Parallel Bat Optimization on GPU using CUDA

Jean Carlo Machado
Univerisdade do Estado de Santa Catarina
Mestrado de Computao Aplicada
UDESC
Santa Catarina, Joinville,
Email: contato@jeancarlomachado.com.br

Rafael Stubs Parpinelli
Univerisdade do Estado de Santa Catarina
UDESC
Santa Catarina, Joinville

***Abstract*—The ever increasing parallel processing power of GPU's is an compelling motive to implement performance demanding algorithms, like optimization techniques, using this technology. This work aimed to develop a GPU version of the bat metaheuristic, a CPU version were developed as well, as a way of comparison. A set of experiments where conducted in order to measure the speedup difference. The results suggests that the GPU version is able to achieve relevant speedups in highly populational problems but for simpler cases the CPU version might outperform GPU.**

I. Introduction

With the aid of parallel computing, specially GPU computing, it's possible to tackle ever increasing computational problems in decreasing amounts of time. A good way of solving complex problems, like NP one's is through the use of optimizaton algorithms, specially meta-heuristics. In the group of meta-heuristics a promising field for parallel computings is the swarm intelligence group. Which encompases many bio-inspiratons liks PSO, ACO, BAT, etc.

Parpinelli [10], when reffering to swarm intelligence algorithms, define that they are:

> compounded by a distributed society/population of individuals where the control is also distributed among the individuals (there is no centralized control); and the individuals decision-making is stochastic and based only on local information .

The fact that swarm intelligence algorithms have composable that run conceptually independent from one another makes them ideal for parallel enviroments.

This work attempts to investigate the applicability of a relatively new and promising swarm algorithm: the BAT algorithm in the GPU using CUDA. Previously some demonstrations of the bat algorithm parallelized on CPU were presented in [7] and [6]. A novel publication about the use of the bat optimization can be found on ??.

However no case was used til this moment to the applicability of the bat on gpu when comparing standard benchmark functions.

The rest of the paper is organized as following: section II gives a overview of cuda and its components. Section III studies the bat meta-heuristic, it's design and details. Section I details the experiments performed. Section V analyses the results and concludes.

II. CUDA

Compute Unified Device Architecture (CUDA) is a general purpose parallel computing programming model for parallel computations [3]. CUDA uses a SIMD *single data multiple execution* approach, where the concurrent code executes the same instruction but with divergent data.

Previous researches suggests that highly parallel applications may speedup up to 450 times [9].

Cuda is organized on threads that are separated blocks which in their

The CUDA platform uses a parallelization schema in "each cuda device supports the Single-Program Multiple-Data (SPMD)" [9], where all concurrent threads are based on the same code but they may follow different paths.

A good approach is to use the threaded model since it has the great benefit in performance.

"..when attempting to achieve an application's maximum performance, the primary concern often is managing global memory latency." [9]

III. Bat algorithm

The bat algorithm is a populational meta-heuristic introduced by Yang in 2010. It uses the inspiration of micro-bats which uses a type of sonar, called echolocation, to detect prey, avoid obstacles, and locate their roosting crevices in the dark [1].

All populational meta-heuristics theoretically are able to benefit from implicit parallelization, which is the approach that each individual of the population executes concurrently.

A. Bat Design on CPU

The bat original paper don't clarify all the implementation details working of the algoritm.

In this work the bath algorithm used was the one proposed by [2], since it represents a concrete demonstration of how the bat metaheuristic might work.

Some distinctions of the original paper are worth noticing

- The selection of new results on the original paper tends to be more greedy (line 14) [improve the descritpion of the other paper].
- On this paper the or operator were used but in the original one an And were proposed.

