Mon. Not. R. Astron. Soc. **000**, 1–7 (2011) Printed 8 August 2011 (MN LATEX style file v2.2)

Accelerating NBODY6 with Graphics Processing Units

Keigo Nitadori[1⋆] and Sverre J. Aarseth[2⋆]
[1] *Center for Computational Science, University of Tsukuba, 1-1-1, Tennodai, Tsukuba, Ibaraki 305-8577, Japan*
[2] *Institute of Astronomy, University of Cambridge, Madingley Road, Cambridge, CB3 0HA, UK*

Accepted 2011. Received 2011; in original form 2011 August 1

ABSTRACT
We describe the use of Graphics Processing Units (GPUs) for speeding up the code $NBODY6$ which is widely used for direct N-body simulations. Over the years, the N^2 nature of the direct force calculation has proved a barrier for extending the particle number. Following an early introduction of force polynomials and individual time-steps, the calculation cost was first reduced by the introduction of a neighbour scheme. After a decade of GRAPE computers which speeded up the force calculation further, we are now in the era of GPUs where relatively small hardware systems are highly cost-effective. A significant gain in efficiency is achieved by employing the GPU to obtain the so-called regular force which typically involves some 99 percent of the particles, while the remaining local forces are evaluated on the host. However, the latter operation is performed up to 20 times more frequently and may still account for a significant cost. This effort is reduced by parallel SSE procedures where each interaction term is calculated in C++ and single precision. We also discuss further strategies connected with coordinate and velocity prediction required by the integration scheme. This leaves hard binaries and multiple close encounters which are treated by several regularization methods. The present $NBODY6$*–GPU* code is well balanced for simulations in the particle range $10^4 - 2 \times 10^5$ for a dual GPU system attached to a standard PC.

Key words: globular clusters: general – methods: numerical

1 INTRODUCTION

The quest to perform efficient N-body calculations has challenged astronomers and computer scientists ever since the early 1960s. For a long time progress was slow but so was the increase in computing power. The first significant advance was achieved by the Ahmad–Cohen (1973, AC) neighbour scheme which splits the total force into a distant slowly changing part and a local contribution with shorter time-scale, hereafter denoted as the regular and irregular force. Considerable progress on the hardware side was made when the GRAPE-type special-purpose computers were developed in the early 1990s (Makino, Kokubo & Taiji 1994) and later improved to GRAPE-4 and GRAPE-6 (Makino et al. 1997, Makino et al. 2003). More recently, the general availability of Graphics Processing Units (GPUs) and the corresponding CUDA programming language have facilitated large speed-up of N-body simulations at modest cost. This revolution has already led to Petaflop performance by combining several thousand GPUs.

In this paper, we are mainly concerned with small stand-alone systems using one or two GPUs with the code $NBODY6$*–GPU*. However, mention should also be made of the parallel version $NBODY6^{++}$ which is capable of reaching somewhat larger particle numbers using several types of hardware (Spurzem 1999) and is also intended for GPUs.

This paper is organized as follows. After reviewing some relevant aspects in the standard $NBODY6$ code we discuss the new treatment of the regular and irregular force. As a result of improvements in the regular force calculation, the irregular force now becomes relatively expensive. Although the latter may also be evaluated on the GPU, the overheads are too large. Instead we perform this calculation in parallel using SSE [1] (Streamlined SIMD Extensions) and OpenMP in C++ with GCC built-in functions. The presence of hard binaries requires careful attention, especially because the regularization scheme involves different precision for the c.m. motion as evaluated by FORTRAN and SSE. A later implementation with AVX accelerated the irregular force calculation. The performance gain is illustrated by comparison with the basic version at relatively small particle numbers and we estimate the cost of doing large simulations for two types of hardware. Finally, we summarize current experience with small GPU systems and point to possible future developments.

2 BASIC NBODY6 CODE

The code $NBODY6$ was developed during the late 1990s (Aarseth 1999). It was based on a previous code $NBODY5$ which also employed the AC neighbour scheme. Here the main improvement was

⋆ E-mail:keigo@css.tsukuba.ac.jp (KN); sverre@ast.cam.ac.uk (SJA)

[1] For computational terms, see glossary in Appendix A.

to replace the fourth-order Adams method by an equivalent Hermite formulation (Makino 1991) and this was extended to the case of two force polynomials (Makino & Aarseth 1992). An intermediate step was made with the Hermite individual time-step code NBODY4 (Aarseth 1996) designed for use by several generations of GRAPE-type computers.

The combination of Hermite integration with block-steps has proved a powerful tool in N-body simulations. At earlier times its simple form was beneficial for using together with the special-purpose GRAPE hardware which supplies the force and its first derivative. This allows for the construction of an efficient and accurate fourth-order integration method. The subsequent introduction of hierarchical block-steps reduces the overheads of coordinate and velocity predictions considerably and also facilitates parallel procedures. Experience has shown that the so-called HACS scheme is highly cost-effective in the standard NBODY6 code.

Efficient implementation benefits from access to contiguous arrays of the variables. The main data structure consists of three separate categories which are required for different purposes. Given $N_{\rm p}$ regularized pairs, the first $2N_{\rm p}$ members of the particle arrays refer to the pairwise components, then comes the single particles up to N followed by the corresponding c.m. particles. Consequently, the force on a single particle involves summation from $2N_{\rm p}+1$ to $N+N_{\rm p}$, where any nearby c.m. systems are replaced by the components. Centre of mass corrections due to perturbers are carried out on the host. Here it is beneficial that the associated neighbour list is sequential, with any other c.m. particles appearing at the end.

The full integration scheme has been described in detail elsewhere (Aarseth 2003). Here we first comment on the choice of time-step which plays a critical role in simulations. Let us denote the force per unit mass (acceleration) on a particle by $\mathbf{F}$. Basic time-steps of the type $(\eta|\mathbf{F}|/|\ddot{\mathbf{F}}|)^{1/2}$ with η a dimensionless parameter provide essential convergence in most cases. However, after considerable experimentation we have adopted a more sensitive composite criterion given by

$$\Delta t_i = \left(\eta \frac{|\mathbf{F}||\ddot{\mathbf{F}}| + |\dot{\mathbf{F}}|^2}{|\dot{\mathbf{F}}||\dddot{\mathbf{F}}| + |\ddot{\mathbf{F}}|^2} \right)^{1/2}. \quad (1)$$

This expression is still well defined in the case of zero force or first derivative. Moreover, all the known terms in the corresponding Taylor series play a part, making it a relative criterion where the higher orders contributes preferentially to step reduction during close encounters. In the limit of large N, it has proved beneficial to employ a secondary criterion based on the difference between the predicted and corrected velocity (Makino 1991, Aarseth 2003). The new time-step is chosen according to the rules of the block-step scheme, where an increase by factor 2 is only permitted every other step but reduction by 2 or even 4 may occur every time. The hierarchical time-steps typically extend over about 15 levels, with the lower limit representing the selection criterion for two-body regularization. For convenience, the regular time-steps are also commensurate and usually a factor of 10–20 times larger than the corresponding irregular steps.

