Scalable Parallel Simulator for Vehicular Collision Detection

Ilan Grinberg and Yair Wiseman
Lucidlogix Technologies Ltd. Holon Institute of technology
Israel
wiseman@cs.huji.ac.il

***Abstract*— Several simulations for parallel vehicular collision detection have been suggested during the last years. Such a simulator saves the need to physically cause the collision. The algorithms usually greatly depend on the parallel infrastructure and this dependency causes in many times non-scalability performance. The dependency also harms the portability of the simulation. This paper suggests a scalable and portable parallel algorithm for a vehicular collision detection simulation that fits both clusters and MPI machines. This paper explains how this Simulator was designed and implemented in a large transportation company.**

I. INTRODUCTION

THERE are many vehicular simulation software tools in the civilian and military markets for many purposes [1], for example Crash Detection Simulation [2], Vehicle Survival Performance [3], Sensor Calibration Optimization, Safety and Simulation of Car accidents. All of these tools are based on three-dimensional shapes that made up of elementary polygons. Such simulation systems are designed to illustrate the real world; hence, they require high accuracy. High accuracy is obtained by using ten thousands to millions of polygons [4]. Handling so many polygons obviously requires enormous computation resources.

To accommodate the many requirements of the computational geometry functions (e.g. Polygon-Polygon intersections in a space where each polygon has its own velocity and acceleration) there will be a need for:

- Clever algorithms that can reduce the complexity of the function.
- Utilization of as many as possible of processors i.e. parallel or distributed computation.

In this paper, we present a parallel algorithm for Collision Detection Simulation. The suggested algorithm is based on the Locality Principle and the Load Balance Principle. The algorithm is suitable for both complex and simple geometry models with no dependency on the parallel environment and the architecture of the machines [5].

II. SCALABLE COLLISION DETECTION

The new parallel simulation that is suggested in this paper includes several advantages over the known parallel collision detection simulations [6,7,8,9]. The main idea of the suggested simulation is keeping the scalability principle while not abandoning the locality principle and the load balancing of the system [10].

We can use one of the known algorithms for Bounding Volumes Hierarchies to check the intersection or a collision of two models [11]. Let us define the smallest "work unit" as one operation (like a collision detection) on a complex geometry model or one operation between two complex geometry models. Indeed, a finer split into smaller unit could have been done like the author of [6] suggest; however, the cost of execution of one "work unit" that we suggest will be still very small, even if the geometry model is very complex. Experiments show that a splitting of geometry models into too many smallest units can produce too much overhead.

Let us define "processing unit" as one process that gets some parts of the collision detection procedure and returns the results to the master process. Any process in the parallel system can migrate from one processor to another processor in the same machine or migrate from one node (machine) to another node in the same computer cluster, if this is the policy of the parallel infrastructure.

The algorithm uses the Vector Space technique [12] to find similarity of scenarios ("work units") and machines ("processing units") in a similar way of queries in document sets in the Information Retrieval field.

Let us assume that we have two geometry models consist of basic polygons, n different scenarios where the models are placed in various places and various orientations and there is a collision in each scenario.

The work is defined as finding $\sum_1^n k_i$ intersection points of two objects in the n different scenarios where ki is the number of the intersection points in scenario i. In such a case, the finding of one single collision will be denoted as one "work unit".

TABLE 1
Example of a "work unit" list

scenario	*BB1*	*BB2*	*BB3*	*BB4*	*BB5*	*BB6*
1	0	0	1	0	1	4
2	0	11	1	1	1	0
3	1	11	1	1	0	4
4	0	0	1	0	0	0
5	0	11	1	1	1	0
6	…	…	…	…	…	…

scenario	*BB1*	*BB2*	*BB3*	BB4
1	2	0	14	6
2	2	1	14	6
3	0	0	14	6
4	0	1	0	0
5	0	1	0	6
6	…	…	…	…

A. The Simulation Algorithm

Let us denote np as the maximal processors in our ma-

chine.

