Eur. Phys. J. Special Topics **210**, 201–216 (2012)

DOI: 10.1140/epjst/e2012-1647-6

THE EUROPEAN PHYSICAL JOURNAL SPECIAL TOPICS

Review

A pilgrimage to gravity on GPUs

J. Bédorf [a] and S. Portegies Zwart

Leiden Observatory, Leiden University, PO Box 9513, 2300 RA Leiden, The Netherlands

Received 30 April 2012 / Received in final form 25 June 2012
Published online 6 September 2012

Abstract. In this short review we present the developments over the last 5 decades that have led to the use of Graphics Processing Units (GPUs) for astrophysical simulations. Since the introduction of NVIDIA's Compute Unified Device Architecture (CUDA) in 2007 the GPU has become a valuable tool for N-body simulations and is so popular these days that almost all papers about high precision N-body simulations use methods that are accelerated by GPUs. With the GPU hardware becoming more advanced and being used for more advanced algorithms like gravitational tree-codes we see a bright future for GPU like hardware in computational astrophysics.

1 Introduction

In this review we focus on the hardware and software developments since the 1960s that led to the successful application of Graphics Processing Units (GPUs) for astronomical simulations. The field of N-body simulations is broad, so we will focus on direct N-body and hierarchical tree-code simulations since in these two branches of N-body simulations the GPU is most widely used. We will not cover cosmological simulations despite this being one of the computationally most demanding branches of N-body simulations, however the GPU usage is negligible.

There are many reviews about N-body simulations which all cover a specific branch or topic, some of the most recent reviews are those by Dehnen and Read [12] with a focus on used methods and algorithms as well as the work of Yokota and Barba [53] which especially focus on Fast Multipole Methods and their implementation on GPUs.

In this review we follow a chronological approach divided into two parallel tracks: the collisional direct N-body methods and the collisionless tree-code methods. For the direct simulations we partially follow the papers mentioned by D. Heggie and P. Hut [25,28] as being noteworthy simulations since the 1960s. These papers and the number of bodies used in those simulations are presented in Fig. 1 (adapted from [25,28]); in the figure we show the number of bodies used and Moore's law which is a rough indication of the speed increase of computer chips [35].

Since direct N-body methods scale as $\mathcal{O}(N^2)$ it is understandable that the number of bodies used do not follow Moore's law in Fig. 1. However, in Fig. 2 we show that

[a] e-mail: bedorf@strw.leidenuniv.nl

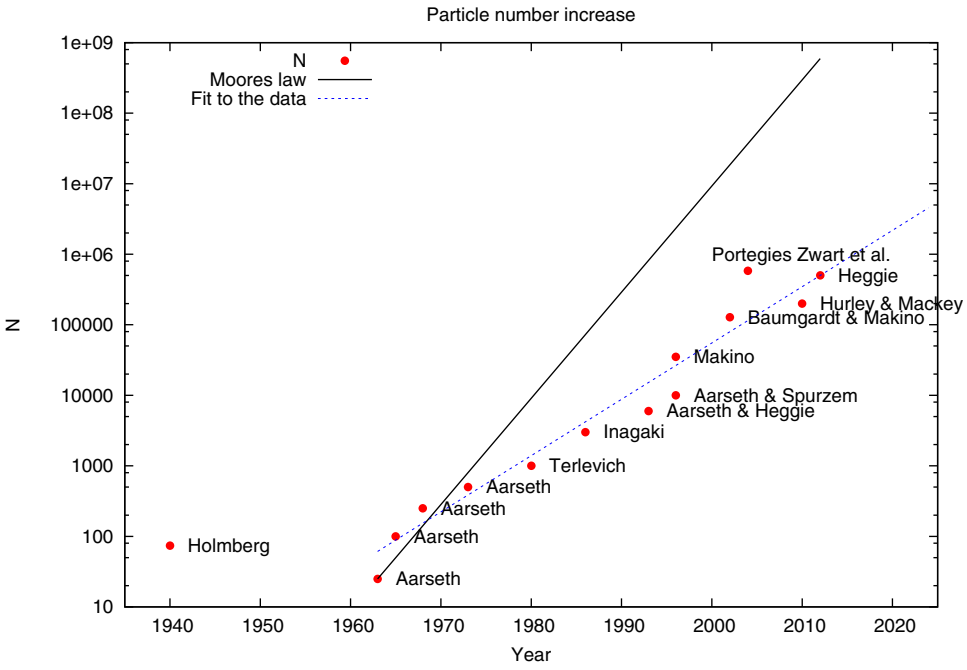

Fig. 1. Number of particles used in collisional simulations over the last 4 decades. The solid line shows Moore's law [35], the circles publications and the dashed-line a fit through the data points. (Adapted from [25,28].)

the increase in the number of bodies is faster than would be explainable by increase in computer speed alone. We show the theoretical number of operations, which is N^2 in the naive situation. And the number of transistors which is an indication of the speed of the computer, whereby we set the start year to 1963. If the increase in N was solely based on the increase in computer speed the line would be horizontal and equal to 1. Everything above 1 indicates that the improvement comes from software or hardware of which the speed doubles every 18 months according to Moore's law. In the figure we tried to indicate (with arrows) what the major reasons for improvements were.

2 The very beginning

Before the first computer simulations the Swedish astronomer Erik Holmberg [26] published simulations of two interacting galaxies which were conducted using light bulbs. In his experiment each galaxy was represented by 37 light bulbs. Holmberg then measured the brightness of the light bulbs, which falls off with $\frac{1}{r^2}$, to compute the gravitational forces and let the galaxies evolve. In Fig. 3 we show one of his results where spiral arms develop, because of the interactions between two galaxies. This experiment was specifically tailored for one problem, namely gravitational interaction, which made it difficult to repeat using other more general hardware available at the time. So, even though it took a lot of manual labour, it would take almost 20 years before digital computers were powerful enough to perform simulations of comparable size and speed. This is the advantage of tailoring the hardware to the specific problem requirements. 50 years later we see the same advantage with the introduction of the special purpose GRAPE hardware (see Sect. 5).

