Developing Efficient Discrete Simulations on Multicore and GPU Architectures

MABICAP project

Cagigas-Muñiz, D.; Diaz-del-Rio, F.; López-Torres, M.R.; Jiménez-Morales, F.; Guisado, J.L. Developing Efficient Discrete Simulations on Multicore and GPU Architectures. *Electronics* **2020**, *9*, 189.

Departamento de
Arquitectura y
Tecnología de Computadores
UNIVERSIDAD DE SEVILLA

Objectives

- Parallel implementations for multicore CPUs and for GPUs of the cellular automaton model of laser dynamics.
- Extract lessons that may be helpful for practitioners implementing discrete simulations of real systems in parallel architectures.
- Previous work in 2012: only CUDA (GPU) and sequential version (single core). Not optimized and not complete.

Laser dynamics

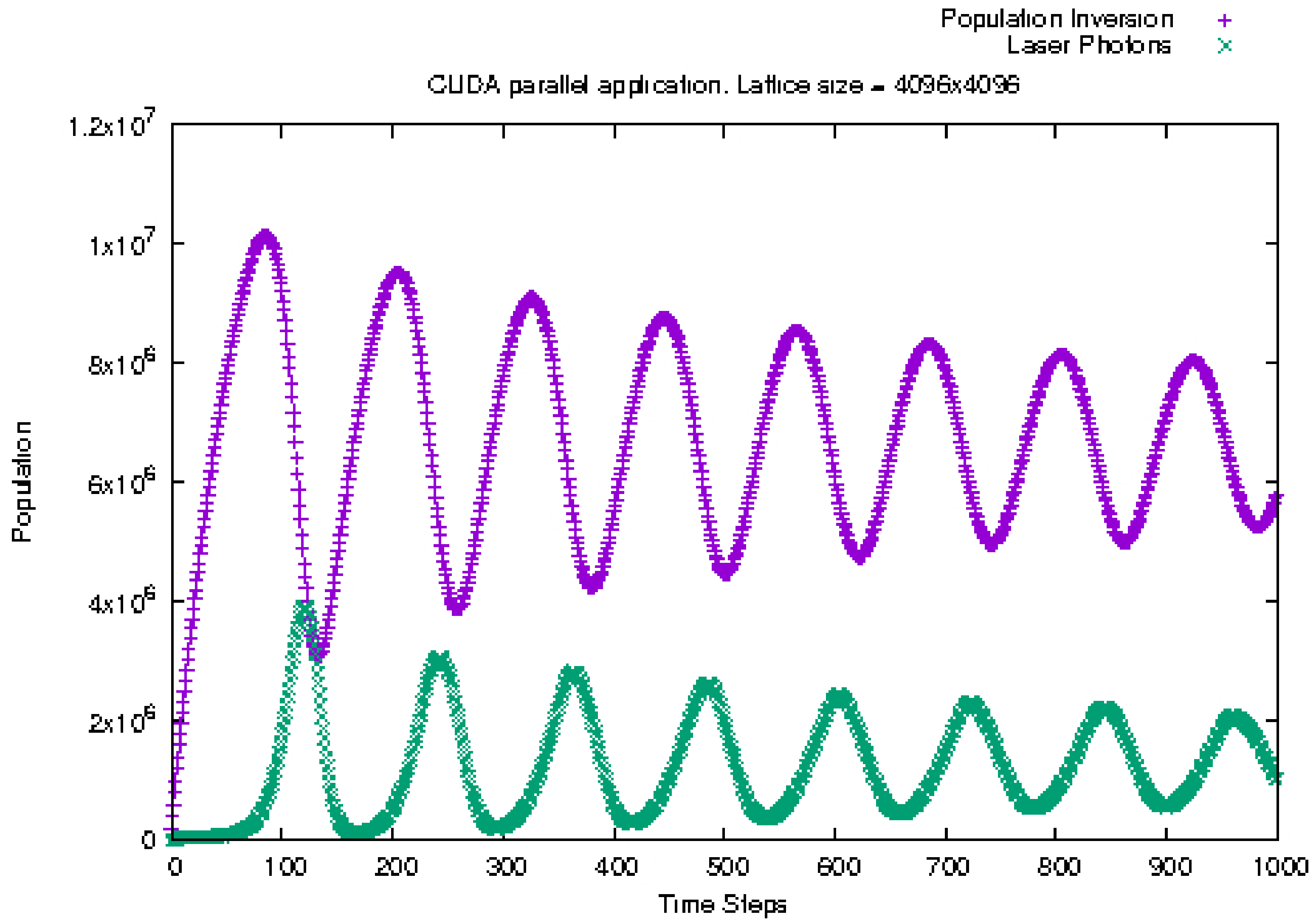
- It is used several matrixes to emulate electron an photon life time (energy) in a grid. (The grid represents a cut on a laser tube)
- Each cell can contain at least one electron and several photons.
- In each time step new photons can be created due to:
  - ‘noise’ (*noise photons creation rule*)
  - rules that take into account neighbours (*stimulated emission rules*).
- Electrons are bumped in each time step (*pumping*).
- Energy decreases in each time step (*photon and electron decay*)

