Procedia Computer Science

Volume 29, 2014, Pages 2111–2120

ICCS 2014. 14th International Conference on Computational Science

A Hybrid MPI+OpenMP Solution of the Distributed Cluster-Based Fish Schooling Simulator

Francisco Borges[1], Albert Gutierrez-Milla[1], Remo Suppi[1], and Emilio Luque[1]

Department of Computer Architecture & Operating Systems
Universitat Autonoma de Barcelona, Bellaterra,08193, Barcelona, Spain
francisco.borges@caos.uab.es, albert.gutierrez@caos.uab.cat, remo.suppi@uab.es, emilio.luque@uab.es

Abstract

Exploring the multi-core architecture is an important issue to obtaining high performance in parallel and distributed discrete-event simulations. However, the simulation features must fit on parallel programming model in order to increase the performance. In this paper we show our experience developing a hybrid MPI+OpenMP version of our parallel and distributed discrete-event individual-oriented fish schooling simulator. In the hybrid approach developed, we fit our simulation features in the following manner: the communication between the Logical Processes happens via message passing whereas the computing of the individuals by OpenMP threads. In addition, we propose a new data structure for partitioning the fish clusters which avoid the critical section in OpenMP code. As a result, the hybrid version significantly improves the total execution time for huge quantity of individuals, because it decreases both the communication and management of processes overhead, whereas it increases the utilization of cores with sharing of resources.

Keywords: Parallel and distributed simulation, Parallel discrete-event simulation, High performance distributed simulation, Individual-oriented Model, Hybrid MPI+OpenMP parallel programming

1 Introduction

Parallel and distributed discrete-event simulations are powerful tool for solving complex problems. These problems require both huge computational resources and the effective use of the underlying architecture. However, the programming models used do not always extract the best of the architecture. In accordance with literature (section 2), it might happen because the solution's features do not fit with the chosen model. The programming models explore the computational resource in different ways and consequently each one of them is more appropriate for distributed memory or shared memory. Therefore the hybrid programming can take advantage of what is the best in each paradigm.

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2014

doi:10.1016/j.procs.2014.05.195

In our previous contributions [18, 19, 20], we have focused on a solution for distributed memory paradigm. Where we have obtained good speedup and scalability through message passing paradigm in our simulator. However, our solution suffers with poor efficiency with MPI processes due the broadcast communication between the nodes. In addition, we have observed that our simulator has attributes which could fit with shared memory paradigm. As an example, there are many individuals in the same processors but the communication between them occurs via MPI and the simulator does not take advantage of the locality and sharing resources. Therefore, in this paper we compared the MPI implementation of our fish schooling simulator with the MPI+OpenMP version. The hybrid MPI+OpenMP version uses the message passing to make the communication among the Logical Processes (LP) and control the conservative algorithm of the simulation. The OpenMP threads are used for computing the Individuals which are distributed over the architecture by using the partitioning cluster algorithm [19]. In addition we propose a new data structure partitioning to avoid the critical section in the OpenMP code. Because the data structure of the MPI version was inefficient in the hybrid version.

In the literature, many papers describe the successes and fails of applying the hybrid programming model. Some of them are presented in section 2. Important aspects and fundamental concepts used in this paper are discussed in the section 3. The hybrid MPI+OpenMP parallelization of the simulator and the new data structure proposed is described in section 4. The experimental results, analysis and comparison between the two versions of the simulator are introduced in section 5. And finally, in section 6 we present the conclusions and some future work.

2 Related Work

Hybrid parallel programming enables to explore the best that is offered by distributed and shared architecture in HPC [9]. The distributed architecture allows the scalability increasing the number of nodes. On the other hand, the shared architecture explores the efficiency, memory savings, the locality of data decreasing the communication and process migration. However, the hybrid programming model can not be regarded as the ideal for all codes[3, 17]. Many studies were conducted to compare the hybrid parallel programming.