- Create np children that will be the "processing units".
- Create a queue of "processing units" in an arbitrary order.
- Construct the Bounding Volumes hierarchy of the two geometry models by one of the known models that have been cited above. The data structure can be saved along with the geometry information so there will be no need to reconstruct the hierarchy many times. The Bounding Volumes hierarchy trees represent the geometry models and any leaf in any tree contains one basic polygon. The indices are put in nodes of no more than level d in each tree from left to right as can be seen in Fig. 1.
- Create a list of scenarios containing for each scenario, the scenario index and the Bounding Volume vector. i. e. for each scenario ("work unit"), the Bounding Volumes (the black nodes in Fig. 1) that are a part of the current check will be put in the list.
- Let us denote the Bounding Volume vector as BB. The value of a BBi that is not intersected in the given scenario will be 0. The value of a BBj that is intersected in the given scenario will be the number of the primitive polygons that the Bounding Volume bounds. An example for such a data structure can be seen in Table 1 -- The first table is the Bounding Volume vectors that are intersected in given scenarios for the geometry model that is depicted in Fig. 1 The second table depicts Bounding Volume vectors for the same scenarios but for the geometry colliding with the first model.
- Create a list of "processing units". Each "processing unit" will contain a vector in the same length as the vectors in the "work unit" list. In the beginning, these vectors are zeroed.
- Allocate q scenarios for each "processing unit" in this way:
- For each "work unit" in the processing queue, the q free scenarios that are most similar to the "work unit" vector will be selected. The most similar scenarios can be chosen by the well-known VS tactic [12] i.e. a scalar multiplication of the scenario vector and the "processing unit" vector.
- These q scenarios will be allocated for these q "processing units" and will be removed from the "work unit" list.
- The "processing unit" vector will be update by a switch of 0 to 1 for any Bounding Volume that was added by the new q scenarios.
- Any "processing unit" finds the collision points of the two geometry models for any scenario that was allocated for this specific "processing unit". If the "processing unit" needs more information on the primitive polygons and it does not have the information, the "processing unit" will call the parent process and will get this information from it. The "processing unit" will cache this information for a possible future use.
- When a "processing unit" finishes its jobs, the "processing unit" will call the parent process and will return the results about the intersections that were found in each scenario. The parent process will add the "processing unit" to the free "processing unit" queue.
- This procedure will be repeated until the "work unit" list is empty.

Fig. 1 depicts an example of a node indexing getting to level 4 in the tree. Each of the filled squares leaves represent one single polygon. The white nodes represent internal nodes whereas the black nodes are the nodes that will be indexed and will be put in the geometry vector. The number within the node indicate the number of the polygons within the volume that the node bounds. The number beside the node is the index of the node in the vector.