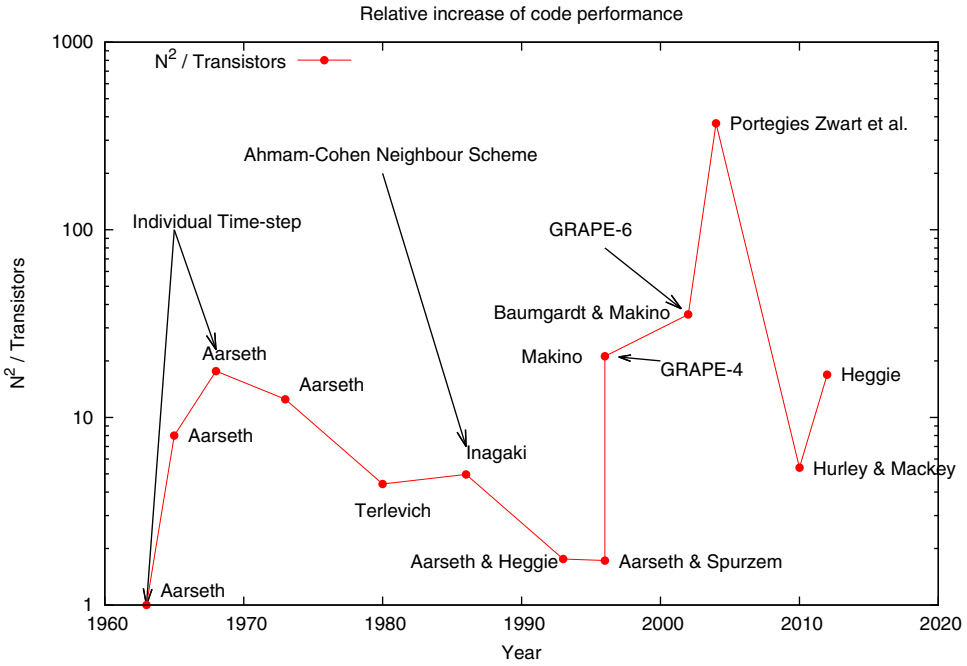

Fig. 2. The number of theoretical operations, N^2, divided by the number of transistors determined using Moore's law where the start year is set and normalized to 1963.

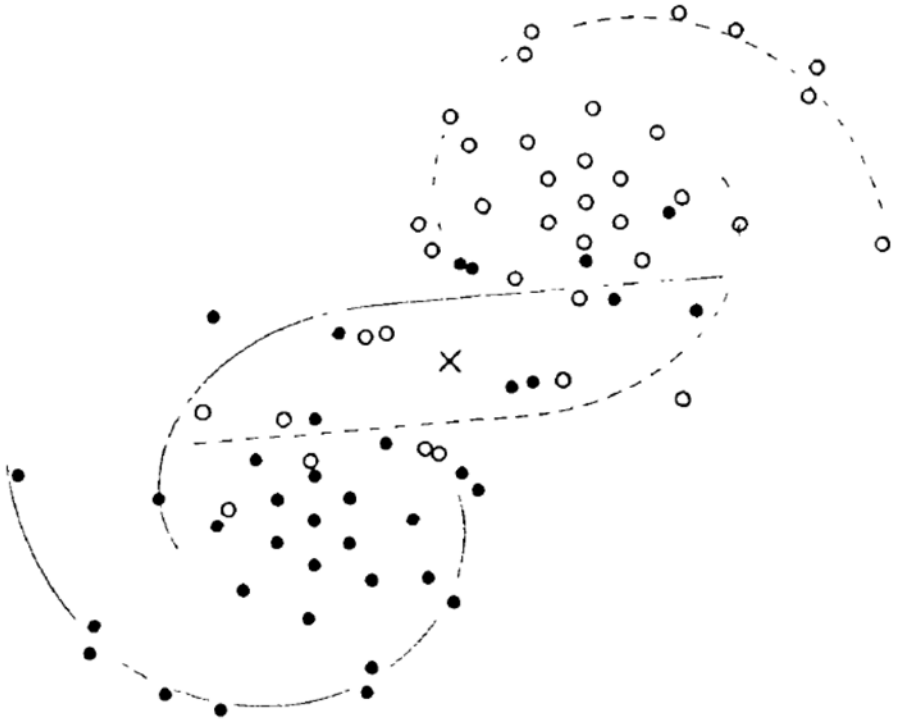

Fig. 3. Spiral arms formed in the experiments by Holmberg in 1941. Image taken from [26].

3 1960–1986: The era of digital computers

The introduction of general purpose digital computers in the 1960s made it easier to buy and use a computer to perform simulations of N-body systems. The digital computers were based on transistors instead of vacuum tubes which made them cheaper to produce and maintain. The first computer simulations of astrophysical N-body systems were performed by von Hoerner in 1960 [51], Aarseth in 1963 [2] and van Albada in 1968 [50]. The number of bodies involved in these simulations was still relatively small and comparable to the experiment of Holmberg ($N = 10$ to 100).

During the first decades of the digital computer there were two ways to increase the number of bodies. One method was to buy a faster computer which allowed you to keep using the same software but increase the number of used particles. This was an efficient method in the sense that the speed of the computer doubled roughly every 18 months, following Moore's Law [35][1]. Another method to increase N was by improving the software either by code optimizations or by algorithmic changes. In direct N-body integrations the number of required interactions scales as N^2 so any improvement to reduce the number of operations is very welcome.

In 1963 Aarseth [2] introduced the individual time-step scheme. To simulate an N-body system the orbits of particles have to be followed exactly. However, particles isolated in space do not have sudden changes in their orbit and therefore can take longer time-steps than particles in the core of the cluster or which are part of a binary. Particles forming a binary change their positions quickly and therefore require many more time-steps to be tracked accurately. This is the basic idea behind the individual time-step scheme, each particle is assigned a simulation time when it is required to update and recompute the gravitational force. When only a few particles take small time-steps then for most of the particles the gravitational forces do not have to be computed. Thereby going from N^2 operations in a shared time-step scheme to $N \cdot N_{\rm active}$ operations where $N_{\rm active}$ is the number of particles that have to be updated. If $N_{\rm active}$ is sufficiently small then the overhead of keeping track of the required time-steps is negligible compared to the gain in speed by not having to compute the gravitational forces for all bodies in the system.

The Ahmam-Cohen Neighbour Scheme (ACS) introduced in 1973 [5] takes another approach to reduce the number of required computations. In ACS the gravitational force computation is split into two parts. In the first part the force between a particle and its nearest neighbours, N_{nn}, (hence the name) is computed in a way similar to the direct N-body scheme with many small time-steps, because of the fast changing dynamical nearby neighbourhood. In the second part the force from the particles that are further away is updated less frequently, since the changes to that part of the gravitational force are less significant. When $N_{nn} \ll N$ the number of total interactions is reduced dramatically and thereby the total wall-clock time required for the simulation is reduced.

