Dynamic Adaptation of Broad Phase Collision Detection Algorithms

Quentin Avril* Valérie Gouranton† Bruno Arnaldi‡
Université Européenne de Bretagne, France INSA, INRIA, IRISA, UMR 6074, F-35043 RENNES

Abstract

In this paper we present a new technique to dynamically adapt the first step (broad phase) of the collision detection process on hardware architecture during simulation. Our approach enables to face the unpredictable evolution of the simulation scenario (this includes addition of complex objects, deletion, split into several objects, ...). Our technique of dynamic adaptation is performed on sequential CPU, multi-core, single GPU and multi-GPU architectures. We propose to use off-line simulations to determine fields of optimal performance for broad phase algorithms and use them during in-line simulation. This is achieved by a features analysis of algorithmic performances on different architectures. In this way we ensure the real time adaptation of the broad-phase algorithm during the simulation, switching it to a more appropriate candidate. We also present a study on how graphics hardware parameters (number of cores, bandwidth, ...) can influence algorithmic performance. The goal of this analysis is to know if it is possible to find a link between variations of algorithms performances and hardware parameters. We test and compare our model on 1,2, 4 and 8 cores architectures and also on 1 Quadro FX 3600M, 2 Quadro FX 4600 and 4 Quadro FX 5800. Our results show that using this technique during the collision detection process provides better performance throughout the simulation and enables to face unpredictable scenarios evolution in large-scale virtual environments.

Index Terms: I.3.1 [Computer Graphics]: Hardware Architecture—Parallel processing; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling

1 Introduction

In the virtual reality field, several thematics are considered as major bottlenecks due to the computation time that some algorithms require. Collision detection is one of them and during the three last decades, the research in this field has intensively look for ways to speed up algorithms. It is a large research field dealing with what appears to be an easy problem: determining if two (or several) objects collide. Collisional computation is used in several fields including computer animation, robotics, physical or mechanical simulations (medical, cars industry, civil engineering..), video games and haptic applications. Virtual environments and 3D objects are constantly evolving to become increasingly large and complex. The performance level for a user in real time becomes harder to ensure in large-scale virtual environments. The use of parallel processing has become a necessity to take advantage of recent gains of Moore's Law. Processors specialists have been able to provide clock frequency increases and parallelism improvement in instruction sets. In that way, single-threaded applications have become faster on a new generation of processors without any modification. To have better management of power consumption, they promote multi-cores architectures. So, software, libraries and any piece of code must be written and specially adapted to take advantage of this hardware evolution. It is no longer possible to rely on the evolution of processing power to overcome the problem of real-time collision detection.

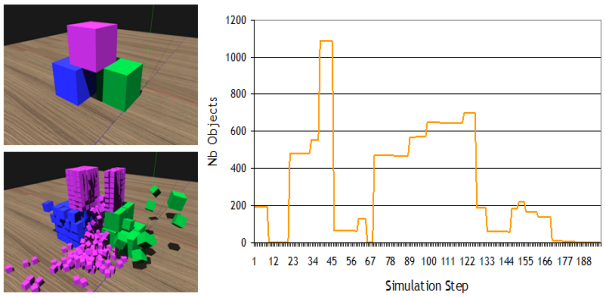

Figure 1: One sample of environment used to perform tests and to compare algorithmic performances in relation to the number of objects. Simulation starts with few cubes (top left) and the user can then split or delete cubes as much as he wants (bottom left). The aim is to vary the number of objects in the scene from few to several thousand as illustrated on the right curve.

With the impressive hardware advances these last years, algorithms for collision detection have greatly improved but remain mostly very specific to certain types of simulation scenarios (convex or non-convex, dynamic or static, rigid or deformable objects, 2 or n body simulation ...). Establishing a comparison between them is not as simple as one would imagine. A lot of physical-based simulations have an unpredictable evolution and can become, at any moment, a huge computations bottleneck (addition of objects, deletion, splitting into thousands of pieces, separate into non convex parts ...). So, using the most efficient and fast collision detection algorithm in an unpredictable scenario will not necessarily insure the best performance throughout the simulation.

We propose a new dynamic adaptation of broad phase algorithms enabling to face scenario evolution during a simulation. We perform off-line simulations to determine the field of optimal performance for each broad phase algorithm on every kind of computer architecture. We then perform an analysis on obtained curves by extracting features points. We have tested our benchmark application on several architectures to highlight architecture parameters and components that influence algorithms performances. In order to better understand performance differences between algorithms, we also present in this study an analysis on how hardware parameters influence the algorithmic performance in collision detection.

