An efficient GPU implementation for large scale individual-based simulation of collective behavior

Ugo Erra*, Bernardino Frola†, Vittorio Scarano†, Iain Couzin‡
*Dipartimento di Matematica e Informatica, Università della Basilicata, Italy
Email: ugo.erra@unibas.it
†Dipartimento di Matematica e Applicazioni, Università di Salerno, Italy
Email: ber.frola@gmail.com, vitsca@dia.unisa.it
‡Department of Ecology and Evolutionary Biology, Princeton University, USA
Email: icouzin@princeton.edu

***Abstract*—In this work we describe a GPU implementation for an individual-based model for fish schooling. In this model each fish aligns its position and orientation with an appropriate average of its neighbors positions and orientations. This carries a very high computational cost in the so-called nearest neighbors search. By leveraging the GPU processing power and the new programming model called CUDA we implement an efficient framework which permits to simulate the collective motion of high-density individual groups. In particular we present as a case study a simulation of motion of millions of fishes. We describe our implementation and present extensive experiments which demonstrate the effectiveness of our GPU implementation.**

***Keywords*-gpu; individual-based model; simulation;**

I. Introduction

Individual-based simulation is a common way to implement autonomous characters or individuals to create crowds and other flock-like coordinated group motion. In this simulation an individual has a local behavior model and it moves by coordinating with the motion of each other individual. The number of individuals involved in such collective motion can be huge, from several hundred birds to millions of individuals.

Biologists consider the collective motion from a mathematical point of view by modeling each organism individually [2]. For instance, Couzin et al. [6] propose biological models for fish schools and bird flocks. Others models can incorporate experimental observations of the behaviors of the organisms [1], predicting that individuals typically interact with a fixed, relatively small, number of near neighbors.

In all of these models, the simulation of local perception is a key aspect for an interactive results. Each individual must take decisions according only with its neighbors and so it must be able to identify efficiently these individuals among all others in the world. This problem from computational point has a solution $O(n^2)$ by using a brute force approach which is a bottleneck for a massive simulation.

In recent years, graphics processing units (GPUs) outperform CPUs in both floating-point performance and memory bandwith. Furthermore, general purpose computation on the GPU (aka GPGPU) is becoming easier thanks to the new programing model CUDA that allows to exploit the GPU processing power by using a C-like programming language.

In this work, we present a GPU-based method for massive simulation of fish schooling model by using CUDA framework for the graphics card NVIDIA G8x series. We adopt the GPU processing power for implementing a uniform data grid to support local perception. The simulation uses grid cells to keep track of the individuals position and offloads the sorting to build up the data grid structure to the GPU. The sorting is performed inside the cells to optimize the search and then to quickly obtain information about neighbors for each individual in parallel.

The remaining of this paper is organized as follow: in the section II we review related work based on hardware solutions. In the section III we give some information related to the GPU and the new programming model CUDA. In the section IV we illustrate our implementation based on CUDA. Finally, the section VI and section VII are dedicated to experimental results and conclusion.

II. Related Work

It is worth noting that computer graphics animation has investigated in past years behaviors models from the point of view of visualization. Such models usually can generate visual pleasing and physical plausible animation in real-time by using simplified or complex models. In fact, since the introduction of the first behavior model introduced by Reynolds [20] it was clear that computing large crowds is a very heavy computing task for real-time computer graphics animation and several hardware based solutions have been investigated in order to alleviate this problem.

In [25], Zhou et al. present a parallel simulation running over a cluster of 16 processors interconnected with Myrinet switch. They were able to simulate in real-time 512 individuals by using an efficient load balancing scheme and a parallel strategy approach. In [18], Quinn et al. simulate 10.000 pedestrians at 45 frame per second(fps) on 10 processors of the SWARM cluster connected by a gigabit Ethernet switch.

Another approach to improve simulation time is to adopt spatial data structure or optimizations that are able to exclude individuals that have no influence. For instance, in Shao et

A video of this work is available at http://isis.dia.unisa.it/projects/behavert/video/hibi09/hibi09.wmv.

978-0-7695-3809-9 2009

DOI 10.1109/HiBi.2009.11

al. [23] and Reynolds [19] used a regular grid to accelerate the behaviors calculation and then exclude individuals too far. While in [24], Silva et al. present an extension of Reynolds model that uses self occlusion in the neighbor computation by using the GPU. The idea is to estimate the number of individuals occluding the viewpoint of each agent and quickly reject parts of the invisible individuals in the behaviors calculation. Experimental results show that this extension runs up to three times faster than the original algorithm.