With the introduction of the digital computer also came the introduction of parallel computing. You can distinguish fine grained and coarse grained parallelisation. The former focuses on tasks that require many communication steps whereas the latter splits the computational domain and distributes it among different processors. These processors can be in the same machine or connected by a network. When connected by a network the communication is slower and therefore only beneficial if the amount of communication is minimal. In the early years of computing the focus was on fine grained parallelism using vector instructions. These instructions helped to increase

[1] Technically Moore does not describe the speed of the computer, but the number of transistors. In practice the speed of a computer chip is roughly related to the number of transistors.

the number of bodies in the simulations, but still N increased much slower than the theoretical speed of the processors. This is because of the N^2 scaling of direct N-body algorithms. Some of the noteworthy publications were the simulation of open clusters containing 1000 stars by Terlevich in 1980 [49] and the simulation of globular clusters using up to 3000 particles by Inagaki in 1986 [29] using the (at the time) commonly used `NBODY5` code.

4 1986–2000: Advances in software

In 1986 Barnes & Hut [7] introduced a collisionless approximation scheme based on a hierarchical data structure. With the introduction of the BH Tree-code we see two parallel tracks in N-body simulations: the first, using high precision direct N-body methods and the second using approximation methods thereby allowing for larger particle simulations. For more information about the collisionless methods see Sect. 7.

The individual time-step method reduced the total number of executed gravitational force computations, since particles are only updated when required. However, if you would use the individual time-step method in a predictor-corrector integration scheme[2] you would have to predict all N particles to the new time while only computing the gravitational force for one particle. This prediction step results in a large overhead and the possibilities to parallelise the algorithm are limited. Since the gravitational force is computed for only one particle. In a shared time-step method there are N particles for which the force is computed which can then be divided over multiple processors. A solution came in the form of the block time-step method in which particles with similar time-steps are grouped together. These groups are then updated using a time-step that is suitable for each particle in the group. Since multiple particles are updated at the same time, the number of prediction steps is reduced and the amount of parallel work is increased [33]. This is an example of i-particle parallelisation in which all j-particles are copied to all nodes and each node works on a subset of the i-particles. The i-particles are the sinks and the j-particles are the sources for the gravitational forces. In hindsight, it might have been more efficient to use j-particle parallelisation in which each processing node would get a part of the total particle set. The i-particles that have to be updated during a time-step are then broadcast to each node. The nodes then compute the gravitational force on those i-particles using their subset of j-particles and finally in a reduction step these partial forces are combined. With the introduction of the GRAPE hardware a few years later (see below), it turned out that this i-particle parallelisation was ideal for special purpose hardware.

The main focus in the development of N-body codes was on increasing N in order to get increasingly detailed simulations, although some research groups focused on specialized problems like planetary stability. The group of Gerry Sussman developed a special machine just for integrating the solar system, The `Digital Orrery` [6]. The machine consisted of a set of specially developed computer chips placed on extension boards which were connected using a special ring network. A photo of the machine is shown in Fig. 4. With this machine the developers were able to find previously unknown chaotic motions in the orbit of Pluto, caused by resonance with Neptune [46].

In 1993 Aarseth & Heggie [4] published the results of a 6000 body simulation containing primordial binaries and unequal mass particles. With this simulation it

[2] In a predictor-corrector scheme the positions are updated in multiple steps. First you predict the new positions using the original computed gravitational forces, then you compute the new gravitational forces using these positions and then you apply a correction to the predicted positions.

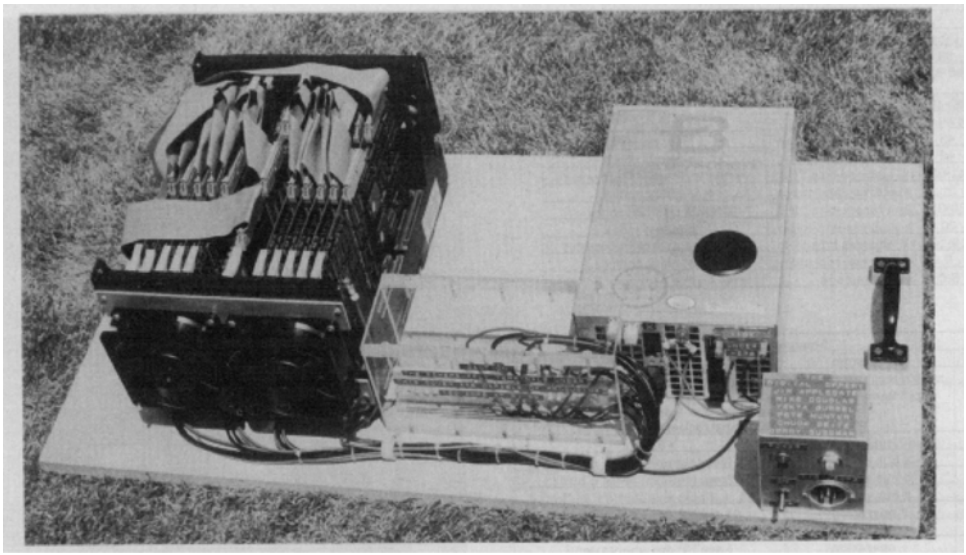

Fig. 4. The digital Orrery. Image taken from [6].

was possible to improve on previous results where only equal mass particles were used. The differences in the unequal mass and equal mass simulations were small, but too large to be ignored, which shows the critical importance of binaries and initial mass functions even though they are computationally expensive.

Spurzem and Aarseth [43] performed a simulation of a star cluster with 10^4 particles in 1996. The simulation was executed on a CRAY machine. This is one of the last simulations in our review that was executed without any special purpose hardware. Since in the same year, Makino *et al.* [30] presented their work which used three times more particles and was executed on GRAPE hardware.

