Artif Intell Rev (2011) 35:53–72
DOI 10.1007/s10462-010-9183-9

Multi-agent based simulations using fast multipole method: application to large scale simulations of flocking dynamical systems

S. N. Razavi · N. Gaud · N. Mozayani · A. Koukam

Published online: 3 November 2010

Abstract This article introduces a novel approach to increase the performances of multi-agent based simulations. We focus on a particular kind of multi-agent based simulation where a collection of interacting autonomous situated entities evolve in a situated environment. Our approach combines the fast multipole method coming from computational physics with agent-based microscopic simulations. The aim is to speed up the execution of a multi-agent based simulation while controlling the precision of the associated approximation. This approach may be considered as the first step of a larger effort aiming at designing a generic kernel to support efficient large-scale multi-agent based simulations. This approach is illustrated in this paper by the simulation of large scale flocking dynamical systems.

Keywords Simulation · Multi-agent · Fast Multipole method · Flocking

1 Introduction

Multi-agent based simulation (MABS) may be considered as one of the approaches which support microscopic simulation. Multi-agent based modeling allows complex natural behavior of various interacting entities to emerge from a set of simple individual rules. Phenomenon such as flocks of birds, schools of fish, and complex biological systems of cells are good

S. N. Razavi (✉) · N. Mozayani
Department of Computer Engineering, Iran University of Science and Technology, Narmak, Tehran, Iran
e-mail: razavi@iust.ac.ir

N. Mozayani
e-mail: mozayani@iust.ac.ir

S. N. Razavi · N. Gaud · A. Koukam
Multi-agent Group, System and Transport laboratory, UTBM, Belfort Cedex 90010, France
e-mail: n.gaud@utbm.fr

A. Koukam
e-mail: abder.koukam@utbm.fr

examples of how systems with simple rules can demonstrate complex emergent behaviors as a result of interactions with other neighboring agents (local to global). However, as soon as we consider a microscopic simulation of several individuals and their relationships, the complexity of the system and associated computational costs increase. Our target application consists in a microscopic simulation tool that is able to execute a model as accurate as possible in a time regarded as acceptable for an end-user. We are therefore faced a dilemma common in the field of simulation: to establish a compromise between performance and accuracy.

The Fast Multipole Method (FMM) has been acclaimed as one of the ten most significant algorithms in scientific computation discovered in the 20th century (along with algorithms such as the Fast Fourier Transform) Dongarra and Sullivan (2000). Its inventors, Greengard and Rokhlin (1987), won the 2001 Steele prize, in addition to getting Greengard the ACM 1987 best dissertation award. The FMM approach and its derivatives (Kernel-independent FMM, Multilevel FMM, etc) enable us to approximately evaluate (with a specified precision) the product of particular dense matrices with a vector in $O(N \log N)$ operations, when direct multiplication requires $O(N^2)$ operations. This algorithm was already applied in various problem areas such as simulation of physical systems (Electromagnetic, Stellar clusters, Turbulence), Computer Graphics and Vision (Light scattering) and Molecular dynamics. This article describes the experiments we conducted to apply Multilevel FMM (MLFMM) to multi-agent based simulations of large scale flocking dynamical systems. Our ultimate aim is indeed to apply MLFMM to (quasi-)real time microscopic simulation of large pedestrian crowds. Flocking dynamical systems was chosen as case study because the underlying behavioral algorithms may be regarded as a simplified version of those used in the pedestrians micro-simulation. It may also be considered as an extension of particle-based models commonly used in physics, which is the traditional application field of the FMM approach. This application is therefore at the confluence of the two domains.

In order to make best use of the available execution resources while minimizing the costs related to the simulated model, our approach consists in dynamically adjusting the precision of the model against the volume of available computational resources. At first, MLFMM helps to reduce the cost of model implementation by reducing its overall complexity ($O(N^2)$ to $O(N \log N)$). The second step is to dynamically change the precision of the model during simulation runs. On this second point, MLFMM also offers the possibility to manage its level of precision, depending on its error in mathematical expansions and clustering size in the associated tree structure.

The primary aim of this paper is to validate the usability of MLFMM in the context of multi-agent based microscopic simulation. It then introduces a general approach for using this method as a support to design efficient large-scale agent-based simulations. The main contributions of this paper consist in adapting existing flocking behavioral models to integrate them within the MLFMM and providing a sensitivity analysis of the proposed method and its associated performances. We show in particular that MLFMM parameters are not sufficient to accurately and effectively control the simulation accuracy.

The paper is organized as follows. Section 2 provides an early glimpse of the scientific background on flocking models and details the fundamentals of fast multipole methods. Our contributions and the integration of the agent-based flocking model within a multilevel kernel-independent fast multipole method are detailed in Sect. 3. Section 4 describes the experimental results and the corresponding analysis. Finally, Sect. 5 summarizes the contributions of the paper and describes some future work directions.

2 Background

2.1 Flocking for multi-agent dynamic systems

Flocking is a form of collective behavior of a large number of interacting agents with a common group objective. Typical flocking phenomena include flocks of birds, schools of fish, animal herds and bee swarms Karaboga and Akay (2009), Reynolds (1987) introduces the three first heuristic rules "*cohesion*", "*separation*", and "*alignment*" that lead to creation of the first computer animation of flocking.

Among the physics-inspired theoretical approaches studying flocking particles systems, we may mention the works of Vicsek et al. (1995) and Shimoyama et al. (1996) that mainly focus on the study of alignment rules and associated emergent phenomena. Mogilner and Edelstein-Keshet (1999) and Toner and Tu (1998) propose continuum models to study alignment and swarming. In Tanner et al. (2003), a model based on artificial potential function (APF) in a network with fixed or dynamic topologies is presented. It proposes a centralized algorithm for particle systems that leads to irregular collapse for generic initial states. It also introduces a distributed version that leads to irregular fragmentation (disintegration of a flock into smaller groups combined with violation of inter-agent constraints). Fragmentation and collapse are two well-known pitfalls of flocking algorithms most likely occurring for generic set of initial states and large number of agents (Zhou et al. 2009; Olfati and Reza 2006).

Our flocking model is based on the theoretical framework for design and analysis of distributed flocking algorithms from Olfati and Reza (2006) and Saber and Murray (2003). This model may be considered as a modified version of Tanner et al. (2003)'s APF trying to avoid traditional pitfalls of flocking. Olfati et al. (2003)'s model provides a definition of "flocking" for particle systems that is independent of the method of trajectory generation for particles. In this sense, it has the same role as "Lyapunov stability" for nonlinear dynamical systems. This model is based on a systematic method for construction of collective potential functions for flocking.