Recently, researchers have begun to investigate the parallelism of the graphical processing units to speed up the simulation of large crowds. In [3], De Chiara et al. present a massive simulation and rendering of a behavioral model using the GPU. In [4], Courty et al. propose a framework called FastCrowd which permits to simulate and render a crowd of 10.000 individuals at 35 fps using the GPU. FastCrowd simulates a complex physics-based animation model that takes into account the influence of gaseous phenomena in the behavior of the crowd. In the same spirit, Richmond et al. in [21] present an efficient GPU implementation, for real-time simulation and visualization. The framework, called ABGPU, allows massive individual based modeling underlying graphical concepts of the GPU. In [11], Li et al. propose an individual-based model for fish schooling on the GPU. They observed a speed up of 230-240 times for the simulation of 100 fishes over the corresponding CPU code but they did not take in consideration a large scale simulation due to lack of optimizations that allows to identify efficiently neighbors among all existing agents in the world. In this work, we adopt the GPU processing power for implementing a uniform data grid to support local perception in the nearest neighbors search. As we well see, this approach fits well with distributed behavioral models and provides a performance increase when compared to the previous results.

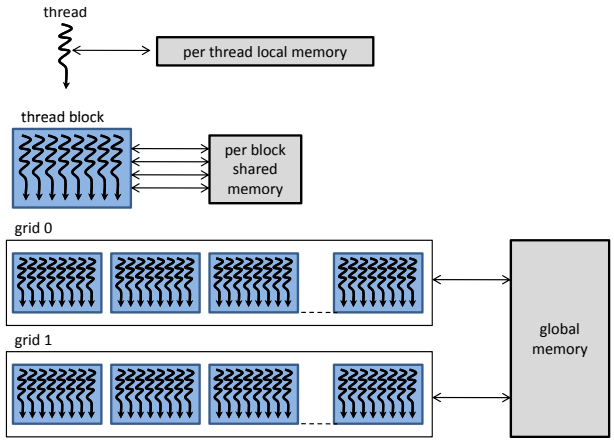

Fig. 1. Levels of parallel granularity and memory sharing on the GPU [14].

III. CUDA: Compute Unified Device Architecture

In the last years, the increasing performance of the GPUs has led researchers to explore mapping general non-graphics computation onto these new parallel architectures. The GPGPU phenomenon has shown some impressive results, but the limitations and difficulties of a mapping a problem via graphics APIs leaved these successful experimentations only to 3D graphics experts. The demand to use the GPU as a more general parallel processor motivated NVIDIA to release in 2006 a new generation of graphics cards (the so called G80 architecture or one of its successors) that significantly extended the GPU beyond graphics through a new unified graphics and computing GPU architecture and the CUDA programming model [14].

From the point of view of hardware model, the GPU architecture is built as a scalable array of multithreaded multiprocessors. Each multiprocessor consist of a number of SIMD ALUs which one called processor. The processor executes at the same time the same instruction in a SIMD fashion and has access to local registers. On the multiprocessor level, all processors of a multiprocessor have read/write access to a shared memory.

From the point of view of software model, CUDA is a minimal extension to C language which permits the writing of a serial program called *kernel*. A kernel executes in parallel across a set of parallel threads. Following the representation in Figure 1, each thread has a private local memory. The programmer organizes these threads into a hierarchy of thread blocks and grids. A thread block is a set of concurrent threads that can cooperate among themselves through barrier synchronization and have access to the shared memory with latency comparable to registers. The grid is a set of thread blocks that may each be executed independently. All threads have access to the same global, constant or texture memory. These three memory spaces are optimized for different memory usages and thus have different time access. For example, the read-only constant cache and texture cache are shared by all scalar processor cores and this speeds up reads from the texture memory space and constant memory space.

The grid and block sizes must be defined for every kernel invocation. Each block is mapped to one multiprocessor and then multiple thread blocks can be mapped on the same multiprocessor and are executed concurrently. Multiprocessor resources (registers and shared memory) are split among the mapped thread block. As a consequence, this limits the number of thread blocks that can be mapped onto the same multiprocessor. In order to maximize the number of threads supported by a multiprocessor it is important to take into account the resources required by each kernel. Then, the choice to design a framework by using several kernels is a crucial point to exploit the resources of the GPU and to maximize the amount of thread parallelism.

Further details on the GPU architecture and CUDA programming model are available in NVIDIA's CUDA Programming Guide [15].

IV. The Fish Schooling Model

Many animal groups such as fish schools and bird flocks clearly display structural order, with the behaviour of the

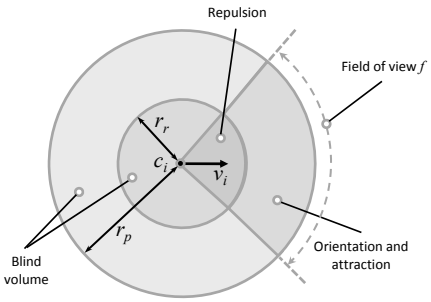

Fig. 2. A schematic representation of the fish schooling model. Each individual i has a position $\boldsymbol{c}_i$ and a direction $\boldsymbol{v}_i$. The zone of repulsion is represented by a circle of radius r_r while the zone of orientation and attraction is represented by an annulus of inner radius r_r and outer radius r_p. All individuals have a field of view of degree f and a blind volume of degree $(2\pi - f)$.

organisms so integrated that even though they may change shape and direction, they appear to move as a single coherent entity. Many of the collective behaviours exhibited by such groups can only be understood by considering the very large number of interactions among group members. Individual-based computer simulations are a very useful analytical tool to study such groups, and using this technique, it has been possible to demonstrate that group leadership, hierarchical control, and global information are not necessary for collective behaviour.

