Advances in Engineering Software 125 (2018) 178–187

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Parallel implementation of hyper-dimensional dynamical particle system on CUDA

Jan Mašek*[,a], Miroslav Vořechovský[b]

[a] Institute of Structural Mechanics, Brno University of Technology, Veveří 331/95, Brno 602 00, Czech Republic
[b] Institute of Structural Mechanics, Brno University of Technology, Veveří 331/95, Brno 602 00, Czech Republic

ARTICLE INFO

Keywords:
Particle dynamical system
Parallel implementation
NVIDIA CUDA
On-chip memory
Global memory
Atomic operations
Serialization of threads

ABSTRACT

The presented paper deals with possible approaches to parallel implementation of solution of a hyper-dimensional dynamical particle system. The proposed implementation approaches are generally applicable for similar particle systems of interest in various research and engineering fields. The original motivation for the present work was a simulation of particles that represent a space-filling design to be optimized for further use in design of experiments. Due to the underlying purpose of this particle system, the dimension of the particle system of interest is considered to be entirely arbitrary. Such a hyper-dimensional space is further folded into a periodically repeated domain.

The theoretical background of the proposed particle system is provided along with the derivation of equations of motion of the dynamical system. As the complexity of the system is not limited by the number of particles nor the number of dimensions, the possibilities of utilizing the GPGPU platform are more restricted in comparison with today's fast parallel implementations of common particle systems.

Two distinct approaches to parallel implementation are presented, one aiming at a generalized usage of the fast on-chip resources, the other entirely relying on the GPU's on-board global memory. Despite unambiguous mutual differences in their performance, both parallel implementations deliver major speedup compared to the single-thread CPU solution as well as a better scaling of execution time when increasing both the number of particles and dimensions.

1. Introduction

During the recent decade, researchers dealing with simulation of particle systems acquired a rather powerful computing platform with the development of general purpose computing on graphic processor units (GPGPU). With the great computational power offered by the GPGPU architecture, increasing number of formerly non-calculable problems are now being solved. Particle systems are nowadays simulated in numerous engineering and research fields. Molecular dynamics, material and mechanical engineering or astrophysics are only a few examples of these.

Namely the astrophysics simulations of vast scenarios of forming galaxies with tens of thousands of planets-particles are now possible to compute, see e.g. [1–3]. Unlike the systems of mutually attracting celestial bodies simulated by the astrophysicists, there exist similar particle models without a direct physical analogy to the purpose of their simulation. This is also the instance of the dynamical particle system as considered further in the presented paper.

The proposed system of *mutually repelling* particles is assembled to serve as an optimization tool for obtaining *uniformly* distributed point samples. Such optimized samples may find utilization in dozens of research problems, an interesting instance of which is the statistical sampling for numerical integration of an arbitrary function – Monte Carlo sampling.

Numerical integration of the Monte-Carlo type requires sampling of points that are uniformly distributed within a *design domain*. The design domain is considered to be the domain of sampling probabilities (values of the joint distribution function - the domain of copulas) which is a unit hypercube $[0, 1]^{N_{\text{var}}}$, where N_{var} is the dimension of the design domain and also the number of random variables of the integrated function.

The problem of using an ideally distributed set of finite number of integration points rises also in numerous engineering and research fields. While sampling from a random vector or integrating an unknown function, achieving a uniform layout of integration points is the only possible way for minimization of the lower bound of the resulting error, see e.g. the Koksma–Hlawka inequality [4–6].

Many criteria for "uniformity" have been put forth over the past

* Corresponding author.
E-mail addresses: jan.masek1@vut.cz (J. Mašek), vorechovsky.m@vut.cz (M. Vořechovský).

https://doi.org/10.1016/j.advengsoft.2018.03.009
Received 18 July 2017; Received in revised form 31 January 2018; Accepted 20 March 2018
Available online 07 April 2018

years, aiming to serve for evaluation or optimization of distribution of N_{sim}points within a unit hypercube of dimension N_{var}. These criteria often investigate mutual distances between integration points with a tendency to prefer designs with points equally distant from each other.

Some other criteria can be shown to have analogies with physical problems. As an elegant instance of these, we consider the Audze-Eglājs (AE) criterion [7] and the generalized ϕ-criterion [8], respectively. The limiting case of the ϕ-criterion is the MaxiMin criterion [9] that prefers designs maximizing the distance between the closest pairs of points.

All of these criteria serve for evaluation of quality of point layouts and can be reinterpreted as a potential energy of a system of charged particles with repulsive forces. The objective of these criteria then lies in minimization of potential energy of a particle system and the positions of the particles are considered to be the coordinates of sampling points in the unit design hypercube.

During the recent years, it has been shown that the Audze-Eglājs criterion suffers from existence of boundaries of the design space [10,11]. A remedy of this behavior was proposed [10,11], assuming periodically extended design hypercube and thus achieving a design domain without boundaries, see Fig. 1b. Since then, it has been proved that optimization of point layouts by the introduced Periodic Audze-Eglājs (PAE) criterion leads to statistically uniform designs (from design to design) and to well distributed set of points for every single point layout.

This paper is based upon the conference paper [12], but the present paper provides much broader context, complementing the already proposed implementation (which relies on the GPU's on-board memory) by the recently assembled implementation utilizing the faster but limited on-chip memory. The in-detail study of thread serialization techniques along with the performance comparison of using atomic operations is not part of this paper and remains as a reference to the conference contribution [12].

Since the initial motivation for simulation of such a particle system was in optimization of point samples using the Audze-Eglājs criterion, the criterion itself is presented in Section 2 and the paper follows with derivation of equations of motion of the physically analogical system of charged particles in Section 3.

A brief review of particle simulation algorithms is presented in Section 4 and the particle system of interest is set into the context of today's GPGPU particle implementations and solution algorithms. Section 5 follows with an analysis of requirements on the solution implementation.

Hardware limitations rising from the unrestrained dimensionality of the particle system at hand are discussed in detail and reasons for both implementation approaches are justified, see Section 6. There, two different ways of data storage and the associated algorithms are presented.

Finally, the performance of both parallel implementations is provided and further compared to the single-thread CPU implementation in Section 7.

2. Audze-Eglājs Optimization criterion

The value of the Audze-Eglājs criterion can be understood as the amount of potential energy stored within a system of mutually repelling particles. The potential energy accumulated in particle interactions depends on distances between all pairs of particles.