In this model, a lattice-type structure is used to model the geometry of desired conformation of agents in a flock. In such conformation, each agent should be equally distanced from all of its neighbors while moving towards its objective that may be static or dynamic. As flocking occurs in presence of multiple obstacles, each agent is equipped with obstacle avoidance capability. Therefore, each agent in this model has to deal with three kinds of objects in the environment: nearby flocking agents (α-agents), obstacles (β-agents) and a virtual leader (γ-agent). For doing so, each agent in the flock applies a control input that consists of three different terms:

$$u_i = u_i^{\alpha} + u_i^{\beta} + u_i^{\gamma}$$

Each agent in the flock is synchronizing itself with nearby flockmates in terms of position and speed. In this model, an interaction range r is defined between two agents. All agents with a distance less than r to agent i are considered as its neighbors. During flocking, agent i tries to maintain an equal distance with its neighbors and also to match its velocity with theirs. The term u_i^{α} in the control input is responsible to coordinate agent i with its neighbors to form the lattice-type structure as illustrated in the left part of Fig. 1.

Coordinating each agent with its neighbors is not sufficient to produce a real flocking behavior. In real situations, all agents in the flock have to move towards the same destination. In this model, the target position is represented by a virtual leader (γ-agent). A γ-agent has the role of a virtual leader in charge of navigation and control of the behavior of a flock as a whole. It is a mechanism that provides a common objective for a group of flocking agents.

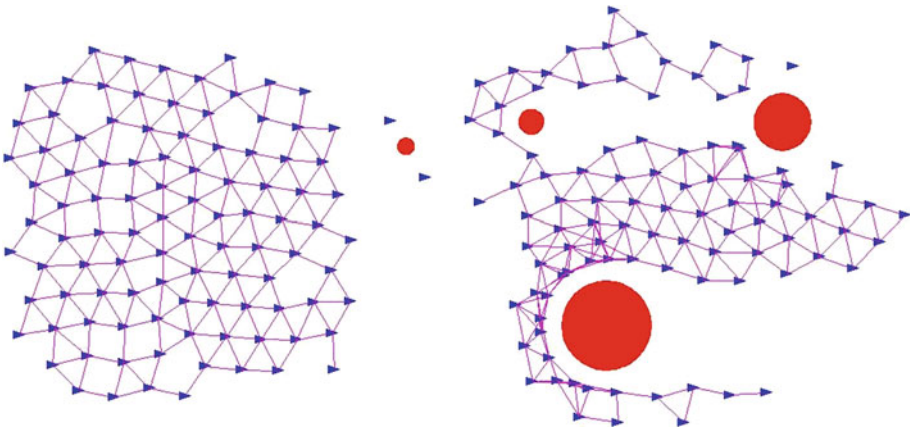

Fig. 1 Lattice-type conformation for a group of flocking agents (*left*) and associated obstacle avoidance capability (*right*)

This objective may be static (fixed position) or dynamic (moving object). The term u_i^{γ} in the control input manages this leader tracking procedure; it may be considered as a navigational feedback.

Beside interactions with nearby agents and tracking the virtual leader, each agent in the flock has the ability to avoid obstacles. Obstacle avoidance is achieved by introducing a third type of agent called β-agent. Whenever an agent is in close proximity of an obstacle, a β-agent will be induced on the border of that obstacle. The interaction between α-agent and β-agent prevents α-agent to further approach to the obstacle and hence helps it to avoid that obstacle (see right part of Fig. 1). The term u_i^{β} in the control input contains all of the interactions between agent i and nearby obstacles. For more details on the exact definitions of u_i^{α}, u_i^{β} and u_i^{γ} the reader may refer to Olfati and Reza (2006).

2.2 Fast multipole methods

2.2.1 Overview

Fast Multipole Method (FMM) represents a revolutionary view towards algorithm design for computational tasks. It is an example of a class of algorithms that trades accuracy for reduced complexity. The algorithm allows the product of a dense matrix and a vector to be approximated in $O(N \log N)$ operations within a pre-established error bound. Since many scientific computations only require a certain accuracy (up to the machine precision), FMM is well suited for extremely large computational problems and offer improved efficiency and reduced memory requirement. It is mainly used in computational physics to study the dynamic of particle systems. However, FMM is a complex technique, hard to implement.

FMM achieves its performance by operations called "*expansions*", which expand the expression to be evaluated into the sum of a series. Expansions are done at given points in the considered space. The terms in the series will be similar to those of the expansions from other points and thus can be regrouped before actual calculations are carried out. Combined with operations called "*translations*", which translate the terms in the series from one center to another, the reduction in computational complexity is achieved by computing these terms only once and reusing them for all points in the domain where the expansion is valid. Succes-

sively regrouping particles at higher levels, groups of particles, and so on can also increase the efficiency of FMM. This is achieved by recursively using translations to group together more evaluations, and is commonly referred to as Multilevel FMM (MLFMM). MLFMM algorithm partitions the domain into a tree structure that can be classified as a complete quadtree in 2D space (Samet 1984) or an octree in 3D. It is well suited for problems with a uniform distribution of particles. However, for other problems it may not always be true, and further optimizations may be achieved by partitioning the domain adaptively. For regions with fewer particles, the level of partitioning may be smaller than those with a higher density of particles. This variation of the initial algorithm is usually called adaptive MLFMM.

2.2.2 Mathematical principle

Let us assume that there are N source densities u_i located at N points x_i ($i \in [1, \ldots, N]$) in 2D or 3D space. Our goal is to compute the potential v_j at M target points y_j $j \in [1, \ldots, M]$ induced by a kernel G using the following relation:

$$v_j = \sum_{i=1}^{N} G\left(x_i, y_j\right) u_i = \sum_{i=1}^{N} G_{ij} u_i, \quad j = 1, \ldots, M$$

We can also state the above summation as a matrix-vector multiplication $\boldsymbol{G}.\,\boldsymbol{u} = \boldsymbol{v}$. An algorithm, which uses direct implementation of this summation requires O(MN) operations, or, in the case where $M \approx N$, the order of complexity is O(N^2) to compute the interactions between all pairs of source densities and target points. It is clear that for a very large N such algorithm is not practical, while FMM can compute an approximate potential with a prescribed relative error in O(N) operations for the building phase and O($N \log N$) operations for the solving phase. So, FMM algorithm dramatically reduces the complexity of the above computation.

Here, we use a simple kernel like $G_{ij} = \log \left\| y_j - x_i \right\|$ to explain how FMM works. The main idea of FMM is to encode the potentials of a set of source densities using multipole expansions and local expansions at places far away from these sources. Assume the source densities are located inside a disk centered at z_c (a complex number corresponding to the center of the disk on the plane) with radius r. Then for all points z outside the disk with radius R ($R > r$), we can represent the potential v_z at z from the source densities inside the disk with a set of coefficients a_k ($0 \leq k \leq p$), where

$$v_z = a_0 \log\left(z - z_c\right) + \sum_{k=1}^{p} \frac{a_k}{\left(z - z_c\right)^k} + \mathrm{O}\left(\frac{r^p}{R^P}\right)$$

where $\mathrm{O}\left(\frac{r^p}{R^P}\right)$ is a residual term and a_k ($0 \leq k \leq p$) satisfies:

$$a_0 = \sum_{j=1}^{m} u_j, \quad a_k = \sum_{j=1}^{m} \frac{-u_j \left(z_j - z_c\right)^k}{k}$$

This is called multipole expansion. Also, if the source densities are outside the disk with radius R, the potential at a point z inside the disk with radius r can be represented with a set of coefficients c_k ($0 \leq k \leq p$) where

$$v_z = \sum_{k=1}^{p} c_k \left(z - z_c\right)^k + \mathrm{O}\left(\frac{r^p}{R^P}\right)$$

Where c_k $(0 \leq k \leq p)$ satisfies:

$$c_0 = \sum_{j=1}^{m} u_j \log\left(z_c - z_j\right), c_k = \sum_{j=1}^{m} \frac{-u_j}{k \cdot \left(z_j - z_c\right)^k}$$

We call this expansion as local expansion. In both expansions, the truncation number p is usually a small constant determining the desired accuracy of the result. In the remaining of this paper, we will refer to p as the FMM error bound.