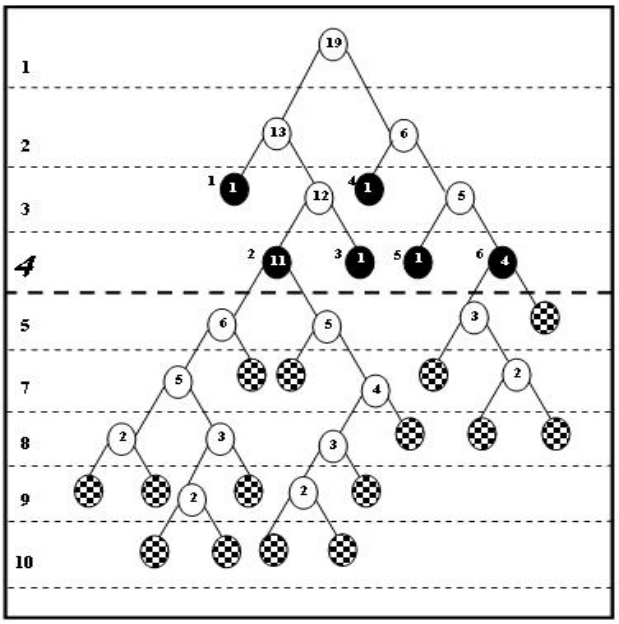


Fig. 1: Example of a node indexing

B. The Simulation Analysis

It can be seen that the "work unit" splitting mechanism enables the simulation to keep the locality principle, because the splitting of a job that contains information on parts of the geometry model is similar to the information that the "processing unit" has worked on it. In this way, the overhead of transferring the geometry models to all the machines in the cluster is prevented. If the geometry models are very complex (This is very common in many simulation tools) and the communication line speed is the common 1Gb/sec, the execution time can be improved significantly.

The simulation keeps the load balancing in the "processing units" by managing a dynamic queue and allocation of a small work portion in each iteration for each "processing unit". In this way, an optimal computation time will be obtained even if the simulation is executed on a complex parallel infrastructure.

The simulation allocates the needed memory for each work portion in each "processing unit" and in that way the simulation saves unnecessary memory allocations in other machines in the cluster [13].

The simulation is generic and it can be independently implemented on any Operating System, Middleware, Hardware or Framework. The simulation is fully portable and can be used in any environment.

There is no harm for the simulation performance in any geometry models. The simulation can handle flawlessly geometry of different sizes or shapes.

III. Implementation

Our collision detection algorithm was implemented based on a wide background including several fields, like: computer graphics, computational geometry, object oriented programming and distributed programming. Our implementation can be divided into two categories: a Serial infrastructure and a Parallel infrastructure [14]. The serial infrastructure includes the following elements:

- Basic mathematical and geometrical operations of matrices and vectors.
- Primitives intersection algorithms, like point, line, segment, plane and triangle.
- 3D interactive viewer implemented in openGL.
- Loading geometry information from a file.
- Generating an OBB tree.
- Intersections between two OBB trees.

The parallel infrastructure includes the following elements:

- Data structures for parallel OBB Trees.
- A particular client-server model.
- A "Matchmaker" object.
- A parallel algorithm for an intersection between two OBB trees.

A. OBB Tree Generation

Given a triangular mesh consisting of a vertices collection and a connectivity list, the basic methodology to constructing an OBB tree is recursive.

The generation process can be divided into three major steps:

1. Generating a bounding box for the set of remained triangles.
2. Splitting the set of triangles into two submeshs.
3. Running the recursive process on the two new split submeshs.

The motivation of splitting the triangles into two submeshs is creating bounding boxes with a minimal volume for the submeshes. This split typically produces a better regional distinction that helps the parallel collision detection algorithm to build distinctive clients and fewer nodes will be needed in each scenario of collision detections.

B. Data Structures for Parallel OBB Trees

Our suggested algorithm has an unusual implementation of bounding box trees. Not all the nodes are known by the clients. This is the reason why we need a different implementation of OBB tree data structure. We call this data structure, Parallel OBB Tree.

The parallel OBB tree inherits its properties from a standard OBB tree object. Additional members and methods provide the parallel OBB tree parallel attributes. The client holds a list of pointers to the nodes, which contain the bounding volume vector of the parallel OBB tree. The client initializes the pointers in the list to null at the beginning of the analysis. Each time the client needs to analyze a collision detection scenario, the client will set all its relevant pointers from the list to the new distributed root subtrees, which play a part in the scenario. These subtrees are received from the master process. A detailed explanation of the implementation can be found at our project website:

http://u.cs.biu.ac.il/~wiseman/Collision.htm

IV. Experimental Results

The comparisons tests that are shown in this section focus on the differences between the "matchmaker" algorithm suggested in this paper to other algorithms of client selection.