$Parameters:\ n, \alpha,\ \lambda$
$initialize\ bats$
$evaluate\ fitness$
$selects\ best\ \vec{x}_*$
while $stop\ criteria\ false$ **do**
  for $each\ bat$ **do**
    $f_i = f_{min} + (f_{max} - f_{min})\beta, \in \beta[0,1]$
    $\vec{v}_i^{t+1} = \vec{v}_i^t + (\vec{x}_i^t + \vec{x}_*^t)f_i$
    $\vec{x}_{temp} = \vec{x}_i^t + \vec{v}_i^{t+1}$
    if $rand < r_i, rand \in [0,1]$ **then**
      $\vec{x}_{temp} = \vec{x}_* + \epsilon A_m, \epsilon \in [-1,1]$
    end if
    $single\ dimension\ perturbation\ in\ x_{temp}$
    if $a < A_i^t$ **or** $f(\vec{x}_{temp}) \leq f(\vec{x}_i), a \in [0,1]$ **then**
      $\vec{x}_i^t = \vec{x}_{temp}$
      $r_i = exp(\lambda * i)$
      $A_i = A_0 * \alpha^i$
    end if
  end for
  $selects\ best\ \vec{x}_*$
end while

Fig. 1. Pseudo-code CPU

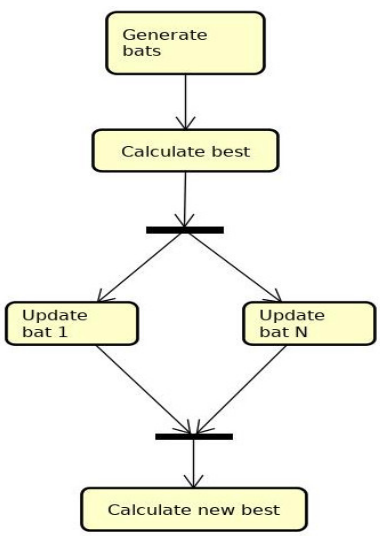

Fig. 2. GPU process flow

- The or operator tends to explore the search space better (more diversity).
- There's a distortion on a single dimension of the search space in order to increase the diversity factor.

The CPU version developed was single threaded. The random algorithm used was the xorshift algorithm.

B. Bat Design on GPU

Since the BAT algorithm uses a population of bats, the most intuitive parallelization method to apply on it is to use each bat on each GPU thread. [4] used a similar method for a GPU implementation for the PSO algorithm.

$Parameters:\ n, \alpha,\ \lambda$
$initialize\ bats\ asynchronously$
$evaluate\ fitness$
$synchronize\ threads$
$selects\ best\ \vec{x}_*$
while $stop\ criteria\ false$ **do**
  for $each\ thread$ **do**
    $f_i = f_{min} + (f_{max} - f_{min})\beta, \in \beta[0,1]$
    $\vec{v}_i^{t+1} = \vec{v}_i^t + (\vec{x}_i^t + \vec{x}_*^t)f_i$
    $\vec{x}_{temp} = \vec{x}_i^t + \vec{v}_i^{t+1}$
    if $rand < r_i, rand \in [0,1]$ **then**
      $\vec{x}_{temp} = \vec{x}_* + \epsilon A_m, \epsilon \in [-1,1]$
    end if
    $single\ dimension\ perturbation\ in\ x_{temp}$
    if $a < A_i^t$ **or** $f(\vec{x}_{temp}) \leq f(\vec{x}_i), a \in [0,1]$ **then**
      $\vec{x}_i^t = \vec{x}_{temp}$
      $r_i = exp(\lambda * i)$
      $A_i = A_0 * \alpha^i$
    end if
  end for
  $synchronize\ threads$
  $selects\ best\ \vec{x}_*$
end while

Fig. 3. Pseudo-code GPU

In the bat algorithm synchronization must occur on the selection of the best individual of the iteration. The best individual is kept in the threaded memory of the GPU which has a limit of 16KB, probably not feasible for more complex problems.

The random number generator used on the GPU was the MTGP32, to maintain the compatibility with the CPU version. Notwithstanding it's not recommended to use more than 256 threads per block with it
[5].

IV. Experiments

For testing the performance of the algorithm a set of experiments where developed using diverse benchmark functions tested against a set of individuals in a highly dimensional problem.

The benchmark functions used were the following:

- Ackley
- Griewank
- Rastringin
- Rosenbrook

The experiments were executed on a machine with the following configuration:
Intel(R) Core(TM) i5-4460 CPU @ 3.20GHz
GK208 GeForce GT 720 1024 MB of vram
Compute capability 3.5
Kepler GM10x

Each experiment was executed a total of 20 times with 10 thousand iterations each and 100 dimensions in each function.

