The Astrophysical Journal, 792:98 (14pp), 2014 September 10 doi:10.1088/0004-637X/792/2/98

EXPANSION TECHNIQUES FOR COLLISIONLESS STELLAR DYNAMICAL SIMULATIONS

Yohai Meiron[1], Baile Li[2], Kelly Holley-Bockelmann[2,3], and Rainer Spurzem[4,1,5]
[1] Kavli Institute for Astronomy and Astrophysics at Peking University, Beijing 100871, China; ymeiron@pku.edu.cn
[2] Department of Physics and Astronomy, Vanderbilt University, Nashville, TN 37235, USA
[3] Department of Physics, Fisk University, Nashville, TN 37208, USA
[4] National Astronomical Observatories of China, Chinese Academy of Sciences, Beijing 100012, China
[5] Astronomisches Rechen-Institut, Zentrum für Astronomie, Universität Heidelberg, Heidelberg D-69120, Germany
Received 2014 February 17; accepted 2014 July 8; published 2014 August 21

ABSTRACT

We present graphics processing unit (GPU) implementations of two fast force calculation methods based on series expansions of the Poisson equation. One method is the self-consistent field (SCF) method, which is a Fourier-like expansion of the density field in some basis set; the other method is the multipole expansion (MEX) method, which is a Taylor-like expansion of the Green's function. MEX, which has been advocated in the past, has not gained as much popularity as SCF. Both are particle-field methods and optimized for collisionless galactic dynamics, but while SCF is a "pure" expansion, MEX is an expansion in just the angular part; thus, MEX is capable of capturing radial structure easily, while SCF needs a large number of radial terms. We show that despite the expansion bias, these methods are more accurate than direct techniques for the same number of particles. The performance of our GPU code, which we call *ETICS*, is profiled and compared to a CPU implementation. On the tested GPU hardware, a full force calculation for one million particles took $\sim$0.1 s (depending on expansion cutoff), making simulations with as many as 10^8 particles fast for a comparatively small number of nodes.

Key word: stars: kinematics and dynamics

Online-only material: color figures

1. INTRODUCTION

A galaxy is a self-gravitating system where stellar dynamics is governed by Newton's law. It could be naively described as a set of $3N_\star$ coupled, second-order, non-linear ordinary differential equations, where $N_\star$ is the number of stars, which ranges between 10^5 and 10^{12} (Binney & Tremaine 2008). Solving such an equation set numerically is practically only possible at the very low end of the $N_\star$-range, and even so is very challenging with current computer hardware. Thus, various techniques are used to simplify the mathematical description of the system; these are often designed to fit a particular problem in stellar dynamics and yield unphysical results when applied to another problem.

Direct N-body simulation is one of the main techniques used to study gravitational systems in general and galaxies in particular. In this technique, the distribution function is sampled at $N \ll N_\star$ points in a Monte-Carlo fashion. This N depends on the computational capabilities, and an astrophysical system with 10^{11} stars might be represented numerically by a sample of just 10^5 "supermassive" particles. This seems to be allowed because of the equivalence principle and the fact that gravitation is scale free, unlike, for example, in molecular dynamics. However, with gravity this simplification can also cause problems, as some dynamical effects depend on the number density rather than just the mass density.

The most well-known N-dependent effect in stellar dynamics is two-body relaxation. The relaxation time, the characteristic time for a particle's velocity to change by an order of itself due to encounters with other particles, scales with the crossing time roughly as $N/\ln N$. Thus, the ratio between the relaxation times in a real and a simulated system is of a similar order of magnitude as the undersampling factor. This could be taken into account when interpreting the result of an undersampled simulation, but a poorly sampled distribution function might have other, unexpected, consequences.

Galaxies are often described as collisionless stellar systems, which means that the relaxation time is much larger than the timescale of interest (except perhaps at the very center). This property could be very useful. Since a particle's orbit is basically what it would be if it were moving in a smooth gravitational field, we could evaluate the field instead of calculating all of the stellar interactions, which is computationally cheaper. Another useful property is that galaxies are often spheroidal in shape. Even highly flattened galaxies will have a spherical dark halo component. Thus, a spherical shape could be used as a zeroth-order approximation for the gravitational field, and higher-order terms could be written using spherical harmonics.

The goal of this paper is to examine two techniques that utilize both of these facts. These techniques are the multipole expansion (MEX) and the self-consistent field (SCF) methods. They historically come from different ideas and, as explained below in detail, they are mathematically distinct. In the context of numerical simulations, however, they serve a similar function: to evaluate the gravitational force on all N particles generated by this same collection of particles in a way that discards spurious small-scale structure (in other words, smooths the field).

MEX was born from the need to ease the computational burden. The idea is that given spherical symmetry, Gauss's law says that the gravitational force on a particle at a radius r from the center is simply $GM(r)/r^2$ toward the center, where $M(r)$ is the enclosed (internal) mass. The gravitational constant, G, will be omitted in the following. This idea was used by Hénon (1964), who simulated clusters with up to 100 particles to study phase mixing due to spherical collapse. This "spherical shells" methods is MEX of order zero and was also used for the same purpose by Bouvier & Janin (1970). The extension of this idea is that when spherical symmetry breaks, corrections to the force can be expressed by summing higher multipoles (dipole, quadruple, etc.) of the other particles, both internal and external to r. Aarseth (1967) used such a code to study a stellar cluster of 1000 stars embedded in a galactic potential, truncating the expansion at $l_{\max} = 4$.

van Albada & van Gorkom (1977) used a variation of this method to study galaxy collision. These authors employed a grid and also conducted simulations of up to $N = 1000$ and $l_{\max} = 4$. They additionally assumed azimuthal symmetry, which reduced the number of terms in the expansion. Fry & Peebles (1980), Villumsen (1982), McGlynn (1984), and White (1983) all use variations of this method with additional features which are partly discussed in Section 5.2. See also Sellwood (1987) for a review.

The prehistory of SCF is rooted in the problem of estimating a disk galaxy's mass distribution from its rotation curve. Toomre (1963) proposed a mathematical method to generate a surface density profile and a corresponding rotation curve (related to the potential) by means of a Hankel transform, and introduced a family of such pairs. Clutton-Brock (1972) used Toomre's idea, but in reverse: to calculate the gravitational field from an arbitrary two-dimensional density, he generated an orthogonal set of density profiles and their corresponding potentials. This solved two problems: (1) with his orthogonal set, it was possible to represent any flat galaxy as a finite linear combination of basis functions, and (2) unwanted collisional relaxation was curbed due to the smooth nature of the reconstructed gravitational field. See a related method by Schaefer et al. (1973). Clutton-Brock (1973) introduced a three-dimensional extension of his method, which was called SCF by Hernquist & Ostriker (1992, hereafter HO92) by analogy to a similar technique used in stellar physics Ostriker & Mark (1968); further historical developments are discussed in Section 2.5.

To exploit recent developments in the world of general purpose computing on graphics processing units (GPUs), we implemented both SCF and MEX routines in a code called *ETICS* (*Expansion Techniques in Collisionless Systems*). In Section 2, we explain the mathematical formalism of both methods and highlight the differences between them. In Section 3, we explain the unique challenges in GPU implementation and measure the code's performance. In Section 4, we discuss the accuracy of expansion and direct techniques. In Section 5, we present a general discussion and finally summarize in Section 6.

1.1. Glossary

Here, we clarify some terms used throughout this work.

Expansion methods. A way to obtain potential and force by summing a series of terms; in this paper, we use either MEX or SCF.

MEX. Multipole expansion method (sometimes known in the literature as the Spherical Harmonics method): expansion of the angular part.

SCF. Self-consistent field method: a "pure" expansion method since both the angular and radial parts are expanded.

ETICS. Expansion Techniques in Collisionless Systems: the name of the code we wrote, which can calculate the force using both MEX and SCF, using a GPU.

GPU. Graphics processing unit: a chip with highly parallel computing capabilities, originally designed to accelerate image rendering, but also used for general-purpose computing. It often lies on a video card[6] that can be inserted into an expansion slot on a computer motherboard.

2. FORMALISM

2.1. Series Expansions

Both the MEX and SCF methods are ways of solving the Poisson equation

$$\nabla^2\Phi(\mathbf{r}) = 4\pi\rho(\mathbf{r}), \tag{1}$$

the formal solution of which is given by the following integral:

$$\Phi(\mathbf{r}) = -\int \frac{\rho(\mathbf{r}')d^3r'}{|\mathbf{r}-\mathbf{r}'|}. \tag{2}$$

The expression $|\mathbf{r}-\mathbf{r}'|^{-1}$ is the Green's function of the Laplace operator in three dimensions and in free space (no boundary conditions), and the integral is over the whole domain of definition of $\rho(\mathbf{r})$. In an N-body simulation, the density field $\rho(\mathbf{r})$ is sampled at N discrete points $\{\mathbf{r}_j\}$, such that

$$\rho(\mathbf{r}) = \sum_{j=1}^{N} m_j\delta(\mathbf{r}-\mathbf{r}_j), \tag{3}$$

where $\delta(\mathbf{r})$ is the three-dimensional Dirac delta function. Direct N-body techniques evaluate integral (2) *directly*:

$$\Phi(\mathbf{r}) = -\sum_{j=1}^{N} \frac{m_j}{|\mathbf{r}-\mathbf{r}_j|}, \tag{4}$$

and thus at each point $\mathbf{r} = \mathbf{r}_i$ where the potential is evaluated, require N calculations of inverse distance, or $N-1$ if $\mathbf{r}_i \in \{\mathbf{r}_j\}$, since there is no self-interaction. In practice, we are interested in evaluating the potential at the same points at which the density field is sampled, and thus a "full" solution of the Poisson equation requires $N(N-1)/2 \approx N^2$ inverse distance calculations. In both MEX and SCF, the integrand in Equation (2) is expanded as a series of terms, each of which is more easily numerically integrable; this is done in two different ways, lending the two methods quite different properties. In both methods, the reduction in numerical effort comes at the expense of accuracy compared to direct-summation, but this statement is arguable since, in practice, direct N-body techniques use a very small number of particles to sample the phase space.

To demonstrate the difference between the two approaches in the following section, let us consider a one-dimensional version of integral (2), and let us further assume that the density exists in the interval $0 \leqslant x \leqslant 1$:

$$I(x) = \int_0^1 \frac{\rho(x')dx'}{|x-x'|} = \int_0^x \frac{\rho(x')dx'}{x-x'} - \int_x^1 \frac{\rho(x')dx'}{x-x'}. \tag{5}$$

Note that this is not a solution for a one-dimensional Poisson equation (hence the notation I instead of Φ), but just a simplification which we will use to illustrate the properties of each method. We will conveniently ignore the fact that this integral is generally divergent in one dimension, as it does not affect the

[6] Many GPUs lie on GPU accelerator cards which lack video output.

following discussion. In brief, MEX is a Taylor-like expansion of the Green's function, while SCF is a Fourier-like expansion of the density. This already hints at the most critical difference between MEX and SCF: while the former, like a Taylor series, is local in nature, the latter is global. Another way to look at it is that in both methods the integrand is written as a series of functions (of x) with coefficients: in MEX, one uses the given density to evaluate the functions, while their coefficients are known in advance; in SCF, one evaluates coefficients, while the functions are known in advance.