The rest of our paper is organized as follows: In Section 2 we report related work on broad phase in collision detection and parallel algorithms. Section 3 describes the technical context from sequential CPU algorithms to multi-GPU ones. In Section 4 we present our solution of a dynamic algorithm adaptation. Results are presented in Section 5. Section 6 presents the study on the influence of hardware parameters on algorithmic performances. Then, we conclude and open on future work on Sections 8.

*e-mail: quentin.avril@irisa.fr
†e-mail: valerie.gouranton@irisa.fr
‡e-mail: bruno.arnaldi@irisa.fr

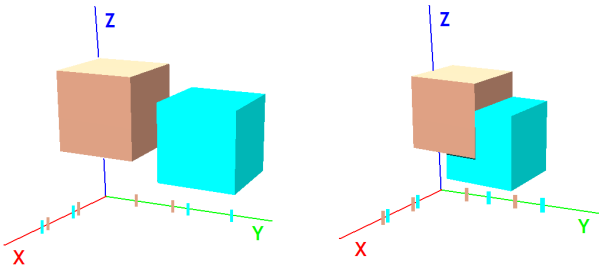

Figure 2: Example of the "Sweep and Prune" algorithm with a non-overlapping condition (left) and an overlapping one (right).

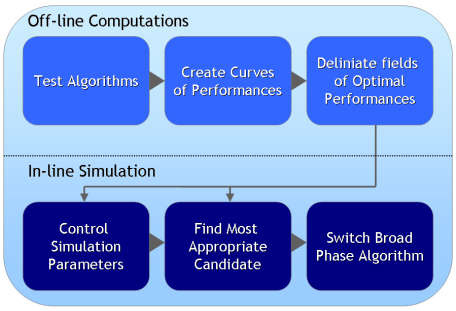

Figure 3: Off-line computations are performed to evaluate fields of optimal performance of each algorithm. Results are then used during the run-time process to switch the algorithm by a more appropriate one.

2 Related Work

Collision detection problem has been intensively studied for many years in the virtual reality field [11, 17, 23]. Our review will focus on the broad phase process in the collision detection pipeline and parallel algorithms.

2.1 Broad Phase in Collision Detection

Testing all object pairs during an n-body simulation in a virtual environment tend to perform n^2 pairwise checks. For this reason, collision detection is built like a pipeline [10] composed by two main parts: broad and narrow phases. The principal goal of this pipeline is to apply successive filters with a growing precision to determine the presence or the absence of collisions.

The broad phase is in charge of an efficient and quick removal of the object pairs that are not in collision. Several techniques have been proposed to speed-up or improve algorithms. Brute force approach is based on the comparison of the overall bounding volumes of objects. Because of its n^2 pairwise checks. This test is a bit exhaustive in a sequential execution but is the most suitable for a massive parallel one. In the bounding volume family many models have been proposed, such as spheres, Axis-Aligned-Bounding-Box (AABB) [4], Oriented-Bounding-Box (OBB) [6], discrete oriented polytopes (k-DOP) and many others. Topological approach is based on the positions of objects in relation to others. Methods proposed to use spatial partitioning and divide space into unit cells: regular grid, octree, Binary Space Partitioning (BSP) [18] or k-d tree structure [3].

In I-COLLIDE [5] used *"Sweep and Prune"*, a pseudo-dynamic object collision pruning method that reduced 3D collision detection between AABBs into three separate 1D problems. It is one of the most used methods in the broad-phase algorithms because it provides an efficient pairs removal and it does not depend on the objects complexity. The sequential algorithm of *"Sweep and Prune"* takes in input the overall objects of the environment and feeds in output a collided object pairs list. AABBs appear, in general, as perfect candidates for the algorithm because of their alignment on the environment axis (cf. Figure 2). The algorithm is in charge of the detection of overlaps between objects. A projection of higher and upper bounds of each AABB is made on the three environment axis. Three lists with overlapping pairs on each axis (x, y and z) are then obtained. We can notice in related work two related but different concepts on the way the *"Sweep and Prune"* operates internally: by starting from scratch each time or by updating internal structures. The first type is called brute-force and the second type persistent. Recently, the persistent algorithm has been enhanced by using a segmented interval list combined with subdivision [25] and it provides faster sequential execution in large scale virtual environments.

2.2 Parallel Collision Detection

In the high performance computing field, parallel solutions of collision detection algorithms are booming. We can distinguish two main ways of research, the one focusing on GPU and the other on CPU.