All the experimental results were conducted on the same setting consists of an heterogenic computer cluster with 5 computers of 2 dual cpus, Intel Xeon 2.8Ghz Dual Core and 3GB RAM, 4 computers of 2 dual cpu, Xeon 2.4Ghz Single Core, and 2GB RAM, altogether 28 cores in the cluster. The cluster was interconnected with 1Gbps LAN network. Linux Kernel 2.6 with Mosix 2.0 was installed on all the computers. We selected two different geometries for our tests. The first one represents a heavy vehicle model that is represented by approximately 300,000 triangles and its serialized Parallel OBB tree consumes approximately 62MB. The second geometry represents a small car model with approximately 90,000 triangles and its serialized Parallel OBB tree consumes approximately 19MB.

The number of collisions scenarios chosen for each test depended on the geometry size. With the aim of emphasize the matchmaker's contribution to the simulation, it is important to limit the number of scenarios. At the first step of the computation, when the computer cluster spends a lot of time on acquiring the geometry data but it is required to quickly analyze scenarios, the matchmaker is mostly needed. The number limit of the scenarios was chosen in a trial and error way - for each test we seek out the limit that caused a performance drop comparing to the other algorithms.

The matchmaker algorithm described in this paper is based on finding out the client with the most similar geometry parts to the given collision detection scenario. With the purpose of testing this algorithm, we need to compare different matching strategies between a client and a scenario. We compared the following algorithms:

- **Best Match** - The suggested algorithm of this paper
- **Random Match** - For each scenario, a random unclaimed client will be picked. This algorithm represents a strategy that actually does not find a connection between clients and scenarios.
- **Lowest Match** - For each scenario, the unclaimed client with the geometry that less resembles the scenario is selected.
- **Best Match-Load** - One can argue that if a client is loaded with many scenarios, this client should not be preferred to analyze the next scenario over a less loaded client even if the less loaded client's geometry is less similar. A client load is calculated by dividing the buf-

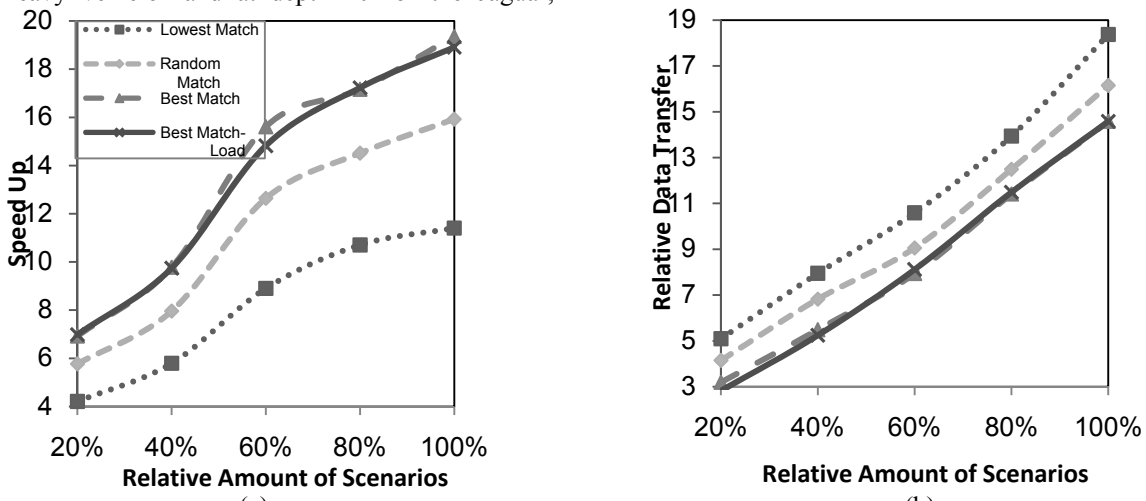
fered scenarios in the client by its buffer maximal size. We took into consideration both the load and the geometry similarity to the client.