2.2. MEX

If we define $z \equiv x'/x$ and expand the Green's function equivalent in Equation (5) around $z = 0$, for $z < 1$ or $x' < x$ we get that

$$\frac{1}{|x - x'|} = \frac{1}{x}\frac{1}{|1 - z|} = \frac{1}{x}\sum_{l=0}^{\infty} z^l, \tag{6}$$

while for $z > 1$ or $x' > x$ we can expand around $z^{-1} = 0$:

$$\frac{1}{|x - x'|} = \frac{1}{x'}\frac{1}{|z^{-1} - 1|} = \frac{1}{x'}\sum_{l=0}^{\infty} z^{-l}. \tag{7}$$

The first and second terms of the integral (5) define the functions $q_l(x)$ and $p_l(x)$ (utilizing the commutativity of the sum and integral operations):

$$\int_0^x \frac{\rho(x')dx'}{|x - x'|} = \sum_{l=0}^{\infty} x^{-(l+1)} \int_0^x \rho(x')x'^l dx' \equiv \sum_{l=0}^{\infty} x^{-(l+1)} q_l(x), \tag{8}$$

$$\int_x^1 \frac{\rho(x')dx'}{|x - x'|} = \sum_{l=0}^{\infty} x^l \int_x^1 \rho(x')x'^{-(l+1)} dx' \equiv \sum_{l=0}^{\infty} x^l p_l(x), \tag{9}$$

and thus

$$I(x) = \sum_{l=0}^{\infty} [q_l(x)x^{-(l+1)} + p_l(x)x^l]. \tag{10}$$

While we seemingly made things worse (instead of one integral to evaluate, we now have a series of integrals), the fact that x has moved from the integrand to the integral's limit greatly simplifies things. Let the density $\rho(x)$ be sampled at $N \gg 1$ discrete and ordered points $\{x_j : x_j < x_{j+1}\}$; it is easy to show that

$$q_l(x_i) = \sum_{j<i} m_j x_j^l \tag{11}$$

$$p_l(x_i) = \sum_{j>i} m_j x_j^{-(l+1)}. \tag{12}$$

In other words, each of these functions is a cumulative sum of simple terms and can be evaluated at all $\{x_j\}$ in just one pass, but a sorting operation is required.

2.3. SCF

Let us leave the Green's function as it is, and instead expand the density as a generalized Fourier series:

$$\rho(x) = \sum_{n=0}^{\infty} a_n \rho_n(x) \tag{13}$$
$$a_n = \int_0^1 \rho(x')\rho_n^*(x')dx',$$

where $\{\rho_n(x)\}$ is a complete set of real or complex functions (the basis functions); the orthonormality of the basis functions is assumed above. The integral (5) becomes

$$I(x) = \sum_{n=0}^{\infty} a_n \int \frac{\rho_n(x')dx'}{|x - x'|} \equiv \sum_{n=0}^{\infty} a_n I_n(x). \tag{14}$$

The function set $\{I_n(x)\}$ is defined by the above integral. In essence, we replaced the integral over an arbitrary density $\rho(x)$ with an integral over some predefined "densities" $\rho_n(x)$. The advantage is that we can calculate the corresponding potentials, $I_n(x)$, in advance, and then the problem is reduced to numerically determining the coefficients a_n. The choice of the basis is not unique, and an efficient SCF scheme requires the following.

1. The functions $\rho_n(x)$ and $I_n(x)$ are easy to evaluate numerically.
2. The sum (13) converges quickly, or in other words, $\rho_0(x)$ is already close to $\rho(x)$.

2.4. Properties in Three Dimensions

The standard form of MEX in three dimensions is

$$\Phi(\mathbf{r}) = -\sum_{l=0}^{\infty} \frac{4\pi}{2l+1} \sum_{m=-l}^{l} [q_{lm}(r)r^{-(l+1)} + p_{lm}(r)r^l] Y_{lm}(\theta, \phi) \tag{15}$$

$$q_{lm}(r) = \int_{r'<r} r'^l \rho(\mathbf{r}') Y_{lm}^*(\theta', \phi') d^3r' \tag{16}$$

$$p_{lm}(r) = \int_{r<r'} r'^{-(l+1)} \rho(\mathbf{r}') Y_{lm}^*(\theta', \phi') d^3r'. \tag{17}$$

All together there are $(1/2)(l_{\max}+1)(l_{\max}+2)$ complex function pairs (not counting negative m, which are complex conjugates of the others) that need be calculated from the density. Since in practice the density field is made of N discrete points, they must be sorted by r in order for the above integrals to be evaluated in one pass.

The standard form for SCF is

$$\Phi(\mathbf{r}) = \sum_{n=0}^{\infty}\sum_{l=0}^{\infty}\sum_{m=-l}^{l} A_{nlm}\Phi_{nl}(r)Y_{lm}(\theta, \phi). \tag{18}$$

All together, there are $(1/2)(n_{\max}+1)(l_{\max}+1)(l_{\max}+2)$ complex coefficients (not counting negative m) that need be calculated from the density. A typical choice is $(n_{\max}, l_{\max}) = (10, 6)$, for which there would be 308 coefficients. The radial basis functions and coefficients for SCF are discussed in the next section. Spherical harmonics are used in both cases to expand the angular part, but alternatives exist, such as spherical wavelets (e.g., Schröder & Sweldens 1995). MEX has two sums (one infinite), while SCF has three sums (two infinite). In practice, the radial and angular infinite sums must be cut off at $n_{\max}$ and $l_{\max}$, respectively. The finite sum could in principle also be truncated to discard azimuthal information.

Simply equating the expressions gives the following relation between the two methods:

$$Q_{lm}(r) \equiv -\frac{4\pi}{2l+1}[q_{lm}(r)r^{-(l+1)} + p_{lm}(r)r^{l}] = \sum_{n=0}^{\infty} A_{nlm}\Phi_{nl}(r), \tag{19}$$

where $Q_{lm}(r)$ is the (l, m)-pole. In case the system is azimuthally symmetric, $Q_{lm} = 0$ for all $|m| > 0$. Also, the same azimuthal information is carried in positive and negative m terms, which are related to each other by complex conjugation.

If one decomposes the density to a spherical average $\bar{\rho}(r)$ and the non-spherical deviation $\tilde{\rho}(r, \theta, \phi)$, then it is easy to show that $Q_{00}(r)$ depends only on the spherical average, while all other terms depend only on the deviation. In a spherically symmetric system, only Q_{00} is nonzero, and setting $l_{\max} = 0$ yields an accurate result. While the choice of $l_{\max}$ depends only on the deviation of the system from spherical symmetry, the choice of $n_{\max}$ in SCF depends on how well the system is described by the zeroth radial basis function, and is usually determined by trial and error (see Section 4.1).

It is interesting to note a nontrivial mathematical difference between the two methods. One can show that the Laplacian of Equation (15) is zero when substituting the appropriate expressions for q_{lm} and p_{lm} from Equations (16) and (17); the proof is mathematically cumbersome and will not be displayed here. This is surprising, since according to the Poisson equation the result should be proportional to the density. One cannot appeal to series truncation to resolve this apparent contradiction; indeed, each term in the formally non-truncated infinite series yields zero density, despite representing the multipoles as continuous functions. The solution is that the potential at point $\mathbf{r}$ has contributions from all internal (i.e., at $r' < r$) particles (represented by q_{lm}) and all external particles (represented by p_{lm}), but has no information about the density at $\mathbf{r}$ itself. This is also the case when the potential is constructed by a direct summation of all gravitational point sources, so one may say that MEX is similar to direct methods in this sense. In SCF, by construction, taking the Laplacian of Equation (18) leads right back to the density field (Equation (1)). One can thus use the coefficients A_{nlm} to represent a smoothed field. One can also use MEX for this purpose if the derivatives of $Q_{lm}(r)$ are calculated on a grid or with a spline.

2.5. *Radial Basis*

A key difference between MEX and SCF is the freedom of choice of the *radial* basis. There are, in fact, two function sets: the radial densities $\{\rho_{nl}(r)\}$ and the radial potentials $\{\Phi_{nl}(r)\}$. These sets are related via the Poisson equation $\nabla^2\Phi_{nl} = 4\pi\rho_{nl}$ (in this case ∇^2 only contains derivatives with respect to r). The choice of basis is not unique, and the basis functions themselves need not represent physical densities and potentials (i.e., ρ_{nl} could be negative). However, it is convenient to take the zeroth term ($n = l = 0$) to represent some physical system, and to construct the rest of the set by some orthogonalization method, such as the Gram–Schmidt process.

The idea of Clutton-Brock (1973) was to use a Plummer (1911) model as the zeroth term and construct the next orders using the Gegenbauer (ultraspherical) polynomials and spherical harmonics (see Allen et al. 1990, who developed a virtually identical method for finite stellar systems using spherical Bessel functions for the radial part). HO92 constructed a new radial basis (also using Gegenbauer polynomials) for which the zeroth order was a Hernquist (1990) model; this is the basis set we adopt in *ETICS*. They argued that this basis was better suited to study galaxies.

More basis sets followed. Syer (1995) used the idea of Saha (1993) that the basis does not have to be biorthogonal to construct a set for which the zeroth order was oblate. Zhao (1996) gave a radial basis set for the more general α model (of which both Plummer and Hernquist are special cases) and Earn (1996) introduced a basis for thick disks in cylindrical coordinates. Brown & Papaloizou (1998) and Weinberg (1999) describe the numerical derivation of the radial basis set so that the lowest order matches any initial spherically symmetric model, so-called designer basis functions. Rahmati & Jalali (2009) introduced an analytical set for which the zeroth order is the perfect sphere of de Zeeuw (1985).

3. IMPLEMENTATION

3.1. *Parallelism*

There are several levels of task parallelism available when writing computer codes. At one level, tasks are performed in parallel on different computational units (such as CPUs), but only one copy of the data exists, which is accessed by all tasks; this is called a *shared memory* scheme. The tasks are called "threads" and are generally managed within one "process" of the program. A higher level of parallelism is called a *distributed memory* scheme, where tasks are performed on different units (often called "nodes") but each unit only has access to its own memory; thus, data must be copied and passed. In this case, the parallel tasks are different processes and cooperation between them is facilitated by a message passing interface (MPI). The parallel programming model is different between shared and distributed memory; the former is considered easier since threads can cooperate faster and more easily. A high-performance supercomputer will generally enable parallelism on both levels: these machines are made of multiple nodes, each of which has its own memory and multiple computational units.