2.2.3 Expansion principle within the FMM

If we write $G\left(x_i, y_j\right)$ as

$$G\left(x_i, y_j\right) = \sum_{m=0}^{\infty} a_m\left(x_i - x_*\right) f_m\left(y_j - x_*\right) \cong \sum_{m=0}^{p-1} a_m\left(x_i - x_*\right) f_m\left(y_j - x_*\right)$$

then, v_j can be computed using the following method:

$$\begin{aligned} v_j &= \sum_{i=1}^{N} G\left(x_i, y_j\right) u_i \\ &\cong \sum_{i=1}^{N} \sum_{m=0}^{p-1} a_m\left(x_i - x_*\right) f_m\left(y_j - x_*\right) u_i \\ &= \sum_{m=01}^{p-1} \sum_{i=1}^{N} a_m\left(x_i - x_*\right) f_m\left(y_j - x_*\right) u_i \\ &= \sum_{m=01}^{p-1} f_m\left(y_j - x_*\right) \sum_{i=1}^{N} a_m\left(x_i - x_*\right) u_i \\ &= \sum_{m=01}^{p-1} c_m f_m\left(y_j - x_*\right) \end{aligned}$$

where $c_m = \sum_{i=1}^{N} a_m\left(x_i - x_*\right) u_i$.

As the reader may notice, the set of coefficients c_m does not depend on the target point y_j. So, we can compute these coefficients only one time and reuse them for all target points in the domain where the expansion is valid. Using the above factorization, the common coefficients can be computed in O(pN) and the potential v_j for M target points in O(pM) operations. So, the total complexity will be equal to $\mathrm{O}\left(pN + pM\right)$. As while as p is equal to a small constant, this complexity is practical even for very large values of N.

FMM employs the above representations in a recursive way. The computational domain, a box large enough to contain all source and target points, is hierarchically partitioned into a tree structure (a quadtree in 2D or an octree in 3D). Each node of the tree corresponds to a geometric box (square or cube). The tree is constructed so that the leaves contain no more than a pre-specified number of points. This number is defined as the *clustering size* of the tree. For each box, the potential induced by its source densities is represented using a multipole expansion, while the potential induced by the sources from non-adjacent boxes is encoded in a local expansion.

Fig. 2 Translation principle within the FMM (Ying et al. 2004)

2.2.4 Translation principle within the FMM

In addition to multipole and local expansions, which can be used for efficient evaluation, translations between these expansions are also available to make an O(N) algorithm possible. The various types of FMM translations are described below and illustrated in Fig. 2 (Ying et al. 2004). In this figure, the multipole expansion at z_S represents the contribution from source densities (marked with "+") inside the small box in the left to the far field. Similarly, the local expansion at z_T represents the contribution from the far field to the target points (marked with "Δ") inside the small box in the right.

M2M: *Multipole-to-Multipole* translation transforms the multipole expansions of a box's children to its own multipole expansion: z_S to z_M in Fig. 2 (expansion of boxes in adjacent levels).

M2L: *Multipole-to-Local* translation transforms the multipole expansion of a box to the local expansion of another non-adjacent box: z_M to z_L in Fig. 2 (expansion of non-adjacent boxes).

L2L: Finally, L2L is *Local-to-Local* translation of the local expansion of a box's parent to its own local expansion: z_L to z_T in Fig. 2 (re-expansion between adjacent levels).

2.2.5 FMM general algorithm

By exploiting the previously described tree structure and by combining Expansions' and Translations' principles, FMM algorithm computes the potentials in the target points in three basic steps:

Upward pass. During the first step, the multipole expansion for each box is computed in a bottom-up manner. At the leaves of the tree the multipole expansions are built using source densities inside the boxes. Then multipole expansions for each non-leaf box are built by shifting the multipole expansions of its children using M2M translation.

Downward pass. During the second step, the tree is traversed in a top-down manner to compute the local expansion of each box. For each box B, the local expansion is the sum of two parts: firstly the local-to-local transformation collects the local expansion of B's parent. This way, we can compute contributions from the sources in all the boxes, which are not adjacent to B's parent. Secondly multipole-to-local transformation collects the multipole expansions of the boxes, which are the children of the neighbors of B's parent but are not adjacent to B (these boxes compose the interaction list of B). The sum of these two parts encodes all the contributions from the sources in the boxes, which are not adjacent to B.

Final summation. In this last step, for each box B, the far interaction, which is evaluated using local expansion on this box, is combined with near interactions evaluated by iterating

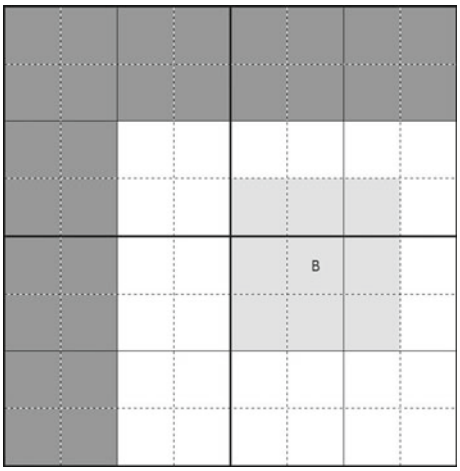

Fig. 3 Contributions from the far field and near field to box B

over all the source points in the neighborhood of the target box to obtain the potential of each target point.

Figure 3 describes the various domains used to support this algorithm. The objective is to compute the potentials from all boxes in the computational domain to box B in this figure. For the considered B box, the near field is defined to be the region containing B and its adjacent neighbors (the light grey boxes in the figure). All other boxes compose the far field of the box B. The near field interactions of the box B are computed directly in the final summation step in the FMM algorithm. The contribution from the far field consists of two main parts: the contribution from white boxes (the *interaction list* of box B) and the contribution from the dark grey boxes. Contributions from white boxes are computed using M2L translations during downward pass. While contributions from dark grey boxes are calculated by first computing their influences on the parent box of B (using M2L translations) and then by L2L translation from parent of B to B.