The presence of binaries invariably leads to multiple encounters with frequent ejection of high-velocity members. Special precautions are needed to prevent over-shooting with abrupt changes in the irregular force. Strong interactions involving 3–5 particles are treated by chain regularization which has proved itself (Mikkola & Aarseth 1993). Because the regularized time is non-linear, the commensurability requirement frequently results in very small time-steps when initializing force polynomials of both types at termination. However, an increase by factor of 4 is often possible and only a small number of particles are affected.

On conventional computers, the regular force calculation dominates the CPU time, with only a weak dependence on the neighbour strategy. Thus there are compensating factors when the number of neighbours (n_i) is varied. The new neighbours are chosen at the time of a regular force calculation. Hence all particles inside the corresponding neighbour radius are selected, together with any particles in an outer shell approaching with small impact parameter. Individual neighbour radii $R_{\rm s}$ are adjusted according to the local density contrast, with additional modifications near the upper and lower boundary. New values are obtained by the expression

$$R_{\rm s}^{\rm new} = R_{\rm s}^{\rm old} \left(\frac{n_{\rm p}}{n_i} \right)^{1/3}, \quad (2)$$

where $n_{\rm p}$ is predicted from the local density contrast. Note that the case of zero neighbour number which may occur for distant particles is also catered for. Formally, explicit derivative corrections to the force polynomial should be carried out for each neighbour change. However, the same terms are added and subtracted from the respective polynomials so that this overhead may be omitted, provided the desired results are obtained at times commensurate with the maximum time-step (Makino & Aarseth 1992).

The treatment of close encounters forms a large part of the code. We distinguish between two-body and multiple encounters which are studied by the tools of Kustaanheimo-Stiefel (1965, KS) and chain regularization (Mikkola & Aarseth 1993). Several methods for integrating the KS equations of motion have been used over the years. The preferred method is a high-order Hermite scheme (Mikkola & Aarseth 1998) and an iterative solution without recalculating the external perturbation. This allows the physical time to be obtained by a sixth-order Taylor series expansion of the time transformation $t' = R$. The so-called Stumpff method maintains machine accuracy in the limit of small perturbations and only requires half the number of steps per orbit compared to a standard fourth-order polynomial method employed by NBODY5. Even so, small systematic errors are present over long time intervals. Hence a rectification of the KS variables $\mathbf{u}, \mathbf{u}'$ is performed to be consistent with the *integrated* value of the binding energy (Fukushima 2005). Further speed-up can be achieved for small perturbations by employing the slow-down concept in which the time and perturbing force is magnified according to the principle of adiabatic invariance (Mikkola & Aarseth 1996).

We also mention the useful simplification of assuming unperturbed two-body motion. The limit adopted for the dimensionless perturbation γ is 1×10^{-6} which corresponds to a distance ratio $d/R = 100$ for equal masses in the tidal approximation. This enables keeping constant Kepler elements and KS variables during the next period. It is also possible to extend the unperturbed motion over many orbits by examining the velocity field of the c.m. neighbours (Aarseth 2003). Note that the c.m. force acting on an unperturbed binary is obtained in the same direct way as for single particles, otherwise the contribution from perturbers are added to each component.

Given a population of hard binaries, close encounters between binaries and field stars or other binaries are an interesting feature particularly because collisions or violent ejections may occur. The chain concept led to a powerful method for studying strong interactions where two-body singularities are removed (Mikkola & Aarseth 1993). Implementation of perturbed chain regularization introduces many complexities, as well as dealing with internal tidal effects, membership changes or post-Newtonian terms (Aarseth

2003). As far as the N-body code is concerned, the associated c.m. is integrated like a single particle with the force and first derivative obtained by mass-weighted summation over the components. The internal motions are advanced by a high-order integrator (Bulirsch & Stoer 1966) with energy conservation better than 10^{-10}. Moreover, the solutions for internal KS or chain are continued until the end of the new block-step before the other particles are treated. As for KS termination, this is implemented exactly at the end by a simple iteration. Internal chain integration, on the other hand, is only advanced while the new time is less than the block-time and any coordinates required by other particles are obtained by prediction. Consequently, chain termination is performed at an arbitrary time and a new current commensurate time is constructed by suitable subdivision. This in turn reduces the block-step further but is compensated by the relatively small number of chain treatments.

The question of when to use chain regularization needs to be addressed. Note that the data structure only allows one perturbed three- or four-body configuration at a time. However, the integration interval is usually short and there is also the facility of selecting one unperturbed three-body (Aarseth & Zare 1974) or four-body (Heggie 1974, Mikkola 1985) system for regularization treatment. Since at least one binary is involved when choosing multiple regularization, it is natural to begin with a strongly perturbed binary of small c.m. time-step. Only suitably compact configurations are selected and for convenience the standard criteria used by KS are employed, making for a uniform treatment. For the purpose of force calculation, the associated c.m. is distinguished by a unique identifier (i.e. `NAME = 0`) which refers to one of the original members. The other components are assigned zero mass and do not contribute to the force summation. This data structure facilitates termination in the form of one or two KS systems, or one KS binary and up to four single particles as appropriate (Aarseth 2003).

Finally, in this paper, we omit a discussion of synthetic stellar evolution which is optional and forms a large part of the code. Hence for the purpose of GPU developments these aspects may be ignored for simplicity. In any case, the additional CPU time required is relatively small.

3 NEW IMPLEMENTATIONS

We now turn to describing some relevant procedures associated with the new implementations.