In the tests performed in the following sections, the following parameters have been used:

- Collisions between "heavy vehicle" to Jaguar scenarios and collisions between two Jaguars scenarios.
- Scenarios buffer size (q) has been set to 20, which is approximately the optimal value.
- The fixed geometry has been distributed at depth 12 for the "heavy vehicle" and at depth 10 for the Jaguar, which is approximately the optimal value.

A. Amount of Scenarios Influence Analysis

The main data transmission overhead occurs at the early stages of the work, when all the clients have small portions of the geometries parts. At this stage, each new starting job consumes more time due to many geometry parts transmission. Testing the influence of the number of scenarios from the early stage to the saturation stage, when there is a little data transmission overhead for each job submission, is significant.



(a) (b)

Fig. 2. Amount of scenarios influence analysis on (a) speedup and (b) relative data transfer of "heavy vehicle" with Jaguar collisions

In Fig. 2 we tested this influence from two different aspects, speedup and relative data transfer. The speedup is calculated by dividing the weighted serial time by the parallel time measured for each test. The relative data transfer is calculated by dividing the total amount of data transmitted to the remote clients by the entire serial size of the two test geometries, as is mentioned at the beginning of the experimental results section.

It can be seen in Fig. 2 that Best Match and Best Match-Load algorithms give the best speedup. When the number of scenarios gets bigger, the ratio between the performances of these two algorithms and the other two algorithms will get bigger. At the final stage (100% of the scenarios), the speedup gets to 20 out of 25 comparing to Lowest Match, which gets only to speedup of 12. As long as there are more scenarios the clients will collect more geometries portions. Best Match algorithms utilize this way of collecting to reduce the data transmission overhead. The Best Match algorithms reduce transferred data, which causes less memory allocations at the clients, comparing to the two other algorithms. It can be seen that Best Match-Load algorithm does not give any significant performance improvement comparing to the standard Best Match algorithm.

B. Distribution Depth Analysis

We have examined the influence of various depths of the OBB tree on the matchmaker algorithms. The results are shown in Fig. 3. It can be noticed that Best Match algorithms give better performance, both in speedup and relative data transfer.

Fig. 3 also shows that each geometry model has an optimal distribution depth, particularly, for "heavy vehicle", the optimal distribution depth is 12 and for Jaguar, the optimal distribution depth is 10.

It can be concluded from Fig. 3 that if the given geometry model is bigger, the relative performance of the Best Match algorithm will be better.

We can see in Fig. 3 trimmed lines in the low distribution depth of Lowest Match and Random Match algorithms. This missing data has been ensued as a result of a crash at the transmitting machine due to a lack of memory. At the initial stage, when the first jobs are transmitted, all the remote clients are still located as a process at the local (transmitting) machine waiting for being migrated to other machine in the cluster. When using a memory wasteful algorithm like Lowest Match or Random Match, the machine can quickly run out of memory and crash [15].

We also checked the influence of several scenarios buffer sizes on the matchmaker algorithms. We concluded that a buffer size 20MB gives the best performance in most of the algorithms. Increasing the buffer size can cause a performance drop because the clients are occupied for too long

time and consequently they lose scenarios that other less suitable clients will take.

Another reason that can cause the performance drop is the increasing transmission time that the main process takes to transmit its first tasks to the clients. The initial time of the process is crucial for decent performance because at this time period, the clients are idle and parallelism still does not take place.