In the collective behaviors model two rules play a crucial part in the simulations. In the first rule, individuals attempt to maintain a minimum distance between themselves and others at all times. This rule has the highest priority and corresponds to a frequently observed behaviour of animals in nature [10]. In the second rule, if individuals are not performing an avoidance manoeuvre to maintain a minimum distance they tend to be attracted towards other individuals (to avoid being isolated) and to align themselves with neighbours [17]. These behavioural tendencies are simulated using local perception and simple response behaviours.

In our approach, we implemented the fish schooling model proposed originally by Couzin [7] and modified by Li [11]. In this model each individual has a strictly local perception of the space it occupies. None of the creatures being part of group has a full knowledge of the entire group. Hence, the decisions must be taken by every individual taking into account only local neighbors that are perceived from its fields of view *f*.

Groups are composed of n individuals. At time t, each individual i has a position $\boldsymbol{c}_i(t)$, a direction $\boldsymbol{v}_i(t)$, a speed s_i and maximum turning rate θ (Figure 2). Individuals simultaneously determine a new desired direction of travel $\boldsymbol{d}_i$ by considering neighbors within two behavioral zones. The first zone, called zone of repulsion, has a local interaction range r_r. Each individual attempts to avoid collision between itself and others individual j in this zone by turning away

$$\boldsymbol{d}_i(t+\Delta t) = -\sum_{j\neq i} \frac{\boldsymbol{c}_j(t) - \boldsymbol{c}_i(t)}{|\boldsymbol{c}_j(t) - \boldsymbol{c}_i(t)|}$$

This behavior has always the highest priority. If neighbors are not detected in the zone of repulsion then the individual i tend to align with j neighbors in the second zone called zone of orientation and attraction. This zone is a annulus of inner radius r_r and outer radius r_p around the individual. The desired direction of travel in this case is:

$$\boldsymbol{d}_i(t+\Delta t) = \omega_a \sum_{j\neq i} \frac{\boldsymbol{c}_j(t) - \boldsymbol{c}_i(t)}{|\boldsymbol{c}_j(t) - \boldsymbol{c}_i(t)|} + \omega_o \sum_{j=1} \frac{\boldsymbol{v}_j(t)}{|\boldsymbol{v}_j(t)|}$$

The terms ω_a and ω_o are the weights of attraction and orientation, respectively. As discussed in [11], by using these terms the individuals can balance their orientation and attraction preferences. For example, if $\omega_o > \omega_a$, individuals are more interested in orienting their direction of travel with their neighbors than attraction towards them while if the weights are approximately the same individuals balance their orientation and attraction preferences. Give the desired direction of travel randomness is included in the model by rotating the vector $\boldsymbol{d}_i$ by an angle draw from a normal distribution. Since individuals can only turn $\theta\Delta t$ radians in one time step, if the angle between $\widehat{\boldsymbol{d}}_i(t)$ and $\widehat{\boldsymbol{d}}_i(t+\Delta t)$ is greater than $\theta\Delta t$ the desired direction is not take into account and individuals rotate $\theta\Delta t$ towards it. While, if the angle is smaller than $\theta\Delta t$ the individuals position is updated as

$$\boldsymbol{c}_i(t+\Delta t) = \boldsymbol{c}_i(t) + s\widehat{\boldsymbol{d}}_i(t+\Delta t)$$

where $\widehat{\boldsymbol{d}}_i$ is the normalized vector. Note that during the computing of the desired direction we consider that orientation and repulsion zone have a blind area of degree $(2\pi - f)$ for which neighbors within these zone are undetectable while for zone of repulsion and zone of orientation individuals communicate with their neighbors (Figure 2).

The model illustrated above requires to identify neighbors out of whole world and in particular determines all individuals that fall inside the zone of repulsion and zone of orientation and attraction. A brute force approach requires $O(n^2)$ steps for a proximity screening i.e., compares each individual to all others and gathers all individuals within the range r_r and r_p. This approach is sufficient for a hundred individuals but it is clear that is computationally inefficient for the simulation of thousands individuals in real-time.