GPUs are powerful and cost-effective devices for high-performance parallel computing. They are used to accelerate many scientific calculations, especially in astrophysics, such as the dynamics of dense star clusters and galaxy centers (Hamada & Iitaka 2007; Portegies Zwart et al. 2007; Schive et al. 2008; Just et al. 2011; see review by Spurzem et al. 2012). The GPU contains its own memory and many computational units, and thus it is a shared memory device.[7] SCF force calculation is particularly easy to parallelize, since the contribution of each particle to the coefficients A_{nlm} is completely independent of all other particles. Particle data can be split into smaller chunks (each could be on a different node) and from each chunk partial A_{nlm}-s are calculated. Then, the partial coefficients are summed up and the result communicated to all the nodes. This was done

[7] A GPU behaves as a shared memory device since all threads have transparent access to the device's global memory, which has a single address space. However, there is a hierarchy in the memory and thread structure, with some kinds of memory being private at the thread or block level. Thus, GPUs also have distributed memory characteristics.

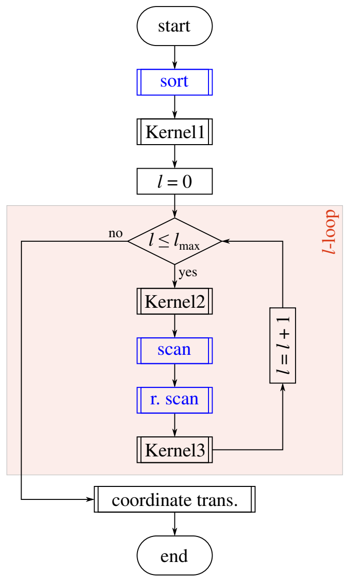

Figure 1. Flowchart of the MEX routine. Boxes with double-struck vertical edges indicate a GPU-accelerated operation. Blue color represents a call to a *Thrust* library subroutine.

(A color version of this figure is available in the online journal.)

by (Hernquist et al. 1995, hereafter H95), whose code used the MPI call `MPI_Allreduce` to combine the partial coefficients. This parallelization scheme, however, is not suitable for GPUs, as discussed in Section 3.3. MEX force calculation is harder to parallelize since the contribution of each particle depends on its position in a sorted list (by radius). However, in a shared memory scheme, this too could be achieved relatively easily as explained in the following section.

3.2. MEX

The current implementation of the MEX method relies on *Thrust* (Bell & Hoberock 2011), a C++ template library of parallel algorithms which is part of the CUDA framework. It makes parallel programming on a shared memory device (either a GPU or a multicore CPU) transparent, meaning that the task is performed in parallel with a single subroutine call and the device setup and even the choice of algorithm is performed by the library. *Thrust* provides a sorting routine which selects one of two algorithms depending on input type. In the current version of MEX and using version 1.6 of *Thrust*, a general Merge Sort algorithm (Satish et al. 2009) is used.

A flowchart of the entire MEX routine is shown in Figure 1. The flow is controlled by the CPU, and boxes with double-struck vertical edges indicate a GPU-accelerated operation. The blue double-struck boxes represent *Thrust* calls, while the black boxes are regular CUDA kernel calls. When a GPU operation is in progress, the CPU flow is paused. Figure 2 shows the four main memory structures of the program and how the *Thrust* subroutines and kernels in the program operate on them. The particle array contains all of the particle coordinates and also the distance square from the center, which needs to reside in this structure for the sorting operation (in practice, the particle array contains additional data such as ID and velocity, but this is not used by the MEX routine); the cache structure contains functions of particle coordinates which are needed to calculate the multipoles. Kernel1, which is executed once, reads the coordinates, calculates those functions, and fills the cache structure.

Kernel2 calculates the spherical harmonics at the current l level and from that the contributions of the particle to q_{lm} and p_{lm}, which are saved in global memory. When this kernel returns, the *Thrust* subroutines are dispatched to perform the cumulative sum. The "scan" (forward cumulative sum) and "r. scan" (reverse scan) are in fact both calls to the `exclusive_scan` subroutine, but to perform the reverse scan, we wrap p_{lm} with a special *Thrust* structure called `reverse_iterator`. Not shown in the flowchart, the two scan subroutines have to be called l times at each l level since they work on one m value at a time.

Kernel3 has both cache and compute operations: it calculates the partial forces in spherical coordinates (i.e., the l order correction to the force) and/or potentials by evaluating all the spherical harmonics again (and their derivatives with respect to spherical coordinates). Later, it advances r^l and $r^{-(l+1)}$ to the next l level (except at the last iteration). Finally, the last kernel operates on the force structure and transforms it to Cartesian coordinates. Figure 6 shows the relative time it takes to perform the internal operation.

We note that the potential could be calculated at the same time as the force (in Kernel3) and stored at another memory structure (not shown in Figure 2) but is skipped if only forces are needed. Alternatively, only the potential could be calculated (this is faster since the derivatives of the special functions are not calculated). The choice between calculating force, potential, or both is made with C++ templates.

3.3. SCF

We first briefly explain the serial algorithm used by HO92. The force (and potential) calculation had two parts: (1) calculation of all the A_{nlm}-s (the plural suffix "-s" to emphasize that there are hundreds of coefficients in this three-dimensional structure) and (2) calculation of all the forces using the coefficients.

In both parts, the particle loop (the j-loop) was the external one, inside of which there are again two main steps. In step (1a), all of the necessary special functions were calculated using recursion relations. Step (2a) was identical but the derivatives of those functions were additionally calculated. In step (1b), there was a nested loop ($n-l-m$ structure) in which a particle's contribution to every A_{nlm} was calculated and added serially. In step (2b), there was also such a loop, which used all the A_{nlm}-s to calculate the force on each particle.

In the parallel algorithm used by H95, another part was added between the two parts mentioned above: communicating all partial A_{nlm}-s from the various MPI processes, adding them up, and distributing the results. In practice, this was achieved using just one command, `MPI_Allreduce`. There are two main reasons why this algorithm could not be used effectively on a GPU, and both are related to the difference between how the GPU and CPU access and cache memory. The first difficulty is performing the sum. The partial sums from the different parallel threads could, in principle, be stored on a part of the GPU memory called *global memory*, and then summed in parallel. However, a modern GPU can execute tens of thousands of threads per kernel (note that the concept of a thread in CUDA is abstract, and the number of threads far exceeds the number of thread processors on the GPU chip), and every partial A_{nlm} is 5 kbyte in size (depending on $n_{\max}$ and $l_{\max}$). Thus, writing and summing the partial coefficients would require extensive access to global memory, which is slow compared to the actual calculation part. The second difficulty is that if one thread uses too much memory, for example, to store all necessary Legendre and Gegenbauer polynomials as well as a complex exponent

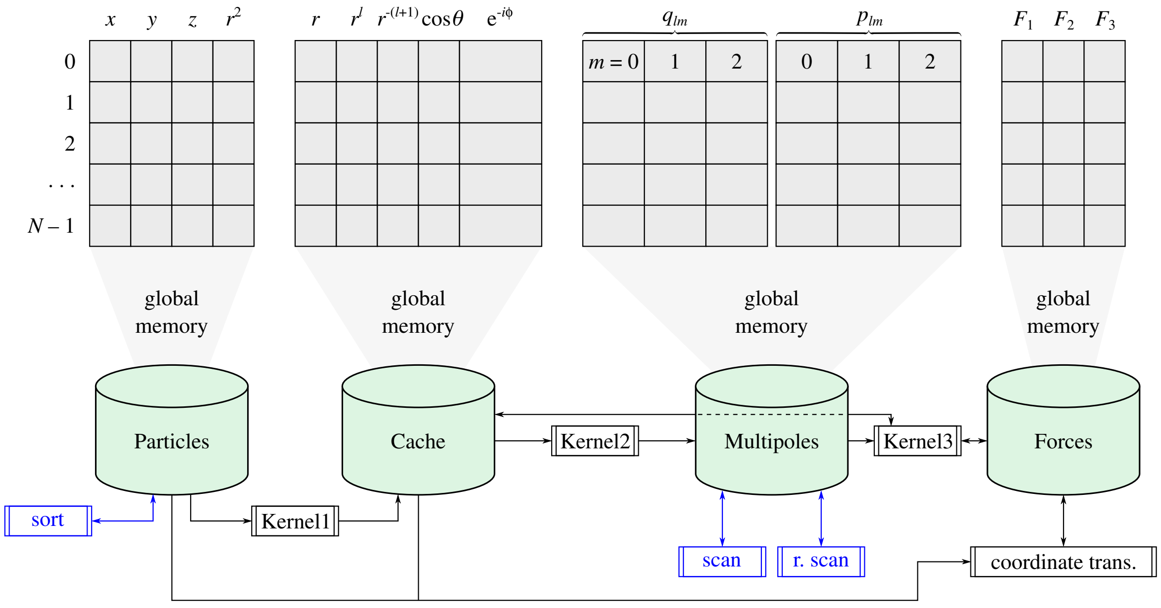

Figure 2. Main memory structures in the MEX routine and a scheme of the action of the *Thrust* subroutines (blue) and kernels on them. The wider boxes for the exponent (in the cache structure) and the multipoles represent complex numbers (require twice the memory). The layout of the multipole structure (shown here for $l_{\max} = 2$) is actually rotated in memory by 90° with respect to the other structures, since it is easier for the scan subroutines.

(A color version of this figure is available in the online journal.)

(as is done in the HO92 code), then this may lead to an issue called *register spilling*, where instead of using the very fast register memory, the thread will store the values on the slow global memory, which again we wish to avoid on performance grounds.

To tackle those issues, we utilized another type of GPU memory called *shared memory*.[8] This memory is "on chip" (on the multiprocessor circuit rather than elsewhere on the video card) and has $\times 100$ lower latency than global memory. Threads in a CUDA program are grouped into blocks. Threads in the same block share this fast memory (hence the name). It is also much less abundant than global memory. The Nvidia Tesla K20 GPUs have just 64 kbyte of shared memory per block, while they have 5 gbyte of global memory.

In order to use shared memory to calculate the coefficients, each thread would serially add contributions from particles to the partial A_{nlm}-s on shared memory; then, they would be summed up in parallel in each block. However, there are usually hundreds of different A_{nlm}-s, as well as tens or hundreds of threads per block (depending on hardware; which is required for efficient loading of the GPU); there is not enough shared memory for that (by far). To solve this, we changed the order of the loops: the external loop is the l loop, then comes the n loop. For each (n, l) pair, a CUDA kernel is executed where the j loop is performed in parallel on different threads, inside of which the m loop is done. Now each thread has to deal with far fewer A_{nlm}-s (no more than $l_{\max} + 1$), for which there is usually enough shared memory.

