Using an Improved Data Structure in Hybrid Memory for Agent-Based Simulation

Silvana Gallo[1,4], Francisco Borges[2], Laura De Giusti[1], Marcelo Naiouf[1], and Remo Suppi[3]

[1] Instituto de Investigación en Informática III-LIDI, Facultad de Informática, Universidad Nacional de La Plata, La Plata, Buenos Aires, Argentina
[2] Federal Institute of Bahia, Santo Amaro, Bahia, Brazil
[3] Universitat Autònoma de Barcelona, Bellaterra, Barcelona, España
[4] CONICET, Facultad de Informática, UNLP, La Plata, Buenos Aires, Argentina
{sgallo,ldgiusti,mnaiouf}@lidi.info.unlp.edu.ar
franciscoborges@ifba.edu.br,remo.suppi@uab.cat

Abstract. Data structure is an important issue to get good performance in parallel and distributed applications. These data structures have to be designed with the memory paradigm in mind where the data structure will be used in order to explore the architecture in a better way and subsequently obtain the best Speedup. Current parallel programming languages enable us to easily transform a parallel solution developed for a distributed paradigm to a hybrid solution just by adding pragma codes. At first approach, this is an interesting solution because it does not require several code modifications. Nevertheless, this interchange can cause a slowdown if an appropriate and deep adaptation is not carried out in the code. In this paper, we present our experience when we migrated a data structure developed for a distributed paradigm to a hybrid paradigm. This data structure was implemented in our Fish Schooling Agent-Based simulator where it might be useful either as a distributed paradigm or a hybrid paradigm. The results show the importance of customizing the data structure for the appropriate infrastructure and parallel programming paradigm. We believe that the data structure should have a flexible and dynamic behavior in accordance with the paradigm used.

Keywords: Parallel and distributed simulation, High performance simulation, Hybrid MPI+OpenMP programming, Individual-oriented model

1 Introduction

Parallel and distributed architectures have been providing more power computing for the scientific community. They enable researchers to develop more complex and realistic models. However, the computing resources available can be underused if the software does not take advantage of these architectures. Data structures are an important issue that must be considered in order to

obtain maximum speedup in the parallel and distributed solutions. Data structures should fit the modeled problem on the parallel and distributed architecture and with the programming paradigm. Otherwise, the solution will not reach an acceptable level of performance.

In fact, the parallel and distributed architecture and the programming paradigm are strongly related. The concepts of the memory paradigm: Distributed Memory (DM) and Shared Memory (SM) are the key to understand how to explore the architecture in a better way. DM paradigm uses an addressing memory for each process. In this paradigm, each process has only access to the variables that are inside of its addressing memory. Hence, there are no parallel and concurrent writing in the same variables and data structures. On the other hand, the SM paradigm shares the same addressing memory space among threads, where each thread can concurrently access the same variables and data structures. This paradigm requires that lock mechanisms be applied in order to maintain data coherence. Generally, lock mechanism implies a sequential access to variable and data structures and can become a bottleneck in a parallel execution.

DM is a very appropriate solution for scalability [5] because it enables us to easily add other processing nodes in order to execute a solution. The communication among the processes occurs by message passing. Communication among the processes is the main cause of the inefficiency of DM solutions. The SM allows the solution to take advantage of the locality of the data using the cache hierarchy. In addition, SM decreases the communication volume in the network because the communication occurs by memory copy. Therefore, the Hybrid Memory (HM) paradigm emerges as an approach that can use the main advantages of both DM and SM paradigms. Accordingly, these were the main motivations in using the data structure proposed by [14] in an HM paradigm. The volume communication in MPI is too high; therefore, we would decrease the communication volume among the agents because they are in the same core and an MPI communication routine is not required. In addition, we can take advantage of data locality. The data structure proposed by [14] is a data structure used to partition the data over the parallel and distributed architecture. This data structure is implemented in our Fish Schooling Agent-Based simulator. Our group has been researching High Performance Parallel and Distributed Simulation over the last few years [2, 14]. That is why we were interested in analyzing the data structure behavior in the hybrid memory.