2.2.1 GPU-based Algorithms

Image-based algorithms have been proposed to exploit the growing computational power of graphics hardware because it is very efficient in rasterization of polygons. An advantage of using GPU is the un-use of precomputed volumetric data structures and its use with rigid or deformable objects. Image-space visibility queries have been proposed to perform the broad phase process [8]. Cinder [15] is an algorithm exploiting GPU to implement a ray-casting method to detect collision. GPU-based algorithms for self-collision and cloth animation have also been introduced [7].

2.2.2 CPU-based Algorithms

Several algorithms have been proposed working on multi-processors machines [14]. Depth-first traversal of bounding volumes tree traversal (BVTT) [21] and parallel cloth simulation [20] are good instances of this kind of work. Few papers also presented multi-threading use on single processor (Lewis et al. [16] propose a multithreaded algorithm to simulate planetary rings). Broad phase has also been developed on a Field-Programmable Gate Array (FPGA) [26].

Few papers appeared dealing with new parallel collision detection algorithms using multi-cores. A new task splitting approach for implicit time integration and collision handling on multi-cores architecture has been proposed [24]. Tang et al. [22] propose to use a hierarchical representation to accelerate collision detection queries and an incremental algorithm exploiting temporal coherence, the overall is distributed among multiple cores. They obtained a 4X-6X speed-up on a 8-cores based on several deformable models. Kim et al [12] propose to use a feature-based bounding volume hierarchy (BVH) to improve the performance of continuous collision detection. They also propose novel task decomposition methods for their BVH-based collision detection and dynamic task assignment methods. They obtained a 7X-8X speed-up using a 8-cores compared to a single-core. Hermann et al. [9] propose a parallelization of interactive physical simulations. They obtain a 14X-16X speed-up on a 16-cores compared to a single-core. A first adaptive parallelization of the pipeline stages have also been proposed running on a multi-core architecture[2]. The broad and narrow phase are executed in the same time using a buffer structure and the model enables to dynamically adapt the threads repartition during the simulation.

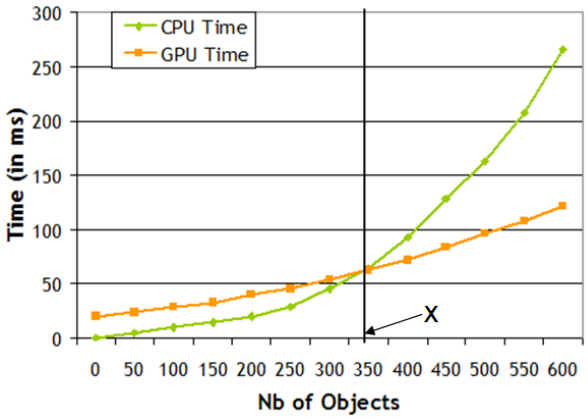

Figure 4: Example of computation time spent by the broad phase algorithm running on a CPU (green line) and GPU (orange line). Optimal performance fields we obtain are shown by the two areas. The X value has to be determine to precisely know when GPU solution becomes better, in terms of computation time, than CPU.

2.2.3 Hybrid-based Algorithms

More and more papers appear dealing with new hybrid solutions that run on multi-core and multi-GPU architecture. Kim and al [13] have presented an hybrid parallel continuous collision detection method *HPCCD* based on a bounding volume hierarchy. Recently, Pabst and al[19] have presented a new hybrid CPU/GPU method for rigid and deformable objects based on spatial subdivision. Broad and narrow phases are both executed on a multi-GPU architecture.

2.3 Positionning

Related work shows that a lot of accurate and fast parallel adaptation have been proposed for collision detection algorithm. But, as previously mentioned, using the most accurate and fast algorithm in a physical simulation with an unpredictable scenario evolution will not necessarily provide best performances. Algorithms have to be adapted during the simulation to to be as efficient as possible.

3 Technical Context

We present in this section the hardware-based algorithms we used to dynamically adapt collision detection algorithms. We explain the development and the adaptation of these algorithms on multi-cores, single GPU and multi-GPUs architectures. To determine what we call "fields of optimal performance", we implemented several algorithms running on different architectures. All of them are broad phase algorithms based on the well-known "Sweep and Prune" [5] and it has been transformed in a generic way to work on each available computation units (Sequential CPU, Multi-cores, Single GPU and Multi-GPUs). We work with the brute force method (cf Section 2.1) because it is the best candidate for the parallelization due to the independence in computations. There is no internal structure to update, we start from scratch each time. After projection on axis there are $\frac{(n^2-n)}{2}$ object pairs to test. Working with the persistent method that updates internal structure is hard to parallelize because computations need to be made on the overall axis to check among upper and lower bounds if there is an overlap or not. A way to parallelize it is to separate axis computation, so it means to divide it by three which is not really advantageous for parallelization.

