Actor-based Scalable Simulation of N-body Problem[⋆]

Kamil Szarek, Wojciech Turek[1][0000−0002−1853−2529], Łukasz Bratek[2][0000−0002−0272−8236], Marek Kisiel-Dorohinicki[1][0000−0002−8459−1877], Aleksander Byrski[1][0000−0001−6317−7012]

[1] AGH University of Krakow
Al. Mickiewicza 30, 30-059 Krakow, Poland
[2] Cracow University of Technology
ul. Warszawska 24, 31-155 Krakow, Poland
szarekkam@gmail.com,wojciech.turek@agh.edu.pl
lukasz.bratek@pk.edu.pl,{doroh,olekb}@agh.edu.pl

Abstract. Efficient solutions of the N-body problem make it possible to conduct large-scale physical research on the rules governing our universe. Vast amount of communication needed in order to make each body acquainted with the information on position of other bodies renders the accurate solutions very quickly inefficient and unreasonable. Many approximate approaches have been proposed, and the one introduced in this paper relies on actor-based concurrency, making the whole design and implementation significantly easier than using, e.g. MPI. In addition to presenting three methods, we provide the reader with tangible preliminary results that pave the way for future development of the constructed simulation system.

Keywords: actor-based concurrency · N-body problem · agent-based computing and simulation.

1 Introduction

The classical N-body problem consists in determining the motion of N objects, each with a specific mass, position, and initial velocity, and all interacting with each other by gravity. The very concept of the N-body problem was presented by Isaac Newton in his work *Philosophiae Naturalis Principia Mathematica*, in which he formulated the laws underlying classical mechanics. However, an analytical solution of the problem is possible only for a two-body system and for some small systems with appropriately defined initial conditions. A similar problem to the purely gravitational one can be considered for point electric charges, provided that magnetic forces and the emission of radiation accompanying the

[⋆] This research was partially supported by funds from the Polish Ministry of Education and Science assigned to AGH University of Science and Technology (AB, WT, MKD) and Cracow University of Science and Technology (ŁB). This research was supported by PLGrid Infrastructure.

moving and accelerated charges can be neglected. The concept of the N-body problem can be extended to other kinds of interaction, including the interaction of objects with an external background field. There are also more complicated effective interactions, such as those assumed between atoms in molecular dynamics simulations. In any case, irrespective of the particular form of interactions between bodies, for many bodies, a simulation is performed, changing the position of objects with time step, taking into account their interactions. This approach is used in many branches of science: astrophysical analyses (e.g. [25], [15]), validation of models in biophysical simulations (e.g. [5]), etc. The critical element for an effective simulation is the knowledge of the entire system required to calculate the force acting on each body.

To date, methods and techniques developed for performing N-body simulations [3] have different advantages and disadvantages, sacrifice accuracy for efficiency, and utilize very particular solutions (like dedicated hardware). For us, the concept of easy-to-implement, natural and appropriately accurate design and implementation of such a system, leveraging the available High Performance Computing (HPC) infrastructure, is paramount. Therefore, in this paper, we show the background and derive the base for utilizing the actor model to create a program that on the example of the classical N-body problem performs a simulation with limited communication and propagation of information about the bodies in the system to achieve scalability and maintain the accuracy of the results. We treat the actor-based approach as very close to the paradigm of agency, which is an interesting and easy-to-apply idea for modeling and implementation of complex, interacting systems(see, e.g., [10,18]).

The actor model of concurrency has already been successfully used to implement high-scalability simulations (see, e.g. [21]), and its advantages include full asynchronicity of message exchange and simplicity of implementation by focusing on modeling communication and agent algorithm. The Akka tool was used to implement the solution, providing a lightweight implementation of the actor with support for efficient handling of a huge number of instances that work simultaneously.

In this paper we start by giving a proper background; then we describe the most important aspects of the proposed system (with several variants of the implementation of the communication) and we provide the reader with sufficient insight into our experimental result.

2 Existing efficient N-body solutions

The classical N-body problem, as originally stated, aims at determining the position changes over time of N objects, knowing their masses, initial positions and velocities. The objects are moving in a three-dimensional space and are not affected by any external force, except for a mutual gravitational interaction. It is assumed that the objects can be described as material points in space governed by Newton's laws of motion and universal gravitation. The vector of force $\vec{F}_i$

exerted on ith body by all other bodies in the system reads

$$\vec{F}_i = Gm_i \cdot \sum_{j\neq i} \frac{m_j}{r_{ij}^2}\frac{\vec{r}_{ij}}{r_{ij}}, \qquad \vec{r}_{ij} \equiv \vec{r}_j - \vec{r}_i, \quad r_{ij} \equiv |\vec{r}_{ij}|,, \quad i = 1, 2, \ldots, N$$

where $\vec{r}_k$, $k = 1, 2, \ldots, N$ are position vectors (with respective velocity vectors denoted by $\vec{r}_k$), m_k are masses and G is the gravitational constant. To speed up the simulation of largely collisionless systems (like galaxies), the forces can be regularized at short distances. This allows to avoid numerically more demanding computations of very close encounters of concern only in much smaller collisional systems (celestial mechanics, globular clusters). A possible regularisation is to replace r_{ij}^2 with $r_{ij}^2 + \epsilon^2$ in all denominators, where ϵ is a softening length parameter much smaller than the characteristic mean distance between bodies.