For more details on FMM and MLFMM, the reader may refer to Gumerov et al. (2003), Gumerov and Duraiswami (2005) and Darve (2000).

2.3 Kernel independent FMM (KIFMM)

In the classical FMM described above, expansions and translations depend on the kernel G in an analytic way. For some kernels, these analytic expansions and translations can be quite difficult to obtain. The main advantage of the kernel independent algorithm of Ying et al. (2004) is that it removes such kernel dependency. In a word, using the original FMM would limit usability and potential generalization of the proposed approach. Complex agents' behaviors could not be easily implemented. These issues are further detailed in the Sect. 3.

For these reasons, we have opted for a kernel-independent FMM-like algorithm Ying et al. (2004) that has the structure of adaptive FMM algorithm but requires only kernel evaluations, and it does not sacrifice the efficiency of the original algorithm. Like analytic FMM algorithm, the method has $O(N)$ asymptotic complexity and works well for non-uniform particle distributions. The difference between original FMM and KIFMM is how multipole and local expansions are represented and how translations (M2M, M2L and L2L) are done. In KIFMM, there are two surfaces (circles in 2D and cubes in 3D) associated to each box: the *upward equivalent surface* and *downward equivalent surface*. For a box B, upward

equivalent surface approximates in the far field of B the potential generated by the densities in B and so it replaces the multipole expansion used in the original FMM. Also for a box B, downward equivalent surface approximates in B the potential generated by densities in the far field of B and so it replaces the local expansion used in the original FMM.

During upward pass, multipole expansions are replaced by solving local exterior inverse problems. To represent potential generated by the sources inside a box on the far field, it uses a surface enclosing the box with an equivalent density. To find this equivalent density on the surface for a box B, first we compute the potential generated by the densities in the box B on a surface called *upward check surface*. The check surface for a box B is an enclosing surface in the far field of box B. Then we match the potential of the upward equivalent surface to the potential of this check surface. Similarly during downward pass, the far field interaction on a surface enclosing the target box is evaluated by solving an interior Dirichlet-type integral equation to compute an equivalent surface. The computed surface represents the far field interaction on particles inside the target box. After construction of these equivalent surfaces, M2L translations are done by direct evaluations on the far field, sparsified by Singular Value Decomposition (Yarvin and Rokhlin 1999) or Fast Fourier Transform.

In this method, computing surfaces and doing translations requires solving an Integral equation. In order to solve these integral equations, they need to be discretized. Solving each of them consists of two steps: Firstly we need to evaluate the check potential at box B using equivalent density from another box. This step is discretized using a simple numerical quadrature. Secondly we need to compute equivalent density at B from the check potential computed in the previous step. This requires the numerical solution of a first-kind Fredholm equation. This equation is denoted as

$$K\phi = q \tag{1}$$

where ϕ is a column vector of size p, which represents equivalent density of B, q is also a column vector of size p and represents the check potential of B and finally K which evaluates q from kernel and ϕ, is a $p \times p$ matrix. Here p is a discretization parameter determining the precision of the calculations (error bound). In Eq. 1, the matrix K and the vector q are known. This equation forms an inverse problem, Tikhonov regularization is then used to solve it:

$$\phi = \left(\alpha I + K^* K\right)^{-1} K^* q$$

where α is the Tikhonov factor and I is the identity matrix. This becomes a second-kind Fredholm integral equation. In our implementation, the Nyström method is adopted to solve it.

Unlike the original FMM method, in the kernel independent method, some additional approximations associated with M2M, M2L and L2L are introduced. In the original analytic method, approximations are only introduced by the S2M (source to multipole) and L2T (local to target) operators. Error analysis and time complexity of KIFMM are beyond the scope of this paper and so for more details the reader may refer to Ying et al. (2004).

3 Core contributions

3.1 Agent behavior

This section details the integration of the (Olfati and Reza 2006)'s flocking model within a kernel-independent FMM. It also describes the related mathematical issues that prevent (at least limit) the use of FMM approach.

As mentioned in Sect. 2.1, the behavior of an agent in our simulation is driven by a control input that consists of three different terms, as described below

$$u_i = u_i^{\alpha} + u_i^{\beta} + u_i^{\gamma} \tag{2}$$

The primary objective of an agent in a flock is to form a lattice-type structure with its neighboring agents. In such a configuration, each agent is at equal distance from all nearby flockmates. The term u_i^{α} aims at satisfying this objective and it is defined as follows:

$$u_i^{\alpha} = c_1^{\alpha} \sum_{j \in N_i^{\alpha}} \phi_{\alpha}\left(\left\|q_j - q_i\right\|_{\sigma}\right) n_{ij} + c_2^{\alpha} \sum_{j \in N_i^{\alpha}} a_{ij}\left(q\right)\left(p_j - p_i\right) \tag{3}$$

where $n_{ij} = q_j - q_i / \sqrt{1 + \varepsilon \left\|q_j - q_i\right\|^2}$ is a vector along the line connecting q_i to q_j and $\varepsilon \in (0, 1)$ is a fixed parameter of the σ-norm. The first term in the definition of u_i^{α} is a *gradient-based* term and the second term is a *velocity consensus* term that acts as a damping force. To see the definitions of various functions used in u_i^{α}, please refer to (Olfati and Reza 2006).

To run a simulation using this flocking algorithm, one needs to compute the control input for all agents in each cycle. Here, the u_i^{α} term is the main bottleneck in computing this control input because if there are N simulated agents, then $O(N^2)$ agent-to-agent interactions should be considered in the worst case. To speed up the computation of the control input, an FMM-like algorithm will be used to calculate u_i^{α}.

In this context, using the original FMM would lead to a set of mathematical issues. Indeed, to apply it, we must be able to define kernel expansions. Let's define $\Phi\left(q_j, q_i\right)$ the considered kernel as

$$\Phi\left(q_j, q_i\right) = \phi_{\alpha}\left(\left\|q_j - q_i\right\|_{\sigma}\right) n_{ij} \tag{4}$$

Finding a factorization of the kernel function Φ defined in Eq. (4) requires its approximation as a sum of series like this:

$$\begin{aligned} \Phi\left(q_j, q_i\right) &= \sum_{m=0}^{\infty} A_m\left(q_i, q_*\right) \cdot F_m\left(q_j - q_*\right) \\ &= \sum_{m=0}^{p-1} A_m\left(q_i, q_*\right) \cdot F_m\left(q_j - q_*\right) + Error\left(p, q_i, q_j\right) \end{aligned}$$

The function $A_m\left(x_i, x_*\right)$ defines expansion coefficients and the function $F_m\left(y_j - x_*\right)$ is the basis function. Here, the important thing is that each term of the above series is a multiplication of two functions A and F such that A is a function of x_i and F is a function of y_j. So, in this expansion we have factorized the function F. It is important to notice that x_* is any point other than x_i and is called the center of expansion.