Cappello and Etiemble [3] consider that the performance of model depends on the level of shared memory model parallelization; the communication patterns; and the memory access patterns. They compare hybrid models with existing MPI codes by using the NAS 2.3 benchmark. In addition, they show that MPI approach is better for most of the benchmarks whereas the hybrid approach is better when the benchmark has high communication time and the level of parallelization is sufficient. Other authors [4, 14] also did the model programming comparison through NAS benchmark.

Chan and Yang [4] argue that MPI can be more favorable with the scalability of clusters. However, OpenMP can favor the speed of shared memory. In addition, the application performance can be affected by the type of problem that is being solved and its size. They show that the effect of MPI communication is the main weakness of this programming model. And finally, they conclude that OpenMP prevails over MPI especially with using a multi-core processor. Hybrid programming models can match better the architecture characteristics of an SMP cluster, and that would replace message passing communication with synchronized thread-level memory access [10].

Smith and Bull [17] discuss situations where the hybrid model may be more efficient than its corresponding MPI implementation on an SMP cluster, such as: codes which scale poorly with

MPI; poorly optimised intra-node MPI; poor scaling of the MPI implementation. Chow and Hysom [7] try to understand the performance of hybrid MPI and OpenMP programs. These authors give an overall review about factors and parameters, which affect the hybrid program performance. Jin et al. [14] bring that the no well-defined interaction between MPI processes and OpenMP threads impedes a wider adoption of hybrid programming. In fact, this limitation requires a strong effort of the programmer in order to develop hybrid solutions. However, the MPI community are working on to improve the interface with threads in MPI 3 [14].

The literature presents many aspects that influence the performance of hybrid solutions. Each approach could have advantages or disadvantages depending on the: architecture [7, 11, 16], network interconnection [6], and application [3, 7]. It is possible to find many researches reporting comparisons between programming model and orientations about how to explore appropriately these techniques. For a list with additional references see Adhianto and Chapman [1]. These authors show other related works that: has shown performance improvement using the hybrid model; others that report poor hybrid performance; and some which present minor benefits to adding OpenMP to an MPI program.

3 Background

In this background section, we focus on the most important aspects of the pure MPI fish schooling simulator. The understanding of these aspects will help to bear the experimental results obtained in this paper. For more details and other informations about our distributed fish schooling simulator consult our previous contributions in [18, 19, 20].

3.1 Fish Schooling Simulator

During the last few years, our group have been researching about individual-oriented models in high performance time-driven simulation with the purpose of decreasing the total execution time and providing close-to-reality simulation results. Individual-oriented models (IoM) are discrete models which are developed in order to understand how global patterns emerge from individual interactions in ecosystems. These models consist of a fixed number of autonomous individuals, interaction rules, individual attributes, which are maintained through time, and an environment where interactions occur.

Our distributed cluster-based individual-oriented fish schooling simulator is based on the biological model described in [12] and [13]. Fish Schools are one of the most frequent social groups in the animal world [12]. This social aggregation shows complex emergent properties, such as: strong group cohesion and high level of synchronization. In order to describe the fish behavior the model considers that each fish changes its position and orientation in discrete step of time (time-driven simulation) and the new fish's position and orientation depend on the position and orientation of a fixed number of nearest-neighbors. The neighbor's influence on a single fish depends on its space-time position. In order to select appropriately the neighbors' influence the model identifies three vision areas: attraction, repulsion and parallel orientation.

3.2 Cluster-Based Partitioning

One of the challenges in distributed individual-oriented simulation is how to distribute individuals on the distributed architecture in order to obtain the best scalability and efficiency. Solar et al. [19] have implemented a partitioning algorithm using a cluster-based approach. This approach consists in assigning to each node a fixed set of individuals.

The partitioning method uses a hybrid partitioning method based on Voronoi diagrams and covering radius criterion. Voronoi diagram is a data structure in computational geometry where given some number of objects in the space, their Voronoi diagram divides the space according to the nearest-neighbor rule [2]. In our simulation, each object is represented by an individual (fish) and it is associated with the area closest to it. Covering radius criterion consists in trying to bound the area c_i by considering a sphere centered at c_i. These centroids (c_i) contain all the objects of the problem domain that lie in the area [5].