Laser dynamics

General algorithm:

```
Initialize system
Input data
for time step = 1 to maximum time step do
    for each cell in the array do
            Apply noise photons creation rule
            Apply photon and electron decay and evolution of temporal variables
            Apply pumping and stimulated emission rules
    end for
    Refresh value of c matrix with contents of c' matrix
    Calculate populations after this time step
    Optional additional calculations on intermediate results (Shannon entropy)
end for
Final calculations
Output results
```

Laser dynamics



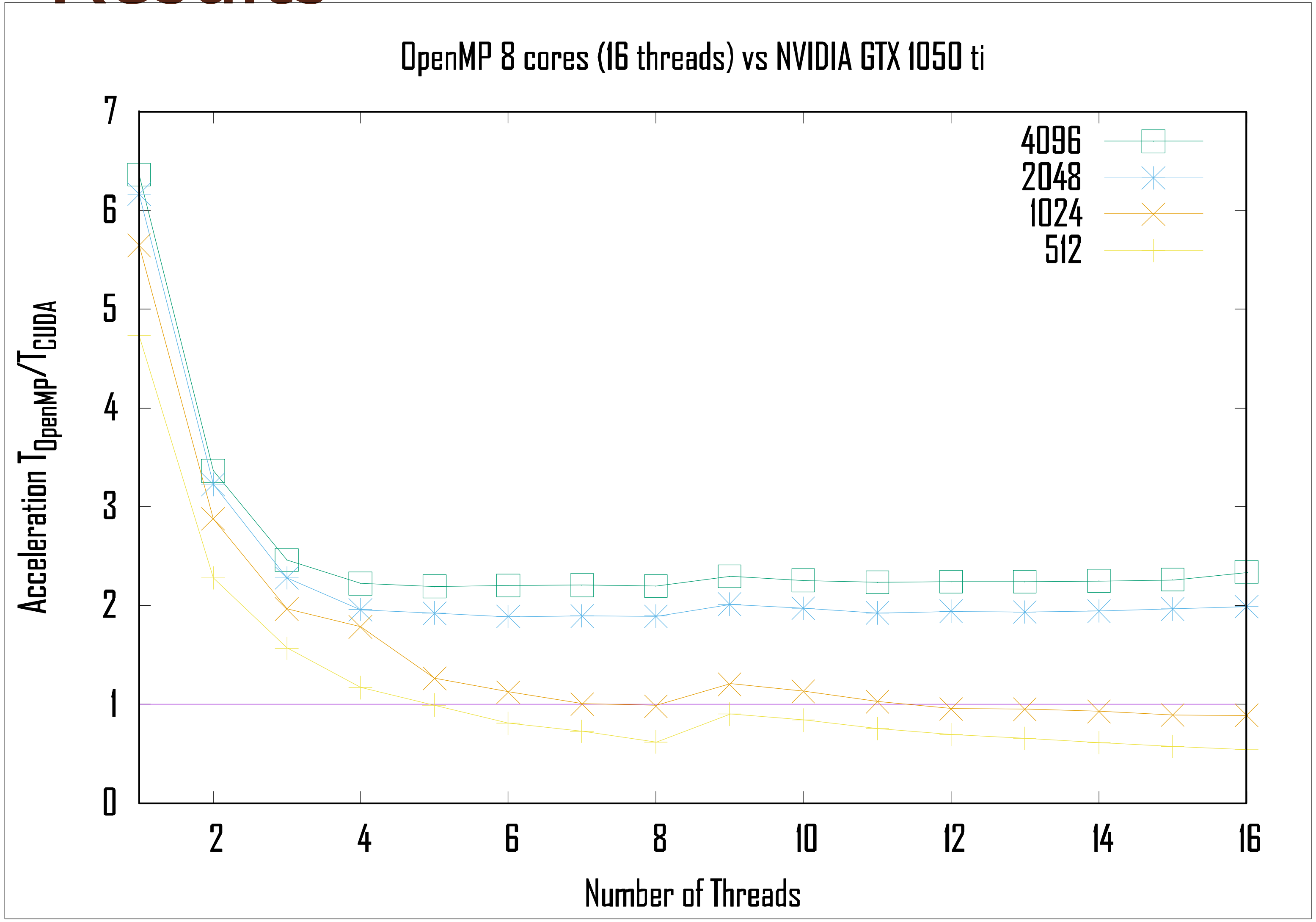
Experiments

- Multiprocessor and OpenMP: two versions: optimized and not optimized. The optimized version:
    - Reduce cell data type: short int instead int.
    - Loop splitting: electron and photons are computed in different loops due to cache constraints.
    - Loop vectorizations: some branchs are transformed into functions. *E.g. q=+p where p is boolean*
- Rand() was be changed with PCG library.

