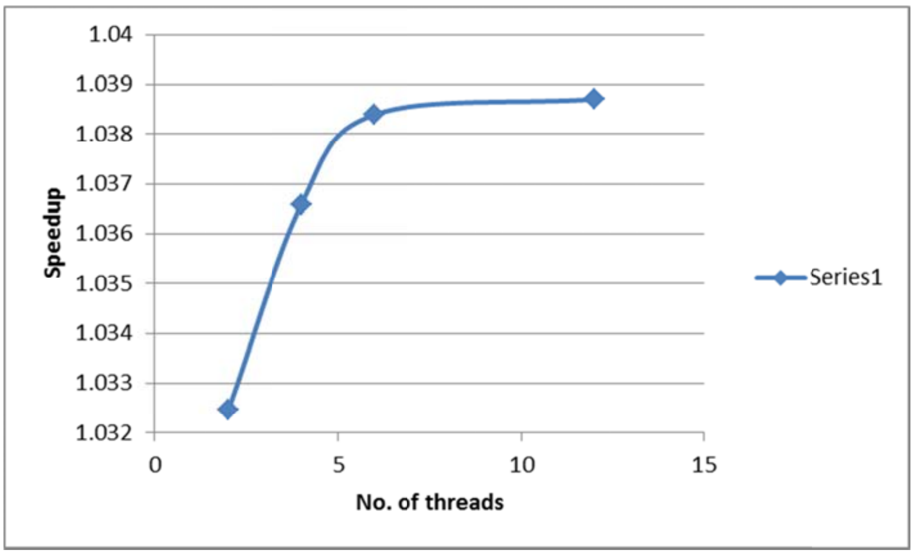
```
	if((13600-tau)>=0 && (30000-sigma)>=0 && (Xtr[2]-Xtr[0])>=0 && (pc-6000)>=0 )
	{f1Xtr=evalfunc(Xtr[0],Xtr[1],Xtr[2],Xtr[3]);
	f2Xtr=evalfunc2(Xtr[2],Xtr[3]);
	}
	else
	{f1Xtr = 1E10;
	f2Xtr = 1E10;}

		if (f1Xt>f1Xtr && f2Xt>f2Xtr)
		{for(int jh=0;jh<D;jh++)
		 pop1[i][jh]=Xtr[jh];
		   f1[i]=f1Xtr;
		 f2[i]=f2Xtr;     };
	}


	for (int ii=1; ii<Np ;ii++)
	printf("%g    \t    %g    \t    %g    \t    %g    \t    %g    \t    %g \n",f1[ii],f2[ii],pop1[ii][0],pop1[ii][1],pop1[ii][2],pop1[ii][3]);

	double end=omp_get_wtime();
	double cpu_time_used = end - start;
	printf("%g \n",cpu_time_used);
	}

	double evalfunc(double h,double l,double b,double t)
	{
	double funcvalue;
	funcvalue = 1.10471*(pow(h,2.))*l+0.04811*t*b*(14.0+l);
	return(funcvalue);
	}

	double evalfunc2(double b,double t)
	{
	double funcvalue;
	funcvalue = 2.1952/(pow(t,3.)*b);
	return(funcvalue);
	}
```

A pareto front is obtained for the number of generations 1000 and population size 200. There seems to be some noise in the results obtained. The extreme solutions obtained are f1*_minimum=1.14091, with (h, l, t, b)=( 0.320643, 6.89286, 0.717135, 0.496701) and f2*_minimum=0.000561314, with (h, l, t, b)=(2.63488, 4.58245, 4.97471, 9.22926).

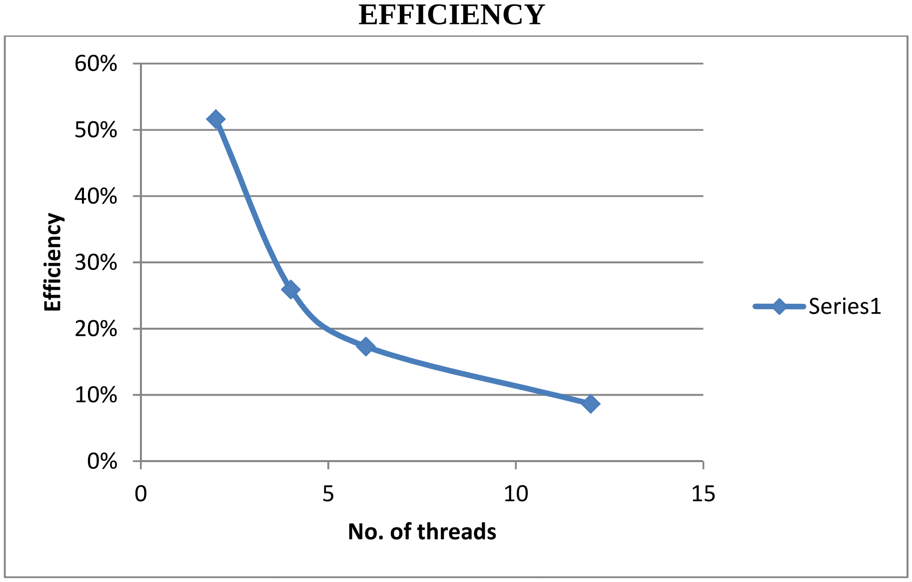
The code is parallelized by introducing a **“#pragma omp for”** before the generation loop. The performance measures are shown below.

| No. of Threads | Time for parallel execution | Speedup | Efficiency |
|---|---|---|---|
| 2 | 0.101191 | 1.032453 | 52% |
| 4 | 0.100788 | 1.036582 | 26% |
| 6 | 0.10061 | 1.038395 | 17% |
| 12 | 0.100582 | 1.038705 | 9% |

**SPEEDUP**




**EFFICIENCY**



## 4. Conclusions

The parallel performance is good for less number of threads and also parallelizing by just one simple "omp for" resulted in a maximum performance improvement of about 52% (observed in this case). This shows the applicability of high performance of computing in varied fields. There's some noise in the pareto front obtained which might be the result of some bug in the code.

## 5. Works Cited


B.V.Babu, & Jehan, M. M. (2003). Differential Evolution for Multi-Objective Optimization. *In proceeding of: Evolutionary Computation.*

Deb, K. (2001). *Multi-Objective Optimization using Evolutionary Algorithms.*

Deb, K., & Srinivasan, A. (n.d.). *Monotonicity Analysis, Evolutionary Multi-Objective Optimization, and Discovery of Design Principles.* Kanpur Genetic Algorithms Laboratory (KanGAL).

Gujarathi, A. M., & B.V.Babu. (December 14-17, 2005). Multi-Objective Differential Evolution (MODE): A New Algorithm of Solving Multi-Objective Optimization Problems. *Proceedings of International Symposium & 58th Annual Session of IIChE.* New Delhi: CHEMCON-2005.

Gujarathi, A. M., & Purohit, S. (2010). Jumping gene adaption of Multi-objective differential evolution algorithm.