A flowchart of the entire SCF routine is shown in Figure 3. The flow is controlled by the CPU, and boxes with double-struck vertical edges indicate a CUDA kernel call. When a GPU operation is in progress, the CPU flow is paused. Figure 4 shows the four main memory structures of the program and how the five kernels in the program operate on them. The particle array contains all of the particle coordinates (in practice, it

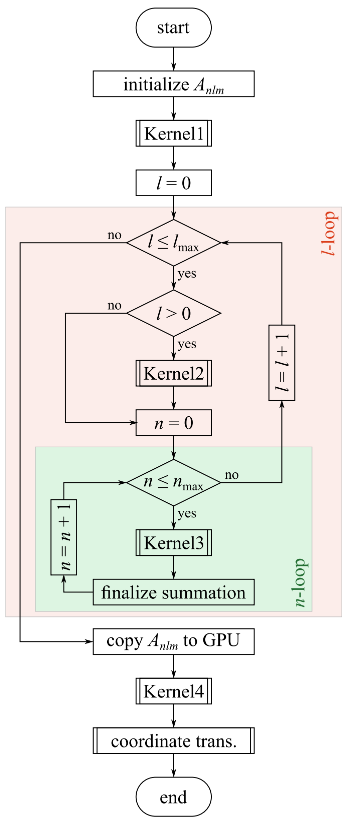

Figure 3. Same as Figure 1, but for the SCF routine.

(A color version of this figure is available in the online journal.)

contains additional data such as ID and velocity, but this is not used by the SCF routine); the cache structure contains those functions of particle coordinates which are needed to calculate the basis functions. Kernel1, which is executed once, reads the coordinates, calculates those functions, and fills the cache

[8] Not to be confused with the concept of a shared memory device.

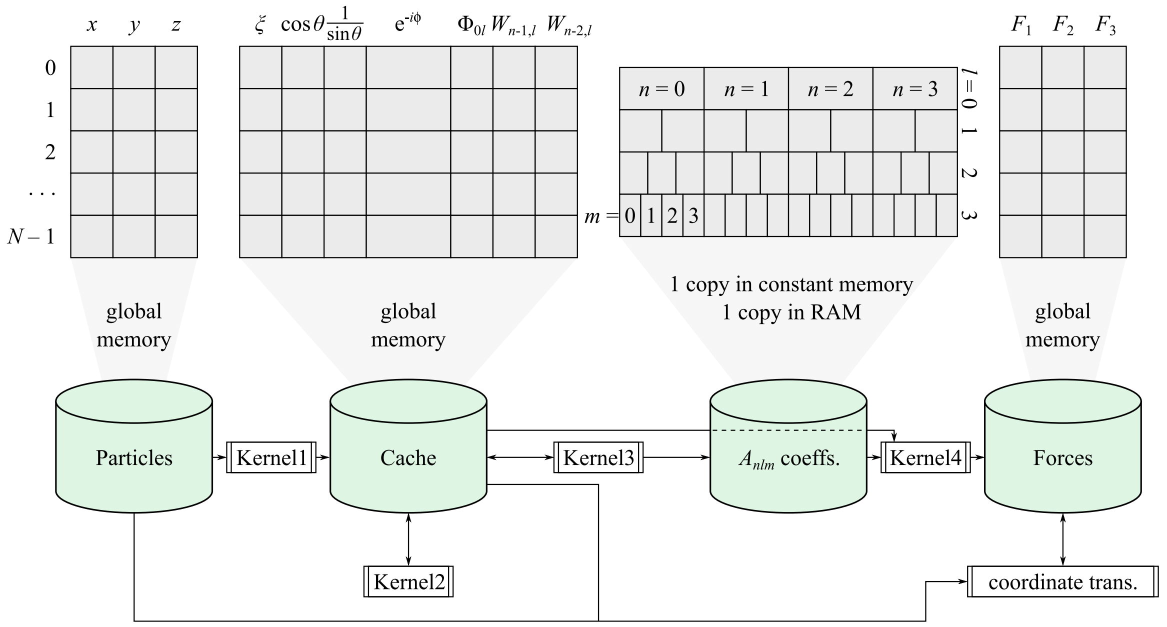

Figure 4. Same as Figure 2, but for the SCF routine. The main memory structures in the SCF routine, and a scheme of the action of the kernels on them. Every cell in the coefficient structure (shown here for $(n_{\max}, l_{\max}) = (3, 3)$) is a complex number.

(A color version of this figure is available in the online journal.)

structure. Kernel2 only operates on the cache structure. It has just one function which is to advance Φ_{0l} by one level; thus, it needs to be executed at the beginning of each iteration of the l loop. As shown in the flowchart, it is skipped for $l = 0$ because Kernel1 calculates and caches Φ_{00}.

Kernel3 has both cache and compute operations: it calculates the current W_{nl} using recursion relations from the cached $W_{n-1,l}$ and $W_{n-2,l}$ and then updates the cache. Later, it calculates the spherical harmonics and from that the contribution of the particle to the all A_{nlm} in the current (n, l) level, which are saved in shared memory. When all threads in the block have finished calculating the contributions of the particles assigned to them, they are synchronized and a parallel reduction is performed. Since threads from different blocks cannot share memory, the data from each block must be transferred to the host machine's memory and the CPU finishes the summation process.

For the force calculation, only a reading of the A_{nlm}-s is required. The GPU has yet another type of memory which is ideal for the storage of coefficient or constant parameters. It is fittingly called "constant memory," and is as fast as shared memory when every thread in a warp accesses the same memory element. It is also very limited (usually to 64 kbyte per device), but the A_{nlm} structure could still fit there nicely. Once calculation of all the coefficients is complete, it is transferred back to the GPU constant memory to be used to calculate the forces. Since only reading the coefficient is required, in Kernel4, which calculates the forces and/or potentials by evaluating all the basis functions again (and their derivatives with respect to spherical coordinates), the j loop is the external one. To avoid register spilling, we keep the internal loop structure as l–n–m, and thus we only need to recalculate the complex exponents which is relatively cheap. Finally, the last kernel operates on the force structure and transforms it to Cartesian coordinates. Figure 7 shows the relative time it takes to do the internal operation.

We note that the potential could be calculated at the same time as the force (in Kernel4) and stored at another memory structure (not shown in Figure 4) but is skipped if only forces are needed. Alternatively, only the potential could be calculated (this is faster since the derivatives of the special functions are not calculated). The choice between calculating force, potential, or both is made with C++ templates.

3.4. *Performance*

We tested the performance of *ETICS* (both MEX and SCF) on a single Nvidia Tesla K20 GPU on the Laohu supercomputer at the NAOC in Beijing. For comparison, we also tested the Fortran CPU SCF code by Lars Hernquist on the ACCRE cluster at Vanderbilt University in Nashville, Tennessee (we used a node with an Intel Xeon E5520 CPU). If the initial conditions are not sorted by r in advance, then the first MEX force calculation is more costly than all of the following since the sorting of an already nearly sorted particle list is faster. Thus, all measurements of the MEX code are done after the system is evolved one very short leapfrog time step. Figure 5 shows the time it takes to do one full force calculation as a function of N, $l_{\max}$, and $n_{\max}$. Each point represents the mean time of 10 different calculations. The dispersion is generally very low, with the exception of *ETICS*-MEX with $l_{\max} \gtrsim 6$; for which we only show error bars. Note that the timing only depends on the number of particles (and expansion cutoffs) and not on their spatial distribution.

The CPU and GPU SCF codes are both theoretically $O(l_{\max}^2 n_{\max} N)$. At low N, the GPU is not fully loaded and *ETICS* performance seems superlinear with N. *ETICS*-MEX is theoretically $O(l_{\max}^2 N \log N)$, but this again is an asymptotic behavior which is not observed. The lack of good GPU load for $N \lesssim 10^6$ is much more evident than the $N \log N$ nature of the algorithm. The GPU global memory was the limiting factor in how many particles could be used with both methods. The dotted lines show the performance of *ETICS* using single-precision instead of double. The speed increase is 61% for SCF and 65% for MEX, but there is a price to pay in accuracy, as noted in Section 4.2. The speedup factor could be very different for different GPUs.

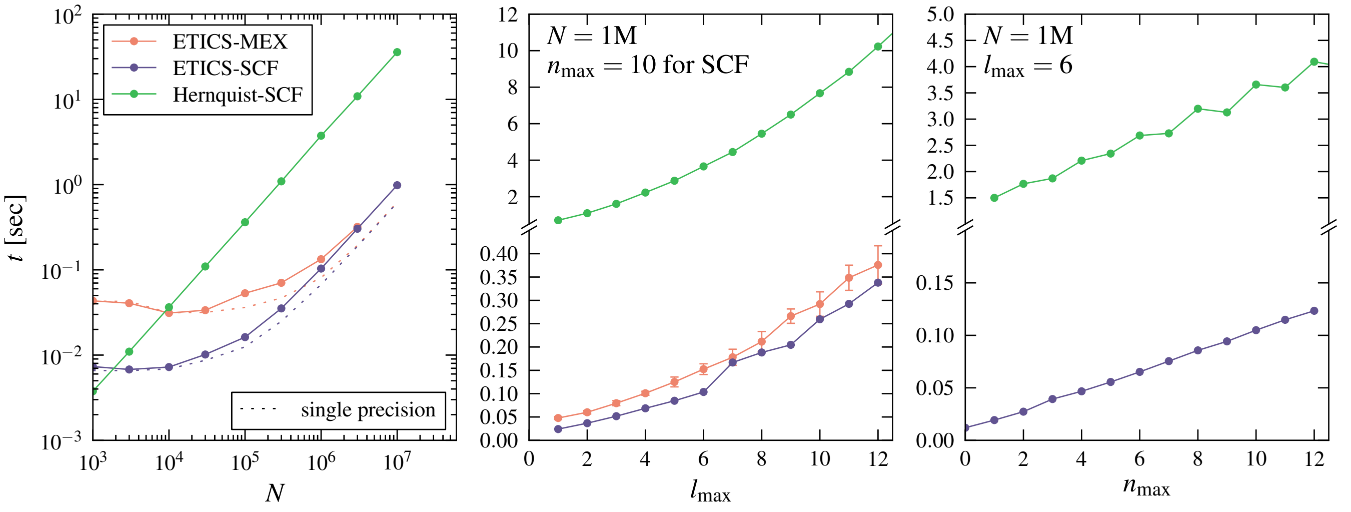

Figure 5. Scaling of one full force calculation time. Hernquist's SCF code (in green) is a CPU code and was tested on an Intel Xeon E5520 CPU (one core). *ETICS* is a GPU code with both MEX (red) and SCF (blue) methods, and was tested with an Nvidia Tesla K20 GPU; for the GPU codes, dotted lines show the performance in single-precision mode. For the scaling with N, we set $l_{\max} = 6$, and for the SCF codes $n_{\max} = 10$. The scaling is theoretically linear with N for SCF and $N \log N$ for MEX, but the theoretical behavior is only seen asymptotically for the GPU codes since the GPU is not fully loaded at low N. Both methods scale quadratically with $l_{\max}$ (the tests were performed with $N = 10^6$ and $n_{\max} = 10$ for SCF). SCF scales linearly with $n_{\max}$ (the tests were performed with $N = 10^6$ and $l_{\max} = 6$). The CPU code shows some erratic behavior due to compiler optimization. Note that the tests are performed on different hardware.

