SUPERMASSIVE BLACK HOLE BINARY COALESCENCE AND ORBITAL STRUCTURE OF THE HOST GALAXY

By

Baile Li

Dissertation
Submitted to the Faculty of the
Graduate School of Vanderbilt University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY
in

PHYSICS

May, 2015

Nashville, Tennessee

DEDICATION

Figure 1: My parents and me in Beijing Airport on Aug. 14, 2013.

I would like to dedicate this thesis to my parents Xianhou Li and Guiying Wang for their love, encouragement, unconditional support and belief in me.

ACKNOWLEDGMENTS

First and foremost, I am most thankful for the supervision of Dr. Kelly Holley-Bockelmann. Without her enthusiastic encouragement and thoughtful guidance this thesis would not be possible. Every week in my meeting with Kelly, her ideas always enlightened me, from the big picture to detailed technical questions. She was so considerate and always there when I needed help. She really helped me a lot during every single step of my career as a PhD student: helped me with my AJC talks, my poster, qualifier exam, courses and finding a job... I cannot count them all. I am so lucky to have Kelly as my supervisor. What I learned from her is not just the astronomy knowledge and research, but how to understand people and help people.

I would like to thank other members of my PhD committee: Dr. Andreas Berlind, Dr. David Weintraub, Dr. Thomas Weiler and Dr. Sait Umar. I would like to thank them for their help in many ways including the very clear lectures they gave, their patience of answering my questions and the recommendation letters they wrote for me. They have all played critical roles in my development as a scientist.

I am also extremely grateful for my collaborators: Dr. Rainer Spurzem, Dr. Yohai Meiron and Dr. Fazeel Khan. We had a very happy time working together. Especially when I visited NAOC in 2011 and 2013, Dr. Rainer Spurzem gave me very considerate accommodation and very helpful suggestion on my projects.

I would also like to acknowledge my fellow graduate students at Vanderbilt University. They really cared about me. Any time when I met difficulties, they offered prompt help without any hesitation. I am so proud to be a member of this community.

Finally, I would like to thank my father Xianhou Li and my mother Guiying Wang. I have been encouraged to be a scientist since my childhood. Now I am on the way to it. It is my parents who let me have a dream and work towards it. They made me who I am today. I thank them for their unconditional love, support and belief in me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

Introduction

I.1 Introduction

Supermassive black holes (SMBHs) are thought to be a key component of nearly every galaxy. These black holes, with masses 10^6 to 10^{10} $M_\odot$, dwell in the galactic center where they may power jets (Genzel et al., 2003; Falcke et al., 2004; Allen et al., 2006; Doeleman et al., 2012), regulate the energy in the interstellar medium (Cattaneo et al., 2009), and even transform the shape of the host galaxy (Gerhard & Binney, 1985; Norman, May, & van Albada, 1985; Merritt & Quinlan, 1998; Wachlin & Ferraz-Mello, 1998; Valluri & Merritt, 1998; Holley-Bockelmann et al., 2002). Despite playing such an integral role in galaxy evolution, how SMBHs form and evolve is a mystery. In general, most theoretical work on SMBH growth deals with how they evolve in mass from seed black holes in protogalaxies to supermassive ones today (Volonteri, Haardt, & Madau, 2003; Shapiro, 2005; King & Pringle, 2006; Lodato & Natarajan, 2006; Pelupessy et al., 2007). From these calculations, it is learned that SMBHs may grow rapidly during galaxy mergers; this process causes the black holes in each galaxy to sink to the center of the newly merged galaxy and coalesce. The merger process is violent and drives copious amounts of gas to the new galactic center, which provides new fuel to the black hole (Hernquist, 1989; Barnes & Hernquist, 1991; Mihos & Hernquist, 1994; Hernquist & Mihos, 1995; Barnes & Hernquist, 1996; Mihos & Hernquist, 1996; Hopkins et al., 2005). Through a combination of merger-driven gas accretion and direct coalescence it is possible that the remnants of the first stars may grow into the SMBHs of the current epoch (Volonteri, Haardt, & Madau, 2003; Volonteri et al., 2005; Volonteri & Rees, 2005).