In $d = 2, 3$ spatial dimensions, the resulting gravitational dynamics is described by a system of $2dN$ first-order equations

$$\begin{aligned} \frac{\mathrm{d}\vec{v}_i}{\mathrm{d}t} &= \sum_{j\neq i} Gm_j \frac{\vec{r}_j - \vec{r}_i}{|\vec{r}_j - \vec{r}_i|^3}, \\ \frac{\mathrm{d}\vec{r}_i}{\mathrm{d}t} &= \vec{v}_i, \end{aligned} \qquad i = 1, 2, \ldots, N \tag{1}$$

for unknown positions and velocities. Although conceptually simple in origin, this is a highly nonlinear and chaotic system of differential equations. It is a textbook problem to find an exact solution for a two-body system (N=2). For more numerous systems ($N\geqslant 3$) exact finite-form solutions cannot be found with method of first integrals, except for systems with high symmetry (e.g, identical masses constrained at the vertices of a regular polygon, or systems with more complicated periodic orbits offered by N-body choreography strategies [8,22]). There were attempts to rigorously analyze methods using series expansions [19,1,16,2].

When finding numerical solutions, useful are conserved quantities that can play the role of control parameters. There are $(d + 1)(d + 2)/2$ such constants following from space-time symmetries of the considered system (1) which is isolated as a whole: energy E, d components of linear momentum $\vec{P}$, $d(d - 1)/2$ of angular momentum $\vec{L}$ and d components of the center-of-mass motion $\vec{K}$. Upon setting the initial data, all components of $\vec{P}$ and $\vec{K}$ can be set to zero by transforming the initial state to the center of mass frame (this choice of reference frame prevents unbound increase of position components due to motion of the system as a whole during the integration process). Furthermore, for $d = 3$ two components of $\vec{L}$ in this frame can be set to zero by performing a suitable rotation of the system about the center of mass. In this new inertial frame, the new conserved E and the magnitude J of the new $\vec{L}$ will in general be non-zero. The dimensional constants provide a characteristic length scale $J/\sqrt{EM}$ and a characteristic time scale J/E (the scale of the system's rotation period), which together with the total mass M provide a complete set of independent mechanical units characterizing that system. With the help of these or other convenient triad of reference units, it is computationally advantageous that the system of equations be finally rewritten in a rescaled form involving only non-dimensional quantities.

For a small number of particles, the system of equations (1) can be solved with a high degree of precision using the Runge-Kutta methods with adaptive step control [9]. This requires the forces to be computed several times per single time step of the simulation. For a large number of particles, one of the obvious difficulties that arise for a gravitational system characterised by long-ranged forces that cannot be neglected between distant objects is the need to trace and compute all of the $N(N-1)/2$ elementary two-body interactions, and this problem cannot be simply avoided. There are dedicated integration methods for simulations with large numbers of particles, however, they are plagued with at least quadratic complexity, rendering them completely useless when 10^5 or more bodies are considered. Many algorithms have been created to reduce complexity while sacrificing accuracy. They very often use dedicated data structures for the description of the space and bodies located there, where compromises in the data contained affect the nature of algorithms that compute the influences [3].

On the side of the integration method used, the number of times all forces have to be recomputed per a single time step of the simulation can be minimised at the cost of reducing the accuracy of the integrator. A possible choice for updating/advancing the phase space data at consecutive time steps is the Störmer-Verlet mid point method approach [23].

Essential reduction of the computational time, however, can be achieved only on the side of the algorithm for determining the forces. In the first method applied in the case of large simulations ($N \sim 10^5$), namely Particle-Mesh method, the space is described as a mesh and the particles are placed inside, determining the density of matter over the mesh. Then the discrete Fourier transform is used to solve the Poisson differential equation for the gravitational potential that depicts the influences based on this mesh. Local forces are then determined simply from local gradients of the potential. This method is easy for parallelization, however, it may be inaccurate for a smaller number of bodies because it trades accuracy for efficiency [3].