(A color version of this figure is available in the online journal.)

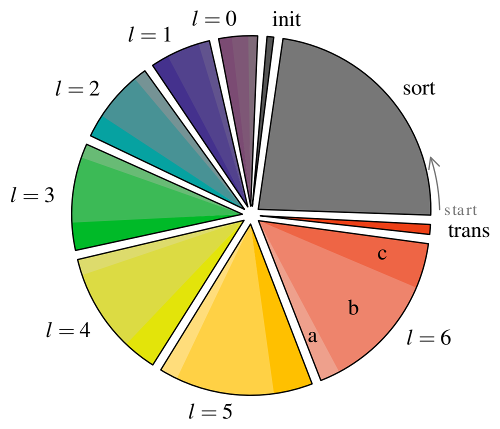

Figure 6. Pie chart showing the relative time of each operation required to perform one full MEX force calculation with *ETICS* (double-precision, $N = 10^6$ and $l_{\max} = 6$); the total time is 0.15 s on an Nvidia Tesla K20 GPU. The results may differ significantly on different hardware and if single-precision is used instead. The first operation is to sort, followed counter-clockwise by initialization of the cache arrays, the l loop where each iteration is divided into (a) summand calculation, (b) cumulative sum, and (c) partial force calculation. The final operation is coordinate transformation from spherical to Cartesian.

(A color version of this figure is available in the online journal.)

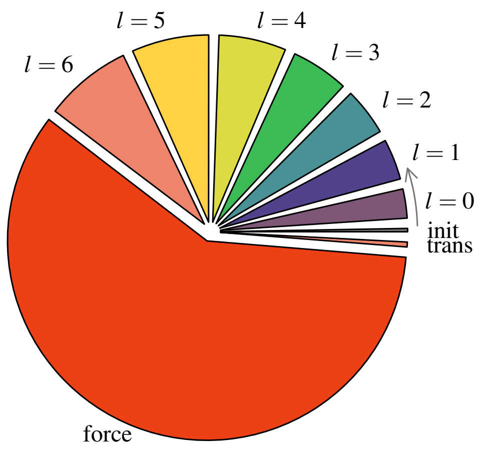

Figure 7. Same as Figure 6 for a full SCF force calculation with *ETICS* ($N = 10^6$ particles, $l_{\max} = 6$, and $n_{\max} = 10$); the total time is 0.16 s on an Nvidia Tesla K20 GPU. The first operation is initialization, followed counter-clockwise by the l loop (in which the n loop is nested). The partial force calculation is a single CUDA kernel, inside of which all the loops are performed.

(A color version of this figure is available in the online journal.)

All codes should scale quadratically with $l_{\max}$, but as the middle panel of Figure 5 shows, this behavior is not as clear for *ETICS*-MEX. This is due to the extensive memory access this code requires, which rivals the calculation time. Memory latency on GPUs is not easy to predict; due to caching and the way memory is copied in blocks, and the latency depends not only on the amount of memory accessed but also on the memory access pattern.

SCF codes theoretically scale linearly with $n_{\max}$. We note a strange behavior of the CPU code: it seems that the time increases with $n_{\max}$ in a "zigzag" fashion (the measurement error of the times is much smaller than this effect and is reproducible). This is paradoxical: it takes a shorter time to calculate with $n_{\max} = 9$ than with $n_{\max} = 8$, even though more operations are required. It is not simple to understand why this is, but it seems that the compiler performs some optimization on the first j loop (coefficient computation) that only helps when $n_{\max}$ is odd but not when it is even.

The comparison between *ETICS*-GPU and Hernquist's code is not exactly fair since they use different types of hardware. Specifically, for the hardware we tested, *ETICS*-GPU outperforms Hernquist's code by a factor of about 20 (which depends little on all of the parameters). However, Hernquist's code can utilize a multicore CPU (using MPI). The Xeon CPU we used has four cores, and two such CPUs are mounted on a single ACCRE node. We could use the Fortran code in MPI mode on all eight effective cores with almost no overhead, and the calculation is accelerated by a factor of eight. Also, Hernquist's code calculate the jerk (force derivative), which *ETICS*-GPU does not; this takes ∼14% of the total time.

Figures 6 and 7 show the fraction of time it takes to perform the internal operations for the force calculation for *ETICS*-MEX and -SCF, respectively; both use $N = 10^6$ and $l_{\max} = 6$, and $n_{\max} = 10$ for SCF. For MEX, operations inside each iteration of the l loop are shown in different shades (also denoted by letters corresponding to stages (3a), (3b), and (3c) as explained in Section 3.2). The most costly operations are those we entrust to *Thrust*, namely, the sorting and cumulative sum. In Figure 7, the internal structure of each l iteration is not shown (since there are too many internal operations, including the n loop). The force calculation is executed as one operation (a single CUDA kernel call) and includes the l loop nested inside it (unlike MEX where only a partial force was calculated at every l iteration, step (3c)).

4. EXPANSION ACCURACY

4.1. *Infinite Particle Limit*

Two separate questions come up when discussing the accuracy of expansion methods: how well the expansion approximates the N-body force (i.e., direct-summation), and how well it approximates the smooth force in the limit of infinite particles (which we will refer to as the "real" force in the following discussion). Both questions depend on N, $l_{\max}$, and (for SCF) $n_{\max}$. A related question is how well the N-body force approximates the real force as a function of N. All of these questions depend not only on the expansion cutoff and N, but also on the stellar distribution as well (e.g., global shape, central concentration, fractality, etc.); we do not fully explore this in this work.

There are two types of error when considering the expansion methods versus the real force, analogous to systematic and random errors. The first, systematic-like error, comes from the expansion cutoff, which is called the *bias*. For example, a system which is highly flattened could not be described by keeping just the quadrupole moment, so both MEX and SCF being cut off at $l_{\max} = 2$ would exhibit this type of error, regardless of N (see Merritt 1996; Athanassoula et al. 2000 for discussion about bias due to softening). The second, random-like error, comes from the finite number of particles and their coarse grainy distribution; it is the equivalent of N-body noise (also referred to as particle noise or sampling noise).

HO92 attempted to estimate the accuracy of SCF by showing the convergence of the coefficient amplitudes with increasing n for the density profiles of some well-known stellar models. They showed that A_{n00} decayed exponentially or like a power law with n, depending on the model. This analysis was not satisfactory because it applied to the limit of infinite N, thus ignoring the random-like error. Furthermore, showing the convergence of the coefficients does not provide information concerning the force error. The bias and random error are not easy to distinguish. The bias could be calculated, in principle, only if the true mass density $\rho(\mathbf{r})$ is known, which is not generally the case; however, it is still useful to look at some particular examples where it is known.

To test the accuracy of the expansions techniques, we used two simple models for the mass density. Both of our models are Ferrers ellipsoids (Ferrers 1877; often called Ferrers bars) with index[9] $n = 1$: `model1` is a mildly oblate spheroid with an axis ratio of 10:10:9, `model2` is triaxial with an axis ratio of 3:2:1. Ferrers ellipsoids are often used in stellar dynamics, especially in the modeling of bars (e.g., Athanassoula 1992). They have a very simple mass density,

$$\rho(\mu^2) = \begin{cases} \rho_0[1-(\mu/a)^2]^n & \mu \leqslant a \\ 0 & \mu > a, \end{cases} \tag{20}$$

where a, b, and c are the axes, ρ_0 is the central density, $n \geqslant 0$ is the index, and μ is the ellipsoidal radius, defined by

$$\mu^2 = x^2 + \left(\frac{a}{b}y\right)^2 + \left(\frac{a}{c}z\right)^2. \tag{21}$$

The potential due to this family of models is simply a polynomial in (x, y, z) if n is an integer. The coefficients could be calculated numerically (also analytically for some cases) by solving a one-dimensional integral (Binney & Tremaine 2008, Chapter 2.5). For both of our models, we used the mathematical software *Sage* to calculate the coefficients to better than 10^{-13}. The force vector components are trivially derived from the potential polynomial; this is the "real" force.

We created many realizations of these two models, ranging from only 100 particles to 10^6. Our goal is, for each realization, to compare the force calculated using MEX, SCF, and direct-summation (no softening) with the real force. All calculations were performed using double-precision, and the direct-summation force is not softened. For each realization, we obtain a distribution of N values of the relative force error,

$$\epsilon_i \equiv \frac{|\mathbf{F}_i - \mathbf{F}_{i,\mathrm{real}}|}{|\mathbf{F}_{i,\mathrm{real}}|}, \tag{22}$$

where i is the particle's index. It is not practical to show to full distribution for all cases, so in Figures 8 and 9 we show the mean and the full distribution for only selected cases.

The left panel of Figure 8 shows the mean relative force error $\bar{\epsilon}$ in `model1` for direct-summation and MEX with even $l_{\max}$ between 0 and 10; odd terms are, in principle, zero if the expansion center coincides with the center of mass, and in practice are very small. For this model, $\bar{\epsilon}$ decreases with N for all cases except $l_{\max} = 0$ (monopole only). The smallest error is for $l_{\max} = 2$ (monopole and quadrupole only). Unintuitively, adding correction terms *increases* the error (for constant N). This is because the model's deviation from sphericity is so mild that the quadruple describes it well enough; the following terms just capture some of the N-body noise in the realization and cause more harm than good. In the right panel, we show the full log-distribution for selected cases. The histograms for $N = 10^3$ are made by the stacking of 10^3 realizations, so that there are 10^6 values of ϵ in all histograms. In all cases, the ϵ distributions are close to log-normal; the logarithmic horizontal axis hides the fact that the distributions on the right are much wider in terms of standard deviation due to a very long and fat tail when viewed in linear space. Note that while the number of particles increased by 1000, in all cases the error distribution shifted down by just a factor of $\sim$10.

Figure 9 is the same, but for the triaxial `model2`. While in the N-body cases the ϵ distributions are much the same, MEX shows different behavior. The most prominent feature is the bump on the right side of the $N = 10^6$, $l_{\max} = 6$ error distribution, which demonstrates the issue of bias. Most of the particles which make up this bump are located in the lobes of the ellipsoid where many angular terms are required. When $l_{\max}$ is increased to 10, this bump disappears. It also is not present in the $N = 10^3$, $l_{\max} = 6$ case, probably because it is overwhelmed by the random error. This bump causes the mean error to saturate

[9] The Ferrers index should not be confused with the SCF radial index, both denoted with n.