The dependency of the FMM implementation on analytic expansions of the kernel, makes implementation of efficient and accurate FMM somewhat tedious (Ying et al. 2003; Greengard and Huang 2002; Popov and Power 2001; Yoshida et al. 2001; Yuhong et al. 1998). Another difficulty related to FMM implementation is to devise work-efficient multipole-to-local translation schemes in three dimensions. For example, Ying et al. (2003) have claimed that it may take more than ten years to obtain such a scheme for the Laplace-based FMM scheme.

To avoid the above problems, a multilevel adaptive kernel-independent FMM (MLK-IFMM) was adopted to compute u_i^{α} for all agents at each simulation cycle. The other terms

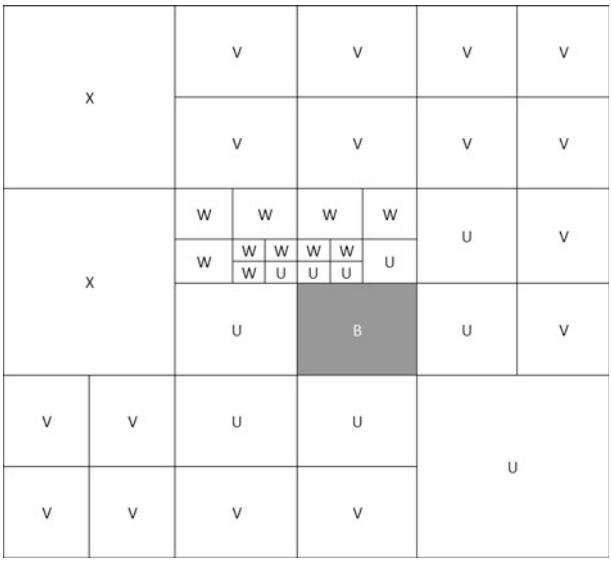

Fig. 4 Lists L_B^U, L_B^V, L_B^W and L_B^X for box B

in the control input are computed in the same manner as presented in Olfati and Reza 2006. As the results shows, this will lead to a much more efficient algorithm.

3.2 The complete algorithm

Before describing the complete algorithm, we need to define different lists associated to each box B in the adaptive tree used in the kernel-independent FMM algorithm. These lists contain all boxes whose contribution needs to be processed by B itself when we use an adaptive way to partition the computational domain. Contributions from more distant boxes are considered by B's ancestors using L2L translations. Each list is defined as below. Figure 4 provides the corresponding spatial representation.

- For a leaf box, L_B^U contains B and its adjacent boxes. If B is not a leaf box, L_B^U will be empty. Since the boxes in L_B^U are not well separated from B, their contribution on B should be computed directly.
- L_B^V contains the set of the children of the neighbors of the parent of B, which are not adjacent to B. The interaction from a box $v \in L_B^V$ to B is computed using M2L translations since V and B are well separated.
- For a leaf box, L_B^W includes all the descendants of B's neighbors, which are not adjacent to B but their parents are adjacent to B. Again, If B is not a leaf box, L_B^W will be empty. Since B is in the far range of $w \in L_B^w$, the contribution from W to B is evaluated directly using upward equivalent density.
- Finally, if box B is in the list L_A^W, then L_B^X contains box A. The contribution from A to B is evaluated directly using downward equivalent density since B is in the near range of A. For each simulation cycle, we use the kernel-independent method to compute u_i^α for all agents. KIFMM computes the potential on every target point; Here, the potential is the term and target points are the agents. The corresponding algorithm is provided in

Listing 1. Assuming N is the total number of agents, m is the number of levels in the tree and q is the clustering size.

Listing 1: Multilevel Adaptive Kernel-Independent Algorithm

```
ASSUME
  N is the total number of agents
  q is the maximum number of agents allowed in a leaf box
STEP 1: SPACE PARTITIONING
  Partition the computational domain recursively until
  there is no more than q agents in each partition
STEP 2: UPWARD PASS
  for each leaf box B do
    compute upward surface from source points in B
  end for
  for each non-leaf box B in post-order traversal of the tree do
    compute upward surface from child boxes of B
  end for
STEP 3: DOWNWARD PASS
  for each non-root box B in preorder traversal of the tree do
    compute downward surface by computing:
      the contribution from each box v ∈ L_B^V to B using M2L translations
      the contribution from agents in each box x ∈ L_B^X
      the contribution of parent box using L2L translation
  end for
STEP 4: FINAL SUMMATION
  for each leaf box B of the tree do
    for each agent A in box A do
      compute the far field interaction on A (contribution from downward surface to A)
      compute the near field interaction on A (contribution from all boxes in L_B^U to A)
    end for
  end for
```

After executing this algorithm in each cycle during simulation, we have all agent-to-agent interactions in the computational domain. To compute the control input for each agent, we need to compute the contributions from all nearby obstacles and virtual leader to each agent. So, the complete algorithm is presented in Listing 2.

Listing 2: Algorithm to compute Agent Control Input

```
for each agent i do
  compute u_i^α (1 ≤ i ≤ N) using MLKIFMM
end for
for each agent i do
  compute agent-to-obstacles interaction u_i^β
  compute agent-to-leader interaction u_i^γ
  compute u_i = u_i^α + u_i^β + u_i^γ
end for
```

4 Experimental results and discussion

In this section, we present a sensitivity analysis of the proposed method and a performance comparison between our MLKIFMM-based implementation and the direct method. By direct

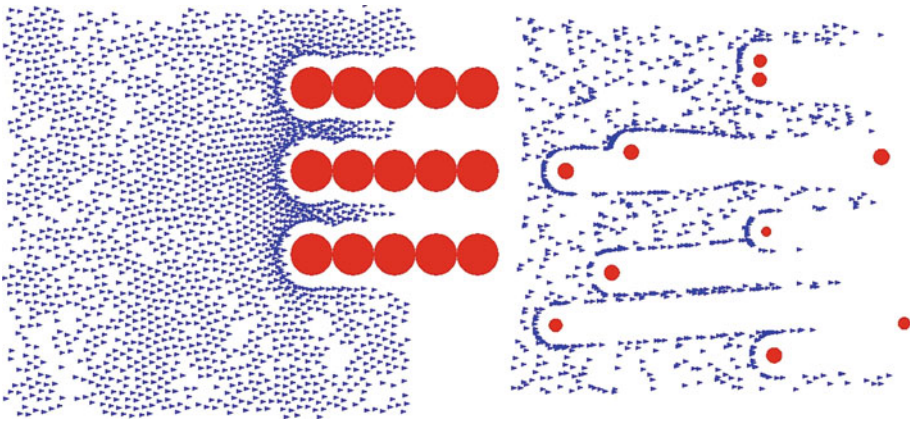

Fig. 5 Snapshots of developed flocking dynamical system

method, we refer to the direct implementation of Olfati and Reza (2006) flocking algorithm. Experiments were performed in a 2D environment, so the hierarchical structure supporting MLKIFMM is a quadtree. A first series of tests were conducted with various environmental settings (random obstacle distributions, corridors, etc) as described in Fig. 5. It is shown that numerical results are not influenced by the environmental configuration. The thorough analysis of the proposed method and its comparison with the direct method were then performed on a 2D environment with a random distribution of obstacles. All corresponding numerical results of the tests are presented in Table 1.