In this paper, we present our experience in data structure migration proposed by [14] to the HM paradigm. We report the solutions and problem in two phases: in the first one, we just add an OpenMP pragma without changes to the data structure [14]. In the second phase, we show the changes and adaptation of data structure and we present, with software point view, the resulting data structure [2]. We have used a Fish Schooling Agent-Based Model simulation [14] in order to test the proposed data structure. Thence, some questions emerge and are answered throughout the paper: could we use a data structure, initially developed for DM, in an HM paradigm? What changes in data structure would

be necessary in order to maintain or improve the previous performance? What are the challenges?

The remainder of the paper is organized as follows: the related works are discussed in Section 2. After that, the hybrid data structure is presented in Section 3. Then, we present the experimentations in Section 4; and, lastly, our conclusions and final considerations are shown in Section 5.

2 Related Work

Data structures drive the computational complexity, and memory is an important issue in HPC. The data structures used depend, directly on the behavior of the model. In our case, we have a fixed amount of nodes and the behavior of the agents inside the node is dynamic, so we use a dynamic list. In previous works, we can find both Distributed memory and Shared memory approaches in order to improve the performance of the solution. In the literature, we have not found an agreement on the use of Hybrid Memory paradigm for parallel and distributed applications. Basically, the works present different applications and results about this point. What we observe is that the gain of performance depends on a combination of several factors, for example, architecture [7, 9, 12], network interconnection [6], and application [4, 7]. Adhianto and Chapman [1] provide an additional list of references on this topic.

In addition, agent based models have been solved by software engineers used widely in software engineering. This is a common problem; therefore, several tools have been created in order to encapsulate the programming complexity. In the case of non-distributed computing, we can find tools such as Netlogo [15], Anylogic [3]. They use dynamic structures such as List of Lists in the case of grid based maps or hash in the case of agent definitions, but all the structures are used without distributing. Some parts of Netlogo [15] are programmed using Scala, which uses Actor programming. Repast HPC [8] is an HPC tool programmed in C and MPI library. Agents communicate their information sharing by using messages. It is programmed using distributed memory with a language description based in environmental agents, networks, coordinating and mobile agents, in a Logo-style. Pandora [13] is an open framework for discrete event simulation and ABM HPC tool that uses shared memory in an OpenMP level and distributed memory in an MPI level. Flame [11] is another framework for agent based modeling. It also has a GPU version that has the same modeling approach but a different computational model and architecture. In the case of the main, Flame uses MPI and it does not use OpenMP, but Pthreads are used for asynchronous communications.

3 Data structure

We organize this section in two parts. First, we review the list of cluster data structure [14], as this is the base for the data structure proposed in [2]. Then, we talk about our experience with using this data structure [14] at hybrid simulator

without appropriate adaptations. In the second part of this section, we present the list of clusters adapted for this solution.

3.1 List of Cluster Data structure

The data structure proposed by [14], see Figure 1, is a dynamic list composed by nodes. Each node has some information and link(s) to other node(s). Each node is called a cluster, therefore, the data structure is called a *list of clusters*.

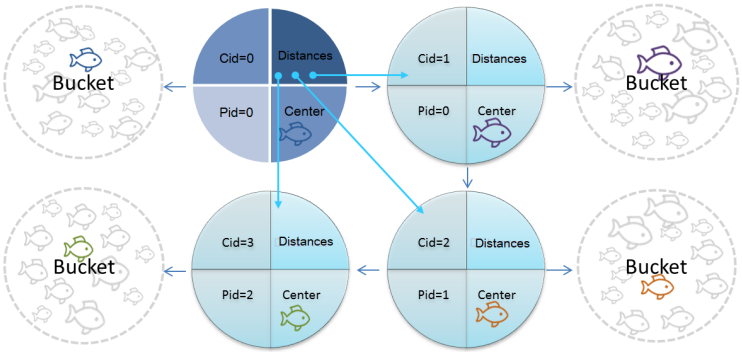
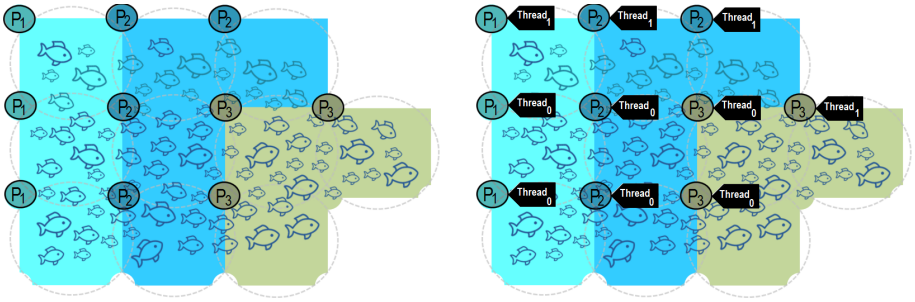