Efficient reduction of the number of force evaluations is achieved by considering clusterisation of the system to smaller subsystems of nearby bodies. Then the force due to the totality of bodies belonging to one subsystem as exerted onto a body from another subsystem can be approximated by a truncated multipole expansion if the distance between the subsystems meets some criteria (i.e. is large enough). An important and widely used method in this respect is the Barnes-Hut approach [4], which uses a tree structure to describe the space. In the leaves, the data of the single bodies are contained, where the nodes comprise information about the lower-level objects. This setting allows the algorithm not to visit every leave (body), but stop at the nodes that are sufficiently far from the current object, using information about the center of the mass, for computing the force. This makes the complexity equal to $O(n\log(n))$, which seems very promising; however, when bodies are distributed among different computing machines, communication can affect the whole complexity to a large extent. Particularly efficient is the Fast Multipole Method, reaching linear complexity for certain accuracy. The accuracy itself is one of the parameters of the algorithm; naturally increasing the accuracy, we lose efficiency. The method was proposed by Greengarg and Rokhlin in 1988 and it divides the space into squares (or cubes in 3D). Instead of building a dynamic tree, a strict structure is imposed, where each level divides the squares (cubes) into four (eight) equal parts [7].

There exist other methods, e.g. hybrid ones (putting together accurate and less accurate methods and balancing the efficiency-accuracy) [3], or the methods focused on leveraging dedicated hardware [13,14] using specially designed modules for computing the gravitational influences. Instead of using dedicated hardware, general purpose one can be adapted, like GPGPU, e.g. one of existing methods put together MPI, OpenMPI and CUDA reaching good speedup and efficiency; however, the limited GPU memory turned out to be a major problem [6]. This project was later developed into [24].

An interesting implementation of the Barnes-Hut algorithm was used in [20], where the authors used the Haskell functional language based on its ability to efficiently manage threads. Another very interesting and accurate implementation of the N-body simulation is [12], which uses the Ruby language and focuses mainly on assessing the accuracy and error of the simulation; however, it avoids delving into matters related to parallelization and HPC.

Each of the methods mentioned here uses selected particular tools and techniques for design and implementation of the N-body simulation. To our knowledge, the existing actor model of concurrency has a great potential in the implementation of scalable HPC grade systems [21], and therefore we would like to focus our proposed approach on this model, providing potential users with a prototype of scalable and reliable implementation. Moreover, thinking about implementations in large scale, we also would like to aggregate certain parts of the bodies (using dedicated, managing agents) and propose first balancing techniques, in order to approach efficient scalability utilizing large, existing HPC infrastructures.

3 The proposed actor-based models

There are numerous ways to use the actor-based model in the computational tasks considered. At least three of them are worth discussing, as they provide significantly different results, efficiency, and scalability. The three approaches can be briefly characterized as follows:

1. Problem-driven actor model, where each actor represents an entity present in the modeled system. In our case, an actor will be equated with a single body in the simulation. The actors will communicate with each other to exchange information about their state.
2. Computation-driven actor model, where an actor is a computing unit, and the number of actors is more related to the computation efficiency that the

model features. Each actor shall be assigned a selected part of the computational problem, which is a set of modeled bodies. The communication will be analogous to the previous case.

3. Limited-communication actor model, where the actors will communicate only with neighbors, other actors responsible for bodies in certain proximity. The information from distant bodies will be forwarded.

The following model descriptions will omit the implementation details regarding initialization, monitoring, and results collection, focusing on the problem model, its mapping to the actor model, and its update algorithm.

3.1 Problem-driven actor model

The most direct way to apply the actor model to the modeling problem is to identify autonomous entities in the modeled system and to assign each entity with a dedicated actor. In the task considered, each body can be represented by a single actor, which knows its *state*: mass, location, and velocity values. The actor is responsible for updating the state, which requires knowledge of other actors' state. Knowledge is exchanged between actors using messages sent directly between all pairs of actors.

The algorithm of a single-body actor is presented in Listing 1.

Algorithm 1 Single-body actor update loop

```
1: $myState = initialState$
2: $allOtherActors = initialDataWithAddresses$
3: **while** $stopCondition == false$ **do**
4:     **for** $otherActor \in allOtherActors$ **do**
5:         $send(otherActor.address, myState)$
6:     **end for**
7:     **for** $allOtherActors$ **do**
8:         $\{senderAddress, senderState\} = receive()$
9:         $allOtherActors[senderAddress].state = senderState$
10:    **end for**
11:    $force = \vec{0}$
12:    **for** $otherActor \in allOtherActors$ **do**
13:        $force = force + computeForce(otherActor.state, myState)$
14:    **end for**
15:    $myState = computeState(myState, force, \Delta t)$
16: **end while**
```

Initially, each actor has to be provided with the addresses of all the other actors and with the initial state of the represented body. In each step of the simulation, an actor sends its state to all other actors, collects states from all other actors, and computes the resultant force and its state after Δt. The results of the algorithm are equivalent to the accurate sequential solution.