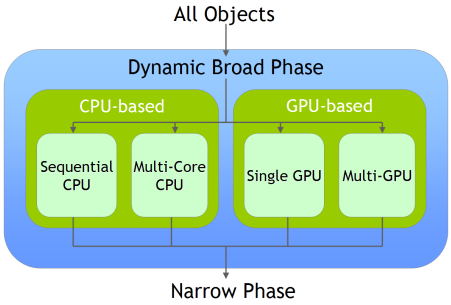

Figure 5: Simplified view of our dynamic broad phase adaptation. Input data are oriented to the most efficient broad phase algorithm.

3.1 Multi-core Algorithm

We proposed a novel approach to parallelize the "Sweep and Prune" algorithm on a multi-cores architecture [1]. The two main parts of the algorithm have been parallelized, the one in charge of updating bounding volumes and the other one in finding overlapping bounding volumes pairs. There is a synchronization point required between these two phases to merge computations of each cores. Critical writing sections and threads idling have been avoided and reduced by adding new data structure for each thread. The algorithm is able to work on a *n*-core architecture reducing computation time by almost 5X-6X on a 8-cores architecture.

3.2 Single GPU Algorithm

We developed a first implementation of the *"Sweep and Prune"* algorithm on GPU. We use the brute force solution of the algorithm and the CPU is only used to detect overlapping on each axis. Three lists of lower and upper bounds of bounding volumes are then obtain and transmitted to the GPU to be computed. The GPU is in charge of extracting pairs common to all three lists. This work is done by a CUDA[1] algorithm that assigns to each GPU threads a kernel function in charge of extracting pairs in a smaller dataset.

3.3 Multi-GPU Algorithm

Recently, we developed a multi-GPUs version of the "Sweep and Prune" based on the single GPU one. To separate computations between GPU devices during the broad phase process we use spatial subdivision technique and more precisely we divide space by number of GPUs. The spatial subdivision technique depends on the density distribution of objects in the environment. This dependence can be explained by wanting to homogeneously balance GPU's computation time.

4 Dynamic Algorithm Adaptation

We present in this section an analysis of off-line algorithmic performances and how they can be used during in-line simulations to perform an algorithmic dynamic adaptation.

4.1 Outline

An overview of our solution is illustrated on Figure 3, it is composed of two parts, namely off-line computations and their use during the in-line process. To illustrate our purpose, Figure 1 shows an example of a complex scenario evolution. The simulation starts with only few objects and then evolves quickly when user decides to add, remove or break objects into thousands of pieces. In this case, number of objects is not stable all along the simulation. Algorithms need to be adapted to face these perturbations. Naively,

[1] www.nvidia.com/object/cudahome.html

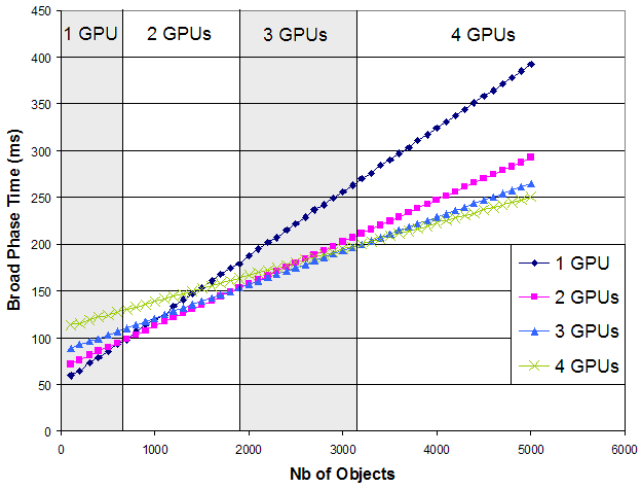

Figure 6: Results of off-line simulations performed on a 4*Quadro FX 5800 platform. We measured time spent by 1, 2, 3 and 4 GPUs in relation to the number of objects in the environment.

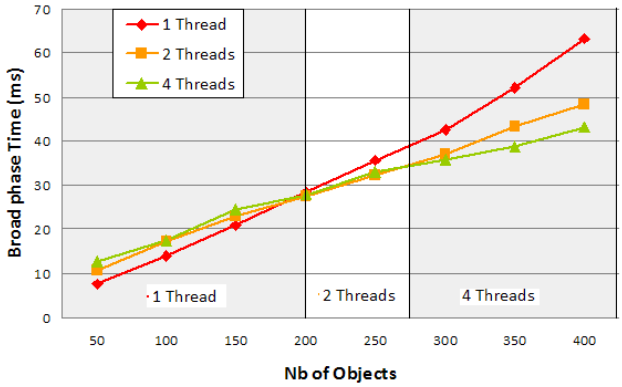

Figure 7: Results of off-line simulations performed on a 4 cores platform. We measured time spent by 1, 2 and 4 threads in relation to the number of objects in the environment.

we can say that a GPU solution for a simulation with thousands of object is the fastest way to compute the broad phase. We can also say that a CPU solution is the best one with very few objects. But we do not know exactly when one becomes better or worse than the other. It is also essential to know how many GPUs provide best performances and when one GPU needs to be included or removed from the broad phase loop.