Fig. 1: The figure shows the data structure proposed by [14] where all communication is using MPI.

Each cluster consists of several data such as: a) Center field that represents a centroid, which is the most representative element of the cluster; b) the processor identifier (PID) indicates in which process each cluster is stored; c) a bucket in which agents belonging to the cluster are stored; d) the cluster identifier (CID), which indicates the cluster position in the list; and e) the distances to other clusters (see Figure 1). This data structure allows for defining areas in which agents can interact only with agents belonging to adjacent areas (see Figure 2a), reducing the computing involved in the neighbour selection process.

In the DM paradigm, the clusters are distributed between processors by MPI processes; therefore, each MPI process can have many allocated clusters (see Figure 2a). Some adjacent clusters are allocated in the same processors according to the PID field. Therefore, each MPI process has a copy of its list of clusters. During the simulation, the agent moves in the environment, thus its centroid can change and it begins to suffer from the influence of other centroids. Therefore, a migration process of agents among the MPI processes occurs. In this moment, the MPI processes send and receive these agents and update their local list of clusters.

The motivation to proof a hybrid implementation for a list of cluster data structure was that we had observed that many agents were in same processor; therefore, the OpenMP strategy in this situation would be interesting as a performance improvement. Thus we developed a hybrid version using the list of clusters



(a) Allocated for different MPI process in the same processor

(b) Allocated same MPI process + threads but the clusters are not manipulated concurrently

Fig. 2: Cluster adjacent areas.

and adding OpenMP pragmas without changes to the data structure. Figure 3 shows the execution time with list of cluster being used as a Distributed Memory paradigm (only MPI) and a Hybrid Memory paradigm (MPI and OpenMP).

As we can note, this Hybrid solution has a slowdown in comparison with the DM solution. The list of clusters works very well in the pure MPI version because each processor has a local copy of this data structure. Therefore, there is no concurrent manipulation of data structure in each MPI process. On the other hand, this hybrid version has many threads of the same processes inserting and updating concurrently the same copy of data structure through OpenMP threads, see Figure 2b.

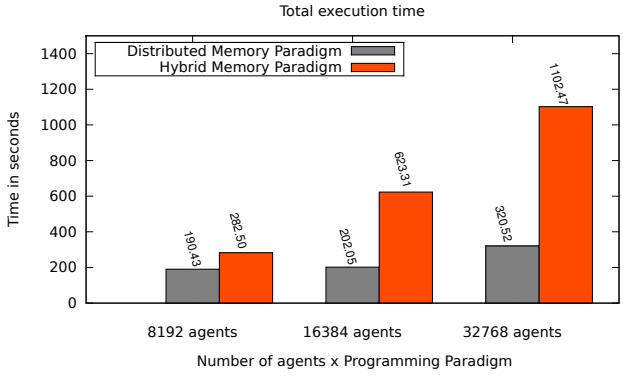


Fig. 3: Total execution time with Distributed Memory and Hybrid Memory paradigm

Therefore, the list of clusters has a performance problem when it is used in a hybrid solution, where many threads can write new information simultaneously in the cluster. This data structure becomes a bottleneck because a sequential access at data structure is required in order to maintain the data coherence. To avoid inconsistency in the simulation results, it is necessary to use OpenMP critical regions. In our DM simulator, the list of cluster is represented by a vector of list of cluster. Each MPI process, manipulates just the agents that are in it. The insert method includes a new agent in the bucket. This has the same behavior when we need to include new clusters inside of the list. The problem with this solution is that many threads execute this operation concurrently through the shared vector of lists. Therefore, the critical section is required. Critical section enables just one thread to have access to this code region at the same time. Thus only one active thread can access the data referenced by the code. As a result, this produces a sequential behavior and possible slowdown of the parallel code.