3.1 Software design

A subroutine `INTGRT` in *NBODY6* drives the numerical integration of the AC neighbour scheme. It uses two libraries during the integration, prepared for the calculation of the regular and the irregular force, named `GPUNB` and `GPUIRR`. Although the names of the libraries include `GPU`, non-GPU implementations exist, e.g. plain C++ versions or high performance versions with SSE/AVX and OpenMP. The present *NBODY6-GPU* code employs a GPU version written in CUDA (with multiple-GPU support) for `GPUNB` and a CPU version with an acceleration by SSE/AVX and OpenMP for `GPUIRR`. A first implementation of `GPUIRR` used GPU. However, it was slower than a fine-tuned CPU code because of the *fine-grained* nature of the irregular force calculation.

Fig. 1 illustrates a schematic diagram for the relation between `intgrt.f`, `GPUNB` and `GPUIRR`. Here, `GPUNB` receives positions, velocities, masses and time-steps of the attracting particles (so called j-particles) and positions, velocities and neighbour radii of the attracted particles (i-particles), and it returns regular forces, their time derivatives and neighbour lists. `GPUIRR` holds the position and up to its third time derivatives, mass, the time of last irregular integration and the current neighbour list of each particle, and returns the irregular force and its time derivative for active particles. After an irregular step of particle i, the variables $\mathbf{R}_i, \dot{\mathbf{R}}_i, \mathbf{F}_i, \dot{\mathbf{F}}_i, t_{\mathrm{I},i}$ and m_i are sent to `GPUIRR` and after a regular step, the list $\mathbb{L}_i$ is sent.

3.2 Velocity neighbour criterion

The traditional neighbour criterion that a particle j is a neighbour of particle i is defined by

$$|\mathbf{R}_{ij}| < h_i, \tag{3}$$

where $\mathbf{R}_{ij} = \mathbf{R}_j - \mathbf{R}_i$ and h_i is the radius of the neighbour sphere. In this implementation, we employ a modified criterion to include the velocities by the condition

$$R_{ij,\min} \stackrel{\mathrm{def}}{=} \min\left(|\mathbf{R}_{ij}|, |\mathbf{R}_{ij} + \Delta t_{\mathrm{R},i}\dot{\mathbf{R}}_{ij}|\right) < h_i, \tag{4}$$

where $\Delta t_{\mathrm{R},i}$ is the regular time-step. Although this increases the computational cost for each pairwise interaction evaluation in the regular force calculation, we can take larger regular time-steps for the same number of neighbour. This safety condition also ensures that high-velocity particles can be added to the neighbour list before they come too close.

3.3 Block-step procedure

Let us examine the sequential procedure for one block-step.

(i) Obtain the next time for integration,

$$t_{\mathrm{next}} = \min_{1 \leqslant i \leqslant N}(t_{\mathrm{I},i} + \Delta t_{\mathrm{I},i}), \tag{5}$$

where $t_{\mathrm{I},i}$ and $\Delta t_{\mathrm{I},i}$ are the time of the last irregular force calculation and irregular time-step of particle i.

(ii) Make the active particle list for regular and irregular force calculation,

$$\mathbb{L}_{\mathrm{act,R}} = \{i \mid t_{\mathrm{R},i} + \Delta t_{\mathrm{R},i} = t_{\mathrm{next}}\}, \tag{6}$$

$$\mathbb{L}_{\mathrm{act,I}} = \{i \mid t_{\mathrm{I},i} + \Delta t_{\mathrm{I},i} = t_{\mathrm{next}}\}, \tag{7}$$

where the subscripts `R` and `I` denote regular and irregular terms. Here, $\{i \mid cond.\}$ defines a set of i such that it satisfies $cond.$

(iii) Predict all particles needed for force evaluation.

(iv) Calculate the irregular force and its time derivative for particle $i \in \mathbb{L}_{\mathrm{act,I}}$,

$$\mathbf{F}_{\mathrm{I},i} = \sum_{j \in \mathbb{L}_i} m_j \frac{\mathbf{R}_{ij}}{|\mathbf{R}_{ij}|^3}, \tag{8}$$

$$\dot{\mathbf{F}}_{\mathrm{I},i} = \sum_{j \in \mathbb{L}_i} m_j \left[\frac{\dot{\mathbf{R}}_{ij}}{|\mathbf{R}_{ij}|^3} - 3\frac{(\mathbf{R}_{ij} \cdot \dot{\mathbf{R}}_{ij})\mathbf{R}_{ij}}{|\mathbf{R}_{ij}|^5}\right]. \tag{9}$$

(v) Apply the corrector for the active irregular particles and decide the next time-step $\Delta t_{\mathrm{I},i}$.

(vi) Accumulate the regular force and its time derivative for each active regular particle $i \in \mathbb{L}_{\mathrm{act,R}}$ and construct the neighbour list,

$$\mathbf{F}_{\mathrm{R},i} = \sum_{j \neq i}^{N} \begin{cases} m_j \dfrac{\mathbf{R}_{ij}}{|\mathbf{R}_{ij}|^3} & (R_{ij,\min} < h_i) \\ 0 & (\text{otherwise}) \end{cases}, \tag{10}$$

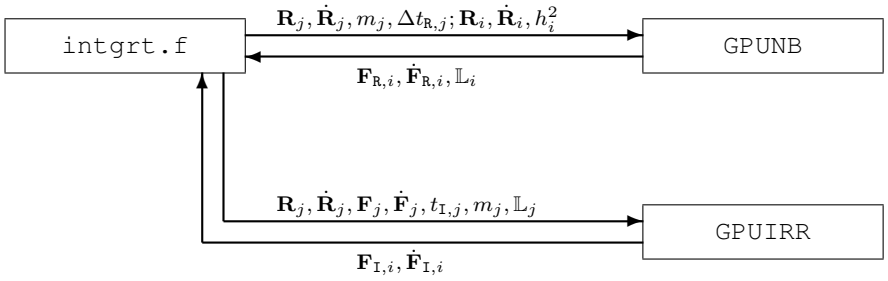

Figure 1. A schematic diagram of the $NBODY6$-GPU code.

$$\dot{\mathbf{F}}_{\mathrm{R},i} = \sum_{j \neq i}^{N} \begin{cases} m_j \left[\frac{\dot{\mathbf{R}}_{ij}}{|\mathbf{R}_{ij}|^3} - 3\frac{(\mathbf{R}_{ij}\cdot\dot{\mathbf{R}}_{ij})\mathbf{R}_{ij}}{|\mathbf{R}_{ij}|^5} \right] & (R_{ij,\min} < h_i) \\ 0 & (\text{otherwise}) \end{cases}, \tag{11}$$

$$\mathbb{L}_i = \{ j \mid j \neq i, R_{ij,\min} < h_i \}. \tag{12}$$

(vii) Execute the regular corrector. Since the neighbour list $\mathbb{L}_i$ has been updated, the force polynomials should be corrected to reflect the difference between the old and new list.