4.2 Off-line Simulations

Our analysis of off-line algorithmic performances consists in an extraction of feature points to delimit fields of optimal performance of broad phase algorithms. The *"Sweep and Prune"* algorithm is only sensitive to the number of objects in the environment. It works with bounding volumes of objects, so time spent to compute overlapping between complex or simple objects is nearly the same. Based on this observation, we propose to analyze and evaluate behavior of each algorithm by varying the number of objects. These evaluations are performed during off-line simulations. Figure 4 shows a sample output, we can see computation time of a CPU and GPU broad phase algorithm in relation to the number of objects. This off-line simulation has been performed on a dual-core with a Quadro FX 3600M. We note that each algorithm has its own behavior and, depending on the number of objects, one of them is more appropriate because it uses less time to compute overlapping pairs. Based on an unpredictable scenario simulation as illustrated in Figure 1, we would like the computation time to be constantly as short as possible. Looking at the graph we are well aware that it is difficult to choose an algorithm better than everyone else but it is possible to determine what kind of algorithm is the most appropriate candidate for a specific range of number of objects.

We use a process of extraction of the intersection curves points. These points delineate fields of optimal performance of each algorithm. They are used during the in-line simulation by a module in charge of controlling and switching the broad phase algorithm. The computed intersection point between two curves is not exactly the real intersection point because we do not have curves functions (f,g) so we can not solve the equation $(f(x) = g(x))$ used to find similar point(s). Each off-line simulation, performed to test one algorithm, generates a data array with computation time information. We proceed to a comparison of time information to build optimal performance fields. Briefly, we have to determine the "X" value illustrated on Figure 4.

4.3 In-line Simulation

Results obtained with the extraction of optimal performance fields are used during in-line simulation. Borders of these fields are stored to be used during the broad phase switching process. We developed a module in charge of controlling number of objects in the environment. This module is also in charge of switching the algorithm by a more appropriate one when performance fields borders are crossed. We include the switching module in the global application and more precisely in the part in charge of the physical world. To insure a switch in real-time during the algorithm transfer, each algorithm takes in input the same data structure and gives in output the same result data. Our module is illustrated on Figure 5.

5 Results

Off-line simulations are required because algorithmic performances are dependent on the run-time architecture. We used 3 different platforms to test our solution, namely:

- Intel Core(x2) CPUX7900 @2.8GHz - Quadro FX 3600M
- Intel Xeon(x4) CPUX5472 @3.0GHz - 2 Quadro FX 4600
- Intel Xeon(x8) CPUX5482 @3.2GHz - 4 Quadro FX 5800

For example, figure 8 illustrates GPU results obtained on the third platform. Each curve shows computation time in relation to the number of objects. We can notice that the order, from the best to the worse computation time, is totally inversed between the beginning and the end.

Each curve has its own behavior compared to the number of objects in the environment. Unsurprisingly, using 4 GPUs to compute broad phase with a hundred of objects appears to be the worse way to proceed. Similarly, using only one single GPU for more than 4.000 Objects brings a loss of time. The interesting point is to notice that each algorithm does have its best moment of use. After measuring these results, we can cross curves data and find intersection points to delimit the field of optimal performance of each algorithm. During the in-line simulation, we know now when pulling out or adding a GPU in the broad phase loop.

Figure 7 presents CPU results obtained on the second platform. As noticed in the GPU results, the order of the best to the worse computation time is inversed between the beginning and the end. We border fields of optimal performance of each algorithm to reveal their "best moment of use". That shows that always using all

	Quadro 3600M	Quadro 4600	Quadro 5800
CPU->GPU (MB/s)	1856,12	2496,3	2463,36
GPU->CPU (MB/s)	559,52	2276,28	2309,6
Internal GPU (GB/s)	31,99	44,36	71,11
Memory Size (MB)	512	768	4096
Clock Rate (GHz)	1,25	1,2	1,296
Nb of Processors	8	16	30
Nb of Cores	64	128	240

Figure 9: Hardware parameters of the three graphics cards used for the comparison.

available cores is not recommended. We have to take into account the number of objects.