All experiments were conducted on a Dell Precision T7400 configured with two 2.66 GHz Quad-Core Intel Xeon processors and 8 GB RAM running a Windows XP Professional x64 Edition (version 2003 SP2). Our application was developed in Java and the java VM used to execute the tests was configured with one CPU core and 1.6GB memory.

4.1 Effects of agents' population size upon computational performances

The goal of this experiment is to analyze the impact of the growth of the agents' population size on the computational performances. In this experiment, the error bound p of the MLKIFMM is fixed to 12 and also we have used a fixed clustering size equal to 100 in the quadtree. That is, each box in the tree can contain at most 100 source points. In other words, if we have more than 100 agents in a quadtree node, we split it into sub-nodes. More details on this point will be provided in the following section.

Figure 6 depicts the evolution of computational performances (in millisecond) of the two compared methods. These results clearly reveal that the computational cost of the direct method is quadratic while MLKIFMM follows a linear growth. On average, whatever the chosen error bound, the MLKIFMM is almost 50 times faster than the direct method for a population of 200,000 simulated agents. The gain will be even higher for more numbers of simulated agents.

These results therefore confirm the interest of the FMM method to reduce the overall complexity of a MABS model as well as reduce the associated computational costs. However reducing computational costs of a simulation is not sufficient to validate usability of the proposed method and its potential generalization. We must now show that the optimized simulation model remains consistent with the initial model (direct method) and simulated

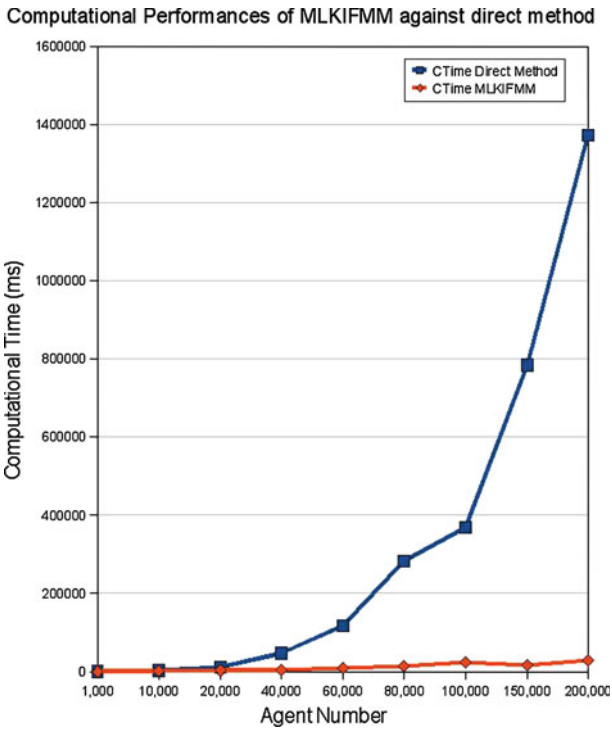

Fig. 6 Comparisons between direct method and MLKIFMM computational performances

phenomenon (flocking dynamical systems). In this context, our approach consists in showing that the optimized model preserves an acceptable level of accuracy in comparison with the direct method. This is precisely the purpose of the tests described in Sect. 4.3.

4.2 Effects of MLKIFMM parameters on computational performances

The goal of this experiment is to study the influence of MLKIFMM parameters on its computational performances. There are two main application-independent parameters in FMM method that directly influence its computational costs: the error bound p in the expansions as well as the clustering size in the quadtree. Our study focuses on these general parameters with a view to develop a generic approach independent of the considered application.

A first series of tests was conducted to study the influence of the clustering size in the quadtree. The results show that this parameter critically impacts the performances of the proposed MLKIFMM method. In brief, use of a technique for dynamically determining the optimal clustering size is so expensive that it makes the method unusable simply because its computational performances are then lower in most cases (especially for population size less than 50 000 agents) than those of the direct method. Therefore, we opted for a fixed and pre-determined threshold. However, a comprehensive study of the influence of the clustering size parameter in the quadtree will not be presented in this article because we are still working on the design of a new efficient method to determine an optimal size. Future works will also attempt to extend the proposed method by using another types of spatial data structures and partition heuristics like icoseptree (Shagam 2003), kd-tree and so on. In the rest of our

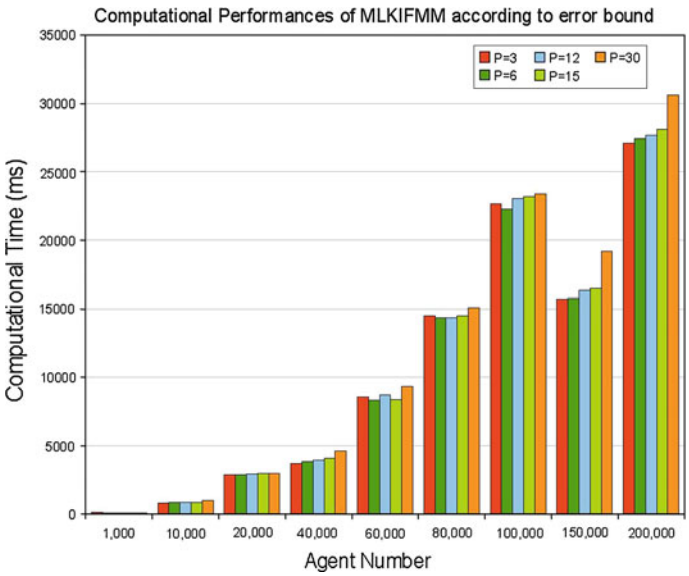

Fig. 7 Computational performances of MLKIFMM according to its error bound

study, the clustering size has thus been set to 100, value offering acceptable computational performances and level of granularity.

Regarding the error bound parameter p, an analysis was conducted to study its impact on the performances of the proposed method, all the associated results are presented in Fig. 7. These results suggest that this parameter has little influence on the performance of the MLKIFMM as while as p remains small. For values between 3 and 15, the results are marginally impacted.

4.3 Effects of MLKIFMM error bound on the experimental error

Within MLKIFMM, the chosen error bound p in the expansions directly determines the level of accuracy. This last experiments aims at validating influence of this parameter by studying its impact on the real experimental error. This latter is determined at each cycle of the simulation for each agent by comparing its potential computed by the direct method and that obtained by the MLKIFMM. As the chosen case study is based on a 2D environment, the potential of an agent is defined in a Cartesian coordinate system. Experimental error is then decomposed into two parts: the error on the X axis and the one on the Y axis. Results of the impact analysis of p on the X axis are shown in Fig. 8, knowing that its impact on Y follows exactly the same progression. This is confirmed by the numerical results presented in Table 1. These results attempt to show that the error bound p does not significantly affect accuracy of the simulation. Or this influence is simply too small to be measured by the chosen experimental protocol.