However, how the black hole binary interacts with the host galaxy during a merger still needs to be investigated, such as the binary hardening rate, the merger time of the binary in different shapes of galaxies and the effect of the binary on the orbital structure of the host galaxy.

In Chapter 1, we present a brief overview on the background of how SMBHs form and why they are important to their host galaxies. Section I.2 gives the observational background of SMBHs. Section I.3 covers the theoretical background, including dark matter halo density profiles, galaxy mergers, seed black hole formation, gravitational recoil, Post-Newtonian dynamics, black hole growth, the final parsec problem and orbits in galaxy potential. In Chapter 2, we analyze the stellar orbits in a middle flattened axisymmetric galaxy with a SMBH embedded in to see if there are enough stars with very small pericenters that can potentially interact with the black hole. In Chapter 3, we add another equal mass black hole to the same axisymmetric galaxy studied in Chapter 2 and check the energy change of stars of particular interest to see if they actually interact with the black hole binary. In Chapter 4 we present a project that transforms the CPU version of self-consistent field (SCF) code that I have been using as a basis to run particles in Chapter 2 to a GPU version and compare the efficiency of them. Appendix is the SCF code with up to 3.5 post-Newtonian terms added in.

I.2 Observational background

I.2.1 Mass of the black hole

I.2.1.1 Measuring the mass of black hole

Since black holes are not able to be detected directly by their very nature, we can only detect them indirectly through the gravitational interaction with the surrounding matter such as gas and dust or with other objects passing by such as stars and other black holes. According to the "no hair" theorem there are only three quantities to distinguish two black holes: mass, spin and electric charge. Mass is the only quantity that all black holes have, so measuring the mass of the SMBH is one of our main goals in this field. In the following paragraphs we show four methods to achieve this goal.

One way of measuring the mass of SMBH is applying stellar dynamics (see Kormendy & Richstone, 1995, and references therein). The mass closed within radius r can be calculated using the first velocity moment of collisionless Boltzmann equation,

$$M(r) = \frac{V^2 r}{G} + \frac{{\sigma_r}^2 r}{G}\left[-\frac{d\ln\nu}{d\ln\mathrm{r}} - \frac{d\ln{\sigma_\mathrm{r}}^2}{d\ln\mathrm{r}} - \left(1 - \frac{{\sigma_\theta}^2}{{\sigma_r}^2}\right) - \left(1 - \frac{{\sigma_\phi}^2}{{\sigma_r}^2}\right)\right] \tag{I.1}$$

where G is the gravitational constant, V is the rotation velocity, σ_r, σ_θ, σ_ϕ are three component of velocity dispersion σ, ν is the density of stars being measured. But usually in practice, the volume brightness is measured to get the mass per volume, i.e., density ν, assuming M/L is constant, independent of radius. All quantities in the above equation are required to be unprojected. However, the quantities that can be measured are only projected, so it is needed to transfer the projected data to unprojected data. In light of the local brightness, V and σ are estimated in order to make a galaxy model, then the data can be "observed" from the model. One can compare the model data with the real data, then adjust the model parameters to make the model data closer to the real one, and iterate this operation until both match up. Once all unprojected quantities are obtained, one can then use this equation to obtain the mass of the black hole.

A second way is to apply gas dynamics, which is relatively straightforward. One just need to measure the velocity of the gas in the accretion disk around the black hole, then apply the Keplerian orbit formula to get the mass of it.