The individuals are associated with a position in a three-dimensional euclidean space and together with the euclidean distance generating a metric space. The distance between objects of this metric space is defined by a set of objects X subset of the universe of valid objects U and a distance function $d : X^2 \rightarrow R$, so $\forall x, y, z \in X$, must be met the following conditions: Positiveness: $d(x, y) \geq 0, d(x, y) = 0 \Rightarrow x = y$; Symmetry: $d(x, y) = d(y, x)$; and Triangular inequality: $d(x, y) + d(y, z) \geq d(x, z)$. These conditions determine the visibility of individuals, which allowing use of similarity or proximity within the distributed simulation. Therefore the partitioning method consist of two phases: the centroids selection by means of covering radius criterion which ensures it a set of centroids far away enough to the others; and the space decomposition by means of Voronoi diagrams which allow it to define similar size areas with similar number of individuals.

3.3 List of Clusters Data Structure

The data structure used to store individuals is called fixed-radius list of clusters[5]. As presented in the previous section, we are using a hybrid Voronoi diagram/covering radius as build criterion for a fixed-radius list of clusters[15]. The radius is fixed in function of the maximum fish vision area. This allow us defining areas in which individuals can interact only with individuals belonging to adjacent areas. The data structure is formed by a linked list of clusters, Figure 1, implemented in C++. Each cluster object is composed of several data such as: 1) *centroid* - which is the most representative element of the cluster; 2) *pid* - the processor identifier indicates in which node each cluster is stored; 3) *bucket* - in which individuals belonging to the cluster are stored; 4) *cid* - the cluster identifier indicating the cluster position in the list; and 5) informations about the distances to other clusters.

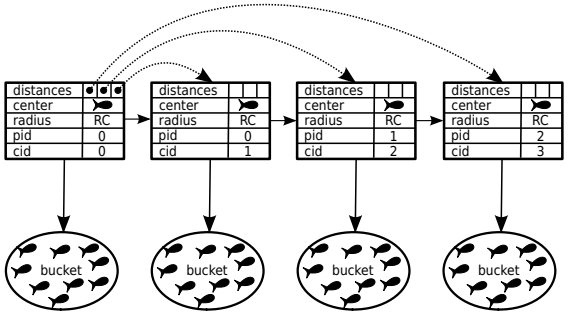

Figure 1: List of clusters used to distribute the individuals through distributed architecture.

The list of clusters is distributed through the distributed architecture. The distribution model used is based on proximity concept. We distribute the list header together a fixed set of different clusters to each node. This fixed set of clusters is determined by the maximum number of individuals that each node can have ($\frac{N_{individuals}}{N_{processors}}$), and how close they are to each other. The

aim is assigning contiguous groups of clusters to each node in order to decrease communication and computing involved in selecting data to transfer. We group clusters by proximity into sets of clusters that we have called *meta-cluster*.

4 Hybrid MPI+OpenMP Parallelization

The original fish schooling simulator was developed for distributed memory. There are several broadcast communications in the simulator because the LPs must exchange information in order to keep the coherence of simulation. Consequently, there is a lack of efficiency when large number of cores are used and these cores are distributed in many nodes. The following two key features support the development of the hybrid version:

- Each LP needs to know all the positions of the centroids for each simulation step. Fish are constantly in motion, therefore the centroid changes according to their position. In addition, the fish do the cluster migration process during all simulation. The migration process occurs when the fish change their location from one LP to other. In order to solve these problems, the pure MPI simulator was designed and developed by using MPI broadcast communication. Currently in the MPI version, each core receives one MPI process. Thus, the quantity of MPI broadcast messages increases when the number of cores increases.

- The partitioning algorithm uses the meta-cluster to store consecutively the clusters in the same process. As fish behavior is influenced by their neighbors the operations of computation and communication occur between the fish that are in same meta-cluster. This way, it is possible to explore the fine-grain parallelism through OpenMP threads.