5 2000–2006: The era of the GRAPE

The introduction of the special purpose GRAvity PipE (GRAPE) hardware caused a breakthrough in direct-summation N-body simulations [31]. The GRAPE chips have the gravitational force calculations implemented in hardware which results in a speed-up of two orders of magnitude compared to the standard software implementations. The GRAPE chips were introduced in the early 1990s [16], but it would take a few years and development cycles before they were widely accepted and being used in production simulations. The GRAPE chips are placed on a PCI-expansion card that can be installed in any general purpose (desktop) computer. The GRAPE came with a set of software libraries that made it relatively easy to add GRAPE support to existing simulation software like NBODY4 [3] and Starlab [41]. The block time-stepping scheme introduced a few years earlier turned out to be ideal for this hardware and when multiple GRAPE chips were used one could combine this in $i-j$-particle parallelisation.

In the early 90s the large computational cost of direct N-body simulations had caused researchers to start using collisionless codes like the Barnes-Hut tree-code (see Sect. 7) in order to do large N simulations. The introduction of the GRAPE, combined with the availability of ready-to-use software caused the opposite effect since suddenly it was possible to do collisional simulations at the same speed as

collisionless simulations. The last generation of the fixed function GRAPE hardware was the GRAPE-6 chip. These were the most commonly used GRAPE chips and when placed on a GRAPE-6Af extension board they had a peak performance of ∼131 GFLOPs and enough memory to store 128k particles.

The GRAPE is designed to offer large amounts of fine grained parallelism since the on-chip communication is fast and specifically designed for the gravity computations. Supercomputers on the other hand are designed for coarse grained parallelisation thereby offering a large amount of computational cores connected using fast networks. But the communication times are still orders of magnitude slower than on-chip communication networks. This means that supercomputers are rarely used for direct N-body simulations and are much more suitable for collisionless simulations (Sect. 7). It would require hundreds of normal processor cores to reach the same performance as the GRAPE offers on one extension board and that is even without taking into account the required communication time. If this is taken into account then the execution time on supercomputers is unrealistically high for high precision (e.g. many small time-steps with few active particles) direct N-body simulations.

In the 2000s it was clear that parallelisation had become one of the requirements to be able to continue increasing the number of particles, because hardware manufacturers shifted their focus from increasing the clock-speed to increasing the number of CPU cores and the introduction of special vector instructions[3]. The clock speed came near the physical limit of the silicon and CPUs used so much energy that the produced heat became a serious problem. For direct N-body simulations with individual or block time-steps often a combination of fine grained and coarse grained parallelisation is used (depending on, for example, the number of particles that is integrated). When the number of particles that have to take a gravity step is small then it is more efficient to not use the external network, but rather let all the work be handled by one machine. On the other hand if the number of particles taking a gravity step is large it could be more efficient to distribute the work over multiple machines.

The introduction of Streaming SIMD Extensions (SSE) vector instructions in modern day processors promised to give a performance boost for optimized code. However this optimization step required deep technical knowledge of the processor architecture. With the introduction of the Phantom GRAPE library by Keigo Nitadori [36] it became possible to benefit from these instructions without having to write the code yourself. As the name suggests the library is compatible with software written for GRAPE hardware, but instead executes the code on the host processor using the special vector instructions for increased performance. Recently this is extended with the new Advanced Vector eXtensions (AVX) which allows for even higher performance on the latest generation of CPUs [47,48].

One of the limitations of the GRAPE is its fixed function pipeline and because of this it can not be used for anything other than gravity computations. For example in the Ahmam-Cohen neighbour Scheme the force computation is split into a near and a far force. The GRAPE cards are suitable to speed-up the computation of the far force, but the near force has to be computed on the host since the GRAPE has no facility to compute the force using only a certain number of neighbours. An alternative like Field Programmable Gate Arrays (FPGAs) allows for more flexibility while still offering high performance at low energy cost, since the hardware can be programmed to match the required computations. The programming is complex, but the benefits can be high since the required power is usually much less than a general purpose CPU cluster offering similar performance. An example of FPGA cards are the MPRACE cards [44], which are designed to speed-up the computation of the neighbour forces

[3] Like Streaming SIMD Extensions (SSE) and Advanced Vector Extensions (AVX).

and thereby eliminating the need to compute the near force on the host computer which would become a bottleneck if only GRAPE cards would be used.

With the increasing availability of the GRAPE hardware at different institutes came the possibility to combine multiple GRAPE clusters for large simulations. A prime example of this is the work by Harfst *et al.* [24] who used two parallel supercomputers which were equipped with GRAPE-6A cards. They showed that for direct N-body simulations it was possible to reach a parallel efficiency of more than 60% and reached over 3TFLOPs of computational speed when integrating 2×10^6 particles. Though the number of GRAPE devices was increasing it was still only a very small fraction of the number of "normal" PCs that was available. In order to use those machines efficiently one had to combine them and run the code in parallel. An example of this is the parallelisation of the N-body integrator in the `Starlab` package [39]. This work showed that it was possible to run parallel N-body simulations over multiple computers, although it was difficult to get good enough scaling in order to compete with the GRAPE hardware. This was also observed in earlier work by Gualandris *et al.* [20] who developed different parallel schemes for N-body simulations thereby observing that the communication time would become a bottleneck for simulations of galaxy size systems. Another approach is the work by Dorband *et al.* [13] in which they implemented a parallel scheme that uses non-blocking communication. They called this a systolic algorithm, since the data rhythmically passes through a network of processors. Using this method they were able to simulate 10^6 particles using direct N-body methods.

6 2006–Today: The era of commercial high performance processing units

With the introduction of programmable Graphics Processing Units (GPU) in 2001 (NVIDIA's Geforce 3) it became possible to program high performance chips using software. However, it would take another 7 years before GPUs were powerful enough to be a viable alternative to the fixed function GRAPE that dominated the N-body simulation field over the previous decade. The GPU was originally designed to improve the rendering speed of computer games. However, over the years these cards became progressively faster and, more importantly, they became programmable. At first one had to use programming languages which were specially designed for the creation of visual effects (e.g. `Cg` and `OpenGL`). The first use of the GPU for N-body simulations was by Nyland *et al.* in 2004 [37] who used `Cg`. Their implementation was mostly a proof-of-concept and lacked advanced time-stepping and higher order integrations which made it unsuitable for production quality N-body simulations. Programming GPUs became somewhat easier with the introduction of the BrookGPU [10] programming language. This language is designed to be a high level programming language and compiles to the `Cg` language. In 2006 Elsen et al. [14] presented an N-body implementation using the BrookGPU language. Around the same time Portegies Zwart *et al.* published an implementation of a higher order N-body integration code with block time-steps written in Cg [42]. Although these publications showed the applicability and power of the GPU, the actual programming was still a complicated endeavor. This changed with the introduction of the Compute Unified Device Architecture (CUDA) programming language by NVIDIA in early 2007. The language and compatible GPUs were specifically designed to let the power of the GPU be harvested in areas other than computer graphics. Shortly after the public release of CUDA efficient implementations of the N-body problem were presented [9,21,38]. Belleman *et al.* [9] showed that it was possible to use the GPU with CUDA for high order