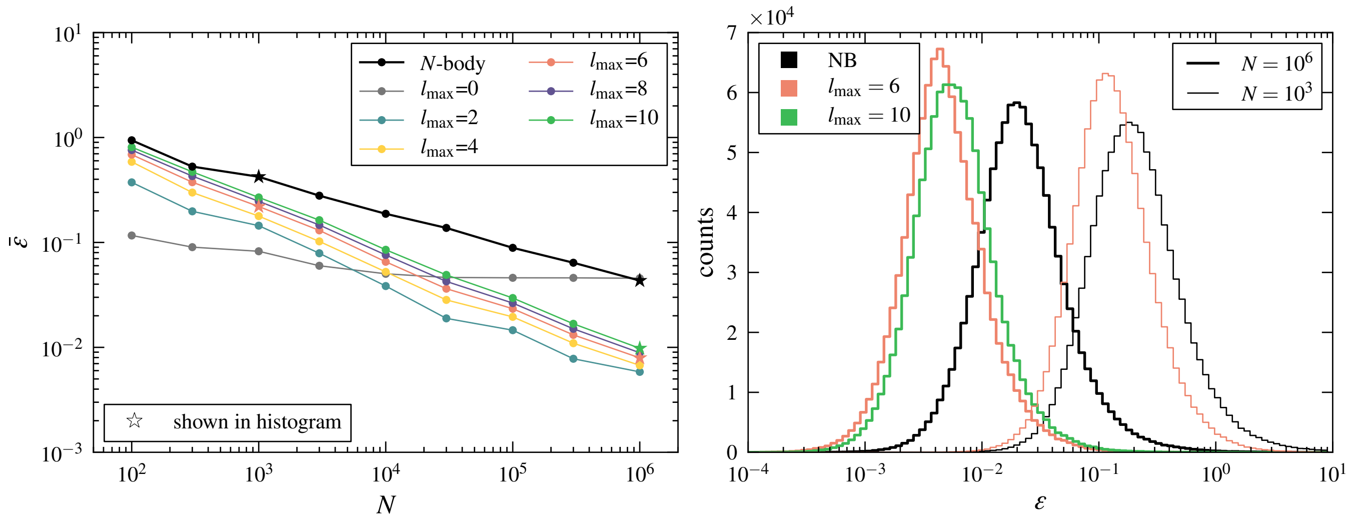

Figure 8. For the mildly oblate `model1`, as a function of N, the left panel shows the mean relative force error ϵ (defined in Equation 22) in direct summation ("N-body"; thick black line) as well as MEX expansions with even $l_{\rm max}$ between 0 and 10 (odd terms have almost no effect). Each point represents a full distribution of error values, obtained by stacking models with the same N. The right panel shows the full log distribution for some selected points in the left panel (shown as stars). Note that since the horizontal axis is logarithmic, the distributions on the right are much wider than those on the left (have larger standard deviation).

(A color version of this figure is available in the online journal.)

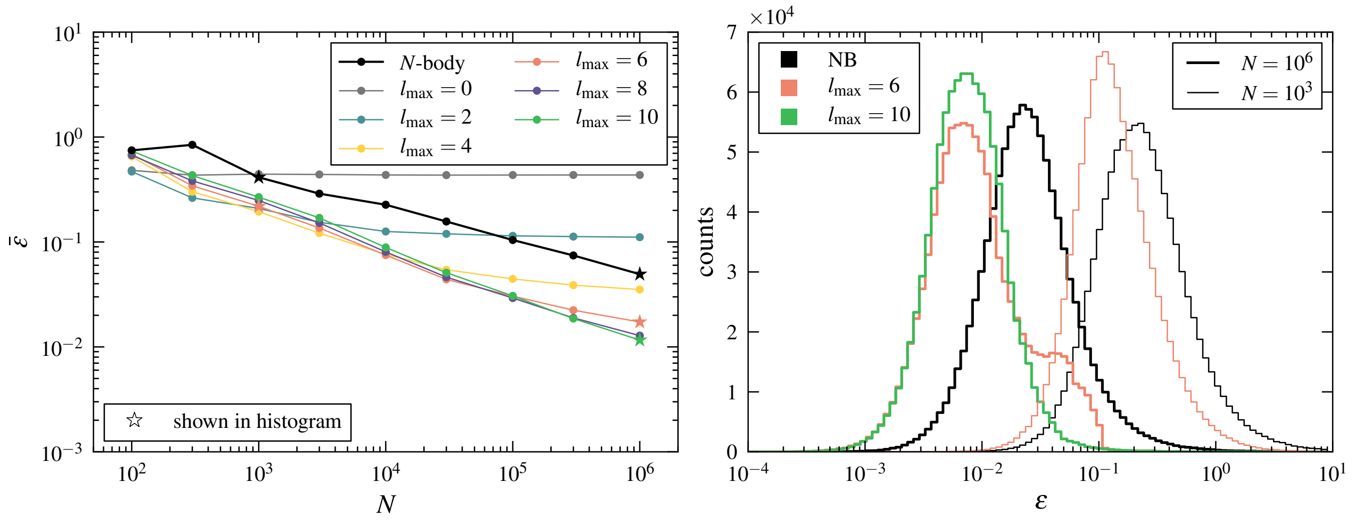

Figure 9. Same as Figure 8, but for the triaxial `model2`. For this model, the behavior of the mean error seems different for MEX (but nearly identical for the N-body): the low multipoles outperform (have smaller mean error than) the higher ones only at low N, but become saturated (increasing N does not improve accuracy). This could be understood from the right panel: while the error distributions are very similar to the previous model, the $l_{\rm max}=6$, $N=10^6$ has a bump on the right. This bump is the bias that causes the saturation and will likely not disappear when N increases (however, as seen, it is diminished for $l_{\rm max}=10$).

(A color version of this figure is available in the online journal.)

with particle number, as the left panel shows. Increasing N will shift the bulk of the bell curve to the left (zero) but will not quench the bump. At much larger N, `model1` will show the same behavior as the random error becomes smaller than the bias, and the high-$l_{\rm max}$ cases would outperform $l_{\rm max}=2$.

We repeat this exercise for SCF, which has an additional source of bias due to the radial expansion cutoff. Figure 10 illustrates this point by showing the relative force error distribution in `model2` for SCF compared to MEX with the same number of particles ($N=10^6$) and the same angular cutoff ($l_{\rm max}=10$). With increasing $n_{\rm max}$, the SCF error distribution approaches that of MEX, demonstrating the point made in Section 2.4 that MEX is equivalent to SCF with $n_{\rm max}\to\infty$. It should be noted that the basis set we programmed in *ETICS* is not at all suitable for Ferrers models (which are finite and have a flat core), and the apparently slow convergence should not disparage one from using SCF, even if it is not known in advance what basis to choose. The overlap between the relative force error distributions of SCF at $n_{\rm max}=10$ and MEX is 77%. A more intelligent choice of basis function is discussed by Kalapotharakos et al. (2008), who used a similar methodology to choose the best basis set for triaxial Dehnen (1993) γ-models (Merritt & Fridman 1996) with $0\leqslant\gamma\leqslant 1$ from a family of basis sets similar to the HO92.

The results presented in this section suggest that there is some optimal expansion cutoff which is different for different models and depends on the number of particles (Weinberg 1996). This is analogous to optimal softening in direct-summation force calculations (Merritt 1996; Athanassoula et al. 1998). If not enough terms are used, then there is a large bias; if too many terms are used, then the particle noise dominates. Vasiliev (2013) addressed this issue by calculating the variance of each

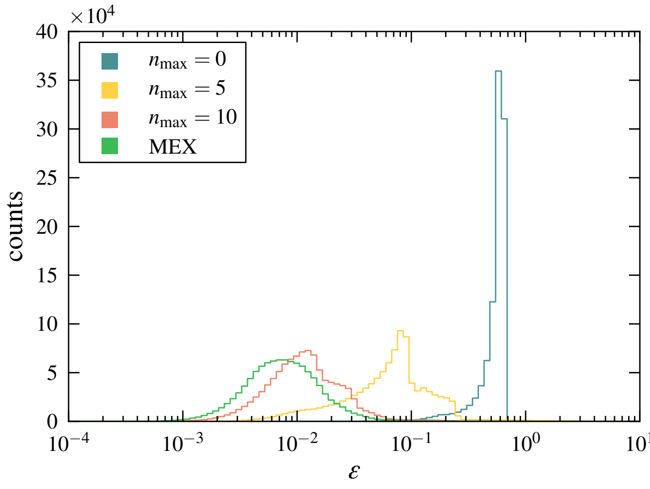

Figure 10. Relative force error distribution for the triaxial `model2` with $N = 10^6$. The green histogram is also shown in the right panel of Figure 9 and represents an MEX expansion with $l_{\max} = 10$. The other histograms represent SCF expansions with $l_{\max} = 10$ and varying $n_{\max}$ values as shown. With increasing $n_{\max}$, the SCF error distribution approaches that of MEX with the same $l_{\max}$. In this case, the model differs greatly from the zeroth-order function of the basis set, showing relatively slow convergence.

(A color version of this figure is available in the online journal.)

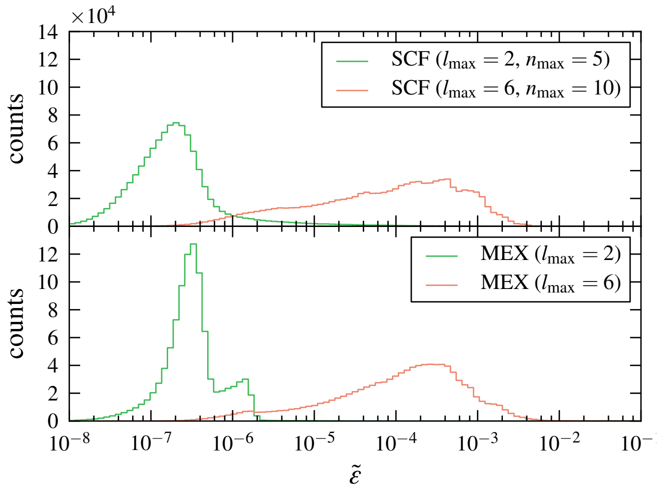

Figure 11. Relative force error distributions of single-precision force expansions compared to double-precision (defined in Equation (23)). The model used is a Hernquist sphere with $N = 10^6$. The top and bottom panels show the SCF and MEX force calculations, respectively. In both cases, the left (green) histogram is a lower-order expansion as indicated in the legend.

(A color version of this figure is available in the online journal.)

SCF coefficient among several realizations of the same triaxial Dehnen model, and found that for $N = 10^6$ particles, the angular terms beyond $l = 8$ are dominated by noise (and that only the first few n, m terms at that l level are reliable).

The force error discussed above is not directly related to energy diffusion or relaxation, which are reduced due to the smoothing but not absent. The mechanism for energy (and angular momentum) diffusion in both expansion methods is temporal fluctuation of the multipoles or coefficients (due to the particle noise). This is somewhat analogous to two-body relaxation, in that the potential felt by every particle fluctuates (although in this case there is no spatial graininess). E. Vasiliev (2014, in preparation) examined energy diffusion in a Plummer sphere with $N = 10^5$ particles using SCF and direct N-body codes and found that SCF demonstrated a diffusion rate only several times lower, which was close to the rate in a direct technique using near-optimal softening for this N. Further reduction was achieved by discarding expansion terms which are nominally zero in any triaxial system centered around the expansion center. Finally, Vasiliev used temporal softening (HO92) where the coefficients (and thus the potential) are updated in longer intervals than the dynamical time step; however, this procedure introduces a global energy error unless some measures are taken to amend this.