We have used the OpenMP parallelization technique called fine-grain. This approach consists in the parallelization of the loop nests in the computation part of the MPI code [3]. We have focus on the main functions of the simulator. The main challenge of our hybrid parallelization is to use the current list of clusters data structure (Figure 1) with OpenMP thread concurrently. Individuals are inserted, deleted and updated on the clusters along simulation. Therefore the same cluster and individual may be computed at same moment. Keeping the coherence and the code free of memory access violation resulted in an undesirable slowdown by using critical section. After preliminary experiments, we observed that changes in the list of clusters data structure were necessary. Therefore we propose a new data structure to deal with the simultaneous inserting, deleting and updating problems. It is presented in Figure 2.

In this new data structure, we create dynamically a vector of meta list of clusters (Figure 2.a). The size of this vector depends on the number of MPI processes created by the simulation. The Meta list of clusters (Figure 2.b) is composed of two fields: 1) ID, which has the MPI rank number; and a vector which has a n-list of clusters, where n is equal to thread number created by MPI process. This data structure enable us to reduce, almost eliminate, the critical sections, because, the clusters are distributed among the threads and each thread executes its operations in its own list of clusters. MPI process continues working with the list of clusters (Figure 1) but the computing is executed in the meta list of clusters (Figure 2) when the OpenMP section begins. After data are computed through the OpenMP threads these data are consolidated in the list of clusters of MPI process. The consolidation operation does a sequential copy of individuals from the OpenMP data structure to the MPI data structure. Therefore this effort is valid just when the time spent in this operation is less than the communication time of the MPI processes. In the pure MPI version, one MPI process is created per core. On the other

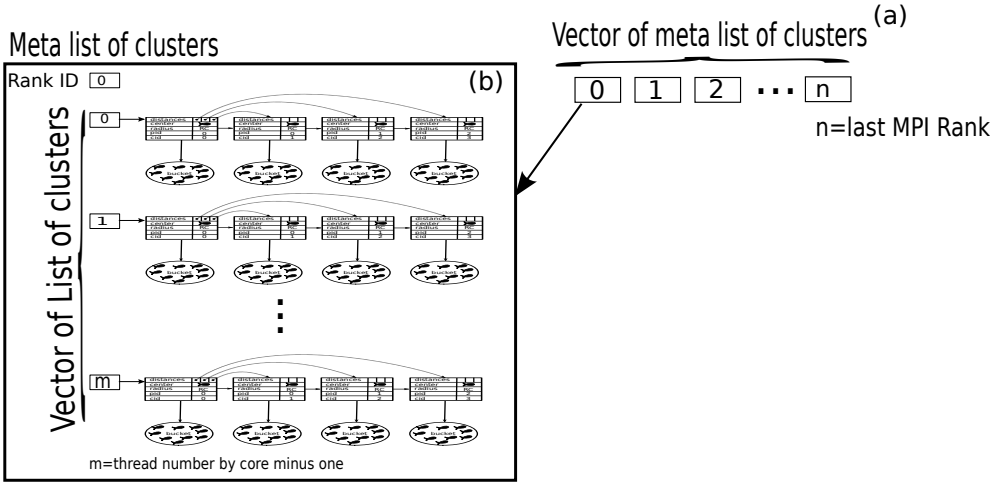

Figure 2: Meta list of clusters proposed in order to avoid critical section in the OpenMP code.

hand, the number of MPI processes is reduced by using the hybrid solution. This decreases considerably the communication time and all the inherent overhead of the management process and its context. In the hybrid solution, there are more clusters doing computing inside the same MPI process. Consequently, some communications and computing among the cluster are eliminated.

5 Experimental Results

The execution environment has the following characteristics: Cluster DELL, AMD Opteron 6200 1.6 GHz, L2 = 2MB, L3 = 6MB, 8 nodes (total = 512 cores), 64 GB RAM per node, Interconnection Gigabit Ethernet. Each node on cluster has 64 cores distributed in 4 socket with three cache level. The two simulator versions were developed by using C++ (gcc 4.3.2), STL (C++ standard template library), MPI namespace (openmpi 1.4.3). For the MPI version experiments, each MPI process was created per core. For MPI+OpenMP version experiments, we have two scenarios where each MPI process creates 8 and 16 OpenMP threads. In addition the number of MPI process created is indicated by: *total number of cores* used divided by *number of threads*.