A coupling of CPU and GPU results is also possible, but it is difficult to present it graphically. However, this coupling is performed during off-line simulations to determine fields of optimal performance of every algorithms. Results show that CPU and GPU results are different in terms on time dependence to the number of objects.

6 Hardware Parameters Influence

Off-line simulations allow to determine fields of optimal performance of each broad phase algorithms. As previously shown, performance curves are dependent on the run-time architecture, we now study this dependency and how hardware parameters can influence algorithms performances. We analyze in this section the influence of the variation of several hardware parameters on the algorithms performances. The goal of the analysis is to know if it is possible to find a link between positions of intersection points and hardware parameters. In other words, it is possible to predict position of those points without performing an off-line simulation just by knowing values of hardware parameters.

We worked on three different kind of Nvidia[2] graphics cards (Quadro FX 3600M, Quadro FX 4600, Quadro FX 5800) (cf. Figure 9). We measured time spent by the broad phase algorithms in relation to the number of objects running on each kind of GPU.

The GPU bandwidth is a very important hardware parameter to study and to take into account. We study three different bandwidths values: from CPU to GPU, from GPU to CPU and the internal GPU bandwidth. Tests were performed with the same size of transfered data (33554432 Bytes = 32 MB). A low bandwidth can highly disturb algorithms performances but we want to determine how the bandwidth influences performances. Figure 9 presents parameters values.

Figure 10 presents a comparison of the hardware parameters of the Quadro FX 5800 and 4600 compared to the 3600M one. The y axis shows how much higher is the parameter compared to the other 3600M one. We can notice that the GPU clock rate and the bandwidth between CPU and GPU do not influence computation time of the broad phase. These two parameters are almost the same between the three GPUs but computation time of the broad phase is still better from one to the other and is decreasing. The bandwidth between GPU and CPU is also similar between the two platform. Algorithm time also seems not to be linked to this parameter. Memory size seems not to highly impact computation time because its value is 1.5 higher on the 4600 and more than 4 times higher on the 5800, and speed-up on the algorithm is not as high as these values. It is difficult to link other parameters with the algorithm time.

To precise the analysis, we proceed to a comparison between the 5800 and the 4600, illustrated on Figure 11. It confirms the fact that the clock rate, the bandwidth CPU-GPU and the GPU-CPU one do not influence algorithm time. The broad phase time is

[2]www.nvidia.com

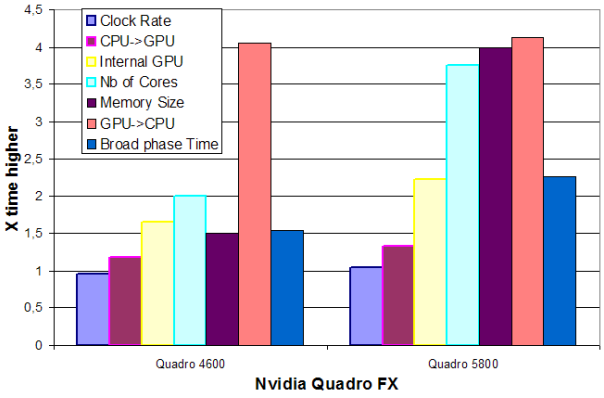

Figure 10: Comparison of the Quadro FX 5800 and 4600 parameters in relation to the 3600M ones.

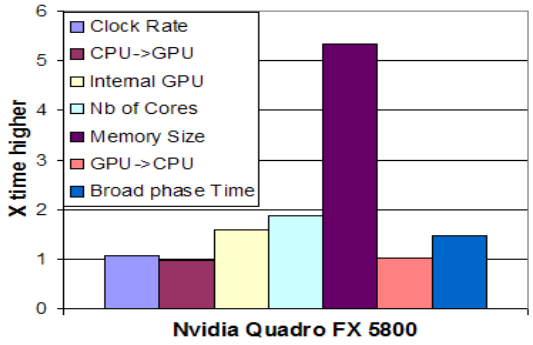

Figure 11: Comparison of the Quadro FX 5800 parameters in relation to the 4600 ones.

1.5X much faster than on the 4600 graphics card and these previous parameters did not change. The memory size of the GPU still seems not to significantly influence the algorithm time. It also seems that internal GPU bandwidth and the number of cores potentially impact the broad phase time.

7 Conclusion

We have presented a first way to dynamically adapt broad phase collision detection algorithms during simulations with unpredictable scenarios. We have developed broad phase algorithm running on several types of computer architectures (sequential CPU, multi-cores, single GPU and multi-GPUs). In our approach, we propose to determine fields of optimal performance for each broad phase algorithm. We evaluate performances of algorithms on different architectures in relation to the number of objects in the environment. From performance curve of an algorithm we extract intersection points with other curves to delineate the area where the algorithm is optimal. Obtained results are then used during in-line simulation to switch, in real time, the broad phase algorithm by a more efficient one. The switching phase is provided by a module in the application in charge of monitoring parameters of the simulation.