4.2. Single Precision

Due to their original intended use, GPUs are not optimized for double-precision arithmetic (indeed, early GPUs completely lacked a double-precision floating-point type). In cards that do support double-precision, arithmetic operations could still be significantly slower than for single. As noted before, in our test, we measured a 60%–65% speed increase when using single-precision. The Nvidia Tesla K20 GPUs we used have enhanced double-precision performance with respect to other GPUs, for which using double-precision may be significantly slower. Those devices are somewhat specialized for scientific use and are thus more expensive (albeit in many applications still superior to parallel CPU architectures in terms of price/performance ratio due to the low energy consumption). CPUs usually take the same time to perform an arithmetic operation in either single- or double-precision, but a program's general performance could be faster in single-precision due to smaller memory load. For the Hernquist-SCF CPU code, we measured a 6% improvement in speed. Using single-precision, however, inevitably reduces the accuracy of the calculated force; here, we examine how bad this *performance-accuracy trade-off* is.

Figure 11 show the relative force error distributions of single-precision calculations compared to double. The relative force error on particle i is now defined as

$$\tilde{\epsilon}_i \equiv \frac{|\mathbf{F}_{i,\mathrm{single}} - \mathbf{F}_{i,\mathrm{double}}|}{|\mathbf{F}_{i,\mathrm{double}}|}. \tag{23}$$

We test an $N = 10^6$ realization of a Hernquist sphere with a characteristic scale of one unit. The top panel shows two SCF force calculations: the green histogram (on the left) is a low-order expansion up to $(n_{\max}, l_{\max}) = (5, 2)$, retaining 36 coefficients; the red histogram is an expansion up to $(n_{\max}, l_{\max}) = (10, 2)$ retaining 308 coefficients. The bottom panel similarly shows two MEX expansions. In both cases, the higher order expansion has relatively large errors. While it is still smaller than the error with respect to the "real" force discussed in Section 4.1, its nature is numeric and it could hinder energy conservation.

The relatively large error is not remedied by the usual methods used to improve the accuracy of floating-point arithmetic, such as the Kahan summation algorithm (Kahan 1965), because the error does not come from the accumulation of round-off errors. Instead, the accuracy bottle neck is the calculation of the spherical harmonics and/or the Gegenbauer polynomials. Particles for which those special functions are calculated with large numerical error will have a large force error, but they also contribute erroneously to all of the coefficient or multipoles, thus causing some error in the force calculation of all of the other particles as well.

There are two groups of particles with large relative error in this implementation: particles that are very far away from the center, and particles which happen to lie very close to the

z axis. The former group is not so problematic since the absolute force is very small as is their contribution to the coefficients or multipoles. The latter group causes large error because the recursion relation used to calculate the associated Legendre polynomials,

$$\begin{aligned} P_{lm}(\theta) &= -2(m-1)\cot\theta\, P_{l,m-1}(\theta) \\ &= -(l+m-1)(l-m+2)P_{l,m-2}(\theta), \end{aligned} \tag{24}$$

is not upwardly stable because of the $\cot\theta$ factor, which diverges when the polar angle θ is very small or very close to π (although the polynomials themselves approach zero in these limits).

The distributions shown in Figure 11 may vary significantly depending on the model. For example, Ferrers ellipsoids are finite and flat at the center, and thus they do not contain the problematic particles described above and have much smaller error in single-precision. A Hernquist sphere is more representative of the general case in galaxies, being infinite and relatively centrally concentrated.

One could conceivably improve the accuracy for single-precision in several ways. In the test described above, everything was calculated in single-precision, apart from some constant coefficients that were only calculated once, in double-precision, and then cast to single. It may be possible to identify the most sensitive parts of the force calculation and use double-precision just for those or use pseudo-double-precision (as in Nitadori 2009) for part of or the entire force calculation routine. Another possibility is to keep using single-precision for everything but prescribe special treatment to those orbits close to the z axis.

5. DISCUSSION

5.1. The Methodology

On their own, expansion techniques are best geared to simulate systems with a dominant single center where it is important to minimize the effects of two-body relaxation, and where the system potential does not change radically (quickly) with time. An ideal class would be long-term secular evolution in a near-equilibrium galaxy.

Both methods presented in this work can be used to quickly calculate the gravitational potential and force on each particle in a many-body system while discarding small-scale structure. MEX comes from a Taylor-like expansion of the Green's function in the formal solution of the Poisson equation, while SCF is a Fourier-like expansion of the density. Both methods are important tools for collisionless dynamics and have been used extensively in astrophysics as discussed in the following sections. They are comparable in terms of both accuracy and performance. In both methods, there are free parameters to be set:

1. center of the expansion;
2. angular cutoff $l_{\max}$.

The center of the expansion could be the center of mass, but a better choice would be the bottom of the potential well. SCF has additional choices:

3. length unit;
4. radial basis set $\{\Phi_{nl}(r)\}$;
5. radial cutoff $n_{\max}$.

The choice of length unit (or model scaling) affects the accuracy of SCF expansion because the zeroth order of the radial basis functions corresponds to a model of a particular scale. For example, the basis set offered by *ETICS* corresponds to a Hernquist (1990) model with a scale length of $a = 1$.

The main difference for the end-user is that SCF smooths the radial direction as well. This could be an advantage when N is very small, since SCF will still provide a rather smooth potential, although it might not represent the real potential well at all due to random error. In MEX, particles are not completely unaware of each other, and every time two particles cross each other's shell, there is a discontinuity in the force which may lead to a large energy error when N is small. This shell crossing occurs when two particles change places in the r-sorted list, and the particles need not be close to each other at all.

Both methods have some problems close to the center. In SCF, the limitation comes from both radial and angular expansions. The radial expansion cutoff induces a bias if the central density profile does not match the zeroth basis function, and a very non-spherical model would cause force bias at the center as well as the lobes. The latter is also a problem for MEX, which has two additional problems: the discrete nature and inevitably small number of particles when one gets arbitrarily close to the center, as well as the numerical error (and/or small step size required) due to having to calculate $1/r$ (the SCF basis functions we use are completely regular at the center).

5.2. MEX

It is clear from the literature that SCF has been more popular by far. However, despite the above, we do not think that most authors intentionally avoided MEX; instead, SCF was better publicized and became the standard. MEX is rarely used in its full form, but more frequently in the spherically symmetric version, sometimes called the "spherical shells" method; in this case, just the monopole term is kept ($l_{\max} = 0$). For example, it is used in the Poisson solvers of Hénon (1975) Monte Carlo method. This hints that it might be easy to extend codes like MOCCA (Giersz et al. 2008) to non-spherical cases using our version of MEX.[10] This monopole approximation has also been used to study dark and stellar halo growth (Lin & Tremaine 1983; Nusser & Sheth 1999; Helmi & White 1999).

The extension of the spherical case using spherical harmonics exists in several variations, divided roughly into two classes: grid and gridless codes. The MEX version presented in this work is gridless and follows from Villumsen (1982) and White (1983). These authors used Cartesian instead of spherical coordinates, and softened the potential at the center. This softening, albeit similar mathematically, is not equivalent to particle-particle softening in direct N-body simulations and was just used to prevent divergence at the center.

The first MEX code, however, was by Aarseth (1967), who divided the simulation volume into thick shells, and calculated the force on a particle by summing the multipoles of all the shells except for its own (own-shell correction was added). Similarly, Fry & Peebles (1980) used an MEX code with $l_{\max} = 3$ to explore galaxy correlation functions; in their version, each shell had six particles and a softened Newtonian interaction was used within a shell. As noted in Section 1, van Albada & van Gorkom (1977) used a variation with axial symmetry (up to $l_{\max} = 4$ but with no azimuthal terms, namely, $m_{\max} = 0$), with a grid in both r and θ. In follow up work (van Albada 1982; Bontekoe & van Albada 1987; Bertin & Stiavelli 1989; Merritt & Stiavelli 1990), the method was extended to three-dimensional geometry. Finally, McGlynn (1984) used a grid in r only with logarithmic spacing. He argues that softening sacrifices the higher resolution

[10] Recently, E. Vasiliev (2014, in preparation) introduced a new Monte Carlo code that uses SCF as a potential solver.

near the center (which is one of the primary advantages of the method) and that a radial grid smooths the potential and prevents shell crossing. Recently, Vasiliev (2013) presented a similar potential solver with a spline instead of a grid.

We note that a virtually identical mathematical treatment to the MEX method has been applied to solve the Fokker–Planck equation under the local approximation (neglecting diffusion in position). The collisional terms of the Fokker–Planck equation can be written by means of the Rosenbluth potentials (Rosenbluth et al. 1957), which are integrals in velocity space very similar in form to Equation (2). Spurzem & Takahashi (1995) assumed azimuthal symmetry and wrote the Rosenbluth potentials using the Legendre polynomials up to $l_{\max} = 4$ in a way exactly analogous to our Equation (15). This treatment was expanded to $l_{\max} = 5$ by Schneider et al. (2011).

5.3. SCF

As noted in the previous section, SCF gained much more popularity than MEX. The SCF formalism has been widely used on galaxy-scale problems. It has been used to model the effect of black hole growth or adiabatic contraction on the structure (density profile) of the dark matter halo (e.g., Sigursson et al. 1995). SCF is also an appropriate tool to model the growth of the stellar and dark matter halos (e.g., Johnston et al. 1996; Lowing et al. 2011) as well as the mass evolution of infalling satellite galaxies (e.g., Holley-Bockelmann & Richstone 1999, 2000). One of the clearest uses of the SCF technique is when the stability of the orbit matters, such as in the study of chaos in galactic potentials (e.g., Holley-Bockelmann et al. 2001, 2002), and in the exchange of energy and angular momentum by mean resonances (e.g., Weinberg & Katz 2002; Holley-Bockelmann et al. 2005). Earn & Sellwood (1995) compare a number of methods and show that SCF is superior for stability work.

The initial motivation for this work was to follow up on Meiron & Laor (2012, 2013), who studied supermassive black hole binaries using a restricted technique. In their method, the stellar potential was held constant while the black holes were treated separately as collisional particles; it was thus not self-consistent in terms of the potential. This class of problems, where there is a small subset of particles which needs to be treated collisionally, has already been attempted using an extension of the expansion technique which hybridizes SCF and direct Aarseth-type gravitational force calculation; in these extensions, either the black holes are the only collisional particle (e.g., Quinlan & Hernquist 1997; Chatterjee et al. 2003), or all of the centrophilic particles are treated collisionally (Hemsendorf et al. 2002). MEX has not been applied to this particular problem to our knowledge, although it is as well suited as SCF.