We can see in Figure 3, as the number of agents increases, the slowdown increases as well. This occurs because the number of updates in the agents is increased; therefore, more threads enter the critical section code. In this hybrid solution, each MPI process receives a set of agents by MPI message. Then, the process through OpenMP loop parallelization distributes the agents to other cores in the same node that execute the computing over the shared data structure. These operations have to be protected inside of the critical section because many threads are inserting and updating the individuals of this data structure at the same time. The slowdown happens because these operations are invoked constantly.

In order to solve the slowdown problem in the hybrid version, we propose an adaptation of the data structure that supports shared memory paradigm. This data structure can be used with either the SM or Hybrid paradigm.

3.2 Hybrid Memory Data structure

Figure 4 shows the data structure proposed. The main idea is to distribute the clusters among the OpenMP threads. In this way, there are no concurrent accesses to the same cluster when insert and update operations occur because each thread is able to manipulate the clusters that were assigned to it (see Figure 2b). Thus, we can take off many critical sections of the code. Since, in this solution, a thread can only access its clusters, there will be no inconsistencies.

The data structure, see Figure 4, is composed by a Metalist where n positions are dynamically created according to the MPI rank size. Suppose there are ten MPI ranks. Then 10 positions will be created. Each position of the Metalist represents an MPI Rank. Therefore, MPI Rank 0 manipulates the clusters of the position 0; MPI Rank 1 manipulates the clusters of the position 1, and so on until the last MPI Rank. Inside of each position, the proposed data structure has a vector of list of cluster. This vector is created dynamically and has the same number of OpenMP threads that were created in the execution time. Therefore, each MPI process distributes the clusters among its threads. Thus each thread will manipulate only the clusters assigned to it.

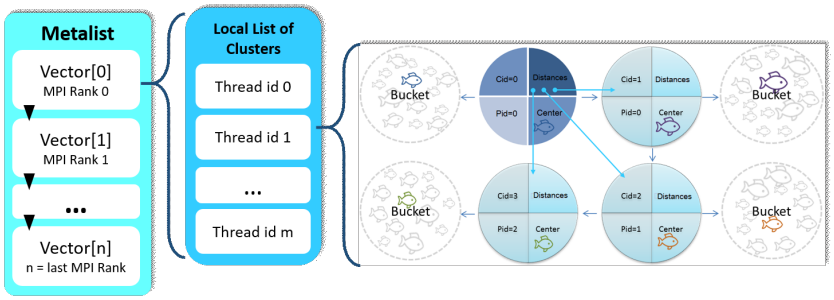


Fig. 4: This figure shows the adaptation of the partitioning data structure to the hybrid simulator version.

The main difference of the proposed data structure is that the pointer to the list of clusters is not shared. This data structure (Figure 4) tries to remove the drawback of list of cluster when applied to the Hybrid solution, see Algorithm 1 for a code example. Each thread receives a pointer to the list of clusters that will be computed. Now, the access to the structure is no longer at only one access point. Thus, the quantity of critical section code is reduced further still.