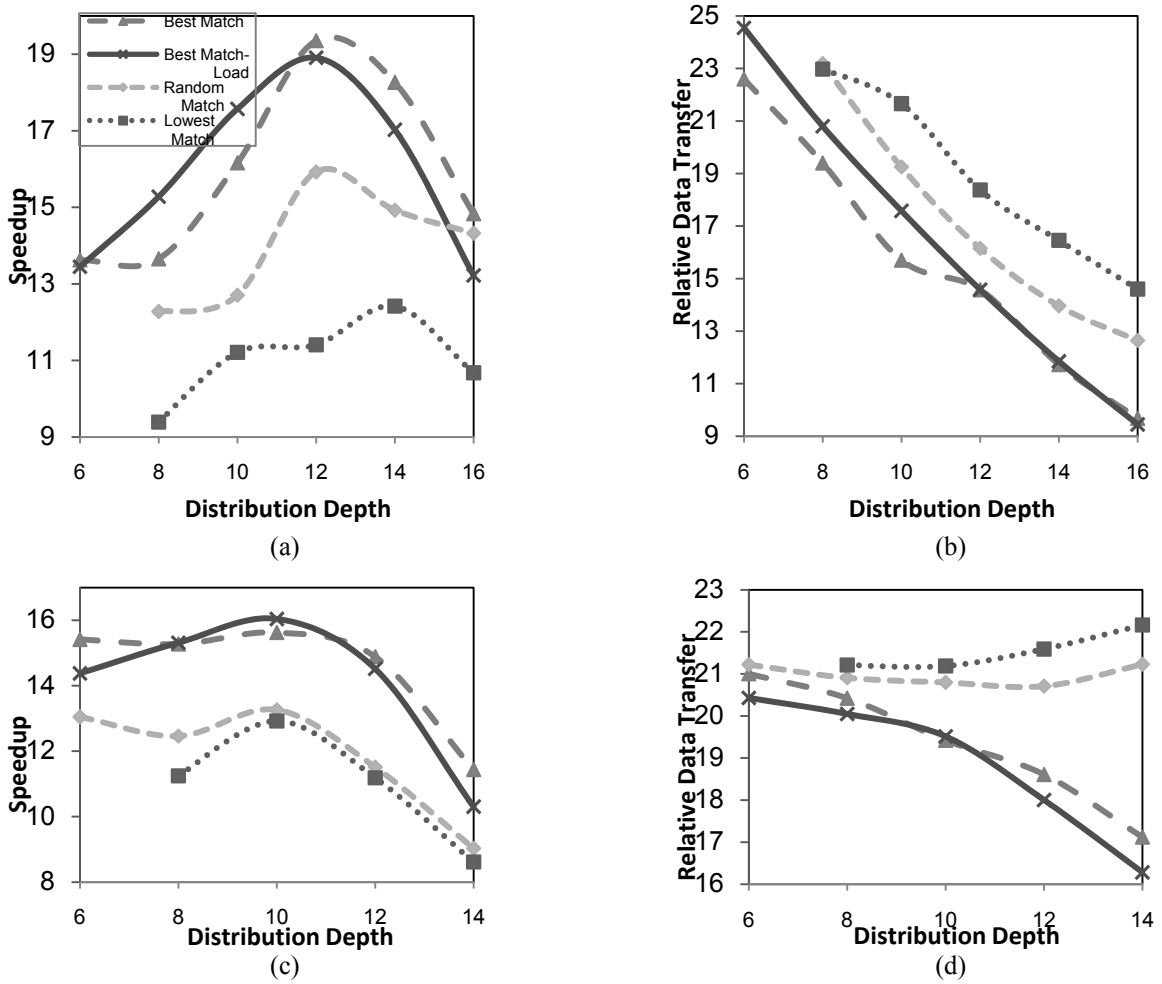


Fig. 3. Distribution depth analysis of (a) speedup and (b) relative data transfer of **"heavy vehicle" with Jaguar** collisions and of (c) speedup and (d) relative data transfer of **two Jaguars** collisions.

C. *Scalability Analysis*

The cluster that was used for the tests contains two types of CPUs; one is almost two times faster than the other. We biased the number of processing units (can be referred as max speedup of the cluster) in respect to the faster CPU. The relative speeds of the two different CPUs were calculated as a ratio of them and the serial results. Table 2 shows the cluster weights calculated for several cluster configurations.

