# Fast Collision Detection for Nanosimulators

Pieter Stroobant*, Luca Felicetti†, Wouter Tavernier*, Didier Colle*,
Mauro Femminella†, Gianluca Reali† and Mario Pickavet*
* IDLab, Ghent University - imec, † Department of Engineering, University of Perugia
Email: pieter.stroobant@ugent.be, ing.luca.felicetti@gmail.com, wouter.tavernier@ugent.be,
didier.colle@ugent.be, mauro.femminella@unipg.it, gianluca.reali@unipg.it, mario.pickavet@ugent.be

***Abstract*—Particle-based nanosimulators are an indispensable tool for studying diffusion-based molecular communication (DMC) processes in which the diffusion process is affected by interactions among the involved particles. Efficiently finding which particles are interacting is a computationally challenging task and can easily become a bottleneck. This paper studies different techniques that allow to detect which particles are interacting/colliding. Recursive and hierarchical grid based approaches are proposed and a GPU and multithreaded CPU implementation are evaluated and compared to an existing nanosimulator.**

***Index Terms*—collision detection, multi-level grids, nanocommunication**

## I. Introduction

Within the context of diffusion-based molecular communication, there are cases in which the impact of interactions between emitted particles cannot be ignored. For example, when the density of the transmitted molecules is high, the collective diffusion behaviour may be affected [1].

Particle-based nanosimulators, such as BiNS2 [2] [3] and N3Sim [4], take the effect of these interactions into account by tracking of the positions of all particles that may interact. Nano-objects are enclosed by a bounding volume, such that interaction between the objects may occur only if the bounding volumes are overlapping.

Starting from a typical nanonetwork setting, this work proposes several algorithms that allow to find collisions between these bounding volumes. Realistic nanonetworks feature a large number of transceivers, possibly resulting in millions of nano-objects. With this in mind, we pay attention to parallelism and aim to significantly improve upon the detection times of existing nanosimulators. The resulting algorithms are compared to a collision detection algorithm that is implemented in the BiNS2 simulator.

## II. Setting

Several properties of nanocommunication and -networks are important to consider when designing a collision algorithm:

- the majority of the objects are moving: datastructures must either allow fast updates, or efficient construction (such that they can be rebuilt at each timestep);
- the object aspect ratio is typically small: nano-objects and biological cells are usually not elongated and bounding spheres allow for a reasonably tight fit;
- object types can be of widely varying sizes: for example, transceivers can exceed the size of the emitted particles by several orders of magnitude;
- the number of objects greatly outnumbers the number of different object types and size variation within a single object type is limited: a simulation may contain several types of nanomachines/cells, but machines of the same type are similar in size;
- objects may be evenly distributed through the environment, but they might also be concentrated in certain areas

## III. Algorithms

Based on the requirements concerning update/build costs and the object shapes, partitioning the environment with uniform grids is straightforward approach. The simulated space is overlaid with a grid and each object is mapped to all overlapping grid cells. Colliding object pairs can be found by iterating over all cells and verifying the whether there is a collision between objects that are mapped to the same cell. Grids have very low construction times and combine well with spherical object shapes [5].

However, uniform grids are inefficient in handling object of varying sizes. Recursive grids solve this problem by mapping all objects to a grid with a size corresponding to the largest object size. This grid is used for finding collisions that involve at least one large object. Collisions between small objects that are mapped the same grid cell are found by recursively constructing a smaller grid.

Since objects may be spread out over a large environment, we cannot use a dense array to store the cells, as such an approach might cause unlimited memory consumption. Therefore, the rastercell coordinates are hashed. We implement two flavours of our recursive grid algorithm: one in which a hashtable stores objects that are mapped to the same cell (HashGrid), and another approach in which an array is filled with pairs of $(cellId, objectId)$s (ArrayGrid). By sorting this array on $cellId$, the objects with the same cellId are stored after each other, and any $cellId$ can quickly be retrieved by employing a binary search. This array-based approach is a well-known technique for grid construction in GPUs [6], [7], but we are unaware of any multi-core CPU implementations.

Both algorithms can easily be parallelized: the size of the hashtable can be precalculated and objects that are mapped to the same slot can be efficiently chained by employing a lock-

free linked lists. The array representation can be efficiently sorted using a parallel radix sort [8].

Additionally, we optimized the array-based recursive algorithms for efficient GPU execution. This approach uses a *hierachical* grid, rather than a *recursive* grid: instead of creating a smaller grained grid for each rastercell separately, all the smaller sized objects are bundled when constructing the smaller grid (EagerGrid). We also implemented an opportunistic/lazy version of this algorithm (LazyGrid). In this variant, smaller sized objects that are mapped to the same rastercell are not passed to the smaller level grid, if their number is limited. In that case, a brute force approach is used to detect the collisions between the smaller sized objects. This technique allows avoiding the construction of smaller grained grids if the concentration of the small sized objects is limited.

