Bat Optimization on GPU

Jean Carlo Machado

Univerisdade do Estado de Santa Catarina

Mestrado de Computao Aplicada

UDESC

Santa Catarina, Joinville,

Email: contato@jeancarlomachado.com.br

Abstract—**This work developed a CPU implementation and a GPU one. And a set of experiments where made in order to measure the speedup. The research shows that the GPU version is able to achieve relevant speedups in highly populational problems.**

I. INTRODUCTION

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-heuristic theoretically can benefit from implicit parallelization, which is the approach that each individual of the populations executes concurrently.

This work attempts to investigate the applicability of the BAT algorithm concurrently on the GPU. Previously some demonstrations of the bat algorithm parallelized on CPU were presented in [6] and [5], however, til the day of this publication no implementation of the bat algorithm was found on GPU. The GPU is a great place to run populational algorithms because GPUs are able to parallelize high loads of concurrent individuals doing part of the hole job.

December 21, 2016

II. BAT DESIGN ON CPU

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

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

A. Pseudo-code

$$\text{Parameters:} n, \alpha\, \lambda$$
$$\text{Initialize Bats}$$
$$\text{Evaluate fitness bats}$$
$$\text{Selects best}$$
$$\textbf{while} \text{ stop criteria false } \textbf{do}$$
$$\textbf{for} \text{ i=1 to n } \textbf{do}$$
$$f_i = f_{min} + (f_{max} - f_{min})\beta, \in \beta[0,1]$$
$$\vec{v}_i^{t+1} = \vec{v}_i^t + (\vec{x}_i^t + \vec{x}_*^t)f_i$$
$$\vec{x}_{temp} = \vec{x}_i^t + \vec{v}_i^{t+1}$$
$$\textbf{if } rand < r_i, rand \in [0,1] \textbf{then} \text{ local search}$$
$$\vec{x}_{temp} = \vec{x}_* + \epsilon A_m, \epsilon \in [-1,1] \quad (1)$$
$$\textbf{end if}$$
$$\text{Single dimension perturbation in} x_{temp}$$
$$\textbf{if} a < A_i^t, a \in [0,1]$$
$$\vec{x}_i^t = \vec{x}_{temp}$$
$$r_i = exp(\lambda * i)$$
$$A_i = A_0 * \alpha^i$$
$$\textbf{end if}$$
$$\text{Selects best}$$
$$\textbf{end for}$$
$$\textbf{end while}$$

III. 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 a GPU core. [3] used a similar method for a GPU implementation for the PSO algorithm. For the GPU version the approach used was the split of each individual in one thread.

The random number generator used on the GPU was the MTGP32, to maintain the compability with the CPU version. Notwithstanding it's not recommended to use more than 256 threads per block with it [4].

A. Pseudo-code GPU

Parameters: n,α, λ
Initialize bats asynchronously
synchronize threads
Evaluate fitness bats
Selects best
while stop criteria false **do**
 for all threads **do**
  $f_i = f_{min} + (f_{max} - f_{min})\beta, \in \beta[0,1]$
$_i^{t+1} = \vec{v}_i^t + (\vec{x}_i^t + \vec{x}_*^t)f_i$
$_{temp} = \vec{x}_i^t + \vec{v}_i^{t+1}$
if rand ¡ $r_i, rand \in [0,1]$ **then**
  $_{temp} = \vec{x}_* + \epsilon A_m, \epsilon \in [-1,1]$
 end if
 Single dimension perturbation in x$_{temp}$**if** $a < A_i^t, a \in [0,1]$ **then**
  $_i^t = \vec{x}_{temp}$
$r_i = exp(\lambda * i)$
$A_i = A_0 * \alpha^i$
end if
synchronize threads
Selects best
end for
 end while

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

Each experiment runned 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
SPEEDUP RESULTS

Name	Time CPU	Time GPU	Speedup
E1	57.3774	??	??
E3	54.6234	20.2427	2.6984
E5	??	32.1898	

TABLE III
CONVERGENCE RESULTS

Name	Fitness CPU	Fitness GPU
E1	1.69691e-06	??
E3	8.34383e-13	2.0095e-15
E3	??	0

V. Results

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

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 FU [5].

References

[1] Xin-She Yang *A New Metaheuristics Bat-Inspired Algorithm*. Department of Engineering, Cambridge, 2010.
[2] Jelson A. Cordeiro, Rafael Stubs Parpinelli Heitor Silvrio Lopes *Anlise de Sensibilidade dos Parmetros do Bat Algorithm e Comparao de Desempenho.*
[3] PSO-GPU: Accelerating Particle Swarm Optimization in CUDA-Based Graphics Processing Unit
[4] http://docs.nvidia.com/cuda/curand/device-api-overview.htmldevice-api-overview
[5] Parallelized Bat Algorithm with a Communication Strategy. Cheng-Fu Tsai, Thin-Kien Dao, Wei-Jie Yang, et al, University of applied sciences
[6] Parallel bat algorithm for optimizing makespan in job scheduling problems, Thi-Kien Dao, Tien-Szu Pan, Trong-The Nguyen, Jeng-Shyang Pan,2015, Springer Sience Review