In order to analyse both versions, we compare the total execution time taking in consideration the total number of cores used on experiment. The details of experiments can be seen in Table 1. The statistical results are guaranteed by using the batch replication techniques described in [8]. In this first experiment, shown in Figure 3, we keep a constant number of individuals and we vary the number of cores of both the MPI version and the MPI+OpenMP version, using 8 and 16 threads in the latter. This experiment is a proof of concept which aim is to analyse the behaviour of both versions with different number of cores. In addition, we verify the influence of the number of MPI processes and threads on the hybrid version.

The MPI version has better results than the hybrid version up to 64 cores. There are no inter-node communication on 32 and 64 cores because these cores lie in the same node. Therefore, the overhead of OpenMP data structure's consolidation does not compensates the MPI communication time. This version scales very well up to 128 cores but it suffers with

Architecture details		MPI experiments
Cores used	Node	MPI processes
32	1	32
64	1	64
128	2	128
256	4	256
512	8	512

Architecture details		MPI+OpenMP experiments			
		Scenario 1		Scenario 2	
Cores used	Node	MPI processes	OpenMP threads	MPI processes	OpenMP threads
32	1	2	16	4	8
64	1	4	16	8	8
128	2	8	16	16	8
256	4	16	16	32	8
512	8	32	16	64	8

Table 1: The Architecture details column contains information about number of cores used and the number of nodes. MPI Processes column indicates how many MPI processes were created. OpenMP Threads column indicates how many OpenMP threads were created by MPI process.

increasing the number of MPI processes as consequence of the broadcast communication.

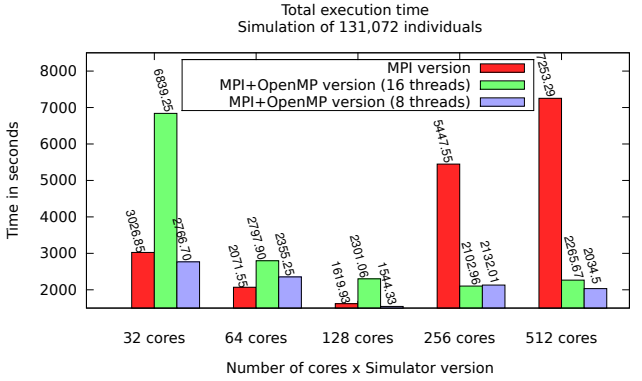

Figure 3: Total execution time comparison between the MPI and the MPI+OpenMP simulator versions.

This experiment (Figure 3) shows that the performance of hybrid version depends on the both number of MPI processes and OpenMP threads. The two MPI+OpenMP approaches have different execution times because their workloads are distinct. Thus, we have to watch out the quantity of individuals inside the OpenMP data structure because it might become a bottleneck. Since the consolidation process of the new list of cluster must copy many individuals between the data structures.

Using more than 128 cores the hybrid version has better results. In this version, we have decreased the number of MPI process consequently the quantity of individuals grew inside the

LP. In other words, each MPI process manipulates and controls more individuals. This implies in decreasing the MPI broadcast communication between the LPs in order to notify others LP about the individuals behaviors. Consequently, the hybrid version provides more intra-node communication rather than inter-node communication. Since intra-node communication is a memory copy among MPI processes it means less MPI communication overhead. On the other hand, the inter-node communication consumes memory bandwidth and slow down the communication because it requires an extra buffer. In order to analyse the scalability of the hybrid version, we increase the quantity of individuals to 262,144. As we can see in Figure 4, the hybrid version scale very well.

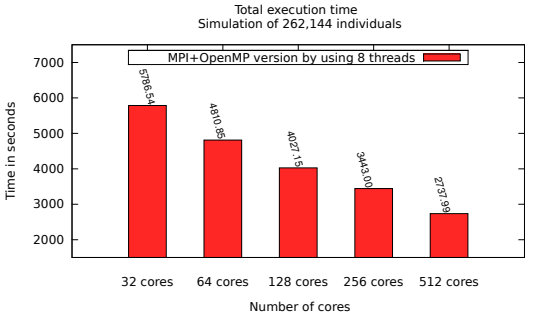

Figure 4: Scalability of the hybrid version by using 8 threads per MPI process.