The Euclidean distance between points i and j in N_{var}-dimensional space, L_{ij}, can be expressed as a function of their coordinates:

$$L_{ij} = \sqrt{\sum_{v=1}^{N_{var}} (x_{i,v} - x_{j,v})^2} = \sqrt{\sum_{v=1}^{N_{var}} (\Delta_{ij,v})^2}, \tag{1}$$

where $\Delta_{ij,v} = |x_{i,v} - x_{j,v}|$ is the difference in their positions projected onto the axis v. Let us assume that the points i and j with their mutual distance L_{ij} are repelled by the force F_{ij} induced by the potential energy E_{ij}:

$$E_{ij}(L_{ij}) = \frac{1}{L_{ij}^2} = \int_{\infty}^{L_{ij}} F_{ij}(x)\mathrm{d}x. \tag{2}$$

By differentiating the energy potential with respect to the distance, L_{ij}, the *repulsive* force is obtained: $F_{ij}(L_{ij}) = 2L_{ij}^{-3}$. As the particle system contains N_{sim} interacting particles, the total potential energy of the system is a sum of contributions from all $\binom{N_{sim}}{2}$ individual pairs:

$$E_p = \sum_{i=1}^{N_{sim}-1} \sum_{j=i+1}^{N_{sim}} E_{ij} = \sum_{i=1}^{N_{sim}-1} \sum_{j=i+1}^{N_{sim}} \frac{1}{L_{ij}^2}. \tag{3}$$

The total potential energy in Eq. (3) represents the value of the Audze-Eglājs criterion to be minimized.

A simple and efficient improvement that considers a periodic extension of the design space has been proposed in [10]. After some simplification, one can derive equations for *periodic* Audze-Eglājs criterion (PAE) by replacing $\Delta_{ij,\,v}$ in Eq. (1) with its periodic variant:

$$\overline{\Delta}_{ij,v} = \min(\Delta_{ij,v},\ 1 - \Delta_{ij,v}). \tag{4}$$

With such a redefined projection, a new metric is obtained and the distance between points i and j, called the periodic length $\overline{L}_{ij}$, becomes the actual shortest distance between point i and the nearest image of point j [10], also see Fig. 1:

$$\overline{L}_{ij} = \sqrt{\sum_{v=1}^{N_{var}} (\overline{\Delta}_{ij,v})^2}. \tag{5}$$

We note that using the nearest image of point j with respect to point i does not cover a true periodic repetition of the design domain. In a complete periodic repetition, infinite number of images of the point j would interact with the point i. The presented approach is a simplification that can be shown [10] to yield identical results to the fully repeated system in case of sufficient point count, N_{sim}.

If the number of points in the original domain is too low for assembly of the desired self-similar pattern[1], considering additional periodical images of particles is advised. As argued in [13], this is due to an insufficient resolution between short and long-range forces in the system. Another remedy is also to use stronger differentiation between short and long-range forces, that is to rise the exponent upon the mutual distance L_{ij} in the energy potential, see Eq. (3), to a greater value. As shown in [13], the exponent shall be no lower than $N_{var} + 1$.

For greater particle systems, there is no practical need to consider more than one image of each particle as long as the energy potential uses a correct value of the exponent. Then, additional periodical images are not considered as a true periodic extension is not necessary for achieving an optimal space-filling design.

The AE criterion is originally meant for evaluation of uniformity of a fixed set of particles by calculating the overall potential energy. This potential energy is stored in all pairwise particle interactions considered in a radial sense.

For the purposes of dynamical simulation, however, each vector of the mutual repelling force $F_{ij} = \ddot{x}_{ij}$ induced by such an energy potential needs to be decomposed into all of its N_{var}orthogonal components which then provide the information about the actual accelerations $\ddot{x}_{ij,v}$ in each of N_{var}:directions. The derivation of equations of motion of the particle system is discussed in what follows.

[1] Simplest self-similar space-filling patterns can be assembled from simplest objects which contain volume in the particular dimension N_{var}: line in 1D (2 points), triangle in 2D (3 points), tetrahedron in 3D (4 points), etc.

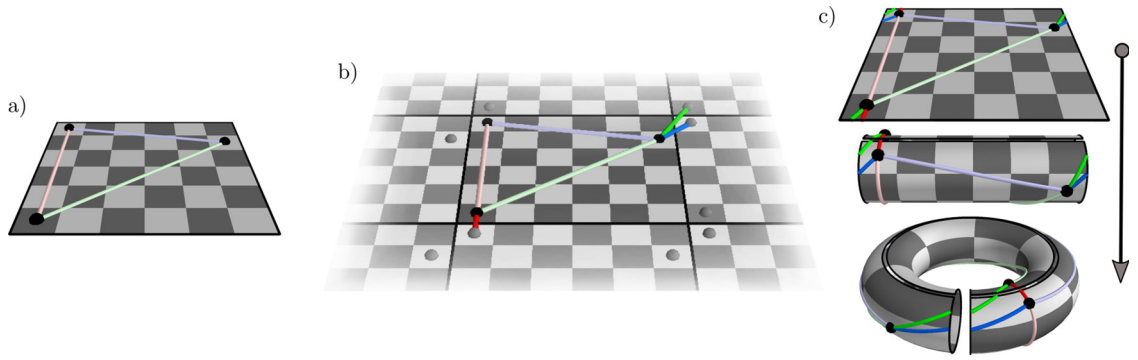

Fig. 1. Illustration of periodically repeated planar domain. a) the original two-dimensional design domain with pale colored distances L_{ij} (Eq. (1)). b) periodically repeated design domain with eight additional images of each particle. Periodic distances $\overline{L}_{ij}$ (Eq. (5)) are rich colored. c) folding the design domain into a torus is another possible illustration of a periodical domain. Note that the computed distances are not defined on the toroidal surface.

3. Physical analogy: a particle system

The (periodic) Audze-Eglājs criterion understands the layout of design points as a system of interacting (mutually repelling) particles and evaluates the amount of potential energy stored within all interactions, see Fig. 1 for illustration.

Instead of utilizing the AE/PAE criterion as a norm minimized using combinatorial or heuristic optimization for a fixed set of coordinates [14], we propose to solve the physically analogical problem by simulating a discrete dynamical system of mutually repelling particles. The coordinates of particles of the dynamical system, after reaching the static equilibrium (after a minimization of potential and kinetic energy), may be directly understood as coordinates of design points within the unit hypercube, see Fig. 2.

As argued in [13], a direct usage the (P)AE energy potential, see Eq. (3), is not advised for it has been shown that the exponent upon the mutual distances shall depend on the dimension of the design domain as well as, partially, on the number of particles. The formulation of the potential energy of a system of charged particles will therefore be generalized and from now on, the exponent q will utilized for derivation of equations of motion of the dynamical particle system, see Appendix I.