3.4 The `GPUNB` library

The `GPUNB` library computes regular forces and creates neighbour lists for a given set of particles with single or multiple GPU(s). First, the predicted position, velocity and mass of N_j particles are sent from the host. The regular force and neighbour lists on N_i particles are then computed. Therefore, $N_i N_j$ pairwise interactions are evaluated in one call. Unless N_i is much smaller than N_j, the cost of the prediction and data transfer is not significant.

The basic method to calculate the force on GPUs is common to the previously existing implementations (Nitadori 2009, Gaburov 2009), except for the treatment of neighbours. During the accumulation of forces from all N_j particles, when j is a neighbour, `GPUNB` skips the accumulation and saves the index j in the neighbour list. This procedure is applied for all N_i particles in parallel, using the many *threads* of the GPU.

Now we discuss more in detail the force calculation procedure of `GPUNB`. Each i-particle is assigned to each thread of GPU. The position, velocity, neighbour radius, force, its time derivative and neighbour count of particle i are held on the registers of each thread. The position, velocity and mass of particle j are broadcast from the memory to all the threads in a thread-block, and forces on multiple i-particles are evaluated in parallel. If particle j turns out to be a neighbour of i, the index j is written to the neighbour list in the memory.

The actual behaviour of each thread is given in Listing B2 with the CUDA C++ language. There is an 'if statement' for the neighbour treatment, which is translated into a mask operation by the compiler and has little impact on performance. If the branch was not removed, it would be a serious overhead for parallel performance.

As well as the i-parallelism for multiple threads, j-parallelism for multiple thread-blocks and multiple GPUs are also exploited. Thus, after the force computations, partial forces and partial neighbour lists of certain particles are distributed for multiple thread-blocks. To minimize the data transfer from GPU to host, we prepared kernels for force reduction and list gathering. Partial forces are summed and sparsely scattered partial lists are serialized to a linear array before they are sent back to the host PC.

Finally, single precision arithmetics turned out to provide sufficiently accurate results for practical use. This is because all the close interactions are skipped in the regular force procedure [2].

3.5 The `GPUIRR` library

The name of the library `GPUIRR` comes from an early effort to accelerate the irregular force calculation on GPU, although the GPU is not used in the current implementation, just because it is slower than a well tuned CPU code. Still, the name of the library and its API are used in the fast version on CPU with SSE/AVX and OpenMP.

Different from `GPUNB`, `GPUIRR` retains many internal states. The position and up to its third derivative, mass and time of the last integration of particle i are held to construct the predictor. These quantities are updated after each irregular step. Additionally, the neighbour list of particle i is saved for the computation of the irregular force. The list is copied from the host routine `intgrt.f` after each regular step. When an irregular force on particle i is requested, the library calculates $\mathbf{F}_{\mathrm{I},i}$ and $\dot{\mathbf{F}}_{\mathrm{I},i}$. Actually, it can be performed in parallel, i.e. the library receives a list of irregular active particles $\mathbb{L}_{\mathrm{act,I}}$ and returns an array of the force and its time derivative.

The library is tuned for multi-core CPUs and SIMD instructions such as SSE and AVX. An OpenMP parallelization for different i-particles is straightforward. On the other hand, the SIMD instructions are exploited for the j-parallelism for one i-particle, which requires technical coding. To calculate pairwise forces on a particle i from multiple (4-way for SSE, 8-way for AVX) j-particles, we need to gather them from non-contiguous addresses indicated by the neighbour list. The gathering process is relatively expensive and the GFLOPS rating or 'number of interactions per second' is deceased to about 40 percent of the case of brute force calculations (Tanikawa et al. 2011) on the same processor with the same AVX instruction set.

Unlike the regular force calculations, full single precision calculation of the irregular force may not be accurate enough during close encounters. Thus, we employ the so-called 'two-float' technique to express the coordinates of each particle (Nitadori 2009, Gaburov 2009). For the summation of force, single precision turned

[2] We still left options using the 'two-float' method for the coordinates and the accumulator.

out to be sufficient since we do not accumulate too many terms (more than a few hundred) in the irregular force calculation.

Prediction of positions and velocities are performed inside the library. When $N_{\rm act,I}$ is small, predicting all the N particles is not efficient. Thus, at some point (cf. NPACT in Table C1), we switch to predict only the necessary particles in this block-step. Even if we only predict the necessary particles, combining and sorting of the neighbour lists or random memory access may be expensive. Thus there is a turn-around point, and an example is shown in the lines 24-29 in Listing C1.

3.6 Binary effects

The presence of binaries poses additional complications when combining results of force calculations from the host and the library for irregular force. As can be seen above, the irregular force on a single particle due to other single particles is of simple form and its calculation can be speeded up in C++ with SSE/AVX and OpenMP. This is no longer the case for regularized systems where differential force corrections are needed on the host to compensate for the c.m. approximation which is employed. For simplicity we assume that only the irregular force needs to be corrected; this in tern implies that the neighbour radius is sufficiently large. A further complication arises in the force evaluation due to perturbers. The numerical problem can be illustrated by considering the new regular force difference during the corresponding time interval $t - t_{{\rm R},i}$,

$$\Delta \mathbf{F}_{\rm R} = (\mathbf{F}_{\rm R}^{\rm new} - \mathbf{F}_{\rm R}^{\rm old}) + (\mathbf{F}_{\rm I}^{\rm new} - \mathbf{F}_{\rm I}^{\rm old}), \quad (13)$$

where $\mathbf{F}_{\rm R}^{\rm old}$ and $\mathbf{F}_{\rm R}^{\rm new}$ denote regular forces evaluated at the old and new times $t_{{\rm R},i}$ and t, respectively, while $\mathbf{F}_{\rm I}^{\rm old}$ and $\mathbf{F}_{\rm I}^{\rm new}$ denote irregular forces both evaluated at the new time t with the old and new neighbour lists. Hence the net change of irregular force is contained within the second bracket. In the case of no neighbour change, this term should be suitably small, otherwise it would tend to reduce the regular time-step. It is therefore important to ensure consistency between the irregular force as calculated at each time-step and that obtained elsewhere at the end of a regular step where both are based on the same *predicted* quantities.