Another way of measuring the mass of SMBHs is to use the maser radiation emitted by water molecules in the accretion flow around the black hole, e.g. for NGC 4258 (Miyoshi et al., 1995; Herrnstein et al., 1999), NGC 1194, NGC 2273, NGC 2960, NGC 4388, NGC 6264 and NGC 6323 (Kuo et al., 2011), etc. Masers are like lasers but in radio band. The Very Long Baseline Array, operated by the National Radio Astronomy Observatory, can be used to image masers with a resolution of 10 microarcseconds, which is 1000 times better than that of the Hubble Space Telescope. The velocity of the gas can be obtained by tracing the gas with masers. Therefore, the mass of the central black hole can be obtained by simply substituting the velocity of the gas into the Keplerian formula when the gas is in a disk around the black hole. Since the distances of the masers to the black hole, to a large extent, are less than 1 light-year, the mass measurement precision can be better than 10%—much better than that of optical band measurement.

Finally, when Active Galactic Nuclei (AGN) and quasars are so bright that the light from the gas/stars around them cannot be easily observed, another important method called reverberation mapping (Blandford & McKee, 1982) works, which measures the structure of the broad line region (see Figure I.1) around a supermassive black hole to calculate the mass of the black hole.

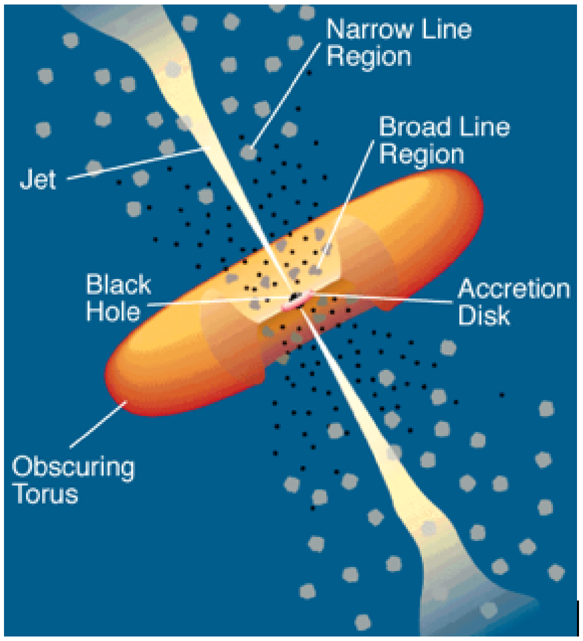


Figure I.1: The Active Galactic Nuclei unified model. Diagram from Figure 1 of Urry & Padovani (1995).

To understand how reverberation mapping works, let's first review the structure of an AGN (see Figure I.1). The black hole is surrounded by an accretion disk, outside of which is the broad line region, and even further away is the narrow line region.The spectrum from AGN/quasar consists of a strong continuum and broad lines. The continuum spectrum is emitted directly by the accretion disk around the black hole, while the broad emission lines are due to the farther gas absorbing the energy of the continuum emission. Since the farther gas is still near to the black hole, when moving around the black hole, it shows the Doppler effect as well, which is why the lines are broad. As the continuum spectrum varies, the broad lines vary accordingly, but with a time delay $T = 2R_{BLR}/C$ (see Figure I.2). The mass of the black hole can be expressed as

$$M_{BH} \approx f R_{BLR} \sigma^2 / G \tag{I.2}$$

where σ is line dispersion and f is a unity order factor depending on the inclination, geometry and kinematics of the broad line region. The narrow line is caused by the emission of the heated gas cloud, since the cloud is clumpy not continuous, the emission line is narrow. With this method the mass of black holes in more than 30 AGN/QSOs are measured (Peterson et al., 2004).

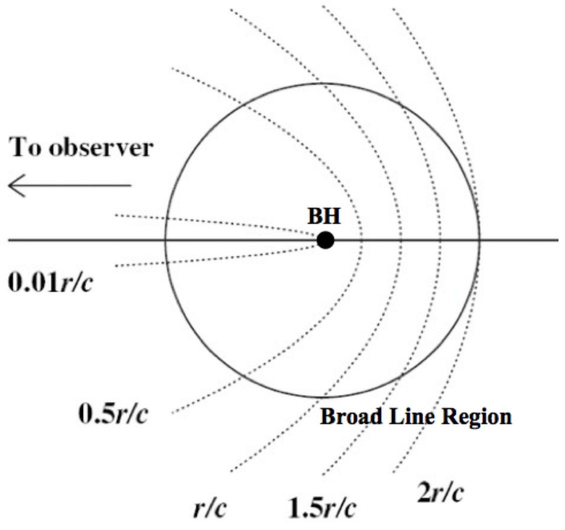


Figure I.2: Time delay of reverberation mapping. Figure adapted from Figure 2 of Peterson (2006).