The computational complexity of the processing performed by a single actor is linearly dependent on the number of all bodies. Additionally, the number of messages sent and received by each actor depends linearly on the size of the simulated system. Therefore, the overall complexity in a single simulation step of a single actor is $\Theta(b)$ and the whole actor system is $\Theta(b^2)$ for b bodies in the simulation. Running the actor system in parallel, it is possible to achieve a computation time proportional to $(b^2)/c$ for c computing cores. It is clearly visible that proper scalability cannot be expected in this case.

3.2 Computation-driven actor model

A more efficient and scalable approach to actor-based modeling of the considered problem is based on the concept of using actors to partition the computation instead of directly representing modeled entities. In the second model, each actor is responsible for processing a set of bodies. Sets are defined at the beginning of the simulation using a clustering algorithm (k-means in our implementation). The communication schema is similar to that before – all actors exchange messages with all other actors in each simulation step. The messages, however, contain only information about the location of the center of mass (CoM) and the mass value of a cluster of bodies. Actors update the state of the assigned bodies using precise data in case of own bodies and the center of mass for bodies in remote clusters. This approach is a common optimization used in the N-body problem, which does not introduce significant errors, provided that the clusters are distant.

The algorithm of a single actor in this case is presented in Listing 2.

Each body-cluster actor is assigned an initial list of bodies that it has to simulate. The optimization based on CoM requires each actor to calculate its CoM at the beginning of each step. The exchange of messages between all pairs of actors remains unchanged. The computation of the change in the state of the assigned bodies requires integrating forces from other assigned bodies and remote clusters $CoMs$.

In a longer simulation, the initial division of bodies between actors may become improper, which can result in significant errors caused by the CoM approximation. If a body approaches another body from a different cluster, its strong mutual impact shall be reduced due to the distance of $CoMs$. Therefore, the bodies migration algorithm has been added to the presented method. It is executed after a fixed number of steps by each of the actors. The actors search for the assigned bodies, which are closer to remote CoM than to the own one. Such bodies are sent to remote actors, which changes their assignment.

In a very long simulation such an approach can lead to imbalance in the number of assigned bodies. This problem can be detected by each actor separately; it is sufficient to compare the current and the initial number of assigned bodies. In such a case, the centralized clustering algorithm has to be executed again.

The total number of actors (a) should be equal to the number of computing cores used $c, c << b$. The number of messages exchanged by each actor is proportional to the total number of actors, a. CoM calculation complexity is

Algorithm 2 Body-cluster actor update loop

```
1: $myBodiesState = initialBodiesState$
2: $allOtherActors = initialDataWithAddresses$
3: while $stopCondition == false$ do
4:     $\{myCoM, myMass\} = calculateCoM(myBodiesState)$
5:     for $otherActor \in allOtherActors$ do
6:         $send(otherActor.address, \{myCoM, myMass\})$
7:     end for
8:     for $allOtherActors$ do
9:         $\{senderAddress, \{senderCoM, senderMass\}\} = receive()$
10:        $allOtherActors[senderAddress].state = \{senderCoM, senderMass\}$
11:    end for
12:    for $myBodyState \in myBodiesState$ do
13:        $force = \vec{0}$
14:        for $otherActor \in allOtherActors$ do
15:            $force = force + computeForce(otherActor.state, myBodyState)$
16:        end for
17:        for $myOtherBodyState \in myBodiesState \setminus \{myBodyState\}$ do
18:            $force = force + computeForce(myOtherBodyState, myBodyState)$
19:        end for
20:        $myBodyState = computeState(myBodyState, force, \delta t)$
21:    end for
22: end while
```

proportional to the number of bodies assigned, which is b/a. Computing the state changes requires $b + (b/a)^2$. So, the overall complexity of a single actor is $\Theta(a + b + (b/a)^2)$. The whole actor system can be executed in proportional time on c cores, assuming $a = c$. Therefore, it can be expected that the computation-driven actor model has a better scalability potential than the problem-driven model. Increasing the size of the simulated system proportionally with the number of cores shall not increase the $(b/a)^2$ element. However, the size of each actor's task grows linearly with the number of actors in the system and the number of bodies. It is caused by the communication schema, where all pairs of actors communicate directly with each other, and the need for updating each assigned body against each remote cluster.

3.3 Limited-communication actor model

In the aforementioned computation-driven actor model, the need to communicate all pairs of actors is definitely the most significant scalability limiter. To address this issue, another mechanism has been added in the third actor model: actors communicate only with "neighbors", which are the actors managing clusters in predefined proximity. In order to preserve the model correctness, the information about the state of non-neighbor clusters has to be forwarded among all actors, which is done using the same messages. Each message contains the most current state of all known clusters.

