Fully Parallel Differential Evolution

Sergio Jhovanne Domínguez González
U.M.S.N.H.
sdominguez@faraday.fie.umich.mx

Norberto García Barriga
U.M.S.N.H
gbarriga@umich.mx

Introduction

A fully parallel Differential Evolution (DE) algorithm based on a Graphic Processing Unit (GPU) approach is presented. The Compute Unified Device Architecture (CUDA) programming platform allows exploiting the inherent parallelism and reducing the computational effort associated to evolutionary operations. The difficulties of generation of random numbers in a multithreaded scheme have been overcome making use of the CURAND library. Mutation, selection and even creation of initial population have been parallelized, remaining only the task associated to the determination of the best solution as a sequential task.

Differential Evolution on the GPU

A DE algorithm based in a greedy scheme is implemented in this work and tested on a set of well-known benchmark optimization problems. A novel mutant vector definition is proposed and the crossover operation is implemented but not used.

Mutation scheme

The mutation operation is carried out using a novel proposal that reduces the number of control variables, avoiding the necessity of tuning procedures or running exhaustive tests in order to achieve accurate results. A certain level of greediness is added by including useful information about the best global solution.

According to this, for each vector in the population at geneartion g, denoted by $\mathbf{x}_i^{(g)}$, a mutant vector $\mathbf{v}_i^{(g)}$ is produced by calculating its components as,

$$v_{i,j}^{(g)} = x_{i,j}^{(g)} + \lambda_j \cdot (x_{r1,j}^{(g)} - x_{r2,j}^{(g)}) + (1-\lambda_j)\cdot(x_{best,j}^{(g)} - x_{i,j}^{(g)})$$

where *i* represents the index of the base vector and the current mutant in the base and test populations of sizes N_p, respectively. $\lambda_j \in (0,\ 1]$ is a random number uniformly generated and is used instead of the constant mutation F to weight the difference between the *j*-th components of two randomly chosen population members r1, r2 ∈ [1,Np] which are different from each other. Besides, the factor $(1-\lambda)$ is used to weight the difference between the *j*-th components of the base vector $\mathbf{x}_i$ and the best global solution $\mathbf{x}_{best}^{(g)}$, which provides a mean to enhance the greediness of the scheme by incorporating information about the best global solution at generation *g*.

Selection scheme

The selection operation is carried out without considering a previous recombination stage. The *i*-th mutant $\mathbf{v}_i^{(g)}$ is selected to be part of the next generation only if its fitness $f(\mathbf{v}_i^{(g)})$ is better than the *i*-th member of the population at the current generation $\mathbf{x}_i^{(g)}$. This is,

$$\mathbf{x}_i^{(g+1)} = \begin{cases} \mathbf{v}_i^{(g)} & \text{if } f(\mathbf{v}_i^{(g)}) > f(\mathbf{x}_i^{(g)}) \\ \mathbf{x}_i^{(g)} & \text{otherwise} \end{cases}$$

By using the proposed scheme, evolutionary parameters are reduced since the mutation constant F was replaced by random numbers and the crossover constant Cr is eliminated.

Parallel processing scheme

Each one of the vectors in population is created, mutated, evaluated and selected independently inside a computing CUDA kernel. In order to accelerate the data reading during the evolutionary operations, populations are stored in texture memory spaces. Results are stored in global memory spaces.

CUDA threads are arranged in one or two-dimensional blocks. Once the blocks size is defined, is necessary to arrange them inside an unidimensional grid which length is calculated as,

$$Grid\_Size = \left\lceil \frac{N_p}{Block\_Size} \right\rceil$$

where Np represents the population size.

Building and execution notes

The sequential and GPU-based source code were compiled and tested in a computing platform consisting of a CPU AMD Phenom II, 3.0 GHz quad-core with 3.9 GB. The GPU used in this work is an NVIDIA Tesla C2050 (“Fermi” architecture) graphic card with 2688 MB of memory, CUDA 3.2 driver, runtime version 3.20 and compute capability 2.0.

Execution parameters

Most of the execution parameters can be defined in the file **`src/de_common.h`**. From this file, it is possible to define the following parameters:

Parameter	Description
`N_p`	Represents the population size N_p.
`D`	Represents de number of dimensions D in the optimization problem.
`G`	Represents g_{max}, the maxium number of generations the aplications is allowed to run.
`BLOCK_SIZE`	For the GPU-based application, it represents the size of the blocks of CUDA threads.
`BLOCK_SIZE_X`	For the GPU-based application, it represents the size of the blocks of CUDA threads in **x** dimension.
`BLOCK_SIZE_Y`	For the GPU-based application, it represents the size of the blocks of CUDA threads in **y** dimension.

In addition to this parameters, it is necessary to set the search space limits or the domain search. The search space limits are set through a two-dimensional array of order 2xD, called **`Restrictions`** for the sequential version and **`d_Restrictions`** for de GPU-based version. This variables are located in the files **`src/DE.cpp`** and **`src/kernels.cu`**, respectively. The values set in the first column correspond to the lower limits of each of the D dimensions, while the values of the second column correspond to the upper limits.

Additional information

This application is part of the master thesis of Sergio Jhovanne Domínguez González, which was carried out thanks to the support and supervision of Norberto García barriga, Research Professor at División de Estudios de Postgrado, Facultad de Ingeniería Eléctrica, Universidad Michoacana de San Nicolás de Hidalgo, Morelia, Michoacán, México. For more information, please contact us: sdominguez@faraday.fie.umich.mx and gbarriga@umich.mx.

Test functions

Four well-known test functions are included in the source code which are useful to assess the application performance. These four functions and its test parameters are described in Table 1.

Table 1: Functions and parameters used to test the application's performance.

Function	N_p	g_{max}	Domain Search
Rosenbrock's function	2048, 4096, 8192	18,000	$[-600,600]^D$
Griewank's function	2048, 4096, 8192	1,000	$[-600,600]^D$
Ackley's function	2048, 4096, 8192, 16384, 32768, 65535	3,000	$[-32,32]^D$
* F4 function	2048, 4096, 8192	2,000	$[-600,600]^D$

* F4 function corresponds to:

$$f(\mathbf{x}) = 0.1\left\{\sin^2(3\pi x_1) + \sum_{j=1}^{D-1}(x_i - 1^2)\left[1 + 10\sin^2(3\pi x_{i+1})\right] + (x_D - 1)^2\left[1 + \sin^2(2\pi x_D)\right]\right\} + \sum_{j=1}^{D} u(x_i, 5, 100, 4)$$

where,

$$u(x,a,k,m) = \begin{cases} k\cdot(x-a)^m, & x > a \\ 0, & -a \le x \le a \\ k\cdot(-x-a)^m, & x < -a \end{cases}$$