In the case of an active KS binary, the old and new force contributions due to perturbers are acquired in double precision on the host and similarly for the c.m. term which is subtracted. Since the latter is evaluated in the single precision GPUIRR library, this means that a small discontinuity is introduced. However, the old and new irregular force are still numerically identical in the absence of neighbour change and hence the same neighbours do not affect the regular force difference. A similar differential force procedure is carried out for single particles having perturbed binaries as neighbours. Note that the sequential ordering of neighbour lists facilitates the identification of KS binaries for various purposes. Finally, force corrections involving a chain c.m. particle are performed analogously, where the internal contributions to $\mathbf{F}$ and $\dot{\mathbf{F}}$ are obtained by summation over the members.

The strategy for controlling the number of neighbours also changes when using the GPU. Unless present initially, binaries eventually become an important feature of N-body simulations. It is therefore desirable to strike a balance between the maximum size of regularized binaries and the neighbour radius. This can readily be done for long-lived binaries, subject to the maximum membership denoted by the parameter NBMAX. The average neighbour number can also be controlled to a certain extent. This is achieved by an optional adjustment of a density contrast parameter α used by equation (2). Fortunately the ratio $R_{\rm s}/a$ becomes more favourable for large N because the semi-major axis for hard binaries is $\propto 1/N$ while the neighbour radius reduces much more slowly. Hence a safe strategy for small and intermediate simulations is to employ a relatively large value of NBMAX together with fine-tuning of α. Also note that the possible problem of too small neighbour radii is mostly relevant for the central cluster region because $R_{\rm s}$ tends to be larger at lower density.

Table 1. Hardware and software configurations.

	System A	System B
CPU	Core i7–920	Core i7–2600K
(spec)	4 cores, 2.66 GHz	4 cores, 3.40 GHz
GPU	2×GeForce GTX 470	2×GeForce GTX 560 Ti
Motherboard	MSI X58	MSI P67
Memory	6 GB, DDR3–1333	8 GB, DDR3–1600
OS	CentOS 5.5 x86_64	CentOS 6.0 x86_64
Compilers	GCC 4.1.2, CUDA 3.0	GCC 4.6.1, CUDA 4.0

Table 2. Performance summary for System B. We show total wall-clock-time, partial time for regular and irregular force. The number of regular and irregular individual steps is also given, together with the time-step weighted average neighbour numbers.

N	32k	64k	128k	256k
$T_{\rm wall}/{\rm sec}$	18	59	209	802
$T_{\rm reg}$ /sec	5.3	19	84	338
$T_{\rm irr}$ /sec	4.7	13	47	230
$\mathcal{N}_{\rm reg}/10^6$	2.8	6.1	13	28
$\mathcal{N}_{\rm irr}/10^6$	33	85	220	550
$\langle N_{\rm nb} \rangle$	46	55	64	83

4 PERFORMANCE

The acid test of any code is measured by its performance. This is especially the case for N-body simulations where an important challenge is to describe the dynamics of globular clusters. Here we report on some performance tests to illustrate the calculation cost for a wide range of particle numbers. It should be emphasized that timings based on idealized systems do not demonstrate the capability of dealing with more advanced stages which are characterized by large density contrasts and the presence of hard binaries. Additional examinations of more realistic conditions are therefore highly desirable, as commented on below.

For the basic tests we study isolated Plummer models in virial equilibrium with equal-mass particles in the range $N = 8$ k–256 k. We use standard N-body units with total energy $E = -0.25$ and total mass 1. The hardware specifications of the test systems are shown in Table 1. Timing tests were first made for System A which employs SSE with OpenMP.

In order to have a more balanced dynamical state, the computing times are measured from $t = 2$ to $t = 4$. The wall-clock time (in sec) as a function of N is shown in Fig. 2 for both systems. Also shown separately are the times for the regular and irregular force calculation. As can be seen, these timings are quite similar for System A, especially in the large-N limit. The minimum cost is achieved for particular choices of the upper limit neighbour number, NBMAX, which are close to values used in long-term simulations. As for the remaining time expended by the FORTRAN part, it forms a diminishing fraction of the total effort, with about 40 percent for $N = 32$ k and 20 percent for $N = 256$ k.

Timing comparisons for the standard $NBODY6$ code have also

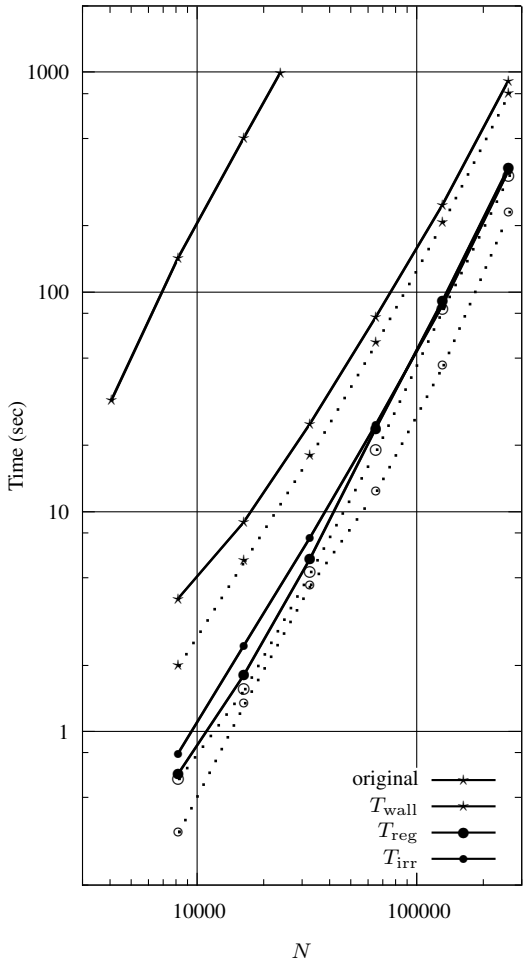

Figure 2. Performance timings of the original $NBODY6$ and the $NBODY6$-GPU code. Stars denote the wall-clock-time for the interval $2 \leqslant t \leqslant 4$. Large and small circles plot the time for the libraries `GPUNB` and `GPUIRR`. Solid and dotted lines show results for systems A and B.

been carried out. Thus at the upper limit of $N = 24000$, the wall-clock time is 56 times the corresponding time for System A, with an asymptotic dependence $T_{\rm comp} \propto N^{1.8}$ in both cases. Excluding the time consumption of the other parts, the performance of the regular force calculation with System A exceeds 1 TFLOPS. As shown in Fig. 2, the CPU times for the irregular force with System B are almost a factor of 2 faster than for System A when the slightly faster clock is included. This improvement is mainly due to the use of AVX instructions which became available very recently. In comparison with Table 1, the actual timings for $N = 256$ k and System A were 911, 366 and 354 sec, respectively [3].