The resulting equations of motion of the dynamical particle system are a system of independent equations. This awareness is of high importance while considering the possibilities for solution method and its computer implementation. This means that each acceleration $\ddot{x}_{i,v}$ can be solved separately, without solving a system of equations. Each undamped acceleration $\ddot{x}_{i,v}$ as resulting from the presented energy potential, see Eqs. (A.2) and (A.3), can be computed as follows, see Appendix I for derivation:

$$\ddot{x}_{i,v} = \frac{1}{m} \sum_{j \neq i}^{N_{\text{sim}}} \frac{\overline{\Delta}_{ij,v}}{\overline{L}_{ij}^{q+2}}. \tag{6}$$

The damping of motion of particles also depends solely on the velocity of each particle. Furthermore, each distance projection $\overline{\Delta}_{ij,v}$ as well as each absolute distance $\overline{L}_{ij}$ can be computed independently. The above-mentioned properties lead to the possibility of utilizing a *parallel* implementation.

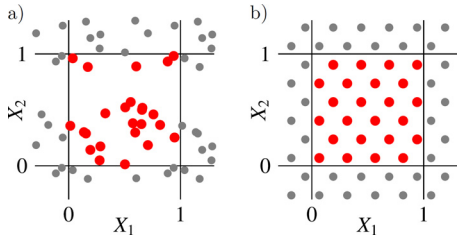

Fig. 2. Illustration of the optimization process of $N_{\text{sim}} = 24$, $N_{\text{var}} = 2$. a) initial randomized sample, b) optimized sample.

As soon as the new accelerations of each particle in each dimension are obtained, the equations of motion are numerically integrated using the semi-implicit Euler method and the new velocities $\dot{x}_{i,v}$ and coordinates $x_{i,v}$ of each particle in each dimension in the new time $(t + \Delta t)$ are computed:

$$\dot{x}_{i,v}(t + \Delta t) = \dot{x}_{i,v}(t) + \Delta t \cdot \ddot{x}_{i,v}(t), \tag{7}$$

$$x_{i,v}(t + \Delta t) = x_{i,v}(t) + \Delta t \cdot \dot{x}_{i,v}(t + \Delta t). \tag{8}$$

For reaching the static equilibrium of the dynamical system, implementing of energy dissipation is desirable. Various types of damping are typically combined. For solving the problem at hand, we add the sum of velocity-dependent damping members into Eq. (7): $\sum_p c_p \dot{x}_{i,v}^p(t)$, where c_p are damping coefficients and p are various powers of the velocity, $\dot{x}_{i,v}$, of ith particle in vth dimension. Note, that the damping part is not derived from the energy potential of the particle system.

4. Particle simulation algorithms

The first simulation of a dynamical particle system was performed in 1960 by von Hoerner [15]. Since then, particle simulations in various research fields were one of great driving forces of development of super and parallel computing.

The $\mathcal{O}(N^2)$ complexity of the brute-force all-pairs solution (here $N \equiv N_{\text{sim}}$ is the number of particles) has been lowered to $\mathcal{O}(N \log N)$ by the Barnes-Hut Treecode algorithm [16] which lowers the complexity by clustering the distant particles into larger groups and approximates their influence on the solved particle. The performance of such an approximation algorithm is however suited dominantly for particle systems where the distribution of particles is highly non-uniform.

Further reduction to the complexity of $\mathcal{O}(N)$ was achieved by the Fast Multipole Method (FMM) [17]. The FMM-type algorithm assembles hierarchical structures not only by clustering the remote particles but also the nearby particles using local expansions. For large particle systems, usage of the FMM algorithm also greatly reduces the number of summands and thus reduces the resulting computation error. In case of smaller particle systems with thousands of particles, the FMM can be shown to yield an intermediate performance between the all-pairs $\mathcal{O}(N^2)$ and the $\mathcal{O}(N)$ complexity, see [18].

These are, nevertheless, approximation-based algorithms which do not suffice for general research of particle systems such as the one proposed above. The particle system of interest here is being a subject of a constant investigation of influence of various interaction laws on the resulting distribution of particles. This means that any premature approximation of particle interaction might deliver confusing results. Not to mention physical reasoning of clustering particles within a periodically repeated domain. Therefore we consider the direct N_{sim}-body

integration that involves the computation of all $\binom{N_{\text{sim}}}{2}$ forces between all pairs of particles.

The direct all-pairs summation is also beneficial for further research of the particle system at hand as we wish to study the evolution of distribution of the potential energy among all mutual interactions. Also, we aim to compute all mutual distances as their distribution is of our interest.

4.1. GPGPU computing of particle systems

Since the very first release of the NVIDIA's Compute Unified Device Architecture (CUDA) in 2006, see [19], simulations of particle systems were among the first applications to exploit the novelty architecture of general purpose parallel computing.

One of the first implementations of GPU solution of a 3D particle system was the so-called Chamomile algorithm [20]. As used in the CUNBODY library [20,21], the Chamomile algorithm considers parallelization of the *source* particles (particles acting on the particles being solved) between thread blocks, requiring a rather large final reduction. Latency of such a reduction might be a challenge to hide.

A more efficient approach was later published by the NVIDIA itself [22], proposing parallelization of the *target* particles (particles being solved) among thread blocks. This implementation therefore does not require large reductions to be performed. Extensive utilization of registers along with *loop unrolling* is also proposed in pursuit of maximal performance.

The idea [22] of using the on-chip shared memory for circulation of all *source* particles while keeping descriptions of the *target* particles stored in registers/L1 cache is also used in the first of the two implementations presented in what follows. Performance of such a solution algorithm depends dominantly on the arithmetic performance of the GPU used and, consequently, on the ability to hide this arithmetic latency.

Note that term *latency hiding* as used further in the paper is meant as an effort to reach hardware's maximum throughput, see [23] and also Chapter 5 of [24]. Typically, the more limiting factor is the latency of accessing the (global) memory rather than the latency of execution of arithmetics. However, for each particular code, their actual ratio may vary.

5. Requirements on solution implementation

Numerical simulation of the proposed particle system consists of several sub-steps which can be, up to certain degree, executed in parallel. This degree is limited dominantly by the nature of the problem at hand and by the possibilities of the hardware used. Currently, the Kepler architecture NVIDIA Tesla K20c (GK110) and the NVIDIA GTX 1080TI driven by Pascal (GP102) are being utilized.

A common problem of any parallel implementation is to control writing requests of threads in a way that multiple requests for writing into the same memory address cannot be executed simultaneously. This scenario is known as the *race condition*. When writing, threads can overwrite values computed and stored by other threads in an incorrect order, which typically renders following computations over such data incorrect. Handling such inconsistency in writing is of high importance in implementations requiring large reductions. Commonly, manual serialization of writing requests in code, parallel reductions or atomic operations are used, see [25] and [26]. Possible ways of solving concurrent writing requests are in-detail discussed and their performance compared in [12].