The algorithm of a single actor in this case is presented in Listing 3.

Algorithm 3 Body-cluster actor update loop

```
$myBodiesState = initialBodiesState$
$allOtherActors = initialDataWithAddresses$
while $stopCondition == false$ do
    $\{myCoM, myMass\} = calculateCoM(myBodiesState)$
    for $otherActor \in allOtherActors$ do
        if $otherActor.isNeighbor$ then
            $send(otherActor.address, \{myCoM, myMass\} \cup allOtherActors)$
        end if
    end for
    for $neighborsCount$ do
        $receivedActors = receive()$
        for $receivedActor \in receivedActors$ do
            if $receivedActor.time > allOtherActors[receivedActor.id].time$ then
                $allOtherActors[receivedActor.id] = receivedActor$
            end if
        end for
    end for
    for $myBodyState \in myBodiesState$ do
        $/*$ body state update same as in Algorithm 2 $*/$
    end for
end while
```

The list of neighbor actors is prepared at the beginning of the simulation. The actors can adjust automatically when they detect a change in distance between the clusters. Each actor sends all available data to all neighbors. After receiving the data, an actor checks if it contains information that is newer than already possessed. As a result, the computation of bodies state changes is based on relatively older information for distant bodies, which is another modification to the model with a minor impact on final results.

The number of messages is significantly limited in this case. Each actor sends and receives a constant number of messages, no matter how numerous the actor system is. Nevertheless, the total volume of the messages is even bigger than in the previous approach and the theoretical processing complexity remains the same. However, the messaging layer is less burdened, which typically has a significant impact on final efficiency.

4 Experimental evaluation

The expected outcome of applying the actor model for the considered simulation task is to achieve a significant performance gain compared to the sequential version. The gain shall result from parallel execution, which is straightforward for actor-based implementation, and from the applied optimizations to the model, which offer better computational complexity. However, optimizations may hinder the accuracy of the model. Therefore, the evaluation will focus on these two aspects: the correctness and the performance.

Implementation has been done using the Akka actor framework for the Scala programming language [17]. All tests were repeated 10 times. In the performance tests 50-100 steps of the simulation were done to measure an average step time.

The experiments were carried out on the Prometheus computing cluster, hosted by the ACK Cyfronet AGH [3]. Prometheus offers 53604 computing cores provided by HP Apollo 8000 servers (24 cores each), which are connected with the 56 Gb/s InfiniBand network. Such resources made it possible to test the system performance and scalability on hundreds of computing cores.

4.1 Correctness evaluation

One of the most common methods for verifying the correctness of the N-body simulation is to monitor the total energy in the simulated system of bodies. With no external forces in the system, the energy should remain constant. In the experiment carried out, 200 bodies were divided into 20 groups, each having 8-20 bodies. The bodies in each group were located at the edges of a regular polygon, which is a theoretically stable figure. The distances between the bodies in groups were significantly smaller than between the groups. The results of the correctness evaluation are presented in Figure 1.

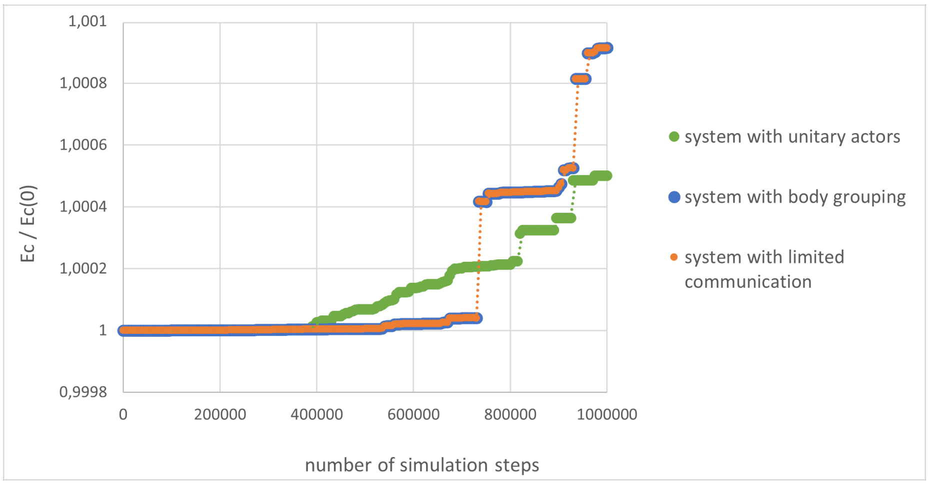

Fig. 1: Comparison of the total energy change of different models

It was expected that the total energy in the system would not be conserved over a large number of steps, due to computer arithmetic inaccuracies. The