V. The GPU-based Method

In order to avoid the $O(n^2)$ complexity of the neighbors search due to the communication of each individual with every other individual in the group, we adopt a common strategy based on the assumption that interaction with some or all steering behaviors drops off with distance. Then, we are interested only to compute efficiently a relatively limited number of individuals. Not only is this computationally efficient but it captures an important aspect of organization in fish schools, that individuals are typically in close proximity (often less than a bodylength apart [17]) which limits the range over which individuals can detect one another [10] [5]. This biologically-based assumption alleviates the computational effort required by the neighbors search as well as the difficult to manage

dynamic data structures which are not trivial to implement on the GPU.

In order to accomplish this task, a static grid subdivides the world into cubic cells. Each cell in the world is of equal size and has a unique ID. To aggregate all the individuals inside the same cell in the GPU memory we assign a hash value to each individual based on its center position. Specifically, given an individual position the hash value is the ID of its cell. At the end of this step, the GPU performs a radix sort based on ID cells. This reordering allows us to identify quickly all individuals inside the same cell as well as to increase the cache hit rate during neighbors search. In fact, to find all individuals within a given region it is sufficient to consider individuals inside the cells that overlap a region of interest.

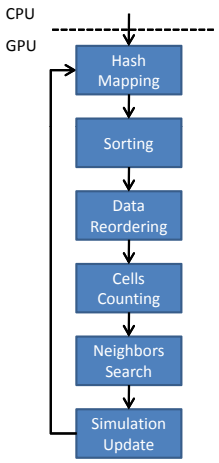

Fig. 3. The execution flow. The CPU is only in charge of the initialization of data structure and the generations of attributes such position, direction and speed.

The execution flow of the simulation can be divided as illustrated in Figure 3. The initialization of data structures is implemented in the CPU and consists of the random generation of individuals' attributes such position, direction and speed. These attributes are float4 data type and are copied in the global memory of the GPU as three distinct 1D arrays. Each subsequent step in the GPU consists of one or more separate kernels that run for each individual (except in the sorting step). When a thread calls a kernel, the kernel first figures out the calling thread's ID (using the CUDA special variable blockIdx and threadIdx). This thread ID is then used as individual ID to access which individual's attributes the current thread will work on. Then, each thread independently will be able to process the individuals' attributes inside the 1D arrays.

The execution flow can be divide into in several steps. The following subsections describe for each step the data structure, the role of sorting and how we implement the fish schooling model.

Hash Mapping

This step consists of a kernel that for each individual computes and stores a pair <CELL ID, INDIVIDUAL ID> in the global memory as a 1D array (see Figure 4a). This operation is performed from scratch at the beginning of each simulation cycle. The CELL ID of each individual is a hash of its centroid's coordinate. As in the work of Green [9], we use the linear CELL ID as the hash function which provides good coherence in memory access.

Sorting

The 1D array of previous step is sorted based on the CELL ID. The sorting is performed using a GPU radix sort described in [13] by Scott Le Grand. At end of this step, the array <CELL ID, INDIVIDUAL ID> maintains a list of individual IDs in cell order (see Figure 4b).

Data Reordering

The data reordering step objective is to reorder the position, the direction and the speed arrays by using the INDIVIDUAL ID order of the previous step (see Figure 4c). For this purpose we use a kernel that takes the INDIVIDUAL ID sorted list and an array as inputs and returns the reordered array. For this step our approach is to assign a separate kernel for each array to reorder and perform three separate kernel execution. In this way, we avoid the situation where there is an insufficient amount of any one or more types of resource needed for the simultaneous execution of the threads. Such a situation could be required by the thread assignment for managing of all attributes' individual in the same kernel.

Cells Counting

From the previous <CELL ID, INDIVIDUAL ID> array, we need to find the index start of any given cell in the sorted array. Then, given a cell, we will able to scan all individuals that fall inside the cell from the sorted list (see Figure 5). This is achieved by running a kernel which uses one thread per individual. For each thread, if the CELL ID of the current individual is different from the CELL ID of the previous individual then the current thread ID is used as index start of the current cell, and is written to another array. At the end of this step, we can scan over all individuals that falls inside. The scanning goes from from the index start up to the current cell index that no longer equals the index of the cell we are examining.

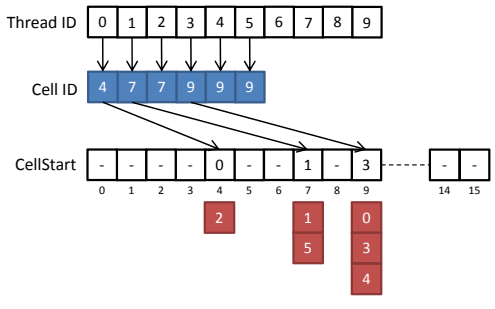

Fig. 5. The cells counting step. The CellStart array mantains the list of individuals that fall inside each cell.