I.2.1.2 $M-\sigma$ and $M-L$ correlations

Given the small mass fraction of the SMBHs compared to their galaxy hosts, we naturally do not expect any correlations between it and the quantities reflecting the characteristics of the host galaxy such as the velocity dispersion σ and luminosity of the bulge L. A series of studies shows that such correlations do exist (Gültekin et al., 2009), among which the M-σ correlation (see Figure I.3) has the smallest intrinsic scatter, smaller than $M-L$ correlation (see Figure I.4). The two relations are also important, because we can achieve the black hole mass through them. Gültekin et al. (2009) adopt 49 black hole masses and 19 mass upper limits of elliptical galaxies and spiral galaxies with bulge, and use $\sigma_e^2 \equiv \frac{\int_0^{R_e}(\sigma^2+V^2)I(r)dr}{\int_0^{R_e}I(r)dr}$, where V is the rotational component of the spheroid, or σ_c, i.e., σ at the center of bulge as σ in the $M-\sigma$ relation, and also apply

$log(L_v/L_{\odot,v}) = 0.4\left(4.83 - M^0_{v,bulge}\right)$ to calculate L_v. They found the $M-\sigma$ relation

$$\log\left(\frac{\mathrm{M_{BH}}}{\mathrm{M_\odot}}\right) = (8.12\pm0.08) + (4.24\pm0.41)\times\log\left(\frac{\sigma_\mathrm{e}}{200\mathrm{kms}^{-1}}\right) \tag{I.3}$$

with an intrinsic rms Gaussian scatter of $\varepsilon_0 = 0.44\pm0.06$, and $M-L$ relation

$$\log\left(\frac{\mathrm{M_{BH}}}{\mathrm{M_\odot}}\right) = (8.95\pm0.11) + (1.11\pm0.18)\times\log\left(\frac{\mathrm{L_v}}{10^{11}\mathrm{L_{\odot,v}}}\right) \tag{I.4}$$

with $\varepsilon_0 = 0.38\pm0.09$.

