Parallel/distributed implementation of cellular training for generative adversarial neural networks

E. Perez, S. Nesmachnow, J. Toutouh, E. Hemberg, U. O'Reily

(a) *Universidad de la República*, Uruguay

(b) *Massachusetts Institute of Technology*, Cambridge, MA, USA

Introduction

Generative adversarial networks (GANs) consist of two networks, a generator and a discriminator, that apply adversarial learning to optimize their parameters

Lipizzaner, a distributed coevolutionary GAN training method, has shown success in overcoming training pathologies. It coevolves two sub-populations (generators and discriminators) on each cell of a spatial grid

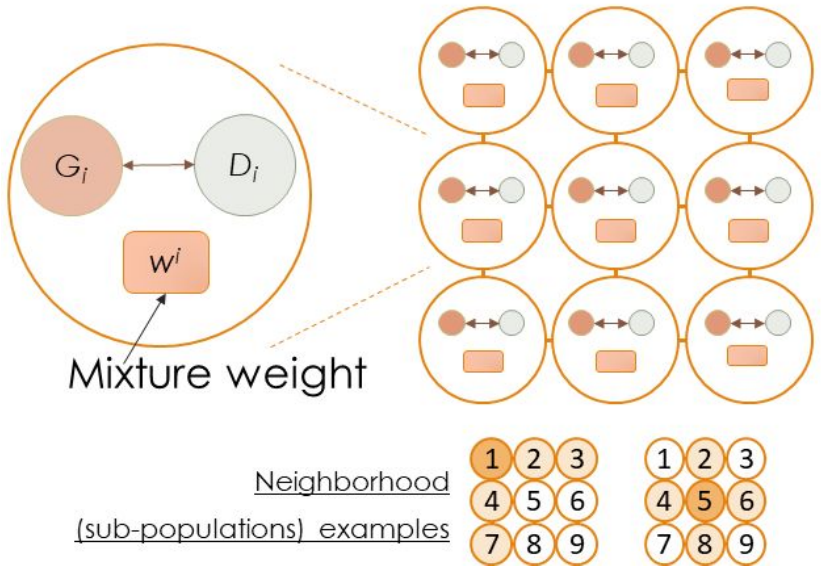
High performance computing: provides computing power for efficient GAN training

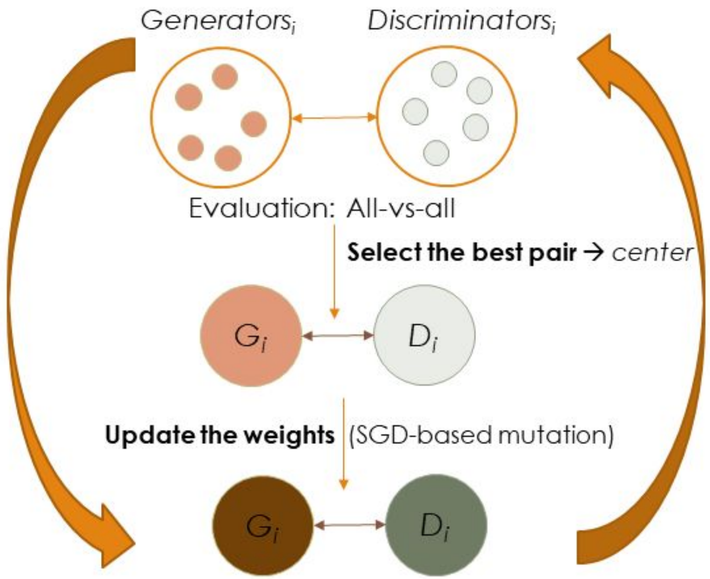
Contributions:

- parallel GAN training: Mustangs/Lipizzaner for (non-dedicated) HPC/supercomputing centers. Two-level parallel model (multithreading and distributed memory via MPI)
- experimental evaluation for MNIST dataset samples generation