The result of a code exhibiting a race condition cannot be predicted as the performance of threads is expected to be identical and the execution time of the same instruction by identically powerful threads depends on hardly predictable circumstances.

Furthermore, in case of the problem at hand, one has to bear in mind that unlike most of the conventional particle simulations, the complexity of the proposed system is not limited by the number of dimensions, N_{var}. As a matter of fact, the unknown number of particles, N_{sim}, and dimensions, N_{var}, at the time of compilation call for a quite general implementation.

With the theoretical size of the solved problem being arbitrary, the fast on-chip resources of the GPU (registers, L1 cache, shared memory) might not suffice. Therefore the second proposed implementation, see [12] and Section 6.2, ignores the possibility of using on-chip resources other than registers needed for conducting arithmetic operations and storing intermediate results. Performance of such an algorithm depends dominantly on the global memory bandwidth and bus size of the GPU used and, consequently, on the ability to hide this memory access latency.

6. Presented approaches to GPU solution

As has been indicated in what preceded, due to the essentially arbitrary extent (number of particles N_{sim} and dimensions N_{var}) of the particle system at hand, we further present two fundamentally different implementation approaches.

6.1. Implementation using on-chip resources

The first presented implementation of particle system simulation is loosely based on the concept [22] that proposes to keep the descriptions of *target* particles statically loaded in registers/L1 cache and circulate the descriptions of *source* particles in the shared memory. In our case, however, the term *descriptions of particles* differs for *target* and *source* particles.

The *target* particles (particles, accelerations of which are to be computed) do not possess only unknown values of accelerations but we also wish to compute the radial stress each particle is experiencing and the amount of potential energy stored in interactions of each particle at any given time. This means that during solution, we need to keep stored the following descriptions of *target* particles on chip:

- N_{var} own coordinates of each particle,
- N_{var} unknown accelerations of each particle,
- one unknown radial stress of each particle,
- one unknown value of potential energy belonging to interactions of each particle.

It also appeared beneficial to store on chip the computed projections $\overline{\Delta}_{ij,v}$ of the mutual distance, $\overline{L}_{ij}$, between the current solved pair of particles. This means storing additional N_{var} values on the chip. All this data, exclusive to each thread in the thread block, may be stored either in registers or the L1 cache/shared memory.

We have, nevertheless, serious objections towards using registers in this case:

- the number of dimensions, N_{var}, is not known at the compilation time, which might easily lead to uncontrolled register spilling into GPU's local memory,
- GPU's registers are not indexable (at least not conveniently), meaning they cannot handle indexed arrays and these are spilled into the local memory right away,
- for an unknown dimension, N_{var}, it is not even possible to arrange the data into vector structures stored in registers,
- in case of further optimization of data reuse, registers are not accessible by other threads in the thread block. However, it is worth noting that since the Kepler architecture (Compute capability 3.0+), the NVIDIA GPUs are capable of fast exchange of data in registers of threads in the same warp using the Shuffle (`shfl()`) method.

Because of the above stated, we propose using the L1 cache/shared memory for storing descriptions of the *target* particles.

The descriptions of the *source* particles consist of their coordinates only, i.e. N_{var} coordinates of each particle. We much prefer to store these coordinates in the shared memory as they are to be accessed by all threads in each thread block.

It can be expected that at one point of the solution process, the loaded part of the *source* particles will represent coordinates of the *target* particles, which are already being loaded on-chip. In this case, it is beneficiary to store N_{var} coordinates of the *target* particles in the shared memory as well for they can be accessed right away, without requesting global memory accesses.

Obviously, the amount of the L1 cache/shared memory requested also depends on the number of threads per block which can be tuned up to certain degree. Nevertheless, it should be mentioned that eventually, for a large dimension, N_{var}, the described implementation will start spilling this data into local memory.

Moreover, the precision of arithmetics matters; empirically, we observe that the *float* precision (32 bit values) is sufficient for solution of the system at hand (not requiring a solution of a system of mutually dependent equations of motion). This means requesting half the resources otherwise needed for the *double* precision (64 bit values).

Before closing the subject of the on-chip resources, it should be noted, that each of 13 streaming multiprocessors (SM) of the Kepler Tesla K20c (GK110) possesses 64KB of configurable on-chip memory to be divided between L1 cache and shared memory.

The GTX 1080TI (GP102) and all the GPUs of the Pascal architecture follow the concept of the previous Maxwell architecture, providing dedicated 96 KB of shared memory for each of its 28 SMs. The functionality of the L1 cache has been merged with the texture cache for Maxwells and Pascals. The L1 cache has been provided only with dedicated 28 KB on GP100 and GP102 cores.

For the future, we aim to the usage of the Pascal Tesla P100 (GP100), therefore we lean towards exclusive usage of the shared memory. To conclude on the on-chip memory allocation approach: we store all $[(4{\cdot}N_{\text{var}} + 2){\cdot}\text{threadsInBlock}]$ values in the shared memory of each SM. The next section discusses the on-chip solution algorithm.

6.1.1. Solution algorithm

As has been already discussed above, the proposed on-chip algorithm distinguishes the *target* particles (particles, unknown properties of which are to be computed) and the *source* particles (particles acting on the *target* particles).

Descriptions of the *target* particles are being held entirely in the shared memory, as justified in the previous section. The *target* particles are maintained by N_{sim} threads divided into $tb = (N_{\text{sim}}/p + 1)$ thread blocks, where $p = \text{threadsInBlock}$. After loading p *target* particles into the shared memory, tiles of *source* particles are consecutively loaded into shared memory and their interaction with *target* particles is solved.

The *source* particles are divided into n tiles, each containing descriptions of m *source* particles. In our implementation, we prefer $m = p$ which leads to possibility of data reuse and also empirically seems optimal for hiding arithmetic latency when good occupancy of GPU is reached. Nevertheless, it might be beneficiary to set $m < p$ to save the on-chip memory in case of high dimension, N_{var}, or when using values of higher precision. A setting of $m > p$ leads to longer arithmetic occupancy of thread blocks which might be useful to hide the latency of global memory accesses if these are too expensive.

The described solution procedure as conducted by a generic thread block is illustratively depicted in Fig. 3 and also further explained by Algorithm 1. As can be seen, the part of solution of interaction with individual tiles is serial for each thread (each particle). This might seem inefficient at first, but keeping the executed thread blocks busy with arithmetics helps to hide the latency of blocks waiting for global memory accesses.