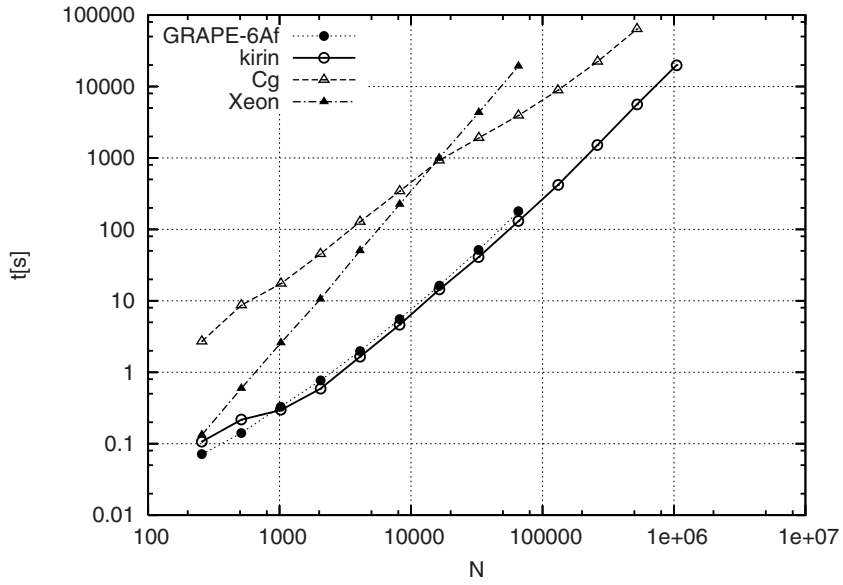

Fig. 5. Performance comparison of N-body implementations. The CUDA GPU implementation `kirin` is represented by the solid line (open circles). The GRAPE is represented as the dotted line (bullets). The Cg GPU implementation is represented as the dashed line (open triangles). The dashed-dotted line (closed triangles) represent the results on the host computer. Figure taken from [9].

N-body methods and block-time steps with an efficiency comparable to the, till then, unbeaten GRAPE hardware (see Fig 5).

With the GRAPE being around for over 15 years most of the production quality astrophysical N-body simulation codes were using the GRAPE when NVIDIA released CUDA. It was therefore relatively easy to shift from the GRAPE to the GPU with the introduction of the GRAPE-compatible GPU library `Sapporo` [17]. This library uses double-single precision[4] and on-device execution of the prediction step, just like the GRAPE-6Af hardware.

The GPU chips are produced in much higher volumes than the GRAPE chips which makes the GPU more cost efficient to produce and cheaper to buy. This, combined with more on-board memory, higher computational speed and the option to reprogram them to your specific needs, is the reason that nowadays more GPUs than GRAPEs are used for N-body simulations. Even though the GRAPE, because of its dedicated design, requires less power than the GPU.

The GRAPE-DR (Greatly Reduced Array of Processor Elements with Data Reduction) [32] is different from the earlier generation GRAPE chips since it does not have the gravitational computations programmed in hardware, but rather consists of a set of programmable processors. This design is similar to how the GPU is built up, but uses less power since it does not have the overhead of graphic tasks and visual output that GPUs have. At the time of its release in 2007 the GRAPE-DR was about two times faster for direct N-body simulations than the GPU.

In Nitadori & Aarseth 2012 (private communication) the authors describe their optimisations to the NBODY6 and NBODY7 [1] simulation codes to make use of the GPU. They have tested which parts of the code are most suitable to be executed

[4] This technique gives precision up to the 14th significant bit while using single precision arithmetic.

on the GPU and came to the conclusion that it was most efficient to execute the so-called 'regular force' on the GPU. This step involves around 99 percent of the number of particles. On the other hand the local force is executed on the host using vector instructions, since using the GPU for this step resulted in a communication overhead which is too large [5]. In this division the bulk of the work is executed on the GPU and the part of the algorithm that requires high precision, complex operations or irregular memory operations is executed on the host machine possibly with the use of special vector instructions. This is a trend we see in many fields where the GPU is used. Although some authors overcome the communication overhead by implementing more methods on the GPU besides the force computation [45].

In 2009 the Khronos group[6] introduced the OpenCL programming language. The language is designed to create parallel applications similar to the way CUDA is used for GPUs, with the difference that programs written in OpenCL also work on systems with only host CPUs. The idea behind this is that the developer has only to write and maintain one software program. In reality, however, the developer will have to write code that is optimized for one platform (GPU or CPU) in order to get the highest performance out of that platform. That CUDA was released a couple of years earlier, has more advanced features and has more supported libraries than OpenCL are the reasons CUDA is currently more commonly used than OpenCL. However, there is no reason this cannot change in the future with updated libraries that offer OpenCL support (e.g. `Sapporo2`, Bédorf *et al.* in preparation).

The two most recent large N simulations have been performed by Hurley and Mackey [27] ($N = 10^5$) and Heggie ($N = 5 \times 10^5$)(private communication) who both used NBODY6 in combination with one or more GPUs.

Simulations that are used to determine planetary stability usually involve only a few particles. This severely limits the amount of parallelism and therefore a different approach has to be taken than in large N simulations. An example of a method that takes a different approach is `Swarm-NG`[7]. This is a software package for integrating ensembles of few-body planetary systems in parallel with a GPU. `Swarm-NG` is specifically designed for low N systems. Instead of breaking up one problem into parallel tasks, `Swarm-NG` integrates thousands of few-body systems in parallel. This makes it especially suited for Monte Carlo-type simulations where the same problem is run multiple times with varying initial conditions.

7 Collisionless

In 1986 Barnes & Hut introduced their collisionless approximation scheme based on a hierarchical data structure, which became known as the Barnes-Hut tree-code [7]. In this hierarchical data structure (tree) the particles are grouped together in boxes (see for an example Fig. 6). These boxes get the combined properties of the underlying particle distribution, like center of mass and total mass. To compute the gravitational force on a particle one does not compute the force between that particle and all other particles in the system, but rather between the particle and a selection of particles and boxes. This selection is determined by traversing the tree-structure and per box deciding if the particle is distant enough or whether the box lies so close that we should use the particles that belong to the box. This decision is made by a 'Multipole

[5] This is similar to how the MPRACE project did the division between the GRAPE and MPRACE cards.