[3] https://www.cyfronet.pl/en/computers/15226,artykul,prometheus.html

version with individual actors, which is equivalent to a sequential version, shows an increase in error starting at around 400,000 steps. This is because much more calculations were performed and the error was accumulated. In the actor systems that use clustering, the simulation accuracy is better for a longer time. However, later on the error builds up more rapidly, which may reflect the fact that a small inaccuracy in one cluster affects all others more significantly. However, the error size introduced by clustering optimizations is surprisingly small compared to the classic sequential version, which confirms the usefulness of the proposed models.

4.2 Simulation performance

To estimate the overall performance of the proposed actor-based simulations, the average time to compute one simulation step has been compared with the classic sequential version. For versions with body grouping, the number of actors increased with the number of bodies. All experiments involved 3 computing nodes of the supercomputer, which provided 48 computing cores. The results are presented in Table 1.

Table 1: Comparison of simulation step execution time (in milliseconds) for different versions of the system

number of bodies	**100**	**500**	**1000**	**5000**	**10000**
Sequential version	1.48	13.6	56.9	1089	3850
Problem-driven actor model	2.40	23.3	55.4	1888	6335
number of clusters	**2**	**5**	**10**	**25**	**50**
Computation-driven actor model	1.37	1.52	4.58	7.47	12.2
Limited-communication model	1.19	1.43	4.51	6.73	10.7

The quadratic increase in the time needed to perform the simulation step in relation to the amount of data for the sequential and problem-driven versions has been expected. It is worth noticing that for 10,000 objects, each step takes 3.85 and 6.34 seconds, respectively, which makes the approach very inefficient for larger problems. The overhead of using large numbers of actors is also clearly visible – intensive context switching and communication burden the efficiency significantly.

The other two versions, in which the bodies' groups were processed by the actors, behave apparently very well. The time of the simulation step for a small problem reaches about 10ms for both versions and does not change significantly in relation to the change in the size of the problem. This result shows how much the optimization related to the division of bodies into groups and the parallelization of calculations for each of them affects the simulation execution time.

4.3 Strong scalability

In the strong scalability testing, the problem remains constant while computational resources increase. Only the versions with body grouping were tested with the scenario of 250000 bodies. The bodies were divided into 50 to 1250 groups, each processed by a dedicated actor. Hardware resources allowed each actor to work on a separate computing core. The results are presented in Figure]2.

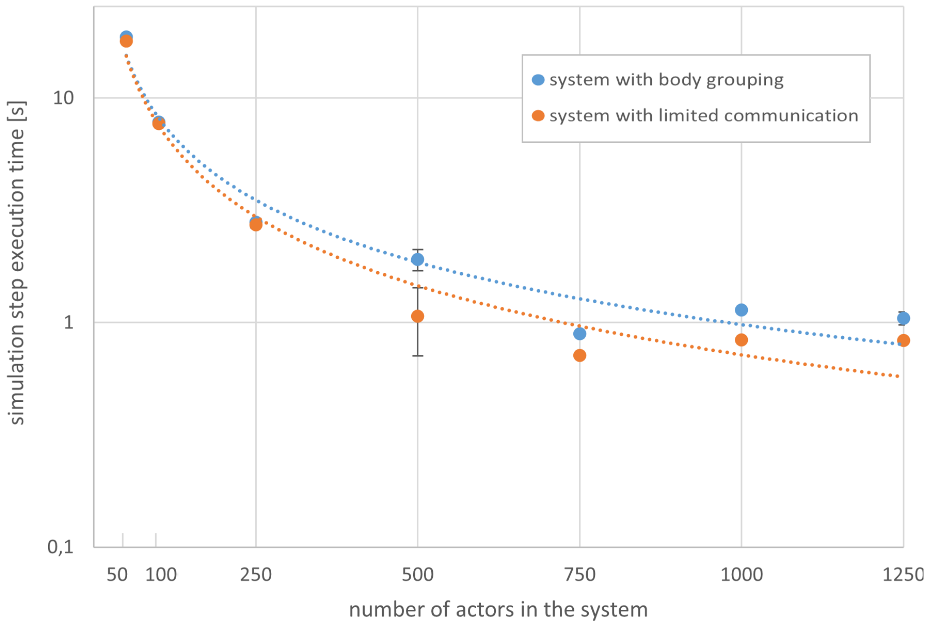

Fig. 2: Strong scalability test results for 250 000 bodies

The execution time of the simulation step decreases as expected, especially with fewer actors used. This shows that sharing the bodies among more processes effectively reduces the time needed to perform the simulation.