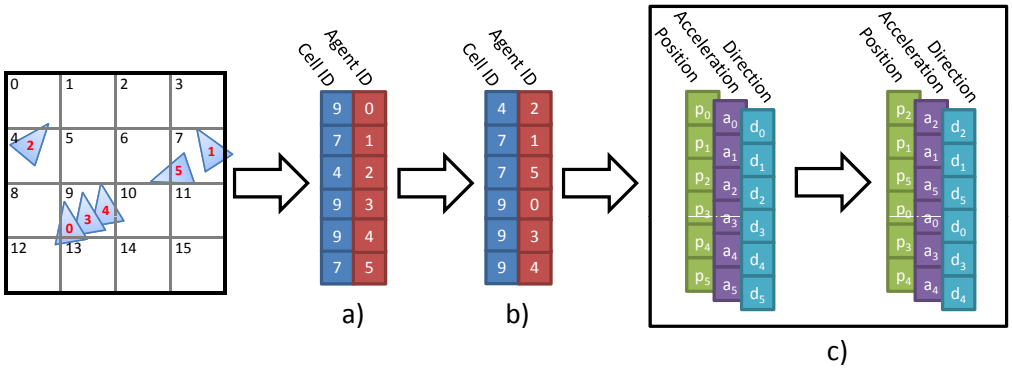

Fig. 4. The first three steps of the execution flow: a)Hash Mapping, b)Sorting, c)Data reordering.

Neighbors Search

This step is responsible to create a list of neighbors from the individual position. As we will see in the evaluation section, this is a critical phase and require a careful choice of graphics hardware resources to obtain superior performance results.

We take in consideration two types of spatial queries: range query and *k*-nearest neighbors (*k*-NN). The range query identifies all the individuals that fall inside a given radius. While the *k*-NN searches the *k* neighbors individuals nearest to the query individual. We adopt a hybrid approach for spatial query that we can called *k*-NN range query where *k* is maximum number of nearest neighbors that fall inside a radius. Furthermore, the search radius is defined in terms of depth search from the current cell. Then, a zero-depth search scans only the cell of the current individual, a one-depth search scans all $3 \times 3 \times 3$ cells, a two-depth scans $5 \times 5 \times 5$ cells, and so on.

In this approach, the cell size is important to trade-off the correct simulation with the accelerations of *k*-NN range query. If the cell is too small, the filling of the uniform grid could be require additional computational times and could be necessary to enlarge the search radius. On the other hand, if the cell is too big there could be too many individuals that fall inside the cell and we could lose the benefits of the uniform data grid for neighbors search. Another important aspect is the the size of the grid that enclose the simulation. We adopt the simple following strategy. Given the grid density d as the number of individuals per grid cells, the grid size v is calculated with the equation $v = \sqrt[3]{n/d}$, where n is the total number of individuals.

Simulation Update

Once we have obtained the list of neighbors for each individual we can calculate the interactions between them. This is achieved by running a kernel for each individual. Then, each thread by using the list of neighbors computes the desired direction of travel described in section IV. Of course, the thread has to distinguish between the individuals at distance r_r and r_p. Figure 9 shows a rendering of the fish schooling model during the execution.

VI. Performance Evaluation and Discussion

We performed a series of experiments in order to measure the performance of our approach. The tests were executed on a Core 2 Duo 2.0Ghz equipped with 2GB RAM and Geforce 8800GTX 768MB (Compute Capability 1.0). All the kernels were written in CUDA 2.1 and the application in C++.

To begin a simulation, individuals are placed in a bounded region (so that each individual initially interacts with at least one other individual), with random positions and directions of travel. The parameters were fixed to be $r_r = 1$, $r_p = 7.0$ (and $r_p = 1.5$), $\omega_o/\omega_a = 16$, $f = 350°$, $s = 1$, $\Delta = 0.2$, $\sigma = 0.01$, and $\theta = 115°$. We considered the average values for 3000 steps which is sufficient to obtain a stable simulation without the influence of initial random position and velocities and allowing individuals to meet and form small local groups.

On the left of Figure 6, we report a simulation considering only 31 nearest neighbors while steering each individual and a cell size equals to r_p that is 7.0 and 1.5 with a density of 0.005. With these values it is possible to simulate about $900K$ individuals per second with $r_p = 7.0$ and $6.2M$ individuals per second with $r_p = 1.5$. This result is obtained with an execution configuration where the block size is 128 and the number of total blocks is always computed as (total threads / block size). In our experimentations, these values fit well with the resources requirements of our kernel code.