[6] The Khronos Group is a group of companies that develops open standards for accelerating graphics, media and parallel computations on a variety of platforms.

[7] `http://www.astro.ufl.edu/~eford/code/swarm/`

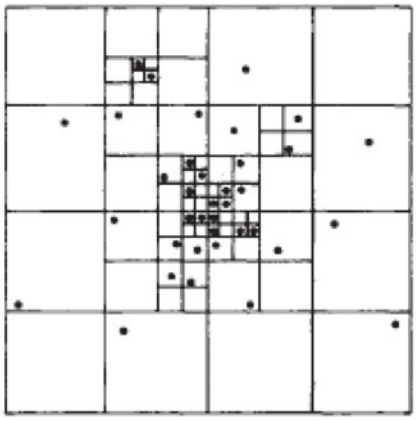

Fig. 6. Particles grouped together in boxes in the tree-code algorithm. Image taken from [7].

Acceptance Criterion' which, in combination with the free parameter θ, is used to get the required precision. In this way one can either get high precision at high computational costs, by using more particles than boxes, or the other way around, lower precision by using more boxes instead of particles. The resulting code that implements this algorithm generally achieves a scaling of $\mathcal{O}(N \log N)$ instead of the $\mathcal{O}(N^2)$ of direct N-body codes.

For large collisionless simulations ($N > 10^7$), supercomputers are required for the available computational resources and for the amount of available memory. Collisionless simulations usually scale as $\mathcal{O}(N \log N)$ in tree methods or even as $\mathcal{O}(N)$ with the Fast Multipole Moment (FMM) methods which allows for simulations with much higher particle numbers [11,53,54]. This requires memory to store the particles, but also computational resources to solve the gravitational equations. The gravity equations can usually be solved using specialized hardware, but those do not have large enough memory buffers to store all particles.

With the introduction of the GRAPE hardware the speed advantage of the tree-codes compared to direct N-body methods was significantly reduced. The tree-code method speeds up the gravity computation, but it still forms the major part of the total computation time. It is therefore beneficial to execute these computations using the GRAPE hardware. By modifying the tree-walk and using a special version of the GRAPE chip, Fukushige et al. [16] were able to execute the computation of the gravitational force of the Barnes & Hut tree-code algorithm using the GRAPE hardware, thereby benefiting from both the fast tree-code algorithm and the efficiency of the GRAPE.

The first result of a tree-code accelerated by a GPU was presented in [9]. The results did show a speed-up compared to the CPU results, however the speed-up was smaller than with direct N-body methods. This is of course understandable since there are fewer force computations that can benefit from the GPU compared to direct N-body methods. Another limiting factor is the amount of communication required between the GPU and CPU in the standard GRAPE tree-code implementations. The high computational speed of the GPU means that communication over the PCI bus can become a bottleneck. In their award-winning papers Hamada et al. [22,23] reduced this overhead by combining multiple tree-walks (executed on the CPU) and transferred these to the GPU in one data transfer. Instead of receiving one set of interaction lists the GPU now receives more than one which increases the amount of parallel work that can be executed and improves the overall efficiency of the GPU. However even with this method the tree-walk is executed on the CPU. In `Octgrav` [18],

the authors execute the tree-walk on the GPU, thereby removing the need to transfer interaction lists between the host and the GPU. Furthermore, since this is a data intensive operation it benefits from the high on device bandwidth of the GPU, which is an order of magnitude higher than that of the CPU.

By removing the traditional bottlenecks of the tree-code algorithm the performance of the code will become limited by new bottlenecks. Parts of the algorithm that took only a few percent of the execution time in the original algorithm suddenly take up a major part of the execution time. In the tree-code method, for example, the construction of the hierarchical data-structure, particle sorting [52] or even just the prediction of the new particle positions become the new bottlenecks. There is only so much one can do to optimize and resolve bottlenecks in an algorithm that scales as $\mathcal{O}(N)$. In the hierarchical GPU tree-code `Bonsai` [8] the authors identified the bottlenecks in `Octgrav` and implemented these on the GPU. This way the algorithms retained their $\mathcal{O}(N)$ and $\mathcal{O}(N \log N)$ scaling, but profit from the high computational speed and bandwidth of the GPU. To prevent any further limits they took it one step further and implemented all parts of the tree-code algorithm on the GPU. This eliminated the need to transfer large amounts of data between the CPU and GPU during each time-step. Using this method the authors are able to develop a GPU tree-code that works efficiently with a shared time-step but also in block time-step hierarchy.

In the parallel version of `Bonsai` the authors need the CPU in order to communicate with other nodes that are connected with each other in a non-shared memory architecture. By overlapping the (network) communication time with computations on the GPU the authors are able to hide most of the required communication overhead of the relatively slow PCI-bus and network cables.

8 BRIDGE; Combining tree and direct

The last few years have seen a huge increase in computational power in the form of special purpose hardware and new supercomputers. And the difference between particle numbers used in collisional and collisionless methods is only increased. Depending on the problem scientists either choose for high precision direct N-body methods or for large particle numbers using approximation methods like the tree-code. Recently, however, methods have been introduced that try to combine the best of both worlds. The high accuracy of direct methods and the speed of tree-codes. In the `BRIDGE` algorithm Fujii et al. [15] combine a direct N-body method and a tree-code to integrate the evolution of star clusters (which requires direct N-body methods) embedded in their host galaxy (which requires an approximation method because of the large number of particles). This allows for detailed simulations in the area of interest while still being able to use large particle numbers. Since the method is based on two well known algorithms it is possible to use existing tools to speed-up the method using GPUs.