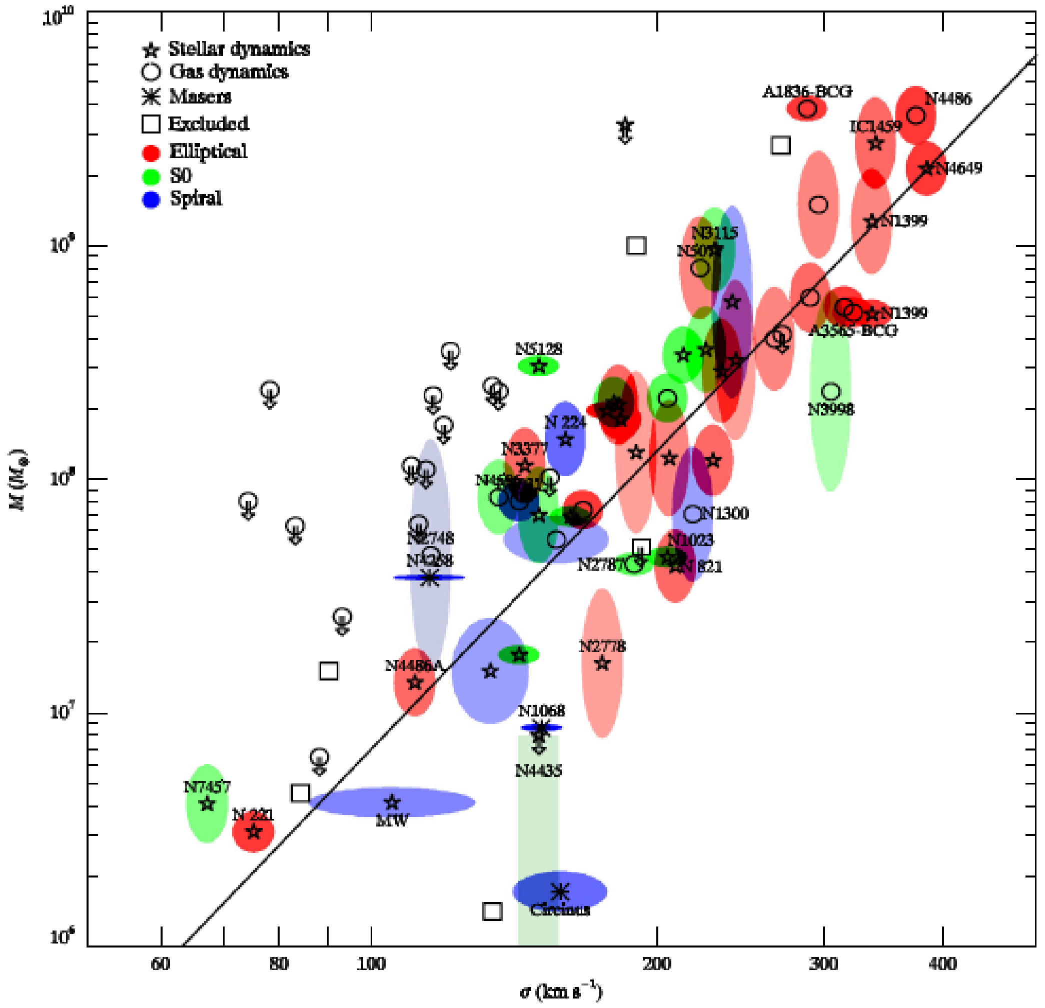


Figure I.3: M-σ correlation for galaxies with dynamical measurements. The figure shows the mass of black hole as a function of velocity dispersion of the bulge of the host galaxy. The measuring methods are stellar dynamics (pentagrams), gas kinematics (circles) and masers (asterisks). The data excluded from fit are marked by squares. The data are colored by galaxy types, elliptical (red), S0 (green) and spiral (blue). The solid line shows the best fit relation $M_{BH} = 10^{8.12}M_\odot(\sigma/200kms^{-1})^{4.24}$. Figure adapted from Figure 1 of Gültekin et al. (2009).