The newly gained accelerations are stored into global memory and afterwards used for numerical integration using the semi-implicit Euler method, see Eq. (7). Numerical integration kernel is executed by $N_{\text{var}}{\cdot}N_{\text{sim}}$ threads, each of which updates its own velocity $\dot{x}_{i,v}$ and coordinate $x_{i,v}$. The task of numerical integration therefore takes only a negligible fraction of the total execution time.

6.2. Implementation using global memory

The second presented implementation approach, as already briefly discussed, aims to rely completely on usage of the GPU's global memory. The main reason is that the *theoretical extent* of the particle system at hand is not limited in terms of the number of particles, N_{sim}, nor in the number of dimensions, N_{var}. Therefore the limited amount of on-chip resources is not going to suffice in scenarios of very high dimensions, N_{var}, and more so if the *double* precision format is used.

The global memory implementation as further described, exhibits a great global memory traffic, hiding the latency of which is a major task. The approach here is to unfold the parallelism of the solution entirely into global memory, executing the highest number of threads possible for each computational step. This way keeps the SMs as busy with simple arithmetics as possible. Of course, it is crucial to ensure that the global memory accesses are *coalesced* and bus utilization is maximized.

Such an implementation requires rather large reductions into the same global memory address. For handling these reductions, fast atomic additions `atomicAdd()` are used, see [26].

Despite the fact that there exist $\binom{N_{\text{sim}}}{2}$ mutual interactions (pairs) of N_{sim} particles in each dimension, it is necessary to realize that each interaction between any two particles i and j results in computing and writing two opposite accelerations, one for the particle i and one for the particle j. This in fact means that it is needed to compute and write $(N_{\text{sim}})^2$ (N_{sim} of which are zero) accelerations of particles in each dimension.

It is indeed possible to compute only $\binom{N_{\text{sim}}}{2}$ of the unique accelerations. One can attempt to use the symmetry of repulsive forces and *mirror* these known accelerations for the other particles in all respective pairs. One has to be aware, however, that such mirroring cannot be conducted between thread blocks on the chip and global memory has to be used for data exchange. In fact, the action of writing and reading in GPU's global memory is much more time expensive than on-chip parallel computing. Therefore it might be beneficiary to compute seemingly redundant data instead of further increasing the global memory workload without providing any other arithmetic tasks to hide this memory latency.

The global memory implementation is divided into several sub-steps (kernel functions):

- computation of $N_{\text{var}}{\cdot}(N_{\text{sim}})^2$ projections $\overline{\Delta}_{ij,v}$ of mutual distances between all pairs of particles in each dimension v. Execution of this step can be conducted by $N_{\text{var}}{\cdot}(N_{\text{sim}})^2$ active threads, each of which stores its result into a unique global memory address,
- computation of $(N_{\text{sim}})^2$ absolute distances $\overline{L}_{ij}$ between all pairs of particles. For obtaining a single value $\overline{L}^2_{ij}$, it is needed to read and add up N_{var} projections $\overline{\Delta}^2_{ij,v}$. This sub-step therefore requires serialization of N_{var} writing procedures into each memory address containing the total squared distance $\overline{L}^2_{ij}$,
- computation of $N_{\text{var}}{\cdot}(N_{\text{sim}})^2$ repulsive accelerations $\ddot{x}_{ij,v}$ between all pairs of particles in each dimension. The result of this sub-step is a vector of $N_{\text{var}}{\cdot}N_{\text{sim}}$ total accelerations of each particle in each dimension. Therefore, it is required to sum N_{sim} repulsive accelerations for each particle in each dimension. Hence, this sub-step requires serialization of N_{sim} writing requests into each of $N_{\text{var}}{\cdot}N_{\text{sim}}$ global memory addresses.
- numerical integration of equations of motion using the semi-implicit Euler method; updating $N_{\text{var}}{\cdot}N_{\text{sim}}$ velocities $\dot{x}_{i,v}$ and coordinates $x_{i,v}$

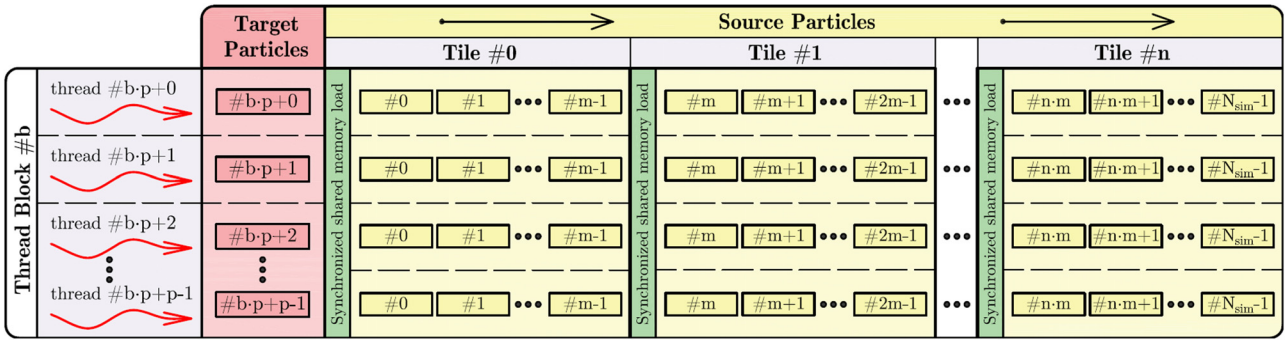

Fig. 3. Solution procedure of the on-chip memory implementation.

of all particles in each dimension. Executed by $N_{var} \cdot N_{sim}$ active threads, this sub-step does not require any serialization of writing as each thread writes into its very own memory address.

6.2.1. Handling concurrent writing requests

In parallel computing, performance *bottlenecks* due to the need of serialization of writing are fairly common. In practice, there exist two most exploited approaches how to conduct serialization of writing of multiple threads into identical memory address.

One way of avoiding the concurrent writing of multiple threads into the same memory address is to eliminate such scenario entirely. Whenever an encounter of n threads writing into a single memory address is anticipated, it is instead possible to run a kernel function with a substitute thread, executing a loop of n instructions (those which would otherwise lead to thread collision in writing). A series of writing into the particular memory address is thus provided without possible code inconsistency.

A less firm approach how to avoid writing inconsistencies is to use GPU's *atomic operations*. Atomic (indivisible) operations are procedures implemented directly by the hardware manufacturer. These provide the possibility of conducting a read-modify-write task of a 32 or 64-bit value as a single uninterruptible action.