Lipizzaner. Distributed coevolutionary GAN training

A **distributed, coevolutionary** framework to train GANs with **gradient-based** optimizers





https://github.com/ALFA-group/lipizzaner-gan

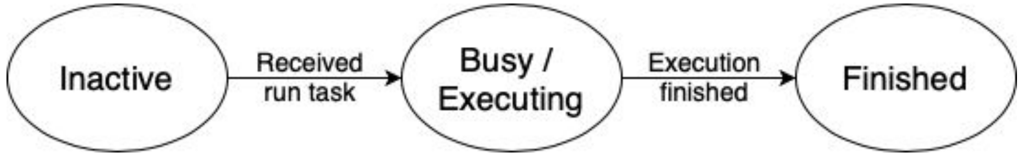
Parallel distributed implementation

- Adapted master/slave parallel model
  - Domain decomposition on the grid used for training
  - Each slave is associated to grid coordinates in MPI_COMM_WORLD. Cartesian topology via MPI_CART_CREATE to optimize communications.

- Master controls the execution flow:
  - gathers computing infrastructure information
  - schedules and assigns workload to each slave (load balancing on each node)
  - shares parameter configuration and launches slaves
  - system monitoring in background, using a heartbeats protocol handled by an execution thread
  - gathers local results from each slave, reduction phase,and returns best result

Parallel distributed implementation

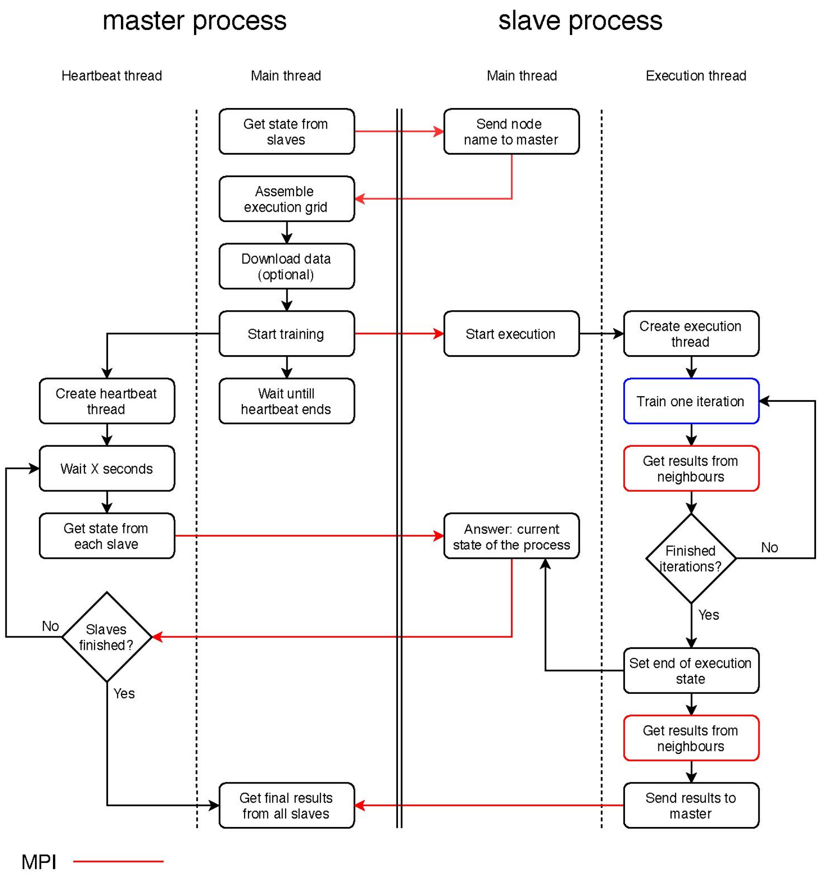
- Slaves perform GAN training
    - Two threads to achieve concurrency: primary (for communications with master) and secondary (for GAN training).
    - Training considers neighborhood info via the grid class (in parameter configuration).
    - Three states: *inactive* (has not received a workload to process yet), *processing* (performing the assigned training), and *finished* (waiting for the master to gather results).



- Main new classes in the parallel implementation:
    - comm-manager: a wrapper of communication functions, defined in an abstract way.
    - grid: defines the grid for execution of each slave. Allows modifying the neighboring structure dynamically (not provided by Mustangs/Lipizzaner) to explore different.
    - The implementation is decoupled, so different modules for communication/training can be applied.