From point of view of visualization our GPU implementation is able to visualize about 8192 (for $r_p = 1.5$) and 65536 (for $r_p = 7.0$) individuals at 60 fps as shown on the right of Figure 6). Note that a frame rate of 60 times per second is more than sufficient for smooth animations allowing to interact and visualize the simulation in real-time. De Chiara et al. [3] simulated 1600 individuals at 60 fps with obstacles avoidance too. While in [19], Reynolds reporting 15000 individuals at 60 fps on PlayStation3 development system. It must also be noted that due to different graphics hardware generations or different architectures these results are difficult to compare

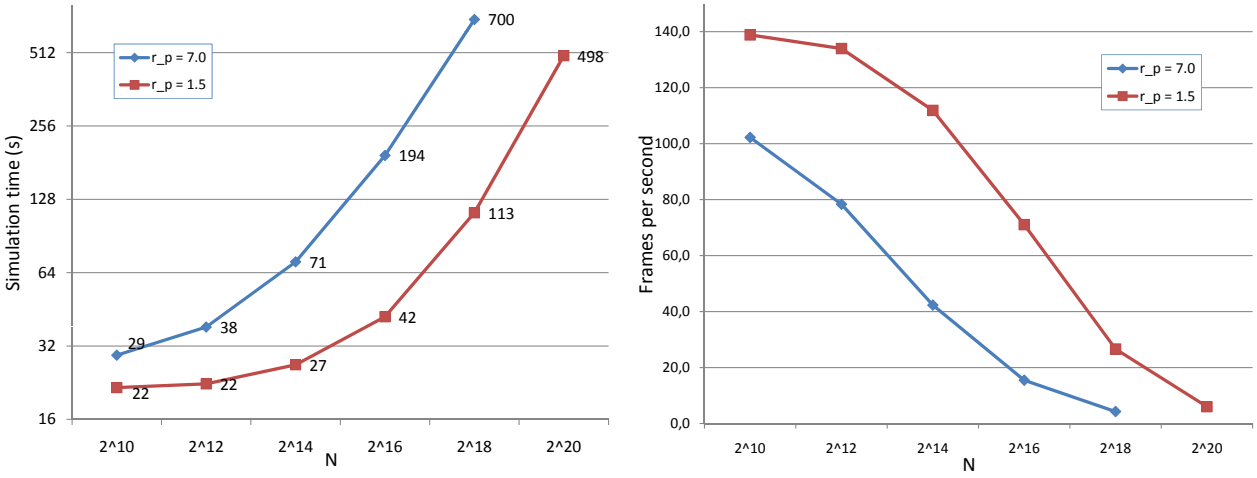

Fig. 6. Performances with increasing number of individuals. The simulation is performed with maximum 31-NN and a cell size equals to r_p. The grid density d is 0.005. On the left, computational times in seconds to complete a simulation of 3000 steps. On the right, visualization time in frames per second with rendering.

and we can only report that while we consider 31 neighbors these implementations considered only 5 neighbors. Moreover, these implementations consider the Reynolds model which is computationally less expensive than the Couzin model. The Reynolds model considers only one zone and the direction of travel is achieved performing a weighted sum of steering behaviors from all the neighbors perceived inside the field of view. In ABGPU [21], authors used a hardware comparable with our configuration to simulate 65536 individuals at 60 fps. In both approaches, up to a million of individuals can be simulated at 5 fps without visualization but with a decreasing number of individuals our approach produces better performance. In particular, our GPU implementation gives as average 25 percent better performance than the ABGPU system with different numbers of individuals ranging from 2^{12} to 2^{20} and also this implementation consider the Reynolds model.

Finally, we performed two experimentations related to the variation of the ratio ω_o/ω_a and k in the k-NN range query for $r_p = 1.5$, $r_p = 7.0$ and 65000 individuals. The Figure 7 shows different values of the ratio ω_o/ω_a. In general for different values of r_p the overall performances are affected slightly. This is very interesting result because the use of uniform static grid in our implementation allows to simulate different orientation and attraction preferences maintaining constant the performances of the simulation. The Figure 8 shows a simulation with different values of k. In this case, for $r_p = 1.5$ the choice of k has no influence because the small radius of the zone of orientation and attraction allows to discard a priori several individuals before the k-NN range query phase starts. While, for $r_p = 7.0$ the overall performances are significantly affect. This is due to the k-NN range query phase that has to performs much more comparison due to a large number of individuals that fall inside the zone of orientation and attraction.

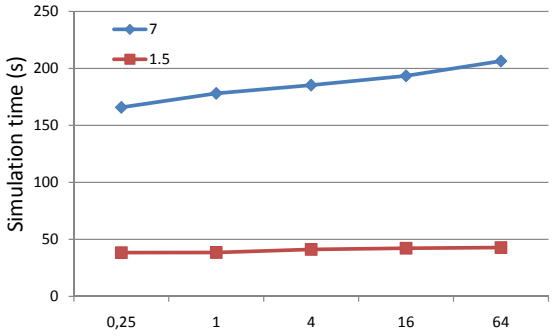

Fig. 7. Recorded performance with different values of ω_o/ω_a.

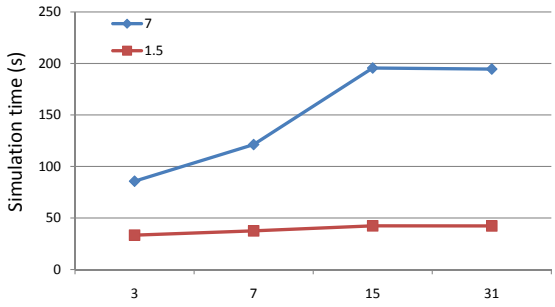

Fig. 8. Recorded performance with different values maximum number of nearest neighbors k.