Fig. 4 shows the influence of several cluster configurations on the matchmaker algorithms. It can be seen that Best Match algorithms are scalable to the number of processing units in the cluster. Lowest Match and Random Match algorithms do not scale up well. We can also see in Fig. 4a that when the tested geometries are big, it will be unworthy to use more than 20 weighted cores for the scenarios analysis for those two algorithms.

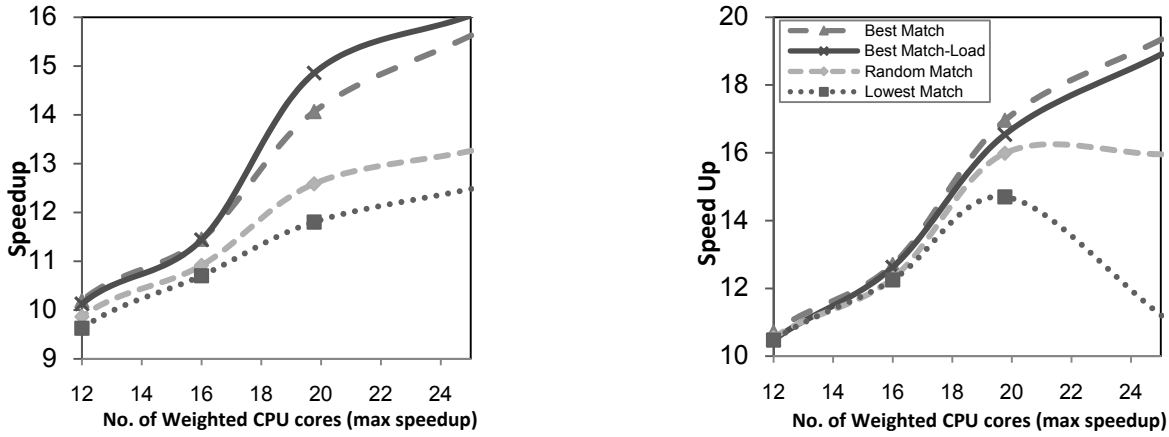
When using less than 16 weighted cores, the performance is equal for all algorithms. This comes about because the geometry transmission to the clients comes to an end very quickly because of the small number of clients.

TABLE 2
Tested Cluster Configurations

No. of CPU Cores		
Fast Computer	Slow Computer	**Weighted Cores**
12	0	**12.0**
16	0	**16.0**
16	6	**19.8**
20	8	**25.0**

V. Conclusions And Future Work

Simulators for collision detection are an intelligent transportation system that helps us to know more about vulnerable points of a vehicle [16] and can help us to make the vehicle safer.



(a) (b)

Fig. 4. Processing units' scalability analysis on (a) speedup of **"heavy vehicle" with Jaguar** collisions and (b) speedup of **two Jaguars** collisions

Given complex geometry models, the simulation can detect an intersection in an efficient execution time. The suggested simulation cuts down the initial overhead of testing a parallel collision between complicated vehicle geometries on a computer cluster. This overhead is cut down by minimizing the dependency of data transfer growth and the number of processing units in the cluster. This is the reason why the suggested simulation scales up well in respect to the cluster size and the geometries size, whereas standard algorithms fail to scale up well. Reducing the amount of clients' memory allocation is another benefit of the suggested simulation. This reducing gives the simulation the flexibility to be ported to any given parallel infrastructure.

In the future we would like to add an ability of transferring geometry data between clients with the intention of reducing the I/O load of the transmitting machine. When a client will need a geometry portion that another client has, the clients-manager will be able to transmit the missing portion to this client.

Another interesting addition we would like to integrate into this simulation is our work on the subject of compressing the transferred data on top of the communication channel [17]. The compression of the different parts of the OBB tree can be done in the preprocessing phase [18,19] and can reduce the overhead of the data transmission.

VI. References

[1] P. Jiménez, F. Thomas, and C. Torras, "3d Collision Detection: A Survey", *Computers and Graphics*, Vol. 25(2), pp. 269-285, 2001.