The manufacturer guarantees that during an atomic operation, no other threads can approach the memory address, or at least these cannot change the value stored. Until the Pascal architecture, NVIDIA GPUs were capable of executing fast atomic operations only with values of the *float* precision. In case of the *double* precision values, the orders of magnitude slower `atomicCAS()` (compare-and-swap) method had to be manually implemented.

Approaches to handling concurrent writing requests are discussed in detail in the conference paper [12] along with the speedup of using fast atomic operations.

7. Performance of parallelized solution

The following section offers performance benchmark of the two developed CUDA implementations when scaling the crucial parameters of the problem; the number of particles, N_{sim}, and the number of dimensions, N_{var}.

First, the performance of the massively parallel solution using NVIDIA Tesla K20c and GTX 1080TI was compared with the single-thread implementation in the C programming language executed by the Intel i7-6700 CPU. All results presented in what follows are achieved using the *float* precision.

Fig. 4 shows the computation time required when solving of 10^5 steps of the $\mathcal{O}(N^2)$ interaction while scaling the number of particles, N_{sim}, and keeping the dimension constant, $N_{var} = 2$. A significant speedup was reached when utilizing massive parallelization. Crucial is the qualitative difference of computation time rise when scaling the number of particles, N_{sim}.

The implementation using the on-chip shared memory (`SMem`) has shown very good performance in terms of execution time as well as its mild linear dependence on the number of particles, N_{sim}. Also, if the dimension, N_{var}, is known at the compilation time, it is possible to take advantage of partial usage of registers for storage of unknown particle accelerations, stresses and energies (`SMem + registers`) which leads to additional speedup.

The implementation relying on the use of the GPU's on-board global memory and fast atomic additions (`GMem + atomicAdd`) did not exhibit as good results compared to the shared memory implementation. The global memory solution performs worse both in execution time as well as in scaling with the number of particles. The only exception are scenarios up to around 200 particles, where the global memory implementation performs slightly better, see the inset in Fig. 4. The reason is that for such a low number of particles, the partially serial solution in shared memory is not able to hide the arithmetic latency as well.

Next, we investigate the execution time when scaling both parameters of the problem; the number of particles, N_{sim}, as well as the number of dimensions, N_{var}. Fig. 5a compares the execution time of solution of 10^3 steps of the $\mathcal{O}(N^2)$ interaction.

The CPU implementation is kept as a benchmark for shared memory implementations executed by the GTX 1080TI. The CUDA implementation delivers major improvement of performance as well as weaker execution time growth with increasing N_{sim}. Fig. 5a shows that the parallel execution starts with linear dependency on N_{sim}. As the device starts to reach its peak bandwidth, the execution time scaling tends to $\mathcal{O}(N^2)$. As shown in Fig. 5b, in the expected range of computation, the speedup in solution time by CUDA grows linearly with the number of particles, reaching up to 200 × .

7.1. Bounds on parallel solution

The limitations of execution of the shared memory implementation, see Section 6.1, are primarily set by the on-chip resources as each SM has to provide storage for $[(4 \cdot N_{var} + 2) \cdot \text{threadsInBlock}]$ values for every thread block it is scheduled to maintain (96kB limit for GTX 1080TI).

The shared memory needed per thread block depends on the number of threads (warps) in each thread block. Therefore, the GPU performance and utilization also depend on this parameter. However, if tuned accordingly, the Pascal GPU approaches its peak bandwidth with each of its 28 SMs busy with around 64 warps and more, depending on the kernel function, see [23]. Roughly, this means approaching GPU's peak bandwidth since around 80,000 solved particles. This behavior has been observed in dimensions, N_{var}, from 2 to 10, see Fig. 5 c. Switching to multi-GPU after reaching a performance peak of a single GPU might be advised.

When increasing the dimensionality of the problem, the solution latency becomes more dependent on global memory bandwidth, bus size and also data structure. Total amount of shared memory requested, however, ultimately sets the upper bound on the size of the problem solved. Switching to a multi-GPU execution after reaching the shared memory limit might be advised.

```
   Input:
1  *gpuGlobMemAccels, *gpuGlobMemCoords,
   *gpuGlobMemStresses, *gpuGlobMemEnergies

   /* Allocate arrays in shared memory                                          */
   *ShMemTargetCoords, *ShMemSourceCoords,

   *ShMemTargetAccels, *ShMemPartialDistances, *ShMemTargetEnergies, *ShMemTargetStresses

   /* Loading static target coordinates                                         */
2  forall the respective threads in block do
3  |  load N_var coordinates of target particle from *gpuGlobMemCoords into *ShMemTargetCoords
4  end
   /* Interaction of source tiles with the target particles                     */
5  for tile → tilesCount do
   |  // loading source coordinates
6  |  if source coords = target coords then
7  |  |  forall the respective threads in block do
8  |  |  |  temporarily switch the pointer *ShMemSourceCoords to *ShMemTargetCoords
9  |  |  end
10 |  else
11 |  |  forall the respective threads in block do
12 |  |  |  load N_var coordinates from *gpuGlobMemCoords into *ShMemSourceCoords
13 |  |  end
14 |  end
   |  // computing tile interaction
15 |  forall the respective threads in block do
16 |  |  forall the particles in tile do
17 |  |  |  compute contributions of: N_var repelling forces, radial stress, potential energy
18 |  |  end
19 |  end
20 end
```

Algorithm 1. Tile interaction in shared memory.

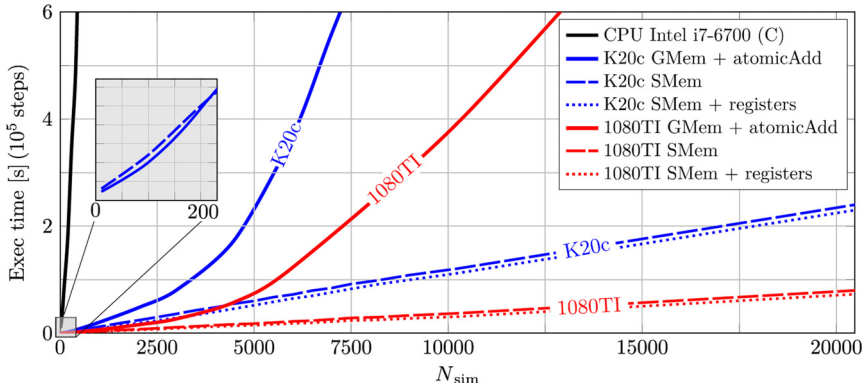

Fig. 4. Performance comparison of the all-pairs $\mathcal{O}(N^2)$ solution while scaling N_{sim} in constant dimension of $N_{var} = 2$.