Our solution is generic because any algorithm can be included in the off-line loop and compared to the others. Results show that this dynamic adaptation of broad phase algorithms enables to use the most efficient algorithm all along the simulation. We knew that multi-GPU collision detection solution was totally unadapted to small physical simulations like sequential CPU solution for large scale simulations, but we are now able to know when one becomes

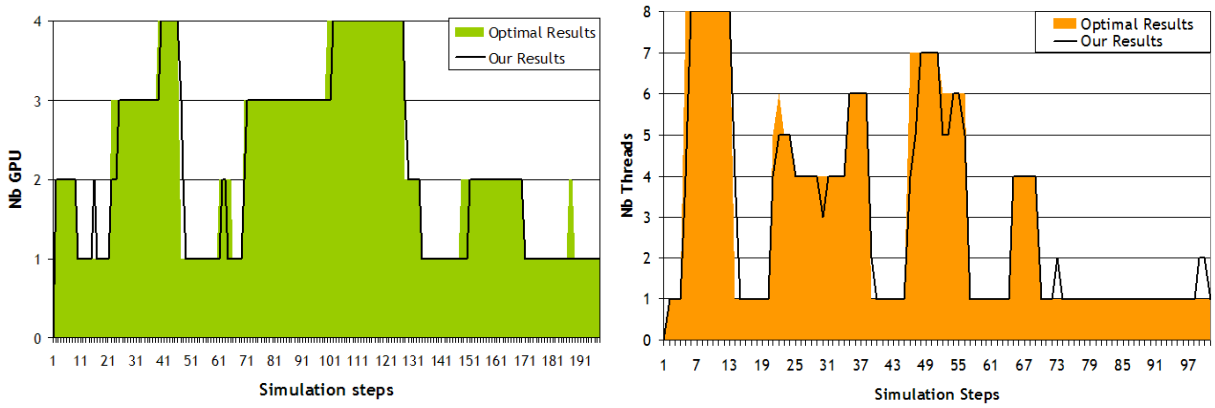

Figure 8: This figure shows two results of our dynamic algorithmic adaptation. The **left** graphic shows a multi-GPU result and the **right** one, a multi-core result. The black line presents choices made by our model to add or remove a GPU or CPU core from the broad phase loop during the simulation. The colored areas presents the optimal results obtained by reproducing the same simulation as the one made by the user but forcing the use of a fixed number of processing unit (CPU or GPU). Tests were performed on 4*Quadro FX 5800 with Intel Xeon(x8) 3.2GHz.

better or worse than the other.

We also presented a study on the dependency between algorithmic performances and hardware architecture. We showed that few parameters (clock rate, CPU-GPU bandwidth and GPU-CPU bandwidth) do not influence the algorithm time. We also showed that other parameters (internal GPU bandwidth and the number of cores) seem to be closely linked to the broad phase time. Finally, GPU memory size appears to be non really impactive to the algorithm time.

There are many ways of optimization in the dynamic adaptation of collision detection algorithms. Broad phase algorithms can still be improved in terms of computation time and efficiency. Data and task repartition is also a good way of improvement for these algorithms that run on multi-core or multi-GPU architecture. Another important and interesting way of improvement is the dynamic algorithms adaptation during the narrow phase process. This step is more complex because there are many parameters to take into account during this phase (object complexity, hierarchies, properties...). Many analysis and implementations have to be done to reveal the essential focus of adapting algorithms on the run-time architecture.

Acknowledgements

The authors want to thank Florian Nouviale (INRIA Rennes) and Colin Moore (Duke University, NC - USA) for their help in the review process of english. This research is supported by the INSA of Rennes (France) with the Bretagne region under project GriRV N°4295.

References

[1] Q. Avril, V. Gouranton, and B. Arnaldi. A broad phase collision detection algorithm adapted to multi-cores architectures. In S. R. . A. Shirai, editor, *VRIC'10 Proceedings*, pages 95–100, April 2010.

[2] Q. Avril, V. Gouranton, and B. Arnaldi. Synchronization-free parallel collision detection pipeline. In *ICAT'2010*, December 2010.

[3] J. L. Bentley and J. H. Friedman. Data structures for range searching. *ACMCS*, 11(4):397–409, 1979.

[4] G. V. D. Bergen. Efficient collision detection of complex deformable models using aabb trees. *J. Graph. Tools*, 2(4):1–13, 1997.