The difference between using one and two GPUs is of interest. For the present System A, the total wall-clock time for $N = 64$ k is reduced from 99 to 77 sec. This raises the possibility of doing two separate simulations at the same time on a system with two GPUs and suffering a small loss of efficiency. Alternatively, further speed-up may soon be possible using four GPUs with System B.

The question of reproducibility is a difficult one, particularly in view of the chaotic nature exhibited by the N-body problem. On the other hand, the calculations are speeded up significantly by using parallel OpenMP procedures. At present, two important parallel treatments (lines 46–50 and 104–115 of Listing C1) are not thread-safe while some other FORTRAN parts are well behaved. Consequently, strict reproducibility can be enforced by omitting these procedures at some loss of efficiency. However, the timing tests were carried out with full optimization, bearing in mind that the early stage is essentially reproducible in any case.

Most N-body simulations, whether they be core collapse or substantial evaporation, are concerned with long time-scales. It is therefore desirable to assess the code behaviour for more advanced stages of evolution with high core density which inevitably leads to binary formation and strong interactions in compact subsystems. At this stage the special treatments of close encounters in the form of KS and chain regularization begin to play an important role. The ejection of high-velocity members is a hallmark of an evolved dynamical state. In general, the numerical problems due to strong interactions are associated with larger energy errors, and are partly the result of regularization switching procedures. Even so, a study of core collapse for an equal-mass system with $N = 32$ k shows that the accumulated changes in total energy are surprisingly small, amounting to $\sum \Delta E_j = -1 \times 10^{-4}$ at minimum core radius. A comparable drift in total energy was also seen in a test calculation well beyond core collapse for $N = 16$ k. Hence the Hermite integration scheme has proved to possess excellent long-term stability.

5 CONCLUSIONS

We have presented new implementations for efficient integration of the N-body problem on GPUs. In the standard $NBODY6$ code, the regular force calculation dominates the CPU time. Consequently, the emphasis here has been on procedures for speeding up the force calculation. First the regular force evaluation was implemented on the GPU using the library `GPUNB` which also forms the neighbour list. This procedure is ideally suited to massively parallel force calculations on GPUs and resulted in significant gains. However, a subsequent attempt to employ the GPU for the irregular force showed that the overheads are too large. It turned out that different strategies are needed for dealing with the regular and irregular force components and this eventually led to the development of the special library `GPUIRR`. The use of SSE and OpenMP speeded up this part such that the wall-clock times are comparable for a range of particle numbers. After the recent hardware with AVX support became available, the library was updated. This led to additional speed-up of the irregular force calculation. It is also essential that the regular force part scales well on multiple GPUs. Thus in the future, further speed-up may be achieved by using four GPUs and an octo-core CPU. Although large N-body simulations are still quite expensive, we have demonstrated that the regular part of the Ahmad–Cohen neighbour scheme is well suited for use with GPU hardware. Moreover, the current formulation of the $NBODY6$-GPU code performs well for a variety of difficult conditions.

6 ACKNOWLEDGMENTS

Much of this work was done over the summers of 2007–2010 when K. N. visited the Institute of Astronomy, supported by the Visitor's Grant. We also acknowledge helpful discussions with members of the $NBODY6$ community. We thank Mark Gieles for making System B available.

[3] Because of cache miss, there is a degradation above $N = 64$ k for the irregular force calculation with AVX.

REFERENCES

Aarseth S. J., 1996, IAU Symp. 174, P. Hut & J. Makino, eds. (Kluwer, Dordrecht) 161
Aarseth S. J., 1999, PASP, 111, 1333
Aarseth S. J., 2003, Gravitational N-Body Simulations, Cambridge Univ. Press, Cambridge
Aarseth S. J., Zare K., 1974, Celes. Mech., 10, 185
Ahmad A., Cohen L., 1973, J. Comp. Phys. 12, 389
Bulirsch R., Stoer J., 1966, Num. Math., 8, 1
Fukushima, T., 2005, AJ, 129, 2496
Gaburov, E., Harfst, S., Portegies Zwart, S., 2009, New Astron., 14, 630
Heggie, D. C., 1974, Celes. Mech., 10, 217
Kustaanheimo P., Stiefel E., 1965, J. Reine Angew. Math. 218, 204
Makino J., 1991, ApJ, 369, 200
Makino J., Aarseth S. J., 1992, PASJ, 44, 141
Makino J., Taiji, M., 1997, ApJ, 480, 432
Makino J., Fukushige T., Koga M., Namura K., 2003, PASJ, 55, 1163
Mikkola S., 1985, MNRAS, 215, 171
Mikkola S., Aarseth S. J., 1993, Celes. Mech. Dyn. Ast., 57, 439
Mikkola S., Aarseth S. J., 1996, Celes. Mech. Dyn. Ast., 64, 197
Mikkola S., Aarseth S. J., 1998, New Astron., 3, 309
Mikkola S., Aarseth S. J., 2002, Celes. Mech. Dyn. Ast., 84, 343
Nitadori, K., 2009, New approaches to high-performance N-body simulations — high-order integrators, new parallel algorithms, and efficient use of SIMD hardware. Ph. D. thesis, University of Tokyo
Spurzem R., 1999, in Computational Astrophysics, JCAM, R. H. Werner ed (Elsevier) 407
Tanikawa, A., Yoshikawa, K., Okamoto, T., Nitadori, K., 2011, New Astron., in press

APPENDIX A: GLOSSARY

GCC GNU Compiler Collection, including `gcc`, `g++`, `gfortran` and other languages.

API Application Programing Interface. This provides definitions or prototypes of FORTRAN subroutines or C functions.

SIMD Single Instruction Multiple Data. A model of parallel computer where an operation such as addition and multiplication is performed for multiple data.

SSE Streaming SIMD Extensions. Additional instruction set for x86. Four words single precision floating point operations on 128-bit registers are supported.