For a given size of the problem, the time decreases to about 750 actors. Further fragmentation of tasks does not improve performance and effectively leads to waste of resources. Here, the time reduction associated with fewer objects per actor is less than the time increase associated with having to process more messages.

4.4 Weak scalability

The weak scalability tests aim to evaluate the system's ability to process growing tasks with growing resources. Typically, the task for each computing core has a constant size in such tests. The task of 100 bodies was defined and tested for 10 – 2000 actors. The resulting simulation step time is presented in Figure 3.

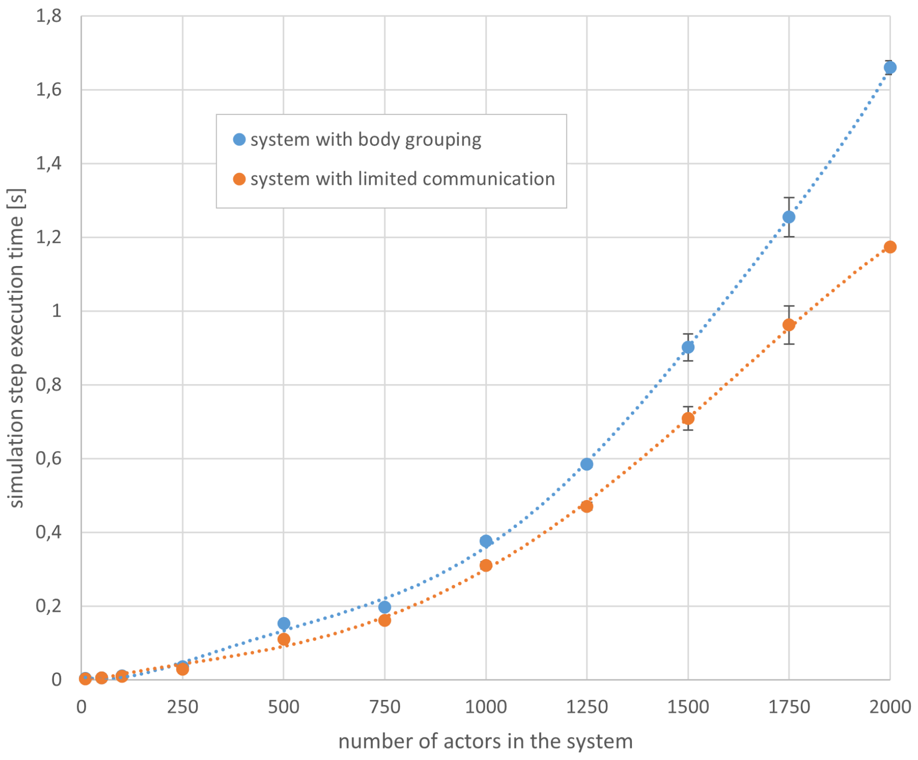

Fig. 3: Strong scalability test results for 100 bodies per actor

The number of 100 bodies per actor has been deliberately selected to demonstrate the significant problem with the scalability of the approach. The computational task performed in each of the actors is constant; however, the number or volume of the messages sent and received grows with the number of actors in the system. This highly undesirable situation, which is forced by the simulation model, is the reason for the growth of the simulation step-time.

The conclusion of the experiments would be to use a proper balance between the number of actors and the number of bodies assigned to each actor. This result is consistent with the theoretical analysis of complexity presented earlier.

5 Conclusion

The proposed method for the simulation of the N-body problem is promising, as we have obtained good quality results, in particular when the bodies were clustered. Moreover, acceptance of a certain delay in delivering the data on the influence of the farther clusters can also be treated as a good direction for further study. Actually using the actor model for concurrent implementation, easy generalization into distributed infrastructure and the "desynchronization" of the communication (cf. [21]) can be treated as the most important aspects of this paper.

As a visible flaw, the results obtained for large number of actors computing small tasks (e.g. 100 bodies) did not turn out to be an efficient case, so these kinds of simulation cannot be sped-up using HPC in such easy way. Further research in this area is needed, leading towards a scale-invariant actor. Its computational task should be constant or depend linearly on the size of the problem fraction assigned to the actor. Thus, improvements to communication methods and perhaps deeper exploration of the desynchronized exchange of messages can lead us further into this interesting research topic.

In the presented work we tested the actor-based approach on a toy-model example of a simple gravitational N-body simulation (that is, with long range-forces). At this stage – as the present work shows – it seems that it will be also appropriate to use the actor approach in N-body simulations with short-range forces such as considered, for example, in molecular dynamics simulations [11]. In molecular dynamics, the time-dependent behavior of a system of particles is simulated at the microscopic level and is widely used in various branches of science and technology, especially in biophysics. In some cases molecular interaction models are used in which inter-atomic or inter-molecular long-range interactions can be neglected and only short-range potentials like Lennard-Jones force law or torsional forces describing bonds, etc suffice. We will continue the work in this promising area, putting together the expertise in-between the IT and physics in order to reach synergistic and valuable output.