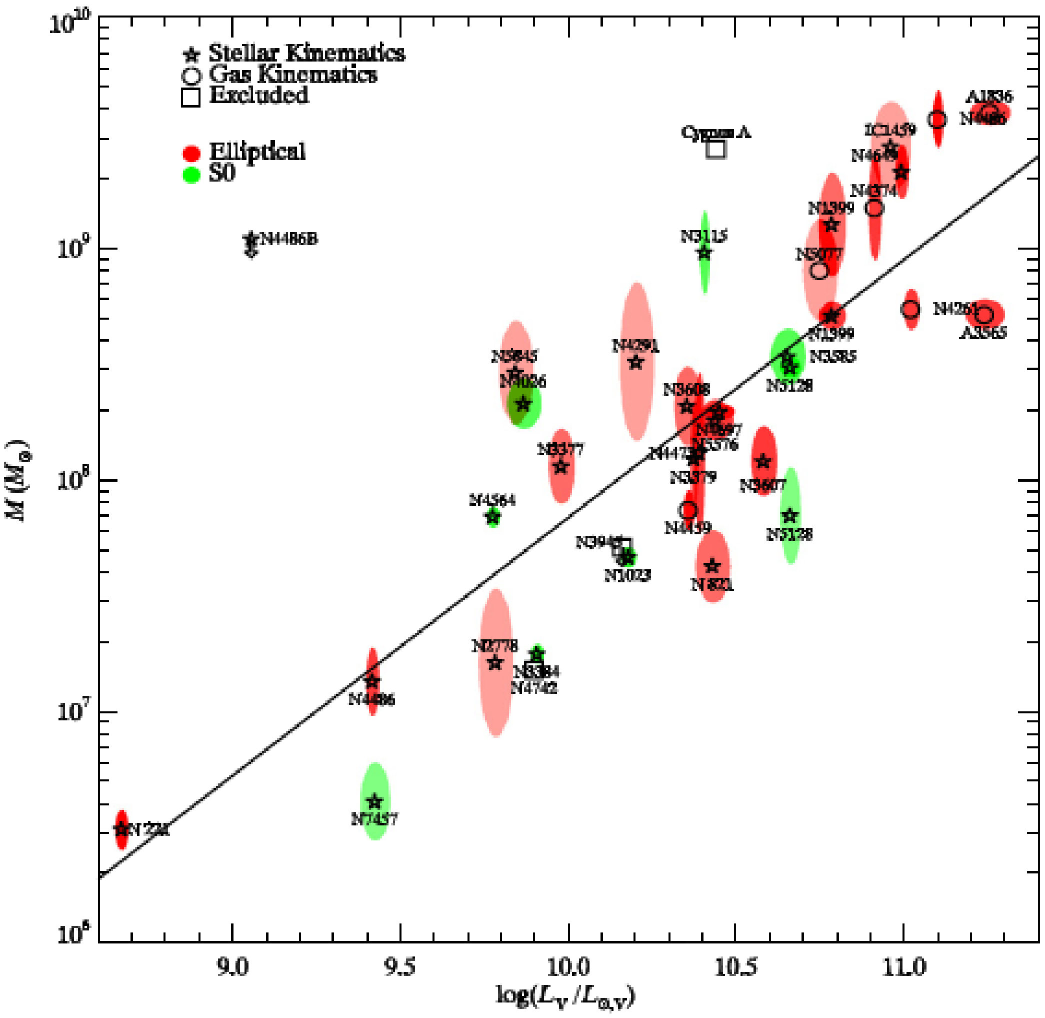


Figure I.4: $M-L$ correlation for galaxies with dynamical measurements. The figure shows the mass of black hole as a function of V-band luminosity of the bulge of the host galaxy. The measuring methods are stellar dynamics (pentagrams) and gas kinematics (circles). The data excluded from fit are marked by squares. The data are colored by galaxy types, elliptical (red) and S0 (green).The solid line shows the best fit relation $M_{BH} = 10^{8.95} M_{\odot} (L_V/10^{11} L_{\odot,V})^{1.11}$. Figure adapted from Figure 4 of Gültekin et al. (2009).

I.2.1.3 SMBH demographics

Once we have a census of accurate SMBH masses, it is useful to understand the demographics of this population—the SMBH mass function and how the mass function behaves in different galaxy hosts at different redshifts. To address this SMBHs are divided into three categories: high redshift quasi-stellar objects (QSOs), local AGNs and the ones in local quiescent galaxies according to their distance to us and their activity stage. (see Figure I.5 and Table I.1, both from Ferrarese (2002)).

As early as 1969, Donald Lynden-Bell made an assumption that QSOs are caused by accretion onto SMBHs. Soltan (1982) gives a quantitative correlation between the QSO 's luminosity and the accretion rate of the black hole (see Equation I.5), in which ε is the efficiency factor which gives the fraction of accretion mass transferring to light, $\dot{M}$ is the accretion rate of the black hole and c is the speed of light. Combining this correlation with the observation of QSOs 's number counts, luminosities and distances, he calculated the black hole mass density as 10^{14} solar masses per cubic Gigaparsec. Then assuming each black hole 's mass is $10^8 \sim 10^9$ solar masses, he derived that there are $10^5 \sim 10^6$ black holes per cubic Gigaparsec. This is called the Soltan argument.

$$L = \varepsilon \dot{M} \mathrm{c}^2 \tag{I.5}$$

For high redshift QSOs, the Soltan argument (Soltan, 1982) concludes that according to the number density of QSOs, the black hole mass density can be calculated(Ferrarese, 2002).