Parallel distributed implementation

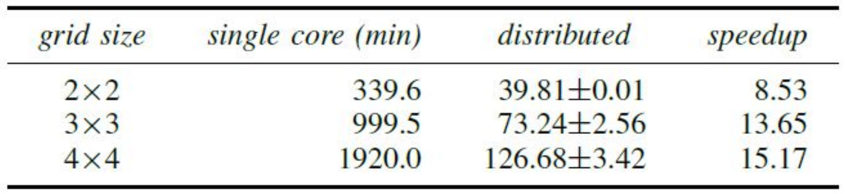
- MPI-based communications in comm-manager
  - Three contexts (MPI communicators)
  - Global WORLD communicator to establish all global configurations, exchange messages to start slaves, and get control messages.
  - LOCAL communicator for collective operations among active slaves in a grid. Allows gathering info without involving the master or inactive slaves (needed for each slave to know training results from neighboring slaves).
  - GLOBAL communicator includes master and all slaves to perform collective operations involving all processes (e.g., master gathering partial results from slaves).



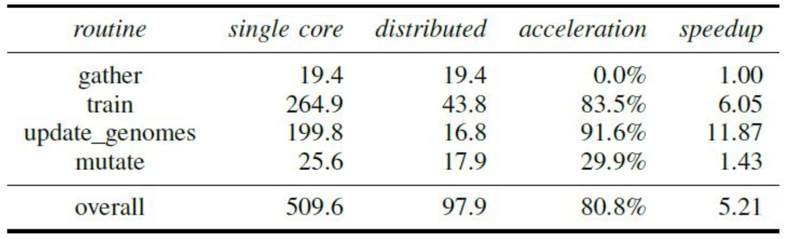
Experimental evaluation

- MNIST dataset
  - 70K grayscale images images (60K for training, 10K for test)
- Efficiency results
  - Speedup up to 15.17 (4x4 grid), good scalability
  - Superlinear speedup for 2x2 and 3x3, mostly due efficient memory management
  - Profiling of the developed implementation:
    - computing time of *update* and *train* is reduced (speedup 6.05 and 11.87)
    - communication time remained the same, suggesting a good scalability behavior

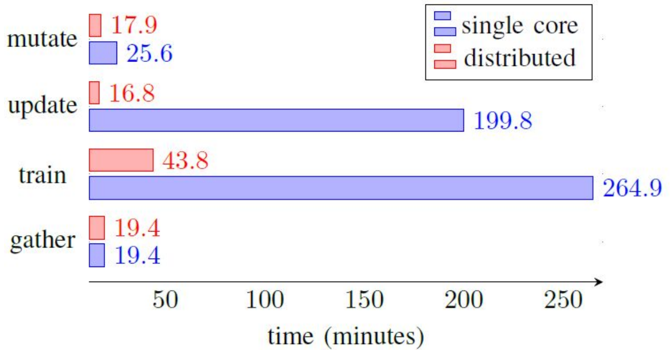
SUMMARY OF EXECUTION TIMES OF GAN TRAINING



grid size	*single core (min)*	*distributed*	*speedup*
2×2	339.6	39.81±0.01	8.53
3×3	999.5	73.24±2.56	13.65
4×4	1920.0	126.68±3.42	15.17



routine	*single core*	*distributed*	*acceleration*	*speedup*
gather	19.4	19.4	0.0%	1.00
train	264.9	43.8	83.5%	6.05
update_genomes	199.8	16.8	91.6%	11.87
mutate	25.6	17.9	29.9%	1.43
overall	509.6	97.9	80.8%	5.21



Conclusions and future work

MPI library and multithreading implementation of an efficient version of Lipizzaner

The proposed parallel implementation is able to effectively reduce the execution times of GAN training, while demonstrating a robust and efficient scalability behavior.

- Speedup values of up to 15.17 were obtained (instance using a grid of size 4x4).
- The proposed implementation reduces the execution time of the most demanding training.

The main lines for future work are related to improving the computational efficiency of the proposed implementation and developing an efficient hybrid CPU/GPU version of the proposed method to take full advantage of nowadays scientific computing platforms.

Thanks!

Comments?

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Sergio Nesmachnow
sergion@fing.edu.uy

Jamal Toutouh
toutouh@mit.edu
@jamtou
jamal.es / necol.net

Massachusetts Institute of Technology