VII. Conclusion and Future Work

This paper has described a GPU-based method for running high performance individual-based simulations based on CUDA. We adopt an approach based on a uniform data grid in order to accelerate the neighbors search by using the parallel architecture of the GPU. Our results show that this approach

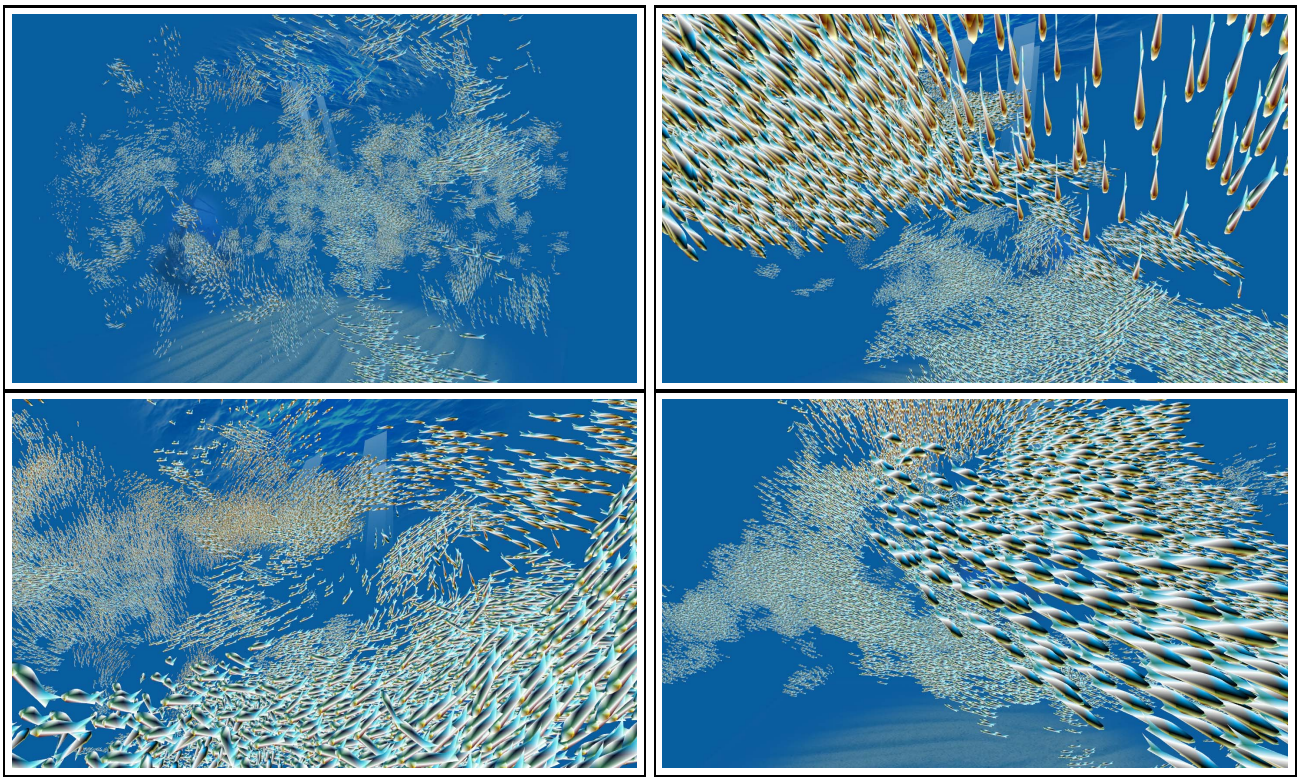

Fig. 9. Two renderings of implemented model. In this case the GPU performs simulation and visualization by using OpenGL [16]. In order do not overload the GPU the 3D fish is a low-poly model with 80 polygons.

can be very effective for such simulations.

We are going to support this work with some extensions. The memory hierarchy of the GPU is quite complex and we need exhaustive performance analysis to maximize the GPU utilization. In particular, an advanced extension concerned with a better use of the shared memory will help to optimize the neighbors search.

By exploiting the GPU processing power, we expect that it will be possible to simulate more complex models or to integrate features like collective motion during escape and pursuit response [22]. Our approach extends beyond the study of fish schools and animal groups. With appropriate modifications of the interaction terms it could be modified to simulate very effectively a large number of systems in which local interactions among mobile elements scale to collective behavior, from cell aggregates, including tumors [8] and bacterial biofilms [12], to aggregates of vertebrates such as migrating wildebeest, and even human crowds. In addition, our capacity to include variation among individuals could have enormous potential for the study of the evolution of collective behavior. For this purpose, we are going to design a flexible and extensible framework based on a plug-in architecture. This framework will permit developers to extend as they see fit with other behavioral models for longer times and over more realizations.