[2] S. Brown, S. Attaway, S. Plimpton, and B. Hendrickson, "Parallel Strategies for Crash and Impact Simulations" *Computer Methods in Applied Mechanics and Engineering*, Vol. 184, pp. 375-390, 2000.

[3] L. Thomas Wasmund. "New Model to Evaluate Weapon Effects and Platform Vulnerability: AJEM. WSTIAC. 2001, Vol. 2, 4, pp. 1-3.

[4] B. Curless and M. Levoy. "A Volumetric Method for Building Complex Models from Range Images", In *Proceedings of ACM Siggraph '96*, pp. 303-312, 1996.

[5] R. B. Yehezkael, Y. Wiseman, H. G. Mendelbaum and I. L. Gordin, Experiments in Separating Computational Algorithm from Program Distribution and Communication, LNCS of Springer Verlag Vol. 1947, pp. 268-278, 2001.

[6] M. Figueiredo and T. Fernando. "An Efficient Parallel Collision Detection Algorithm for Virtual Prototype Environments". ICPADS'04. 2004.

[7] O. Lawlor and L. Kale. "A Voxel-Based Parallel Collision Detection Algorithm". Proceedings of the 16th international conference on Supercomputing. 2002.

[8] U. Assarsson and P. Stenstr. "A Case Study of Load Distribution in Parallel View Frustum Culling and Collision Detection". Lecture Notes in Computer Science Vol. 2150, pp. 663 - end, 2001.

[9] O. Lawlor. "A Grid-Based Parallel Collision Detection Algorithm", Master's thesis, University of Illinois at Urbana-Champaign, March 2001.

[10] I. Grinberg and Y. Wiseman, Scalable Parallel Collision Detection Simulation, Proc. Signal and Image Processing (SIP-2007), Honolulu, Hawaii, pp. 380-385, 2007.

[11] G. van der Bergen . "Collision Detection in interactive 3D Environments" Spatial Data Structures, pp. 171-217, 2004.

[12] Salton, G., Wong, A., and Yang, C. S.. "A Vector Space Model for Automatic Indexing". Commun. ACM vol. 18(11), pp. 613-620, Nov. 1975.

[13] M. Geva and Y. Wiseman, Distributed Shared Memory Integration, Proc. IEEE Conference on Information Reuse and Integration (IEEE IRI-2007), Las Vegas, Nevada, pp. 146-151, 2007.

[14] Y. Wiseman, ASOSI: Asymmetric Operating System Infrastructure, Proc. 21st Conference on Parallel and Distributed Computing and Communication Systems, (PDCCS 2008), New Orleans, Louisiana, pp. 193-198, 2008.

[15] M. Reuven and Y. Wiseman, Medium-Term Scheduler as a Solution for the Thrashing Effect, The Computer Journal, Oxford University Press, Swindon, UK, Vol. 49(3), pp. 297-309, 2006.

[16] A. Avila, G. Korkmaz, Y. Liu, H. Teh, E. Ekici, F. Ozguner, U. Ozguner, K. Redmill, O. Takeshita and K. Tokuda, A complete simulator architecture for inter-vehicle communication based intersection warning systems, Proceedings of the Intelligent Transportation Systems Conference, Vienna, Austria, pp. 461-466, 2005.

[17] Y. Wiseman, K. Schwan and P. Widener, "Efficient End to End Data Exchange Using Configurable Compression", Proc. The 24th IEEE Conference on Distributed Computing Systems (ICDCS 2004), Tokyo, Japan, pp. 228-235, 2004.

[18] S. T. Klein and Y. Wiseman, Parallel Lempel Ziv Coding, Journal of Discrete Applied Mathematics, Vol. 146(2), pp. 180-191, 2005.

[19] S. T. Klein and Y. Wiseman, Parallel Huffman Decoding with Applications to JPEG Files, The Computer Journal, Oxford University Press, Swindon, UK, Vol. 46(5), pp. 487-497, 2003.