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 attractive motive to implement performance demanding algorithms, like optimization techniques, using this approach. 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 made in order to measure the speedup. The research shows 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

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 parallel cation, which is the approach that each individual of the populations executes concurrently.

Compute unified device architecture (CUDA)[ ] is a platform to execute software concurrently, it uses a single data multiple execution approach.

Previous researches suggests that highly parallel general processing application, speedups may vary from 10 to 450 [8].

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 for GPU.

II. The CUDA platform

The CUDA platform uses a parallelization schema in "each cuda device supports the Single-Program Multiple-Data (SPMD)" [8], 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." [8]

1: $Parameters:\ n, \alpha,\ \lambda$
2: $initialize\ bats$
3: $evaluate\ fitness$
4: $selects\ best$
5: **while** $stop\ criteria\ false$ **do**
6:     **for** $each\ bat$ **do**
7:         $f_i = f_{min} + (f_{max} - f_{min})\beta, \in \beta[0,1]$
8:         $\vec{v}_i^{t+1} = \vec{v}_i^t + (\vec{x}_i^t + \vec{x}_*^t)f_i$
9:         $\vec{x}_{temp} = \vec{x}_i^t + \vec{v}_i^{t+1}$
10:         **if** $rand < r_i, rand \in [0,1]$ **then**
11:             $\vec{x}_{temp} = \vec{x}_* + \epsilon A_m, \epsilon \in [-1,1]$
12:         **end if**
13:         $single\ dimension\ perturbation\ in\ x_{temp}$
14:         **if** $a < A_i^t$ **or** $f(\vec{x}_{temp}) \leq f(\vec{x}_i), a \in [0,1]$ **then**
15:             $\vec{x}_i^t = \vec{x}_{temp}$
16:             $r_i = exp(\lambda * i)$
17:             $A_i = A_0 * \alpha^i$
18:         **end if**
19:         $selects\ best$
20:     **end for**
21: **end while**

Fig. 1. Pseudo-code CPU

III. Bat Design on CPU

In this work the bath algorithm used was the one proposed by Jelson et al., since it represents a concrete demonstration of how the bat metaheuristic given that the original paper dont show clearly the expected behavior of the algorithm.

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). On this paper the or operator were used but in the original one an And were proposed. 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 Mersenne twister.

IV. 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

$Parameters:\ n, \alpha,\ \lambda$
$initialize\ bats\ asynchronously$
$evaluate\ fitness$
$synchronize\ threads$
$selects\ best$
while $stop\ criteria\ false$ **do**
  for $eachthread$ **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
    $synchronize\ threads$
    $selects\ best$
  end for
end while

Fig. 2. Pseudo-code GPU

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.

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, not feasible for the most 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 [4].

V. 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 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
Speedup Results

Name	Time CPU	Time GPU	Speedup
E1	57.3774	??	??
E3	54.6234	20.2427	2.6984
E5	42.3531	32.1898	1.3157
E7	24.7752	26.4898	0.9352

TABLE III
Convergence Results

Name	Fitness CPU	Fitness GPU
E1	1.69691e-06	??
E3	8.34383e-13	2.0095e-15
E5	6.50132e-07	0
E7	93.884	96.7034

VI. Results

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

Rosenbrook function had the poorest speedups in relation with the CPU version, this is probably because this function has much float point operations in relation to the others.

The convergence results of the Rosenbrook function are specially worse than the others but it seems to be a problem with the algorithm since it misbehaves equally bad on CPU.

VII. 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.

VIII. 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

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

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