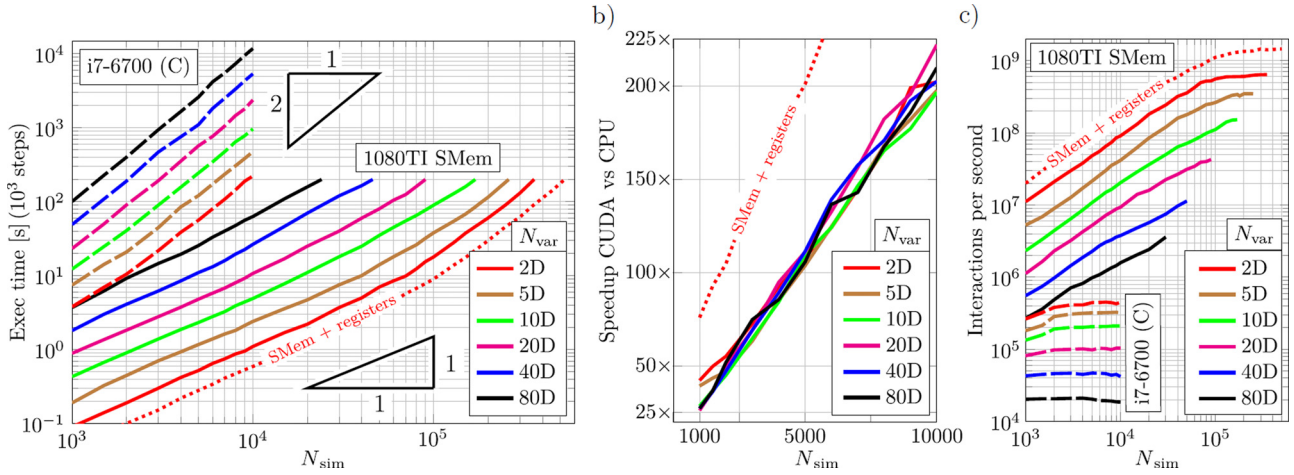

Fig. 5. a) Performance comparison of the $\mathcal{O}(N^2)$ solution as executed by CPU (dashed) and GPU (solid and dotted lines) while scaling both N_{sim} and N_{var}. b) The achieved speedup of solution. c) Comparison of performance in interactions solved per second.

In case of the global memory implementation, see Section 6.2, its limits are set by the size of global memory (11GB on GTX 1080TI). Register usage per thread in kernel functions is rather low (20 registers at most) compared to the maximum of 255 registers per thread on the GTX 1080TI.

8. Conclusion

The presented paper deals with a parallel implementation of a dynamical system of mutually repelling particles assembled as a physical analogy of the Audze-Eglājs and ϕ optimization criteria for obtaining space-filling designs. The unusual property of the particle system of interest is the arbitrary dimension of the design space which stipulates specific restrictions of utilization of GPU's fast on-chip resources. Two fundamentally different parallel implementations are therefore developed and their performance is compared to the initial single-thread CPU implementation.

The parallel implementation proposing utilization of the on-chip memory for a serialized interaction of tiles of particles is built on the concept [22]. The on-chip memory allocation is however a subject of refinement due to the unknown dimensionality of the particle system. The usage of registers for data storage is argued to be inappropriate as well as the utilization of L1 cache is avoided for it has been unified with the texture cache since the Maxwell architecture. Therefore, the usage of the shared memory is proposed for both maintaining the computed data as well as for circulation of tiles of particles.

Along with the on-chip solution, an entirely general implementation has been developed, utilizing solely GPU's on-board global memory. It investigates the approach of unfolding all the possible parallelism into the global memory and hiding the memory latency by computing all possible data in parallel. Large reductions required are handled by fast atomic additions.

Despite the shared memory resources being rather limited, the on-chip implementation turns out to be the best performing solution for the expected range of dimensionality, exhibiting mild linear dependency of execution time on both number of particles, N_{sim}, and number of dimensions, N_{var}.

Generally, however, both GPU implementations provide a major speedup of such a generalized particle system solution compared to the CPU. Although the actual GPU code is going to be a subject of further optimization efforts, the parallelized solution is already capable to serve its research purposes; to compute many runs of vast optimization scenarios.

Acknowledgment

This publication was supported by the Czech Science Foundation under the project no. 16-22230S. This support is gratefully acknowledged.

Appendix A. Derivation of equations of motion

The formulation of the potential energy of a system of charged particles as proposed by the (P)AE criterion, see Section 2, will further be generalized and utilized for derivation of equations of motion of the undamped dynamical particle system. The derivation itself may be conducted by using various approaches, all of which lead, of course, to identical results. Essential remarks about derivation using Lagrangian mechanics are provided in what follows.

To begin, let us state that the dynamical behavior of a mechanical system with a finite number of degrees of freedom can be described by the Lagrange's function, or shortly Lagrangian, $\mathscr{L}$. Sometimes also called a *kinetic potential*, the Lagrangian $\mathscr{L}$ is a functional; a sum of formulations of kinetic and potential energy. In case of the PAE-conditioned dynamical system, the Lagrangian can be described as follows:

$$\mathscr{L} = E_k + E_p, \tag{A.1}$$

with the kinetic energy of the particle system E_k being a simple sum of kinetic energies of all particles of equal mass m:

$$E_k = \frac{1}{2} m \sum_{i=1}^{N_{\text{sim}}} \sum_{v=1}^{N_{\text{var}}} \dot{x}_{i,v}^2, \tag{A.2}$$

where $\dot{x}_{i,v} = \frac{\mathrm{d}}{\mathrm{d}t} x_{i,v}$ is the velocity of ith particle in dimension v.

We now consider a generalized formulation of the potential energy, E_p, employing an arbitrary value of the exponent, q, upon the mutual distances, $\overline{L}_{ij}$. The potential energy of the model can be written as a sum of energies stored within all mutual inter-particle interactions:

$$E_p = \sum_{i=1}^{N_{\text{sim}}-1} \sum_{j=i+1}^{N_{\text{sim}}} \frac{1}{\overline{L}_{ij}^q}, \tag{A.3}$$

where the power has been changed to a general integer, q, (similarly to the ϕ-criterion [8]) and the metric considered is the periodic length, $\overline{L}_{ij}$, see Eq. (5).

