PAMM · Proc. Appl. Math. Mech. **5**, 657–658 (2005) / **DOI** 10.1002/pamm.200510305

Dynamical Particle Simulation with Parallel Cache-Aware Domain Decomposition Strategies

Florian Fleissner* and **Peter Eberhard**

Institute B of Mechanics, University of Stuttgart, Germany

The simulation of large particle systems with the Discrete Element Method can be very time consuming. This is due to the necessity for collision detection between the disordered particles. Various methods, originating from different areas such as computer science, are well established and have been used in various applications. For parallel computations the simulation domain needs to be divided into subdomains to be distributed among the different nodes or machines within a supercomputer or a computer-cluster. The strategy for this domain decomposition has a significant influence on the performance of the calculation. In this paper we discuss some aspects of the development of a hierarchical domain decomposition algorithm that provides flexible adaption of the decomposition pattern to the changing structure of the particle system during the simulation. Thus an even load distribution among the different machines can be maintained. Moreover, the same method is also used to deal with the computational bottleneck caused by the presence of unstructured data.

1 Particle Simulation

For the simulation of granular media with short range interactions the Discrete Element Method (DEM) is a widely used approach. This is in part due to its simplicity that involves particle collision detection as the core part of the method. However, for large systems the method is very memory and CPU extensive. Fluctuating neighborhood relations resulting in unstructured data are the main reason for the method's low CPU efficiency, compared to any method based on regular grids.

2 Parallel Particle Simulation

In a parallel particle simulation objects can be grouped and assigned to different processors. The different ways of object grouping all have in common that collisions between objects within the groups as well as between neighboring objects of different groups need to be detected. Subdividing the simulation domain into fixed subdomains is only advisable if the system is almost static with respect to the objects positions. Objects with large displacements can leave their initial subdomains and must then be reassigned to another processor. So if lots of objects change subdomains it will cause some processors to be idle while others become overloaded. To avoid such an imbalance, adaptive domain decomposition methods are employed that have been designed to maintain an optimal work balance and thus minimize processor idle time.

3 Adaptive Domain Decomposition

Two situations that interfere with performance need to be prevented: In the first one, under-worked processors, the tasks of two or more processors, can be combined and assigned to only one of them, thus setting the others free for other tasks. The second situation, overloaded processors, is more crucial since one such processor can stall the whole calculation. The work-load of critically overloaded processors has to be divided and also assigned to additional machines, e.g. those that had their own work load reassigned to another machine and are now idle. The decision when to divide or combine tasks can simply, but efficiently, be based on the number of objects that are assigned to a given processor: If it exceeds a certain threshold, the subdomain will be split up. If it falls below another threshold, the subdomain will be recombined with a subdomain corresponding to another under-worked processor.

4 Domain Decomposition Strategies

Using fixed tiles as an initial domain decomposition pattern is problematic due to the arising task of dividing and recombining subdomains which may have complicated geometries. It is better to use tree based decomposition patterns. In a binary tree structure, subdividing a parent sub-domain will yield two twin subdomains that can easily be recombined if necessary. The same holds for quadtrees and octrees where quadruples or groups of eight sibling subdomains will result from a subdivision.

* Corresponding author: e-mail: fleissner@mechb.uni-stuttgart.de, Phone: +49 711 685 6888 Fax: +49 711 685 6400

5 Cache-Aware Particle Simulation

The unstructured neighborhood relations of DEM simulations are the reason for a relatively bad CPU-utilization. Even though objects themselves may be neighboring, it is very unlikely for their data to be stored in close proximity in memory as well. As a result, data access during collision detection occurs almost randomly. Because of this the cache memory is the main bottleneck of the calculation. The ratio of cache size to the overall amount of simulation data is a good estimate for the probability for an object's data to be present in the cache memory when randomly accessed. For large simulations this probability is very small. The high number of clock cycles that elapse while the processor is waiting for data to be fetched from main memory instead of being available in cache memory, multiplied by the probability for a cache miss and the number of data requests is an estimate for the processors idle time caused by cache misses. With unstructured data and thus unstructured access patterns this value is very high. To conquer this problem, data is grouped into small chunks that fit into cache memory. Operations on these chunks are also grouped and performed on the data in cache before proceeding with the next chunk of data (temporal locality). If data is also stored in a contiguous memory layout, the increased likelihood for a datum to be present in cache after requesting its neighbor (spatial locality) [1] also yields a minor performance increase. This comes from the fact that data is always loaded as chunks (cache lines) into cache memory. Even though this is a completely different motivation for subgrouping, the same domain decomposition strategies as described above can be applied for the parallel simulation.

6 Collision Detection

Several well established approaches exist for the detection of collisions. Most of them attempt to reduce the worse case complexity $O(n^2)$ for the detection of collisions between n objects to $O(n)$, taking advantage of the fact that an object can only collide with others in its direct vicinity [2] . Two efficient approaches are the DESS algorithm [3] and the Incremental Sort and Update Algorithm [4]. The former is the method of choice if no history (such as informations about collisions from the preceding time step) can be stored due to memory restrictions. The latter detects only those collisions which are newly occurring or ending. Both algorithms perform a predetection of overlaps between the objects' bounding boxes. Therefore they use three sorted lists that contain the upper and lower coordinates of the objects' bounding boxes in the three spatial dimensions. If memory size is not a restriction, the advantages of both methods can be combined, thus creating an algorithm that stores the collisions from the last time step and takes advantage of the DESS' capability to very efficiently detect newly occurring collisions. The detection of collisions between objects of different subdomains is done by an algorithm that is based on the data structures of the DESS, detecting overlaps of bounding boxes in pairs of sorted lists. At the beginning of every time step it is necessary to check which subdomains contain objects that are just colliding with objects from other subdomains. The following hierarchical approach addresses this task.

7 Hierarchical Collision Detection

From the sorted lists with the objects' bounding boxes it is easy to determine the total extension of an object group. Thus groups can be replaced by their bounding box for a rough predetection step. This suggests the following hierarchical approach for the collision detection in a parallel simulation: First we determine the extensions of the objects assigned to the different processors in terms of bounding boxes. These bounding boxes are used in a first collision detection step to determine which processors need to interchange data. With this information we detect the collisions of objects that are assigned to different processors. From this point on the collision detection can be carried out independently for all processors. The processors perform a collision detection step with the bounding boxes of their subdomains to determine which object groups overlap. Those collisions are then detected as well. Finally we detect the remaining collisions of objects inside the subdomains.

8 Conclusion

Tree based domain decomposition is an efficient and flexible approach for particle subgrouping. It can be used to adaptively balance the work-load in parallel particle simulations as well as for cache-aware collision detection. Hierarchical collision detection employs the same detection algorithms on different levels. It minimizes the amount of unnecessary collision checks between objects that are far apart. The method is well suited for parallel cache-aware simulation of large particle systems.

References

[1] M. Kämpe, F. Dahlgren, Exploration of the spatial locality on emerging applications and the consequences for cache performance, Proceedings of the 14th International Parallel and Distributed Processing Symposium, pp. 163ff, 2000
[2] B. Muth, M. K. Müller, P. Eberhard, S. Luding, Contacts between many bodies. Machine Dynamics Problems, Vol. 28, Nr. 1, pp. 101-114, Warschau, 2004
[3] E. Perkins, J. R. Williams, A fast contact detection algorithm insensitive to object sizes, Eng. Comp., Vol. 18, No. 1/2, pp. 48-61, 2001
[4] A. Schinner, Fast algorithms for the simulation of polygonal particles, Granular Matter, Vol. 2, pp. 35-43, 1999