5.4. Implementation

Our SCF implementation on GPU outperformed the serial Hernquist CPU version by a factor of $\sim$35 (for double-precision), but this number depends on the particular GPU and CPU hardware compared. The CPU code is definitely competitive on multi-core CPUs. Intel recently introduced the Many Integrated Core architecture (known as Intel MIC), which are shared memory boards with the equivalent of tens of CPUs. In principle, the Fortran SCF code for CPUs could be adapted for this architecture with little modification, and it will most likely outperform the GPU version. On the other hand, next-generation GPUs (such as Nvidia's Maxwell architecture) would also deliver performance-improving features and it is not clear which one would win. The goal of this project is to ultimately enable simulations of $N \geqslant 10^8$ and to perform them fast enough so that many could be performed, exploring a large parameter space rather than making a few such large-N simulations. To do that, the code will be adapted to a multi-GPU and multi-node machines using MPI. As noted in Section 3.1, this is easy for SCF but not so much for MEX.

Simultaneously, we will attempt to improve the per-GPU performance. We spent a lot of time trying to optimize this first version of *ETICS*, but by no means do we guarantee that out implementation is flawless. Some improvement might come from tweaking of the implementation. For example, we decided not to cache $P_{l0}(\cos\theta)$, but rather we recalculate it in-kernel before every m loop (as a starting point for the recursion relation). Since the Legendre polynomials are "hard-coded" and computed very efficiently, it is not immediately clear whether or not caching is a more efficient approach (it is probably worth while at very high $l_{\max}$). Likewise, we chose to separate the caching operations that are performed once per routine or once per external loop and execute them as independent kernels, while in principle they could be executed as statements inside the inner kernels (so-called kernel fusion, which would save kernel execution overhead) with an if statement making sure that the cache operations are performed only if needed.

Some possible more fundamental changes include trying to get rid of the sorting operation in MEX; while the most basic approach requires that the particle list be sorted and a cumulative sum performed over the multipoles, some alternatives exist, such as logarithmic grid (as in McGlynn 1984) or spline (as in Vasiliev 2013). Also, we might find a more sophisticated way to perform the cumulative sum, since we suspect that the *Thrust* routines are not optimal for our uses. Another improvement might come from the integration side rather than force-calculations, such as implementation of a higher-order integrator instead of the leapfrog. Hernquist's SCF code already contains a fourth order Hermite scheme (Makino 1991), which is not hard to implement for GPUs, but MEX has a fundamental problem with this scheme due to shell crossing, which causes the force derivatives to be discontinuous.

5.5. Final Remarks

ETICS is a powerful code, but as with any computer program, one should understand its limitation. The code in its current form should not be used for highly flattened systems or where two-body interactions are significant. The code is momentarily available upon request from the authors, but we plan to make it public, including a module to integrate it with the AMUSE framework (Portegies Zwart et al. 2009; Pelupessy et al. 2013).

We thank Peter Berczik, Adi Nusser, Marcel Zemp, and Eugene Vasiliev for the interesting and helpful discussions and the referee for useful comments. Y.M. is grateful for support from the China Postdoctoral Science Foundation through grant No. 2013M530471. The special GPU accelerated supercomputer Laohu at the Center of Information and Computing at National Astronomical Observatories, Chinese Academy of Sciences, has been used for the simulations; it was funded by Ministry of Finance of People's Republic of China, under the grant ZDY Z2008-2, and by the "Recruitment Program of Global Experts" (Qianren) for Spurzem (2013). R.S. has been partially supported by NSFC (National Natural Science Foundation of China) grant No. 11073025.

REFERENCES

Aarseth, S. 1967, Les Nouvelles Méthodes de la Dynamique Stellaire (Paris: Editions du Centre National Recherche Scientifique), 47
Allen, A. J., Palmer, P. L., & Papaloizou, J. 1990, MNRAS, 242, 576
Athanassoula, E. 1992, MNRAS, 259, 328
Athanassoula, E., Bosma, A., Lambert, J.-C., & Makino, J. 1998, MNRAS, 293, 369
Athanassoula, E., Fady, E., Lambert, J. C., & Bosma, A. 2000, MNRAS, 314, 475
Bell, N., & Hoberock, J. 2011, in GPU Computing Gems Jade Edition, ed. W.-M W. Hwu (Waltham, MA: Morgan Kaufmann), 359
Bertin, G., & Stiavelli, M. 1989, ApJ, 338, 723
Binney, J., & Tremaine, S. 2008, Galactic Dynamics (2nd ed.; Princeton, NJ: Princeton Univ. Press)
Bontekoe, T. R., & van Albada, T. S. 1987, MNRAS, 224, 349
Bouvier, P., & Janin, G. 1970, A&A, 5, 127
Brown, M. J. W., & Papaloizou, J. C. B. 1998, MNRAS, 300, 135
Chatterjee, P., Hernquist, L., & Loeb, A. 2003, ApJ, 592, 32
Clutton-Brock, M. 1972, Ap&SS, 16, 101
Clutton-Brock, M. 1973, Ap&SS, 23, 55
de Zeeuw, T. 1985, MNRAS, 216, 273
Dehnen, W. 1993, MNRAS, 265, 250
Earn, D. J. D. 1996, ApJ, 465, 91
Earn, D. J. D., & Sellwood, J. A. 1995, ApJ, 451, 533
Ferrers, N. M. 1877, QJPM, 14, 1
Fry, J. N., & Peebles, P. J. E. 1980, ApJ, 236, 343
Giersz, M., Heggie, D. C., & Hurley, J. R. 2008, MNRAS, 388, 429
Hamada, T., & Iitaka, T. 2007, arXiv:astro-ph/0703100
Helmi, A., & White, S. D. M. 1999, MNRAS, 307, 495
Hemsendorf, M., Sigursson, S., & Spurzem, R. 2002, ApJ, 581, 1256
Hénon, M. 1964, AnAp, 27, 83
Hénon, M. 1975, in IAU Symp. vol. 69, Dynamics of the Solar Systems, ed. A. Hayli (Dordrecht: Reidel), 133
Hernquist, L. 1990, ApJ, 356, 359
Hernquist, L., & Ostriker, J. P. 1992, ApJ, 386, 375
Hernquist, L., Sigursson, S., & Bryan, G. L. 1995, ApJ, 446, 717
Holley-Bockelmann, K., Mihos, J. C., Sigursson, S., & Hernquist, L. 2001, ApJ, 549, 862
Holley-Bockelmann, K., Mihos, J. C., Sigursson, S., Hernquist, L., & Norman, C. 2002, ApJ, 567, 817
Holley-Bockelmann, K., & Richstone, D. 1999, ApJ, 517, 92
Holley-Bockelmann, K., & Richstone, D. O. 2000, ApJ, 531, 232
Holley-Bockelmann, K., Weinberg, M., & Katz, N. 2005, MNRAS, 363, 991
Johnston, K. V., Hernquist, L., & Bolte, M. 1996, ApJ, 465, 278
Just, A., Khan, F. M., Berczik, P., Ernst, A., & Spurzem, R. 2011, MNRAS, 411, 653
Kahan, W. 1965, Commun. ACM, 8, 40
Kalapotharakos, C., Efthymiopoulos, C., & Voglis, N. 2008, MNRAS, 383, 971
Lin, D. N. C., & Tremaine, S. 1983, ApJ, 264, 364
Lowing, B., Jenkins, A., Eke, V., & Frenk, C. 2011, MNRAS, 416, 2697
Makino, J. 1991, ApJ, 369, 200
McGlynn, T. A. 1984, ApJ, 281, 13
Meiron, Y., & Laor, A. 2012, MNRAS, 422, 117
Meiron, Y., & Laor, A. 2013, MNRAS, 433, 2502
Merritt, D. 1996, AJ, 111, 2462
Merritt, D., & Fridman, T. 1996, ApJ, 460, 136
Merritt, D., & Stiavelli, M. 1990, ApJ, 358, 399
Nitadori, K. 2009, PhD thesis, Univ. Tokyo
Nusser, A., & Sheth, R. K. 1999, MNRAS, 303, 685
Ostriker, J. P., & Mark, J. W.-K. 1968, ApJ, 151, 1075
Pelupessy, F. I., van Elteren, A., de Vries, N., et al. 2013, A&A, 557, A84
Plummer, H. C. 1911, MNRAS, 71, 460
Portegies Zwart, S., McMillan, S., Harfst, S., et al. 2009, NewA, 14, 369
Portegies Zwart, S. F., Belleman, R. G., & Geldof, P. M. 2007, NewA, 12, 641
Quinlan, G. D., & Hernquist, L. 1997, NewA, 2, 533
Rahmati, A., & Jalali, M. A. 2009, MNRAS, 393, 1459
Rosenbluth, M. N., MacDonald, W. M., & Judd, D. L. 1957, PhRv, 107, 1
Saha, P. 1993, MNRAS, 262, 1062
Satish, N., Harris, M., & Garland, M. 2009, in IEEE International Parallel & Distributed Processing Symposium, ed. A. Mei, S. Barua, S. Jelinek, V. Prasanna, & G. Westrom (Washington, DC: IEEE Computer Society), 1
Schaefer, M. M., Lecar, M., & Rybicki, G. 1973, Ap&SS, 25, 357
Schive, H.-Y., Chien, C.-H., Wong, S.-K., Tsai, Y.-C., & Chiueh, T. 2008, NewA, 13, 418
Schneider, J., Amaro-Seoane, P., & Spurzem, R. 2011, MNRAS, 410, 432
Schröder, P., & Sweldens, W. 1995, in Computer Graphics Proceedings, ed. S. G. Mair & R. Cook (SIGGRAPH 95) (New York: ACM), 161
Sellwood, J. A. 1987, ARA&A, 25, 151
Sigursson, S., Hernquist, L., & Quinlan, G. D. 1995, ApJ, 446, 75
Spurzem, R., Berczik, P., Berentzen, I., et al. 2012, in Large-Scale Computing Techniques for Complex System Simulations, ed. W. Dubitzky, K. Kurowski, & B. Schott (Hoboken, NJ: John Wiley & Sons), 35
Spurzem, R., & Takahashi, K. 1995, MNRAS, 272, 772
Syer, D. 1995, MNRAS, 276, 1009
Toomre, A. 1963, ApJ, 138, 385
van Albada, T. S. 1982, MNRAS, 201, 939
van Albada, T. S., & van Gorkom, J. H. 1977, A&A, 54, 121
Vasiliev, E. 2013, MNRAS, 434, 3174
Villumsen, J. V. 1982, MNRAS, 199, 493
Weinberg, M. D. 1996, ApJ, 470, 715
Weinberg, M. D. 1999, AJ, 117, 629
Weinberg, M. D., & Katz, N. 2002, ApJ, 580, 627
White, S. D. M. 1983, ApJ, 274, 53
Zhao, H. 1996, MNRAS, 278, 488