REFERENCES

[1] M. Ballerini, N. Cabibbo, R. Candelier, A. Cavagna, E. Cisbani, I. Giardina, V. Lecomte, A. Orlandi, G. Parisi, A. Procaccini, M. Viale, and V. Zdravkovic, "Interaction ruling animal collective behaviour depends on topological rather than metric distance: Evidence from a field study," *PNAS*, vol. 105, p. 1232, 2008.

[2] S. Camazine, N. R. Franks, J. Sneyd, E. Bonabeau, J.-L. Deneubourg, and G. Theraula, *Self-Organization in Biological Systems*. Princeton, NJ, USA: Princeton University Press, 2001.

[3] R. D. Chiara, U. Erra, V. Scarano, and M. Tatafiore, "Massive simulation using gpu of a distributed behavioral model of a flock with obstacle avoidance," in *VMV*, 2004, pp. 233–240.

[4] N. Courty and S. R. Musse, "Simulation of large crowds in emergency situations including gaseous phenomena," in *CGI '05: Proceedings of the Computer Graphics International 2005*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 206–212.

[5] I. D. Couzin and Krause, "Self-organization and collective behavior of vertebrates," *Advances in the Study of Behavior*, no. 32, pp. 1–75, 2003.

[6] I. D. Couzin, J. E. N. S. Krause, R. James, G. D. Ruxton, and N. R. Franks, "Collective memory and spatial sorting in animal groups," *Journal of Theoretical Biology*, vol. 218, no. 1, pp. 1–11, September 2002.

[7] I. D. Couzin, J. Krause, N. R. Franks, and S. A. Levin, "Effective leadership and decision-making in animal groups on the move," *Nature*, vol. 433, no. 7025, pp. 513–516, February 2005.

[8] T. Deisboeck and I. D. Couzin, "Collective behavior in cancer cell populations," *Bioessays*, vol. 31, no. 1, pp. 190–197, 2009.

[9] S. Green, "Cuda particles," *nVidia Whitepaper*, November 2007.

[10] J. Krause and G. Ruxton, *Living in Groups*. Oxford University Press, USA, 2002.

[11] H. Li, A. Kolpas, L. Petzold, and J. Moehlis, "Parallel simulation for a fish schooling model on a general-purpose graphics processing unit," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 6, pp. 725–737, 2009.

[12] C. D. Nadell, J. B. Xavier, and K. R. Foster, "The sociobiology of biofilms," *FEMS Microbiology Reviews*, vol. 33, no. 1, pp. 206–224, 2009.
[13] H. Nguyen, *Gpu gems 3*. Addison-Wesley Professional, 2007.
[14] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," *Queue*, vol. 6, no. 2, pp. 40–53, 2008.
[15] Nvdia, *NVIDIA CUDA Compute Unified Device Architecture - Programming guide*.
[16] Opengl, D. Shreiner, M. Woo, J. Neider, and T. Davis, *OpenGL(R) Programming Guide : The Official Guide to Learning OpenGL(R), Version 2 (5th Edition)*. Addison-Wesley Professional, August 2005.
[17] B. L. Partridge, "The structure and function of fish schools," *Scientific American*, pp. 114–123, June 1982.
[18] M. J. Quinn, R. A. Metoyer, and K. Hunter-zaworski, "Parallel implementation of the social forces model," in *in Proceedings of the Second International Conference in Pedestrian and Evacuation Dynamics*, 2003, pp. 63–74.
[19] C. Reynolds, "Big fast crowds on ps3," in *Sandbox '06: Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*. New York, NY, USA: ACM, 2006, pp. 113–121.
[20] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1987, pp. 25–34.
[21] P. Richmond and D. Romano, "Agent based gpu, a real-time 3d simulation and interactive visualisation framework for massive agent based modelling on the gpu," 2008.
[22] P. Romanczuk, I. D. Couzin, and L. Schimansky-Geier1, "Collective motion of animal groups due to escape and pursuit behavior," *Physical Review Letters*, vol. 102, 2009.
[23] W. Shao and D. Terzopoulos, "Autonomous pedestrians," *Graph. Models*, vol. 69, no. 5-6, pp. 246–274, 2007.
[24] A. R. Silva, W. S. Lages, and L. Chaimowicz, "Improving boids algorithm in gpu using estimated self occlusion," in *Proceedings of SBGames'08 - VII Brazilian Symposium on Computer Games and Digital Entertainment*. Sociedade Brasileira de Computação, SBC, 2008, pp. 41–46.
[25] B. Zhou and S. Zhou, "Parallel simulation of group behaviors," in *WSC '04: Proceedings of the 36th conference on Winter simulation*. Winter Simulation Conference, 2004, pp. 364–370.