With simulation codes becoming more complex and having more advanced features it becomes difficult to add new physics to existing codes without breaking other parts of the codes. This is a common problem in computational sciences and software development in general. Ideas often start off simple, but when something works you want to extend it, which complicates matters. In AMUSE [40] a different approach is taken. Codes that are written for different specific purposes are combined into one framework. This simplifies the development of the separate software products. The other advantage is that you can combine simulation codes that have support for GPUs with codes that do not and thereby still have the speed advantage of using GPUs. With AMUSE it is possible to use the same script using different simulation codes and thereby having the choice between speed, accuracy or available

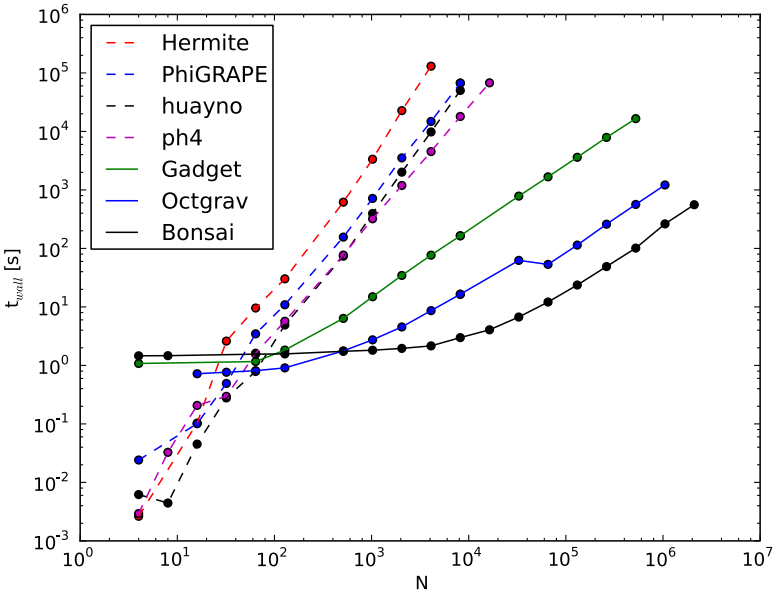

Fig. 7. Performance comparison of a suite of N-body codes. (Taken from S. PZ 2012 in prep.)

hardware. An example of this is shown in Fig. 7 where the execution speed of a set of N-body integration codes is demonstrated. The figure shows the results of 4 direct N-body codes (`Hermite, PhiGRAPE, Huayno and ph4`) and 3 tree-codes (`Gadget2, Octgrav, Bonsai`). Clearly visible is the difference in speed and scaling between the direct code ($\mathcal{O}(N^2)$ scaling) and the tree-codes ($\mathcal{O}(N \log N)$ scaling).

9 The future

With focus shifting to more complex methods and algorithms we see the advantage of the versatility of GPUs and the shift from fixed function methods in the early 90s (like the GRAPE) to programmable chips like GPUs. Even though FPGAs have been around for decades their programming is difficult and expensive, certainly compared to chips that are programmable by software. It is much easier to develop and acquire chips like GPUs, since you can buy them in the computer store around the corner. The availability and price makes the GPU one of the most attractive high performance computing devices that are currently available. It is of course still possible to develop faster chips that require less energy if you make them dedicated, but the development cost and specialized knowledge to build a chip that is competitive against the multi-billion dollar gaming industry is higher than a university research team can afford.

Also, simulation algorithms become more advanced and incorporate different techniques to overcome the painful $\mathcal{O}(N^2)$ scaling. An example of this is the `Pikachu` code by Iwasawa *et al.* (in prep.). In the `BRIDGE` method one has to indicate which particles will be integrated using the direct algorithm and which particles with the tree-code algorithm when the initial conditions are created. The `Pikachu` code improves on this by dynamically deciding which particles can be integrated using a tree-code and which need direct N-body integration.

Fig. 8. The Little Green Machine. The Leiden GPU cluster.

Even though approximation methods (tree-codes, FMM and Particle Mesh) are much faster than direct N-body methods they do not reach the same level of accuracy. With the increase in computational power, direct N-body methods will always be used for new simulations with increasing N either to compare to previous results (e.g. performed with approximate methods) or for new science. The same is valid for the methods used to improve the performance of direct N-body simulations (block time-steps, neighbour schemes, etc.; see Sect. 3). These all have an influence on the precision. Although the difference is smaller than the difference between direct methods and approximations methods it still might be of influence, especially considering the chaotic nature of the N-body problem [19,34]. Therefore, with the increased compute performance we will not only perform simulations with larger N, but also much more detailed simulations with relatively small N to validate previously obtained results. Simulations of globular clusters using high precision shared time-step algorithms are still far out of reach, but one day we will have the computational power to perform exactly this kind of simulations.

The increasing availability of GPUs in supercomputers and in small dedicated GPU clusters (Fig. 8) shows the potential, increased usage and the faith of researchers in GPUs over the last few years. And especially with the installation of GPUs in ordinary desktop computers, as is done, for example, at the Leiden Observatory, this computational power is available at everyone's fingertips without having to request time on expensive supercomputers.

This work was supported by NWO (grants #643.000.802, VICI [#639.073.803], AMUSE [#614.061.608] and LGM [# 612.071.503]), NOVA and the LKBF.