We are interested in realistic and complex simulations. Therefore, in the last experiment, we simulate 131,072, 262,144 and 524,288 individuals in both versions by using 512 cores. As we can see in Figure 5, the hybrid version has better total execution time than the MPI version. In this experiment, we are using 512 cores and increasing the number of individuals. The hybrid version has more individuals together in same node thus these individuals can read directly the memory through OpenMP threads and avoiding the memory copy among MPI processes. The MPI+OpenMP version reduces the memory requirement overhead from multiple processes. We can observe that the hybrid version is a feasible solution for a high number of individuals and cores, since the number of MPI processes and threads could be adjusted.

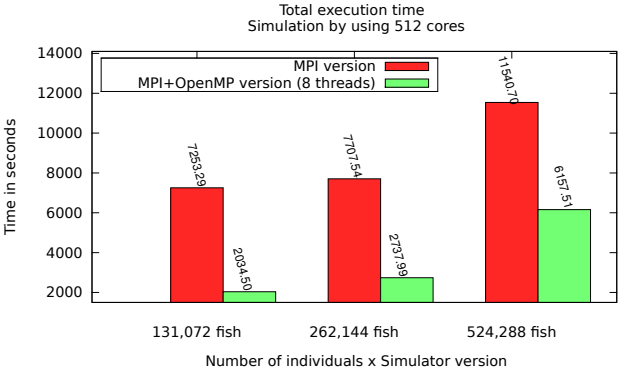

Figure 5: Total execution time comparison between the MPI and the MPI+OpenMP simulator versions by using 512 cores.

6 Conclusions

In this paper we have implemented a hybrid MPI+OpenMP version of a parallel and distributed discrete-event fish schooling simulator. This simulator was modeled by using the Individual-oriented Model where each individual interacts with another and a fine-grain parallelism can be applied. The hybrid parallelization of existing discrete-event individual-oriented MPI implementation may bring excellent results depending on the communication pattern among the MPI processes; partitioning of individuals through the architecture distributed and shared; computing between these individuals; and the relation between the number of MPI processes and its OpenMP threads. Generally the data structure for distributed memory are not appropriate for shared memory because it has to be designed for simultaneous access and free of critical sections. Critical sections are main bottlenecks of OpenMP codes. In order to treat this problem in our simulator, we create a new data structure to manipulate the data inside the OpenMP section. This new data structure is more efficient because it distributes evenly clusters among the threads enabling simultaneous operations over the individuals without using critical sections. We believe that the data structure proposed can be used in other kind of scientific applications which have a similar data partitioning and have fine-grain parallelism code. In addition, we verified that replacing message passing communication with synchronized thread-level memory access is an interesting approach for applications which has intensive broadcast communication process. In our experiments, we reduce the total execution time of the hybrid version in comparison with the previous MPI simulator. We observed that the hybrid version is 3.56, 2.81 and 1.87 times better than the MPI version simulating 131,072, 262,144, and 524,288 individuals by using 512 cores, respectively. Summarizing, the main contributions of this paper are:

- We have developed a hybrid version that has better results than the MPI version for high number of cores and individuals.
- In addition, a new data structure of clusters which avoids the critical section in OpenMP code was proposed.
- Finally, we fit the features of our simulator with the underlying architecture which support the reduction of total execution time of simulation in the hybrid MPI+OpenMP version.

The main objectives for future work are the following: 1) we will discuss and analyze more precisely the portion between the total execution time and communication time; 2) to develop the dynamic load balancing algorithm for the hybrid solution; 3) to explore the thread and process affinity in the hybrid solution; and 4) identify the best relation between the number of processes and its threads for the hybrid version.

7 Acknowledgments

- This research has been supported by the MICINN Spain under contract TIN2007-64974 and the MINECO (MICINN) Spain under contract TIN2011-24384.