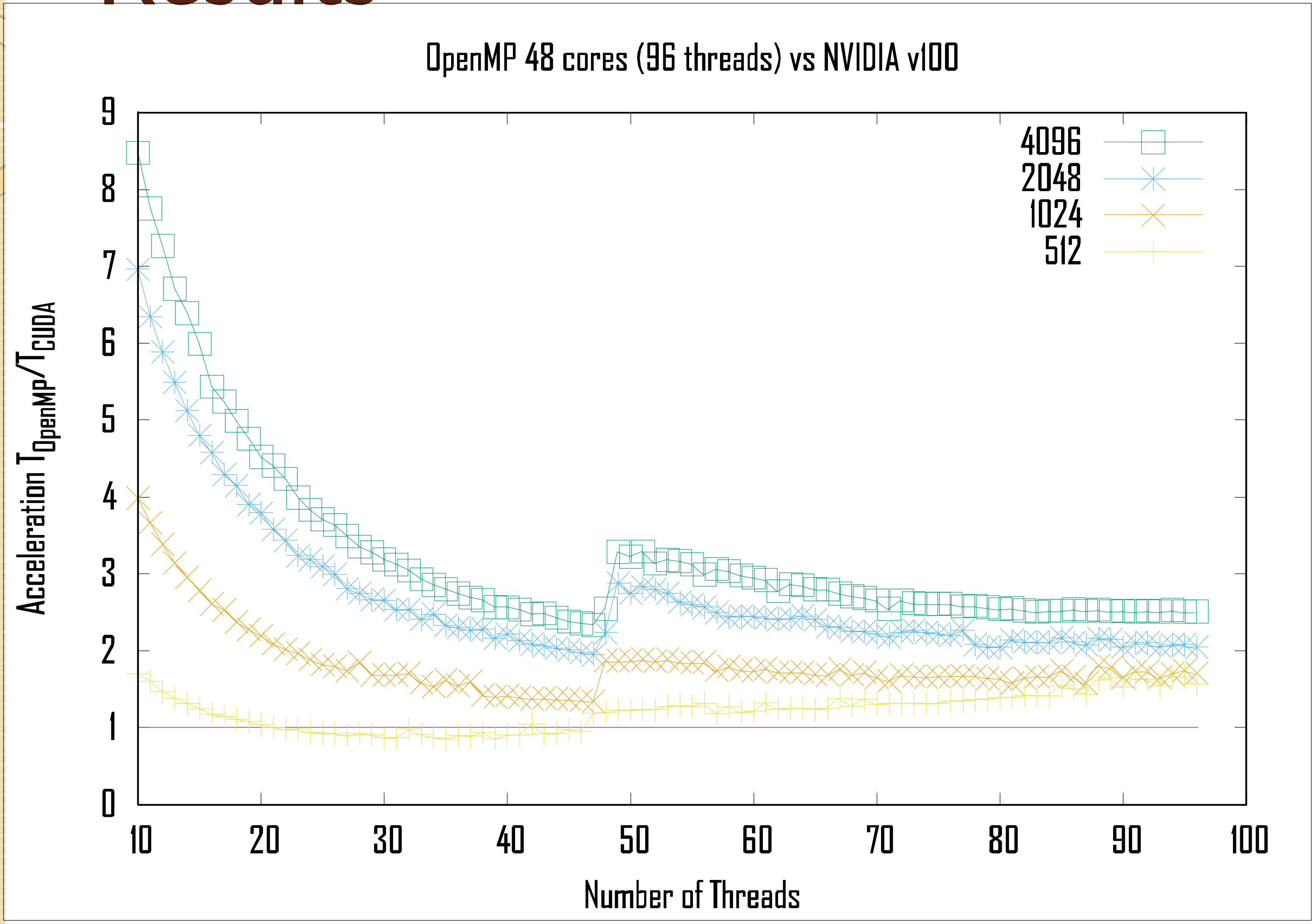
Experiments

- GPU and CUDA:
  - Each algorithm cell operation is a CUDA kernel: noise, decay, pumping and stimulated emission. There is also a shannon entropy kernel.
  - Syncronized kernel execution: each kernel must wait the kernel before.
- Two tests:
  - Geforce GTX 1050TI vs Core i9 9900k
  - NVIDIA v100 vs Intel Xeon Platinum 8259CL (48 cores, 96 HW threads) --» Executed in the cloud (AWS)

Results


OpenMP 8 cores (16 threads) vs NVIDIA GTX 1050 ti
Acceleration T_{OpenMP}/T_{CUDA}
Number of Threads
4096
2048
1024
512
0
1
2
3
4
5
6
7
2
4
6
8
10
12
14
16

Results


OpenMP 48 cores (96 threads) vs NVIDIA v100
Acceleration T_{OpenMP}/T_{CUDA}
Number of Threads
4096
2048
1024
512
0
1
2
3
4
5
6
7
8
9
10
20
30
40
50
60
70
80
90
100

Conclusions

- It was pointed out in previous works that the speedups achieved with GPU when comparing to CPUs where x10 or above (sometimes even ×100 or x200) --» these distances are now significantly shortened (x3)
- Simulation by substituting the pseudorandom number generator by a different type of algorithm can speed up multicore processors.
- GPUs are always the best choice?