4.4 Summary

The previous experimental results confirm applicability and benefits of MLKIFMM approach for large-scale multi-agent based simulations. This approach is effective in reducing signifi-

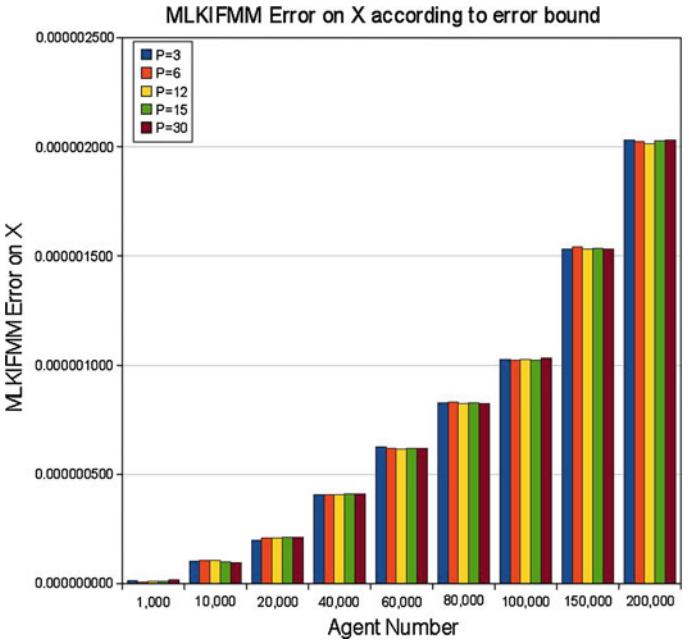

Fig. 8 Evolution on MLKIFMM Error on X in a 2D space according to its error bound

cantly computational costs of a MABS. The low impact of the error bound p the computational costs allows maintaining maximum precision without affecting the overall performances of the application. However, the obtained performances on a relatively simple application such as flocking dynamical system are still very far from desired real-time or quasi real-time performances (simple in comparison to a complete pedestrian simulation). Future works will therefore focus on finding new ways to further reduce computational costs. One approach would be to manage a dynamic accuracy of the simulation model. On this point, a generic approach only based on the MLKIFMM parameters seems to be compromised according to the obtained results. Indeed the error bound p does not significantly impact computational costs. The main perspective remains in the optimization of the spatial data structures and associated partition heuristics, and also in determination of optimal clustering size for these structures.

5 Conclusion and future works

In this work, a derivative of fast multipole method has been successfully applied to a multi-agent based simulation and in particular to the simulation of flocking dynamical systems. Experiments were conducted to study performances and usability of the proposed model. An analytic FMM could be used but it leads to a series of mathematical issues. Moreover, such an approach would reduce generality of the proposed approach since it would be problem-dependent. To avoid these issues, a kernel-independent FMM is used, this method requires only kernel evaluations, and it does not sacrifice efficiency of the original algorithm. Like analytic FMM algorithm, the method has $O(\mathrm{N})$ asymptotic complexity and works well for non-uniform particle distributions. Experimental results confirm the performances of the proposed approach and its applicability to large-scale multi-agent based simulation.

Future works will extend the proposed approach to develop a multilevel simulation model. Multilevel simulation may be regarded as a generalization of micro-macro perspectives in which a range of levels of detail coexist within the same simulation and where transitions between these levels can be performed dynamically (Nicolas et al. 2007; Gaud et al. 2008). To design such a model, our approach will be based on the definition of a dynamic indicator based on the desired simulation accuracy and the volume of available computational resources. This indicator will be used as a mechanism to establish a compromise between accuracy and efficiency. The lattice-type structure in the proposed algorithm has interesting properties because it helps maintain an orderly structure between agents. Thanks to this order, the proposed indicator may determine if it is possible to compute the potential for every agent in a box or if we just compute the potential of a representative agent in the box and reuse it for each agent near to that representative. This decision may be made for the agents in a leaf box or agents in a box in higher levels of the tree. In this way, whenever we have sufficient computational resources, normal FMM computations may be used. But in other cases, it will be possible to save a lot of computations and hence to reduce the complexity by grouping agents and computing just one potential and applying it to the entire group. This principle will be mainly supported by the hierarchical structure used in the FMM.

Appendix: Experimental numerical results

See Table 1.

Table 1 Computational results of MLKIFMM and direct method (clustering size = 100)

Agent number	P	Comp. time of direct method (ms)	Comp. time of MLKIFMM (ms)	Error on X	Error on Y	Performance ratio
1,000	3	88.4	104.33	0.000000013	0.000000009	0.85
10,000	3	2685.4	814.6	0.000000105	0.000000103	3.3
20,000	3	11615.73	2875.07	0.000000200	0.000000203	4.04
40,000	3	47649.07	3683.27	0.000000407	0.000000413	12.94
60,000	3	119847	8568.73	0.000000626	0.000000622	13.99
80,000	3	271055.27	14520.73	0.000000828	0.000000825	18.67
100,000	3	361078.27	22664.47	0.000001030	0.000001035	15.93
150,000	3	797142.73	15697.87	0.000001532	0.000001529	50.78
200,000	3	1410451.2	27095.8	0.000002032	0.000002021	52.05
			Mean	0.000000752	0.000000751	
			Std dev	0.000000683	0.000000681	
1,000	6	66.67	91.67	0.000000009	0.000000012	0.73
10,000	6	2690.6	823.07	0.000000106	0.000000101	3.27
20,000	6	11602.93	2893.87	0.000000209	0.000000212	4.01
40,000	6	45750	3844.8	0.000000408	0.000000412	11.9
60,000	6	144652.2	8320.67	0.000000621	0.000000620	17.38
80,000	6	259705.33	14368.67	0.000000832	0.000000828	18.07
100,000	6	418951.13	22247.87	0.000001026	0.000001029	18.83