References

1. L. K. Babadzanjanz. Existence of the continuations in the n-body problem. *Celestial mechanics*, 20(1):43–57, Jul 1979.
2. L. K. Babadzanjanz. On the global solution of the n-body problem. *Celestial Mechanics and Dynamical Astronomy*, 56(3):427–449, Jul 1993.
3. Jasjeet Singh Bagla. Cosmological N-body simulation: Techniques, scope and status. *Curr. Sci.*, 2005.
4. J. Barnes and P. Hut. A hierarchical o(n log n) force-calculations algorithm. *Nature*, 1986.
5. Sandro Bottaro and Kresten Lindorff-Larsen. Biophysical experiments and biomolecular simulations: A perfect match? *Science*, 361(6400):355–360, 2018.
6. Roberto Capuzzo-Dolcetta, Mario Spera, and Davide Punzo. A fully parallel, high precision, n-body code running on hybrid computing platforms. *Journal of Computational Physics*, 2012.

7. J. Carrier, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm for particle simulations. *SIAM Journal on Scientific and Statistical Computing*, 9(4):669–686, 1988.
8. Alain Chenciner and Richard Montgomery. A remarkable periodic solution of the three-body problem in the case of equal masses. *Annals of Mathematics*, 152(3):881–901, 2000.
9. R. Ellis, A. Perelson, and N. Weisse-Bernstein. The computation of the gravitational influences in an n-body system. *International Amateur-Professional Photoelectric Photometry Communication*, 75, 1999.
10. Lukasz Faber, Kamil Pietak, Aleksander Byrski, and Marek Kisiel-Dorohinicki. Agent-based simulation in age framework. In A. Byrski et al., editor, *Advances in Intelligent Modelling and Simulation - Simulation Tools and Applications*, volume 416 of *Studies in Computational Intelligence*, pages 55–83. Springer, 2012.
11. Adam Hospital, Josep Ramon Goñi, Modesto Orozco, and Josep L Gelpí. Molecular dynamics simulations: advances and applications. *Advances and applications in bioinformatics and chemistry*, pages 37–47, 2015.
12. Piet Hut and Jun Makino. Moving stars around. *The Art of Computational Science*, 2007.
13. Tomoyoshi Ito, Junichiro Makino, Toshikazu Ebisuzaki, and Daiichiro Sugimoto. A special-purpose n-body machine grape-1. *Computer Physics Communications*, 1990.
14. Atsushi Kawai, Toshiyuki Fukushige, and Junichiro Makino Makoto Taiji. Grape-5: A special-purpose computer for n-body simulations. *Publications of the Astronomical Society of Japan*, 2000.
15. Edward Liverts, Evgeny Griv, Michael Gedalin, and David Eichler. *Dynamical Evolution of Galaxies: Supercomputer N-Body Simulations*, pages 340–347. Springer, 2003.
16. Wang Qiu-Dong. The global solution of the n-body problem. *Celestial Mechanics and Dynamical Astronomy*, 50(1):73–88, Mar 1990.
17. Raymond Roestenburg, Rob Williams, and Robertus Bakker. *Akka in action*. Simon and Schuster, 2016.
18. Robert Schaefer, Aleksander Byrski, Joanna Kolodziej, and Maciej Smolka. An agent-based model of hierarchic genetic search. *Comput. Math. Appl.*, 64(12):3763–3776, 2012.
19. Karl F. Sundman. Mémoire sur le problème des trois corps. *Acta Mathematica*, 36(none):105 – 179, 1913.
20. Prabhat Totoo and Hans-Wolfgang Loidl. Parallel haskell implementations of the n-body problem. *Wiley InterScience*, 2013.
21. Wojciech Turek. Erlang-based desynchronized urban traffic simulation for high-performance computing systems. *Future Gener. Comput. Syst.*, 79:645–652, 2018.
22. Robert J. Vanderbei. New orbits for the n-body problem. *Annals of the New York Academy of Sciences*, 1017(1):422–433, 2004.
23. Loup Verlet. Computer ”experiments” on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Phys. Rev.*, 159:98–103, Jul 1967.
24. Long Wang and et al. Nbody6++gpu: ready for the gravitational million-body problem. *Monthly Notices of the Royal Astronomical Society*, 2015.
25. Tiandan Wu. An application of n-body simulation to the rotational motion of solar system bodies. Master’s thesis, Miami University, 2008.