## IV. Results

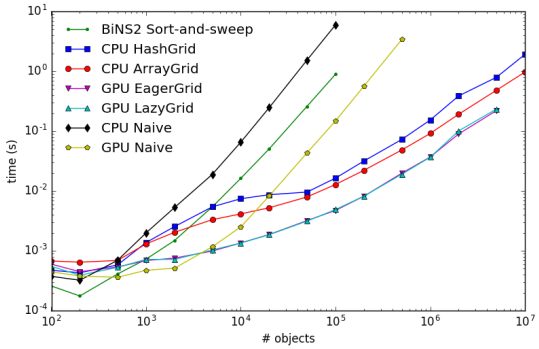
To evaluate the performance of the proposed algorithms for varying numbers of same-sized objects, we randomly generate scenarios with the required number of spherical objects, uniformly distributed in a spherical environment. The radius of the sphere is selected to maintain a constant density of 5% (i.e. a fraction 5% of the volume is filled with nano-objects). The resulting measurements, for an Intel Xeon E5645 processor, connected to an Nvidia Geforce GTX 580 are reported in Fig. 1a. The collision times for performing *naive* all-pairs collision detection are provided for comparison purposes.

All the proposed algorithms have a linearly increasing detection time and therefore outperform the BiNS2 sort-and-sweep algorithm. Furthermore, the arraygrid algorithm clearly outperforms the hashgrid algorithm. This is due to the memory access pattern of the hashgrid algorithm: during construction, it requires random memory accesses to a very large hash table. For equisized objects, both GPU algorithms behave identically, and thus achieve the same performance.

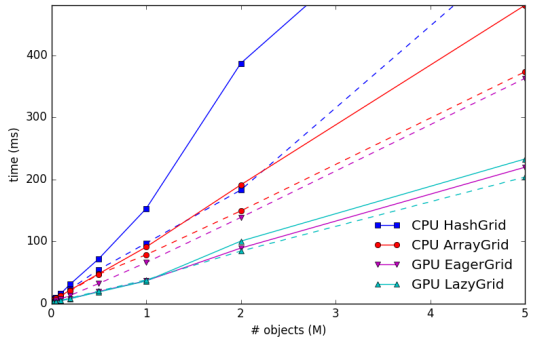
We also measured the collision detection time for scenarios with 5, equally frequent, object types of radius $1/2^0, 1/2^1, ..., 1/2^4$. Figure 1b compares the average computing time for 5 object types to the measurements with one object type. We notice that the collision detection time decreases for the recursive grid algorithms and the GPU lazy grid algorithm. This is because smaller sized objects are mapped to fewer rastercells, speeding up the raster building process. Additionally, these algorithms are able to avoid the construction of small grained grids. This is in contrast to the eager grid algorithm, which always needs to build as many grid levels as there are object sizes.

## V. Conclusion

We proposed, implemented and evaluated several varieties of multi-level grid collision detection algorithms for nanocommmunication simulations, both on CPU and on GPU. The proposed algorithms outperform the existing solutions by a significant margin, and have potential to accelerate the nanosimulators, allowing for more complex simulations and faster results.




(a) Equisized objects




(b) Equisized objects (full line) vs. 5 object sizes (dashed line)

Fig. 1: Average collision detection time for varying numbers of objects


## Acknowledgements

Pieter Stroobant is funded by a Ph.D. grant of Ghent University, Special Research Fund (BOF).


## References


[1] A. J. C. Ladd, H. Gang, J. X. Zhu, and D. A. Weitz, "Time-dependent collective diffusion of colloidal particles," *Phys. Rev. Lett.*, vol. 74, pp. 318–321, Jan 1995.

[2] L. Felicetti, M. Femminella, and G. Reali, "A simulation tool for nanoscale biological networks," *Nano Communication Networks*, vol. 3, no. 1, pp. 2–18, 2012.

[3] L. Felicetti, M. Femminella, and G. Reali, "Simulation of molecular signaling in blood vessels: Software design and application to atherogenesis," *Nano Communication Networks*, vol. 4, no. 3, 2013.

[4] I. Llatser, D. Demiray, A. Cabellos-Aparicio, D. T. Altilar, and E. Alarcón, "N3sim: Simulation framework for diffusion-based molecular communication nanonetworks," *Simulation Modelling Practice and Theory*, vol. 42, pp. 210 – 222, 2014.

[5] C. Ericson, *Real-Time Collision Detection*. Boca Raton, FL, USA: CRC Press, Inc., 2004.

[6] J. Kalojanov and P. Slusallek, "A parallel algorithm for construction of uniform grids," in *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, (New York, NY, USA), pp. 23–28, ACM, 2009.

[7] J. Kalojanov, M. Billeter, and P. Slusallek, "Two-Level Grids for Ray Tracing on GPUs," in *EG 2011 - Full Papers* (O. D. Min Chen, ed.), (Llandudno, UK), pp. 307–314, Eurographics Association, 2011.

[8] J. Wassenberg and P. Sanders, "Engineering a multi-core radix sort.," in *Euro-Par (2)* (E. Jeannot, R. Namyst, and J. Roman, eds.), vol. 6853 of *Lecture Notes in Computer Science*, pp. 160–169, Springer, 2011.