AVX Advanced Vector eXtensions. Further enlargement of SSE. Eight words single precision or four words double precision floating point operations on 256-bit registers.

CUDA Compute Unified Device Architecture. A frame-work for general purpose computing on NVIDA GPUs, including language, compiler, run-time library and device driver.

OpenMP A standard to utilize multiple processors on shared memory from high-level languages, provided as directives of the language.

Thread A unit of parallel execution. Threads of CPU execute different contexts, while a cluster of threads of GPU executes the same context for different data.

Thread-Block A number of GPU threads which share the same context.

Kernel A short program submitted and executed on GPU.

***i*-particle** A particle which feels the gravitational force, named from the outer loop index.

***j*-particle** A particle which is a source of the gravitational force, named from the inner loop index.

***i*-parallelism** Parallelism for the outer loop. Forces on different particles are calculated in parallel.

***j*-parallelism** Parallelism for the inner loop. Forces from different particles are calculated in parallel, and summed later.

APPENDIX B: CUDA CODES

In Listing B1 and B2, we show the innermost regular force kernel written in CUDA C++ as well as some definitions of data structures. Each CUDA thread holds one i-particle in its registers, and evaluates a force from particle j and accumulates it. The functions qualified with `__device__` are forced to be in-line. The argument `nblist` is a pointer to the device memory.

Listing B1: Definitions of structures.

```
struct Jparticle{
    float3 pos;
    float  mass;
    float3 vel;
    float  pad; // 64-byte
};
struct Iparticle{
    float3 pos;
    float  h2;
    float3 vel;
    float  dtr; // 64-byte
};
struct Force{
    float3 acc;
    float  pot;
    float3 jrk;
    int    nnb; //  64-byte
};
```

Listing B2: The innermost gravity calulation.

```
// num of neib per block, must be power of 2
#define NNB_PER_BLOCK 256

__device__ void dev_gravity(
        const int        jidx,
        const Iparticle &ip,
        const Jparticle &jp,
        Force           &fo,
        uint16           nblist[])
{
    float dx  = jp.pos.x - ip.pos.x;
    float dy  = jp.pos.y - ip.pos.y;
    float dz  = jp.pos.z - ip.pos.z;
    float dvx = jp.vel.x - ip.vel.x;
    float dvy = jp.vel.y - ip.vel.y;
    float dvz = jp.vel.z - ip.vel.z;
    float dxp = dx + ip.dtr * dvx;
    float dyp = dy + ip.dtr * dvy;
    float dzp = dz + ip.dtr * dvz;

    float r2  = dx *dx  + dy *dy  + dz *dz;
    float r2p = dxp*dxp + dyp*dyp + dzp*dzp;
    float rv  = dx *dvx + dy *dvy + dz *dvz;

    float rinv1 = rsqrtf(r2);
    if(fminf(r2, r2p)  < ip.h2){
        // address to avoid buffer overflow
        int addr = bo.nnb & (NNB_PER_BLOCK -1);
        nblist[addr] = (uint16)jidx;
        fo.nnb++;
        rinv1 = 0.f;
    }
```

Table C1. List of symbols.

Symbol	Data type	Definition
LMAX	PARAMETER	Size of compiled neighbour arrays
NBMAX	PARAMETER	Maximum neighbour number
NIMAX	PARAMETER	Block-size of regular force loop
NPACT	PARAMETER	Limit for active particle prediction
NPMAX	PARAMETER	Parallel irregular integration
IFIRST	INTEGER	Array index of first single particle
NFR	INTEGER	Number of regular force calculations
NQ	INTEGER	Membership of LISTQ
NTOT	INTEGER	Number of single particles and KS pairs
LISTQ	INT (NMAX)	List of particles due before a given time
NXTLEN	INT (NMAX)	Length of current block-step list
NXTLST	INT (NMAX)	List of block-step members
RS	REAL(NMAX)	Radius of individual neighbour sphere
STEPR	REAL(NMAX)	Regular time-step
T0	REAL(NMAX)	Time of last irregular force calculation
TMIN	REAL(NMAX)	New block-time
TNEW	REAL(NMAX)	Next irregular force time
X	REAL(3,NMAX)	Predicted coordinates
X0	REAL(3,NMAX)	Corrected coordinates
XDOT	REAL(3,NMAX)	Predicted velocities
X0DOT	REAL(3,NMAX)	Corrected velocities

```
33      float rinv2 = rinv1 * rinv1;
34      float mrinv1 = jp.mass * rinv1;
35      float mrinv3 = mrinv1 * rinv2;
36      float alpha  = -3.f * rv * rinv2;
37
38  #ifdef POTENTIAL
39      fo.pot += mrinv1;
40  #endif
41      fo.acc.x += mrinv3 * dx;
42      fo.acc.y += mrinv3 * dy;
43      fo.acc.z += mrinv3 * dz;
44      fo.jrk.x += mrinv3 * (dvx + alpha * dx);
45      fo.jrk.y += mrinv3 * (dvy + alpha * dy);
46      fo.jrk.z += mrinv3 * (dvz + alpha * dz);
47  }
```

In lines 17–19, 22 and 26, we can see the additional operations for the velocity criterion (5 mul, 6 add, 1 min) discussed in Section 3.2.

APPENDIX C: FORTRAN CODES

It is instructive to examine the program flow for the treatment of the regular and irregular force during one block-step. We have copied the relevant FORTRAN parts, omitting some extra features, and display a complete cycle in the general case. Consider the situation when the next block-step time, denoted by TMIN, has been determined at the end of the previous step (as shown below). All particles due for consideration up to a certain time are contained in LISTQ, and the new block-step members NXTLEN which satisfy equation (7) are saved in the list array NXTLST after the first call to INEXT. Procedures for advancing KS and chain solutions which usually follow here are omitted for simplicity. The essential symbols are defined in Table C1, where some are fixed parameters. For completeness, we also include the OPEN and CLOSE statements, normally only used at the start and end. With this preamble and the definitions of Table C1 the code section can now be inspected with assistance from the explanatory comments. The purpose of most special procedures have already been discussed in the text and the naming is intended to be descriptive.