[5] J. D. Cohen, M. C. Lin, D. Manocha, and M. K. Ponamgi. I-collide: An interactive and exact collision detection system for large-scale environments. In *SI3D*, pages 189–196, 218, 1995.

[6] S. Gottschalk, M. Lin, and D. Manocha. Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings of the ACM Conference on Computer Graphics*, pages 171–180, New York, Aug. 4–9 1996. ACM.

[7] N. K. Govindaraju, M. C. Lin, and D. Manocha. Fast and reliable collision detection using graphics processors. In *COMPGEOM: Annual ACM Symposium on Computational Geometry*, 2005.

[8] N. K. Govindaraju, S. Redon, M. C. Lin, and D. Manocha. Cullide: Interactive collision detection between complex models in large environments using graphics hardware. In M. Doggett, W. Heidrich, W. Mark, and A. Schilling, editors, *SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 025–032, San Diego, California, 2003. Eurographics Association.

[9] E. Hermann, B. Raffin, and F. Faure. Interactive physical simulation on multicore architectures. In *Eurographics Workshop on Parallel and Graphics and Visualization, EGPGV'09, March, 2009*, Munich, Allemagne, 2009.

[10] P. M. Hubbard. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230, Sept. 1995. ISSN 1077-2626.

[11] P. Jiménez, F. Thomas, and C. Torras. 3d collision detection: a survey. *Computers & Graphics*, 25(2):269–285, 2001.

[12] D. Kim, J.-P. Heo, and S. eui Yoon. Pccd: Parallel continuous collision detection. Technical report, Dept. of CS, KAIST, 2008.

[13] D. Kim, J.-P. Heo, J. Huh, J. Kim, and S.-E. Yoon. HPCCD: Hybrid parallel continuous collision detection using CPUs and GPUs. *Comput. Graph. Forum*, 28(7):1791–1800, 2009.

[14] Y. Kitamura and A. Smith. Parallel algorithms for real-time colliding face detection. *Robot and Human Communication*, pages 211–218, Nov. 07 1995.

[15] D. Knott and D. K. Pai. Cinder: Collision and interference detection in real-time using graphics hardware. In *Graphics Interface*, pages 73–80, 2003.

[16] M. Lewis and B. L. Massingill. Multithreaded collision detection in java. In H. R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications & Conference on Real-Time Computing Systems and Applications (PDPTA'06)*, volume 1, pages 583–592, Las Vegas, Nevada, USA, June 2006. CSREA Press.

[17] M. C. Lin and S. Gottschalk. Collision detection between geometric models: a survey. In R. Cripps, editor, *Proceedings of the 8th IMA Conference on the Mathematics of Surfaces (IMA-98)*, volume VIII of *Mathematics of Surfaces*, pages 37–56, Winchester, UK, Sept. 1998. Information Geometers.

[18] B. F. Naylor. Interactive solid geometry via partitioning trees. In *Graphics Interface '92*, pages 11–18, May 1992.

[19] S. Pabst, A. Koch, and W. Straβer. Fast and scalable cpu/gpu collision detection for rigid and deformable surfaces. In *Computer Graphics*

Forum, volume 29, pages 1605–16212, July 2010.

[20] A. Selle, J. Su, G. Irving, and R. Fedkiw. Robust high-resolution cloth using parallelism, history-based collisions, and accurate friction. *IEEE Trans. Vis. Comput. Graph*, 15(2):339–350, 2009.

[21] A. Smith, Y. Kitamura, H. Takemura, and F. Kishino. A simple and efficient method for accurate collision detection among deformable polyhedral objects in arbitrary motion. *Proc. IEEE Virtual Reality Annual International Symposium*, pages 136–145, March 1995.

[22] M. Tang, D. Manocha, and R. Tong. Multi-core collision detection between deformable models. In *Computers & Graphics*, 2008.

[23] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Straßer, and P. Volino. Collision detection for deformable objects. *Comput. Graph. Forum*, 24(1):61–81, 2005.

[24] B. Thomaszewski, S. Pabst, and W. Blochinger. Parallel techniques for physically based simulation on multi-core processor architectures. *Computers & Graphics*, 32(1):25–40, 2008.

[25] D. J. Tracy, S. R. Buss, and B. M. Woods. Efficient large-scale sweep and prune methods with AABB insertion and removal. In *VR*, pages 191–198. IEEE, 2009.

[26] M. Woulfe, J. Dingliana, and M. Manzke. Hardware accelerated broad phase collision detection for realtime simulations. *4th Workshop in Virtual Reality Interactions and Physical Simulation (VRIPHYS) (2007)*, pages 79–88, 9 Nov. 2007.