References

[1] L. Adhianto and B. Chapman. Performance modeling of communication and computation in hybrid mpi and openmp applications. In *Parallel and Distributed Systems, 2006. ICPADS 2006. 12th International Conference on*, volume 2, pages 6 pp.–, 2006.

[2] Franz Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23:345–405, September 1991.

[3] F. Cappello and D. Etiemble. Mpi versus mpi+openmp on the ibm sp for the nas benchmarks. In *Supercomputing, ACM/IEEE 2000 Conference*, page 12, nov. 2000.

[4] Michael K. Chan and Lan Yang. Comparative analysis of openmp and mpi on multi-core architecture. In *Proceedings of the 44th Annual Simulation Symposium*, ANSS '11, pages 18–25, San Diego, CA, USA, 2011. Society for Computer Simulation International.

[5] E. Chavez and G. Navarro. An effective clustering algorithm to index high dimensional metric spaces. In *Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE'00)*, pages 75–, Washington, DC, USA, 2000. IEEE Computer Society.

[6] Martin J. Chorley and David W. Walker. Performance analysis of a hybrid mpi/openmp application on multi-core clusters. *Journal of Computational Science*, 1(3):168–174, August 2010.

[7] Edmond Chow and David Hysom. Assessing performance of hybrid mpi/openmp programs on smp clusters. 2001.

[8] Christopher A Chung. *Simulation modeling handbook: a practical approach*. CRC press, 2003.

[9] J. Diaz, C. Munoz-Caro, and A. Nino. A survey of parallel programming models and tools in the multi and many-core era. *Parallel and Distributed Systems, IEEE Transactions on*, 23(8):1369–1386, 2012.

[10] N. Drosinos and N. Koziris. Performance comparison of pure mpi vs hybrid mpi-openmp parallelization models on smp clusters. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 15, april 2004.

[11] Georg Hager, Gabriele Jost, and Rolf Rabenseifner. Communication characteristics and hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes. *Proceedings of Cray User Group Conference*, 4(d):5455, 2009.

[12] Andreas Huth and Christian Wissel. The simulation of the movement of fish schools. *Journal of Theoretical Biology*, 156(3):365 – 385, 1992.

[13] Andreas Huth and Christian Wissel. The simulation of fish schools in comparison with experimental data. *Ecological Modelling*, 75-76:135 – 146, 1994. State-of-the-Art in Ecological Modelling proceedings of ISEM's 8th International Conference.

[14] Haoqiang Jin, Dennis Jespersen, Piyush Mehrotra, Rupak Biswas, Lei Huang, and Barbara Chapman. High performance computing using mpi and openmp on multi-core parallel systems. *Parallel Computing*, 37(9):562 – 575, 2011. Emerging Programming Paradigms for Large-Scale Scientific Computing.

[15] Roberto Uribe Paredes, Claudio Marquez, and Roberto Solar. Construction strategies on metric structures for similarity search. *CLEI Electron. J.*, 12(3), 2009.

[16] R. Rabenseifner, G. Hager, and G. Jost. Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes. In *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*, pages 427 –436, feb. 2009.

[17] Lorna Smith and Mark Bull. Development of mixed mode mpi / openmp applications. *Scientific Programming*, 9(2,3):83–98, August 2001.

[18] Roberto Solar, Francisco Borges, Remo Suppi, and Emilio Luque. Improving communication patterns for distributed cluster-based individual-oriented fish school simulations. *Procedia Computer Science*, 18(0):702 – 711, 2013. 2013 International Conference on Computational Science.

[19] Roberto Solar, Remo Suppi, and Emilio Luque. High performance distributed cluster-based individual-oriented fish school simulation. *Procedia CS*, 4:76–85, 2011.

[20] Roberto Solar, Remo Suppi, and Emilio Luque. Proximity load balancing for distributed cluster-based individual-oriented fish school simulations. *Procedia Computer Science*, 9(0):328 – 337, 2012. Proceedings of the International Conference on Computational Science, ICCS 2012.