Listing C1: extract of `intgrt.omp.f`.

```
1   *       Declare typical parameters for N=64k.
2         PARAMETER(NPACT=150, NPMAX=16, NIMAX=1024)
3   *       Open the regular and irregular libraries.
4         CALL GPUNB_OPEN(NTOT)
5         CALL GPUIRR_OPEN(NTOT,LMAX)
6   *
7   *       Find all particles in next block (TNEW = TMIN) and set TIME.
8         CALL INEXT(NQ,LISTQ,TMIN,NXTLEN,NXTLST)
9         TIME = TMIN
10  *
11  *       Form lists of candidates for new irregular and regular force.
12        NFR = 0
13        DO 28 L = 1,NXTLEN
14            J = NXTLST(L)
15            IF (TNEW(J).GE.T0R(J) + STEPR(J)) THEN
16                NFR = NFR + 1
17                IREG(NFR) = J
18                IRR(L) = J
19            ELSE
20                IRR(L) = 0
21            END IF
22     28 CONTINUE
23  *
24  *       Decide between predicting <= NPACT active (NFR=0) or all particles.
25        IF (NXTLEN.LE.NPACT.AND.NFR.EQ.0) THEN
26            CALL GPUIRR_PRED_ACT(NXTLEN,NXTLST,TIME)
27        ELSE
28            CALL GPUIRR_PRED_ALL(IFIRST,NTOT,TIME)
29        END IF
30  *
31  *       Evaluate new irregular forces & derivatives in the GPUIRR library.
32        CALL GPUIRR_FIRR_VEC(NXTLEN,NXTLST,GF,GFD)
33  *
34  *       Choose between standard and parallel irregular integration.
35        IF (NXTLEN.LE.NPMAX) THEN
36  *
37  *       Correct the irregular steps sequentially.
38        DO 48 II = 1,NXTLEN
39            I = NXTLST(II)
40            CALL NBINT(I,IRR(II),GF(1,II),GFD(1,II))
41     48 CONTINUE
42  *
43        ELSE
44  *
45  *       Perform irregular correction in parallel.
46  !$omp parallel do private(II, I)
47        DO 50 II = 1,NXTLEN
48            I = NXTLST(II)
49            CALL NBINTP(I,IRR(II),GF(1,II),GFD(1,II))
50     50 CONTINUE
51  !$omp end parallel do
52        END IF
53  *
54  *       Check regular force updates (NFR members on block-step).
55        IF (NFR.GT.0) THEN
56  *
57  *       Predict all particles (except TPRED=TIME) in C++ on host.
58        CALL CXVPRED(IFIRST,NTOT,TIME,T0,X0,X0DOT,F,FDOT,X,XDOT,TPRED)
59  *       Send all single particles and c.m. bodies to the GPU.
60        NN = NTOT - IFIRST + 1
61        CALL GPUNB_SEND(NN,BODY(IFIRST),X(1,IFIRST),XDOT(1,IFIRST))
62  *
63  *       Perform regular force loop (blocks of NIMAX=1024).
64        JNEXT = 0
65        DO 55 II = 1,NFR,NIMAX
66            NI = MIN(NFR-JNEXT,NIMAX)
67  *       Copy neighbour radius, STEPR and state vector for each block.
68  !$omp parallel do private(LL, I, K)
69            DO 52 LL = 1,NI
70                I = IREG(JNEXT+LL)
71                H2I(LL) = RS(I)**2
72                DTR(LL) = STEPR(I)
73                DO 51 K = 1,3
74                    XI(K,LL) = X(K,I)
75                    VI(K,LL) = XDOT(K,I)
76     51         CONTINUE
77     52     CONTINUE
78  !$omp end parallel do
79  *
80  *       Evaluate forces, derivatives and neighbour lists for new block.
81          CALL GPUNB_REGF(NI,H2I,DTR,XI,VI,GPUACC,GPUJRK,GPUPHI,LMAX,
82       &                                                  NBMAX,LISTGP)
83  *       Copy neighbour lists from the GPU.
84  !$omp parallel do private(LL, I, ITEMP, NNB, L1, L)
85            DO 56 LL = 1,NI
86                I = IREG(JNEXT + LL)
87                NNB = LISTGP(1,LL)
88                L1 = 1
89                DO 53 L = 2,NNB+1
90  *       Note GPU address starts from 0 (hence add IFIRST to neighbour list).
91                    ITEMP = LISTGP(L,LL) + IFIRST
92                    IF (ITEMP.NE.I) THEN
93                        L1 = L1 + 1
94                        LISTGP(L1,LL) = ITEMP
95                    END IF
96     53         CONTINUE
97                LISTGP(1,LL) = L1 - 1
98                CALL GPUIRR_SET_LIST(I,LISTGP(1,LL))
99     56     CONTINUE
100 !$omp end parallel do
101 *
102 *       Evaluate current irregular forces by vector procedure.
103         CALL GPUIRR_FIRR_VEC(NI,IREG(II),GF(1,1),GFD(1,1))
104 !$omp parallel do private(LL, I, LX)
105           DO 57 LL = 1,NI
106               I = IREG(JNEXT+LL)
```

```
*       Send new irregular force and perform Hermite corrector.
              CALL GPUCOR(I,XI(1,LL),VI(1,LL),GPUACC(1,LL),GPUJRK(1,LL),
     &                                    GF(1,LL),GFD(1,LL),LISTGP(1,LL),LX)
*       Update neighbour lists in GPUIRR library (only if changed: LX > 0).
              IF (LX.GT.0) THEN
                  CALL GPUIRR_SET_LIST(I,LIST(1,I))
              END IF
   57     CONTINUE
!$omp end parallel do
          JNEXT = JNEXT + NI
   55 CONTINUE
      END IF
*
*       Determine next block time (note STEP may shrink in GPUCOR).
      TMIN = 1.0D+10
      DO 60 L = 1,NXTLEN
          I = NXTLST(L)
          IF (TNEW(I).LT.TMIN) THEN
              TMIN = TNEW(I)
          END IF
   60 CONTINUE
*
*       Copy current coordinates & velocities from corrected values.
!$omp parallel do private(I, L, K)
      DO 70 L = 1,NXTLEN
          I = NXTLST(L)
          DO 65 K = 1,3
              X(K,I) = X0(K,I)
              XDOT(K,I) = X0DOT(K,I)
   65     CONTINUE
*       Send corrected active particles to GPUIRR library.
          CALL GPUIRR_SET_JP(I,X0(1,I),X0DOT(1,I),F(1,I),FDOT(1,I),
     &                                              BODY(I),T0(I))
   70 CONTINUE
!$omp end parallel do
      CALL GPUNB_CLOSE
      CALL GPUIRR_CLOSE
```