References

1. S. Aarseth, Mergers and ejections of black holes in globular clusters, ArXiv e-prints, (2012)
2. S.J. Aarseth, MNRAS **126**, 223 (1963)
3. S.J. Aarseth, PASP **111**, 1333 (1999)
4. S.J. Aarseth, D.C. Heggie, G.H. Smith, J.P. Brodie, *The Globular Cluster-Galaxy Connection*, edited by G.H. Smith, J.P. Brodie, vol. 48 of *Astronomical Society of the Pacific Conference Series* (January 1993), p. 701
5. A. Ahmad, L. Cohen, J. Comput. Phys. **12**, 389 (1973)
6. J.H. Applegate, M.R. Douglas, Y. Gürsel, P. Hunter, C.L. Seitz, G.J. Sussman, IEEE Trans. Comput. **34**, 822 (1985) ISSN 0018-9340
7. J. Barnes P. Hut, Nature **324**, 446 (1986)
8. J. Bédorf, E. Gaburov, S. Portegies Zwart, J. Comput. Phys. **231**, 2825 (2012)
9. R.G. Belleman, J. Bédorf, S.F. Portegies Zwart, New Astron. **13**, 103 (2008)
10. I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston P. Hanrahan, *ACM SIGGRAPH 2004 Papers, SIGGRAPH '04*, (New York, NY, USA, 2004), p. 777, ACM
11. W. Dehnen, J. Comput. Phys. **179**, 27 (2002)
12. W. Dehnen, J.I. Read, Eur. Phys. J. Plus **126**, 55 (2011)
13. E.N. Dorband, M. Hemsendorf D. Merritt, J. Comput. Phys. **185**, 484 (2003)
14. E. Elsen, M. Houston, V. Vishal, E. Darve, P. Hanrahan, V. Pande, *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06 (New York, NY, USA, 2006), ACM, ISBN 0-7695-2700-0
15. M. Fujii, M. Iwasawa, Y. Funato, J. Makino, PASJ **59**, 1095 (2007)
16. T. Fukushige, T. Ito, J. Makino, T. Ebisuzaki, D. Sugimoto, M. Umemura, Publ. Astr. Soc. Japan **43**, 841 (1991)
17. E. Gaburov, S. Harfst, S. Portegies Zwart, SAPPORO: New Astronomy **14**, 630 (2009), ISSN 1384-1076
18. E. Gaburov, J. Bédorf, S. Portegies Zwart, *International Conference on Computational Science 2010* (Elsevier, 2010)
19. J. Goodman, D.C. Heggie, P. Hut, ApJ **415**, 715 (1993)
20. A. Gualandris, S. Portegies Zwart, A. Tirado-Ramos, P Parallel Computing **33**, 159 (2007), ISSN 0167-8191
21. T. Hamada, T. Iitaka, ArXiv Astrophysics e-prints (2007)
22. T. Hamada, K. Nitadori, *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, (Washington, DC, USA, 2010), p. 1, IEEE Computer Society, ISBN 978-1-4244-7559-9
23. T. Hamada, K. Nitadori, K. Benkrid, Y. Ohno, G. Morimoto, T. Masada, Y. Shibata, K. Oguri, M. Taiji, *Computer Science - Research and Development* (2009)
24. S. Harfst, A. Gualandris, D. Merritt, R. Spurzem, S. Portegies, Zwart P. Berczik, New Astronomy **12**, 357 (2007)
25. D. Heggie, P. Hut, *The Gravitational Million-Body Problem: A Multidisciplinary Approach to Star Cluster Dynamics* (2003)
26. E. Holmberg, ApJ **94**, 385 (1941)
27. J.R. Hurley, A.D. Mackey, MNRAS **408**, 2353 (2010)
28. P. Hut, Dense New Astronomy Reviews **54**, 163 (2010)
29. S. Inagaki, PASJ **38**, 853 (1986)
30. J. Makino, Postcollapse Evolution of Globular Clusters. ApJ **471**, 796 (1996)
31. J. Makino, M. Taiji, *Scientific simulations with special-purpose computers: The GRAPE systems* (Chichester; Toronto: John Wiley & Sons, c1998., 1998)
32. J. Makino, K. Hiraki, M. Inaba, *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, SC '07 (New York, NY, USA, 2007), p. 18, ACM, ISBN 978-1-59593-764-3
33. S.L.W. McMillan, The Vectorization of Small-N Integrators, edited by P. Hut, S.L.W. McMillan, *The Use of Supercomputers in Stellar Dynamics*, vol. 267, Lecture Notes in Physics (Berlin Springer Verlag, 1986), p. 156

34. R.H. Miller, ApJ **140**, 50 (1964)
35. G.E. Moore, Electronics **38** (1965)
36. K. Nitadori, J. Makino, P. Hut. New Astronomy **12**, 169 (2006)
37. L. Nyland, M. Harris, J. Prins. *The rapid evaluation of potential fields using programmable graphics hardware* (Poster, 2004)
38. L. Nyland, M. Harris, J. Prins, GPU Gems **3**, 677 (2007)
39. S. Portegies Zwart, S. McMillan, D. Groen, A. Gualandris, M. Sipior, W. Vermin, New Astronomy **13**, 285 (2008)
40. S. Portegies Zwart, S. McMillan, S. Harfst, D. Groen, M. Fujii, B.Ó. Nualláin, E. Glebbeek, D. Heggie, J. Lombardi, P. Hut, V. Angelou, S. Banerjee, H. Belkus, T. Fragos, J. Fregeau, E. Gaburov, R. Izzard, M. Jurić, S. Justham, A. Sottoriva, P. Teuben, J. van Bever, O. Yaron, M. Zemp. New Astronomy **14**, 369 (2009)
41. S.F. Portegies Zwart, S.L.W. McMillan, P. Hut, J. Makino. MNRAS **321**, 199 (2001)
42. S.F. Portegies Zwart, R.G. Belleman, P.M. Geldof, New Astronomy **12**, 641 (2007)
43. R. Spurzem, S.J. Aarseth, MNRAS **282**, 19 (1996)
44. R. Spurzem, P. Berczik, I. Berentzen, D. Merritt, N. Nakasato, H.M. Adorf, T. Brüsemeister, P. Schwekendiek, J. Steinacker, J. Wambsganß, G.M. Martinez, G. Lienhart, A. Kugel, R. Männer, A. Burkert, T. Naab, H. Vasquez, M. Wetzstein, J. Phys. Conf. Ser. **78**, 012071 (2007)
45. R. Spurzem, P. Berczik, I. Berentzen, K. Nitadori, T. Hamada, G. Marcus, A. Kugel, R. Mãnner, J. Fiestas, R. Banerjee, R. Klessen, Comput. Sci. Res. Development **26**, 145 (2011) `http://www.springerlink.com/index/10.1007/s00450-011-0173-1`
46. G.J. Sussman J. Wisdom, Science **241**, 433 (1988)
47. A. Tanikawa, K. Yoshikawa, K. Nitadori, T. Okamoto, ArXiv e-prints (2012)
48. A. Tanikawa, K. Yoshikawa, T. Okamoto, K. Nitadori, New Astronomy **17**, 8292 (2012)
49. E. Terlevich. N-Body Simulations of Open Clusters, edited by J.E. Hesser, *Star Clusters*, vol. 85 of *IAU Symposium* (1980), p. 165
50. T.S. van Albada, Numerical integrations of the N-body problem, Bull. Astron. Inst. Netherlands **19**, 479 (1968)
51. S. von Hoerner, ZAp **50**, 184 (1960)
52. M.S. Warren J.K. Salmon, *Supercomputing '93: Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, (New York, NY, USA, 1993), p. 12, ACM, ISBN 0-8186-4340-4
53. R. Yokota, L.A. Barba, ArXiv e-prints, August (2011)
54. R. Yokota, J.P. Bardhan, M.G. Knepley, L.A. Barba, T. Hamada, Comput. Phys. Comm. **182**, 1272 (2011)