TABLE I
EXPERIMENTS

Name	Function	Dimensions	Agents
E1	Ackley	100	256
E2	Ackley	100	768
E3	Griewank	100	256
E4	Griewank	100	768
E5	Rastringin	100	256
E6	Rastringin	100	768
E7	Rosenbrook	100	256
E8	Rosenbrook	100	768

TABLE II
CPU RESULTS

Time Avg	Time SD	Fit Avg	Fit SD
(E1) 49.4428	0.0557314	4.44089e-16	2.05196e-22
(E2) 161.439	0.155131	4.44089e-16	2.82843e-22
(E3) 61.3661	5.06578	0	0
(E4) 162.761	57.8119	0	0
(E5) 52.0624	9.25666	0	0
(E6) 171.986	14.9089	0	0
(E7) 20.4486	0.0847218	98.9875	0.0405622
(E8) 74.3533	0.186482	98.9864	0.0204378

TABLE III
GPU RESULTS

Time Avg	Time SD	Fit Avg	Fit SD
(E1) 17.2255	0.708198	12.8881	2.40027
(E2) 10.9591	0.23902	10.9412	3.23942
(E3) 24.2459	0.740923	2.04281e-15	2.60744e-16
(E4) 15.0012	2.01505e-15	2.55402e-16	0.0394986
(E5) 30.4483	2.0005	0	0
(E6) 14.4247	0.0543432	0	0
(E7) 28.9867	1.24554	105.03	31.2888
(E8) 15.4403	0.326284	101.793	140.393

V. RESULTS

In this section are described the speedup and convergence results. The time spent in each execution of the algorithms is described in seconds.

The fitness of almost all functions presented a slight worse result when compared with the cpu version. This slightly difference, in most cases, probably refers to the case that the subsequent bats of the same iteration don't have a best based on this current iteration.

However the purpose of this research is to focus on the speedups so it seems like a reasonable tradeoff.

VI. CONCLUSION

It was observed speedups with big populations. The original BAT was proposed with 40 individuals and the speedups was seen with 250 individuals.

The advantages of the algorithm may be tested against a threaded CPU implementation to be fair.

With this work it's clear that is possible to speedup the bat metaheuristic using GPU. Notwithstanding the best results are only achievable on really complex problems with many dimensions.

VII. FURTHER WORKS

In the future it may be explored the usage of blocks as representation for the dimensions in which each bat details.

A subpopulation approach may also work, considering each GPU block as it's boundaries, somewhat similar to the work made on parallel bat on CPU by [6].

REFERENCES

[1] Xin-She Yang *A New Metaheuristics Bat-Inspired Algorithm*. Department of Engineering, Cambridge, 2010.

[2] J.A. Cordeiro and R.S. Parpinelli and H.S. Lopes, "Anlise de Sensibilidade dos Parmetros do Bat Algorithm e Comparao de Desempenho," Department of Bioinformatics, UTFPR

[3] NVIDIA, "CUDA C Programming Guide," no. July. NVIDIA Corportation, 2013

[4] PSO-GPU: Accelerating Particle Swarm Optimization in CUDA-Based Graphics Processing Unit

[5] http://docs.nvidia.com/cuda/curand/device-api-overview.html

[6] Parallelized Bat Algorithm with a Communication Strategy. Cheng-Fu Tsai, Thin-Kien Dao, Wei-Jie Yang, et al, University of applied sciences

[7] Parallel bat algorithm for optimizing makespan in job scheduling problems, Thi-Kien Dao, Springer Sience Review

[8] Optimization Principles and Application Performance Evaluation of a Multithreaded GPU using Cuda, Shane Ryo

[9] Optimization Principles and Application Performance Evaluation on a Multithreaded GPU using CUDA, Shane Ryoo, Critopher I Rodrigues et all

[10] Parpinelli, R.S. and Lopes, H.S. (2011) 'New inspierations in swarm intelligence: a survey', Int. J. Bio-Inspired Computation, Vol. 3, No. 1 pp.1-16