Algorithm 1 Cluster insertion with proposed data structure and without critical section.

```
1: vector<list_of_clusters* > vector_lc;                ▷ create a vector of list of cluster
2: ...
3: for (long pid=0l;pid<this→nprocs;pid++) do                    ▷ for each process...
4:     for (unsigned int i=0l;i<this→num_thread;i++) do            ▷ for each thread...
5:         vector_lc.push_back(new list_of_clusters());
6:     end for
7:     metalist_lc.push_back(make_pair(pid,vector_lc));
8: end for
9: ...
10: metalist_lc[pid]→lc_thread[i]→insert(c);          ▷ The variable "pid" is MPI Rank
11:                                                  ▷ process, "i" identifies the thread that
12:                                                  ▷ has to manipulate these data and the
13:                                                  ▷ variable "c" is a new cluster
```

4 Experimentation

The environment configuration used in this experimentation has the following characteristics: AMD Opteron 6200 1.6 GHz, L2 (2MB), L3 (6MB), 8 nodes with 64 cores each distributed in 4 sockets with three cache levels, 64 GB RAM

per node, Interconnection Gigabit Ethernet. The simulator was developed by using C++ (gcc 4.3.2), STL (C++ standard template library), MPI namespace (openmpi 1.4.3). The experimentations were carried out for 131072, 262144 and 524288 agents in 32, 64, 128, 256 and 512 cores. We used one MPI Process per core for the DM paradigm and eight OpenMP threads per MPI Process for the Hybrid paradigm. In addition, the number of MPI processes is calculated as the total number of used cores divided by eight.

In the experimentation, see Figure 5, we compare the total execution time of simulation obtained by using the list of cluster and the hybrid data structure proposed. As we can observe, there is no significant difference between both data structure until 128 cores. But there is a significant difference of total execution time when the simulation is performed with 256 and 512 cores. This occurs basically because the communication volume is higher in DM than in the hybrid approach. As presented before, in Section 3, the MPI processes have to send and receive agents that migrate along the simulation. This migration procedure is implemented with MPI_Broadcast and MPI_Wait methods. Therefore, the impact of this type of message is more visible when the number of nodes is increased.

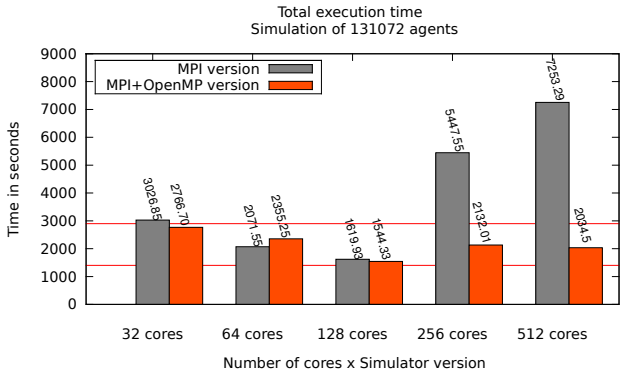


Fig. 5: Total execution time of Fish School simulation.

Also, we can note that the infrastructure with 128 cores is enough to execute 131072 agents. This means that 256 and 512 cores would only be necessary if the problem were bigger. In addition, we can observe that the execution time of the hybrid solution remains within a well-defined range, while the MPI curve has a significant increment. Therefore, if we extrapolate both curves for 1024 and 2048, we will get the hybrid solution with feasible execution times in contrast with the MPI solution. As we can see in the next experimentation, see Figure 6, we maintain the number of cores constant (512 cores) and increase the size of the agents dataset.

The MPI solution continues to show a high growth in total execution time. However, the hybrid solution does not follow the same exponential behavior as it is observed in Figure 6.

The data structure must be created thinking in the paradigm target in order to take advantage of it. Using a data structure developed for a specific paradigm requires adaptation if it is to be used in another one. Otherwise, slowdown at execution time can occur.

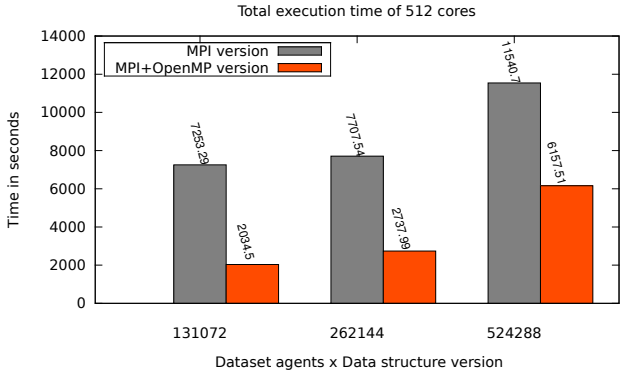


Fig. 6: Total execution time in 512 cores for different sizes of agent datasets.