Table 1 continued

Agent number	*P*	Comp. time of direct method (ms)	Comp. time of MLKIFMM (ms)	Error on X	Error on Y	Performance ratio
150,000	6	796065.6	15734.33	0.000001542	0.000001532	50.59
200,000	6	1367693.87	27418.6	0.000002025	0.000002017	49.88
			Mean	0.000000753	0.000000751	
			Std dev	0.000000682	0.000000679	
1,000	12	26.07	62.47	0.000000011	0.000000006	0.42
10,000	12	2692.73	837.47	0.000000105	0.000000106	3.22
20,000	12	11770.87	2911.4	0.000000208	0.000000203	4.04
40,000	12	47019.73	3952.13	0.000000409	0.000000412	11.9
60,000	12	116636.47	8696.8	0.000000617	0.000000615	13.41
80,000	12	282653.2	14332.2	0.000000827	0.000000823	19.72
100,000	12	368320.93	23017.53	0.000001027	0.000001031	16
150,000	12	784110.53	16335.33	0.000001531	0.000001530	48
200,000	12	1372827.13	27660.53	0.000002015	0.000002023	49.63
			Mean	0.000000750	0.000000750	
			Std dev	0.000000679	0.000000681	
1,000	15	26.07	62.47	0.000000011	0.000000006	0.42
10,000	15	2672.87	860.47	0.000000101	0.000000102	3.11
20,000	15	11604.07	2965.6	0.000000212	0.000000205	3.91
40,000	15	45239.53	4055.27	0.000000412	0.000000412	11.16
60,000	15	178537.47	8347.93	0.000000621	0.000000613	21.39
80,000	15	287644.73	14518.67	0.000000831	0.000000827	19.81
100,000	15	347679.33	23164.4	0.000001026	0.000001024	15.01
150,000	15	784980.27	16491.6	0.000001534	0.000001530	47.6
200,000	15	1398038.53	28109.33	0.000002029	0.000002027	49.74
			Mean	0.000000753	0.000000750	
			Std dev	0.000000682	0.000000682	
1,000	30	25.07	68.73	0.000000016	0.000000009	0.36
10,000	30	2667.67	1004.13	0.000000098	0.000000101	2.66
20,000	30	11604.07	2965.6	0.000000212	0.000000205	3.91
40,000	30	45664.53	4611.53	0.000000412	0.000000412	9.90
60,000	30	118276.13	9358.33	0.000000620	0.000000623	12.64
80,000	30	290969.73	15056.33	0.000000825	0.000000826	19.33
100,000	30	367515.67	23389.67	0.000001034	0.000001041	15.71
150,000	30	761662.47	19202.13	0.000001533	0.000001533	39.76
200,000	30	1382313.4	30590.67	0.000002031	0.000002025	45.19
			Mean	0.000000754	0.000000753	
			Std dev	0.000000682	0.000000682	

References

Darve E (2000) Error analysis and asymptotic complexity. SIAM J Numer Anal 38(1):98–128

Davidsson P (2000) Multi agent based simulation: beyond social simulation. Multi agent based simulation, LNCS series, vol 1979. Springer, Berlin

Dongarra JJ, Sullivan F (2000) The top 10 algorithms. Comput Sci Eng 2:22–23

Duraiswami R, Gumerov NA (2005) Lecture notes on the fast multipole method for copurse AMSC698R. University of Maryland, Maryland

Gaud N, Galland S, Koukam A (2008) Towards a multilevel simulation approach based on holonic multiagent systems, s.n. In: 10th international conference on computer modelling and simulation (EUROSIM), Cambridge, UK, pp 180–185

Greengard L, Huang J (2002) A new version of the fast multipole method for screened Coulomb interactions in three dimensions. J Comput Phys 180:642–658

Greengard L, Rokhlin V (1987) A fast algorithm for particle simulations, 2. Academic Press Professional, Inc., San Diego. J Comput Phys 73:325–348, 0021-9991

Gumerov R, Duraiswami NA (2005) Fast multipole methods for the Helmholtz equation in three dimensions. Elsevier, Oxford

Gumerov NA, Duraiswami R, Borovikov EA (2003) Data structures, optimal choice of parameters, and complexity results for generalized multilevel fast multipole methods in d dimensions. s.l. University of Maryland Institute for advansed computer studies

Helbing D, Farkas I, Vicsek T (2000) Simulating dynamical features of escape panic. Nature 407:487–490

Karaboga D, Akay B (2009) A survey: algorithms simulating bee swarm intelligence. Artif Intell Rev 31(1):61–85

Khatib O (1986) Real-time obstacle avoidance for manipulators and mobile robots. Int J Robot Res 5(1):90–98

Kress R (1999) Linear integral equations, applied mathematical sciences. Springer, Berlin

Mogilner A, Edelstein-Keshet L (1999) A non-local model for a swarm. J Math Biol 38:534–570

Nicolas G et al (2007) Holonic multiagent multilevel simulation: application to real-time pedestrians simulation in urban environment. In: Hyderabad, India: s.n., Twentieth international joint conference on artificial intelligence, IJCAI'07, pp 1275–1280

Olfati Saber R (2006) Flocking for multi-agent dynamic systems: algorithms and theory. IEEE Trans Automat Control 51:401–420

Olfati Saber R, Murray RM (2003) Consensus protocols for networks of dynamic agents. Am Control 951–956

Olfati Saber R, Murray M (2003) Flocking with obstacle avoidance: cooperation with limited communication in mobile networks. s.l.: IEEE. IEEE conference on decision and control, vol 5, pp 2022–2028

Popov V, Power H (2001) An O(N) Taylor sereis multipole boundary element method for three-dimensional elasticity problems. Eng Anal Bound Elem 25:7–18

Reynolds Craig W (1987) Flocks, Herds, and Schools: {A} distributed behavioral model. Comput Graph 21(4):25–34

Saber RO, Murray RM (2004) Consensus problems in networks of agents with switching topology and time-delays. IEEE 49:1520–1533

Samet H (1984) The quadtree and related hierarchical data structures. ACM Comput Surv 16(2):187–260

Shagam J (2003) Dynamic Spatial partitioning for real-time visibility determination. PhD thesis, Department of Computer Science, New Mexico State University

Shimoyama N et al (1996) Collective motion in a system of motile elements. Phys Rev Lett 76(20):3870–3873

Tanner HG, Jadbabaie A, Pappas GJ (2003) Stable flocking of mobile agents, part II: dynamic topology. In: 42nd IEEE conference on decision and control, pp 2010–2015

Toner J, Tu Y (1998) Flocks, herds, and schools: a quantitative theory of flocking. Phys Rev E 58:4828–4858

VanDyke Parunak H, Savit R, Riolo RL (1998) Agent-based modeling vs. equation-based modeling: a case and study and users' guide. In: Sichman JS, Conte R, Gilbert N (eds) Multi-agent systems and agent-based simulation (MABS). Springer, Paris pp 10–26

Vicsek T, Czirook A et al (1995) Novel type of phase transition in a system of self-deriven particles. Phys Rev Lett 75:1226–1229

Yarvin N, Rokhlin V (1999) An improved fast multipole algorithm for potential fields on the line. SIAM J Numer Anal 629–666

Yoshida K, Nishimura N, Kobayashi S (2001) Application of fast multipole Galerkin boundary integral equation method to elastostatic crack problems in 3D. Int J Numer Methods Eng 50(3):525–547

Ying L, Boris G, Zorin D (2003) A kernel-independent fast multipole algorithm. Courant Institute, New York University. Technical Report TR2003-839

Ying L, Biros G, Zorin D (2004) A kernel-independent adaptive fast multipole algorithm in two and three dimensions. J Comput Phys 196(2):591–626

Yuhong F et al (1998) A fast solution method for three-dimensional many-particle problems of linear elasticity. Int J Numer Methods Eng 42(7):1215–1229
Zhou J, Yu W, Wu X et al (2009) Flocking of multi-agent dynamical systems based on pseudo-leader mechanism, eprint arXiv:0905.1037. Cornell University Library