Further, it is needed to calculate the derivatives of Lagrangian $\mathscr{L}$ with respect to all state variables. In the case at hand, the state variables are the coordinates $x_{i,\,v}$ and velocities $\dot{x}_{i,v}$ of all particles in each dimension. Obeying the Lagrange's equations of the second kind:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathscr{L}}{\partial \dot{x}_{i,v}}\right) = \frac{\partial \mathscr{L}}{\partial x_{i,v}}, \tag{A.4}$$

one can start off with the assumption that, apart from the derivatives with respect to the time t, the kinetic energy E_k is further differentiable only with respect to velocities $\dot{x}_{i,v}$ and the potential energy E_p (Eq. (A.3)) is differentiable only with respect to coordinates $x_{i,\,v}$. Therefore, the left-hand side of Eq. (A.4) is rather easily obtainable:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathscr{L}}{\partial \dot{x}_{i,v}}\right) = \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial E_k}{\partial \dot{x}_{i,v}}\right) = m\,\ddot{x}_{i,v}, \tag{A.5}$$

with $\ddot{x}_{i,v} = \frac{\mathrm{d}}{\mathrm{d}t}\dot{x}_{i,v}$ being the acceleration of the ith particle in the dimension v.

The right-hand side of Eq. (A.4) becomes:

$$\frac{\partial \mathscr{L}}{\partial x_{i,v}} = \frac{\partial E_p}{\partial x_{i,v}} = \sum_{j\neq i}^{N_{\text{sim}}} \left(\frac{1}{\overline{L}_{ij}^{q+1}} \frac{\overline{\Delta}_{ij,v}}{\overline{L}_{ij}}\right). \tag{A.6}$$

The motion of the dynamical system is therefore described by a system of independent equations. This awareness is of high importance while considering the possibilities for solution method and its computer implementation. This means that each acceleration $\ddot{x}_{i,v}$ can be solved separately, without solving a system of equations.

The resulting equation of motion of ith particle in vth dimension as assembled from Eqs. (A.4)–(A.6) finally reads:

$$\ddot{x}_{i,v} = \frac{1}{m} \sum_{j\neq i}^{N_{\text{sim}}} \frac{\overline{\Delta}_{ij,v}}{\overline{L}_{ij}^{q+2}}, \tag{A.7}$$

which is identical to Eq. (6). Note that these are equations of motion of a conservative dynamical system as defined by the energy potential (A.1) which does not cover any form of energy dissipation.

References

[1] Belleman RG, Bédorf J, Zwart SFP. High performance direct gravitational n-body simulations on graphics processing units ii: an implementation in cuda. New Astron 2008;13(2):103–12.

[2] Gaburov E, Harfst S, Zwart SP. Sapporo: a way to turn your graphics cards into a grape-6. New Astron 2009;14(7):630–7.

[3] Dindar S, Ford EB, Juric M, Yeo YI, Gao J, Boley AC, et al. Swarm-ng: a cuda library for parallel n-body integrations with focus on simulations of planetary systems. New Astron 2013;23:6–18.

[4] Koksma JF. Een algemeene stelling uit de theorie der gelijkmatige verdeeling modulo 1. Mathematica B 1942/1943;11:7–11.

[5] Fang K-T, Ma C-X. Wrap-around L_2-discrepancy of random sampling, latin hypercube and uniform designs. J Complex 2001;17(4):608–24. http://dx.doi.org/10.1006/jcom.2001.0589.

[6] Niederreiter H. Random number generation and Quasi-Monte carlo methods. Philadelphia, Pennsylvania: Society for Industrial and Applied Mathematics0-89871-295-5; 1992.

[7] Audze P, Eglājs V. New approach for planning out of experiments. Probl. Dyn. Strengths 1977;35:104–7. (in Russian).

[8] Morris MD, Mitchell TJ. Exploratory designs for computational experiments. J Stat Plan Inference 1995;43(3):381–402. http://dx.doi.org/10.1016/0378-3758(94)00035-T.

[9] Johnson M, Moore L, Ylvisaker D. Minimax and maximin distance designs. J Stat Plan Inference 1990;2(26):131–48. http://dx.doi.org/10.1016/0378-3758(90)90122-B.

[10] Eliáš J, Vořechovský M. Modification of the audze–eglājs criterion to achieve a uniform distribution of sampling points. Adv Eng Software 2016;100:82–96.

[11] Vořechovský M, Eliáš J. Improved formulation of audze-eglājs criterion for space-filling designs. In: Haukaas T, editor. ICASP12, the 12th international conference on applications of statistics and probability in civil engineering held in vancouverCanada on July 12–15, 2015: The University of British Columbia; 2015. p. 1–8. http://dx.doi.org/10.14288/1.0076173.

[12] Mašek J, Vořechovský M. Parallel implementation of dynamical particle system for cuda. In: Iványi P, Topping BHV, Várady G, editors. Proceedings of the fifth international conference on parallel, distributed, grid and cloud computing for engineering Civil-Comp Press, Stirlingshire, UK, Paper 34; 2017. http://dx.doi.org/10.4203/ccp.111.34.
[13] Mašek J, Vořechovský M. On the influence of the interaction laws of a dynamical particle system for sample optimization. Trans VŠB â Tech Univ Ostrava 2017;17(1):137–46. http://dx.doi.org/10.1515/tvsb-2017-0016.
[14] Vořechovský M, Novák D. Correlation control in small sample monte carlo type simulations i: a simulated annealing approach. Probab Eng Mech 2009;24(3):452–62https://doi.org/10.1016/j.probengmech.2009.01.004.
[15] von Hoerner S. Die numerische integration des N-Körper-Problemes für sternhaufen. i. Zeitschrift für Astrophysik 1960;50(10–11):184–214.
[16] Barnes J, Hut P. A hierarchical o (n log n) force-calculation algorithm. Nature 1986;324(6096):446–9.
[17] Greengard L, Rokhlin V. A fast algorithm for particle simulations. J Comput Phys 1987;73(2):325–48.
[18] White CA, Head-Gordon M. Derivation and efficient implementation of the fast multipole method. J Chem Phys 1994;101(8):6593–605.
[19] Kirk D, et al. Nvidia cuda software and gpu parallel computing architecture. ISMM. 7. 2007. p. 103–4.
[20] Hamada T, Iitaka T. The chamomile scheme: an optimized algorithm for n-body simulations on programmable graphics processing units. arXiv:astro-ph/0703100 2007.
[21] Hamada T. Internals of the cunbody-1 library: particle/force decomposition and reduction. Princeton; 2007. AstroGPU 2007.
[22] Nyland L, Harris M, Prins J, et al. Fast n-body simulation with cuda. GPU gems 2007;3(31):677–95.
[23] Volkov V. Understanding latency hiding on gpus. University of California, Berkeley; 2016. Ph.D. thesis.
[24] Cook S. CUDA programming: a developer's guide to parallel computing with GPUs. Newnes; 2012.
[25] Harris M. Optimizing cuda. SC07: High Performance Computing With CUDA 2007.
[26] Cheng J, Grossman M, McKercher T. Professional CUDA c programming. John Wiley & Sons; 2014.