5 Conclusion

In this paper, we detail a data structure for shared memory that supports a clustering algorithm. This data structure avoids the creation of critical sections and enables us to obtain more efficient parallel and distributed execution. We have tested this data structure using a Fish Schooling Agent-Based Model as a case of study.

The next stage of our research will be to validate this data structure with a bigger agent dataset in order to confirm the results obtained. In addition, it would be interesting to analyze the data structure with some memory and cache counters in order to evaluate the memory contention and cache misses.

As part of our main findings, we show that data structure must be conceived with the paradigm of programming in mind. The migration to another paradigm can cause slowdown to the solution without appropriate adjustment and changes. Also considering the current parallel and distributed architectures, it would be interesting for the data structure to have a flexible and dynamic behavior according to the paradigm used.

Acknowledgments

This research has been supported by the MINECO (MICINN) Spain under contracts TIN2011-24384 and TIN2014-53172-P. Some images come from [10].

References

1. Adhianto, L., Chapman, B.: Performance modeling of communication and computation in hybrid MPI and OpenMP applications. In: Proceedings of the 12th International Conference on Parallel and Distributed Systems, ICPADS'06. vol. 2, pp. 3–8 (2006)
2. Borges, F., Gutierrez-Milla, A., Suppi, R., Luque, E.: A Hybrid MPI+OpenMP Solution of the Distributed Cluster-based Fish Schooling Simulator. Procedia Computer Science 29(0), 2111–2120 (2014)
3. Borshchev, A.: The Big Book of Simulation Modeling: Multimethod Modeling with AnyLogic 6. AnyLogic North America (2013)
4. Cappello, F., Etiemble, D.: MPI versus MPI+OpenMP on the IBM SP for the NAS Benchmarks. In: Supercomputing, ACM/IEEE Conference. p. 12 (2000)
5. Chan, M.K., Yang, L.: Comparative Analysis of OpenMP and MPI on Multi-core Architecture. In: Proceedings of the 44th Annual Simulation Symposium. pp. 18–25. ANSS '11, Society for Computer Simulation International (2011)
6. Chorley, M.J., Walker, D.W.: Performance analysis of a hybrid MPI/OpenMP application on multi-core clusters. Journal of Computational Science 1(3), 168–174 (2010)
7. Chow, E., Hysom, D.: Assessing Performance of Hybrid MPI/OpenMP Programs on SMP Clusters. Tech. rep., Lawrence Livermore National Laboratory (2001)
8. Collier, N., North, M.: Repast HPC: A platform for large-scale agent based modeling. Wiley (2011)
9. Hager, G., Jost, G., Rabenseifner, R.: Communication Characteristics and Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-core SMP Nodes. Proceedings of Cray User Group Conference 4(d), 54–55 (2009)
10. Icons8: Largest colletion of icons (2016), `https://icons8.com`, icons/maps/draws under: CCBY-ND3.0
11. Kiran, M., Richmond, P., Holcombe, M., Chin, L.S., Worth, D., Greenough, C.: FLAME: Simulating Large Populations of Agents on Parallel Hardware Architectures. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems. AAMAS '10, vol. 1, pp. 1633–1636. International Foundation for Autonomous Agents and Multiagent Systems (2010)
12. Rabenseifner, R., Hager, G., Jost, G.: Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes. In: Parallel, Distributed and Network-based Processing, 17th Euromicro International Conference on. pp. 427–436 (2009)
13. Rubio-Campillo, X.: Pandora: A Versatile Agent-Based Modelling Platform for Social Simulation. In: Conference Proceedings SIMUL 2014, The 6th International Conference on Advances in System Simulation. pp. 29–34 (2014)
14. Solar, R., Suppi, R., Luque, E.: High performance distributed cluster-based individual-oriented fish school simulation. Procedia Computer Science 4, 76–85 (2011)
15. Wilensky, U.: NetLogo Ants model. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. (1997), `http://ccl.northwestern.edu/netlogo/models/Ants`