$$\rho_{QSO}(>M) = \frac{K_{bol}}{\varepsilon c^2} \int_0^\infty \int_L^\infty \frac{L'\Phi(L',z)}{H_0(1+z)\sqrt{\Omega_m(1+z)^3+\Omega_\Lambda}} dL' dz \tag{I.6}$$

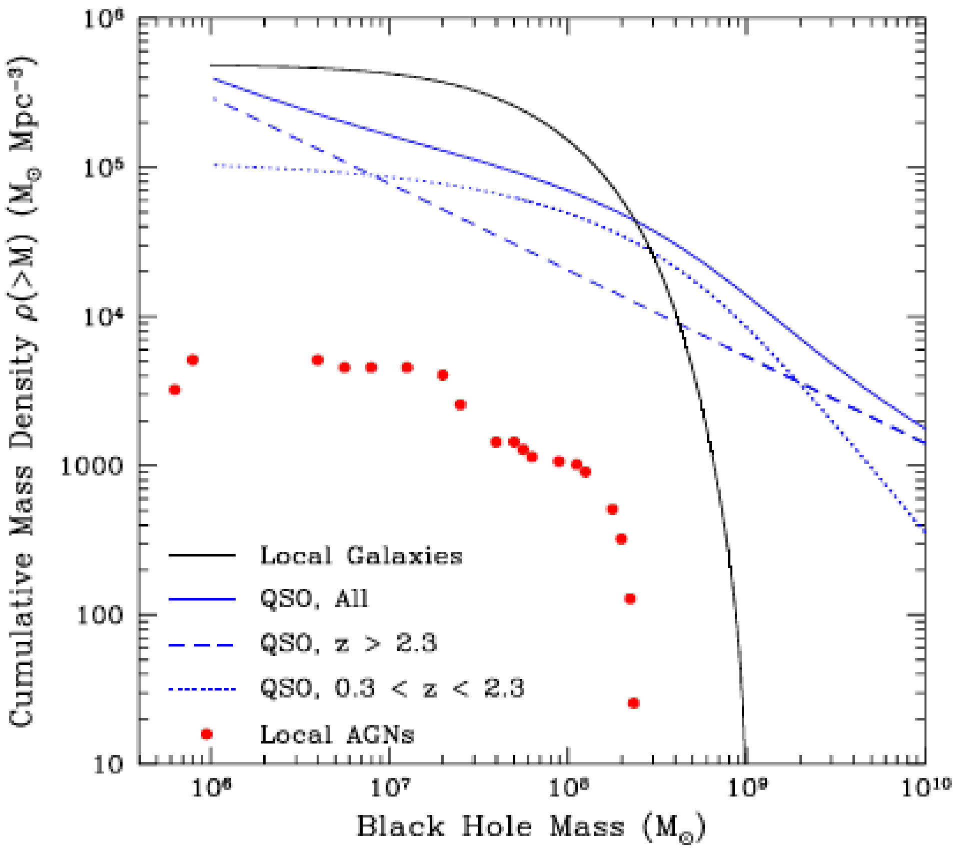


Figure I.5: Comparison of the cumulative mass density of black holes in galaxies with different red-shift and activity stage: local quiescent galaxies (black), QSOs (blue), and local AGNs (red). Figure adapted from Figure 1 of Ferrarese (2002).

For local AGNs, reverberation mapping is adopted to achieve the mass of the black holes. For local quiescent galaxies, $M-\sigma$ and $M-L$ correlations are used to get the black hole masses (see Figure I.6).

Table I.1: Summary of mass densities in supermassive black holes. Table adapted from Ferrarese (2002).

Method	$\rho_\bullet (10^5 M_\odot Mpc^{-3})$
QSO optical counts, $0.3 < z < 5.0$	2-4
AGN X-ray counts, $z > 0.3$	0.6-9
Special fit to the X-ray background, z unknown	2-30
Local AGNs, $z < 0.1$	0.05-0.6
Local Quiescent Galaxies, $z < 0.0003$	4-5

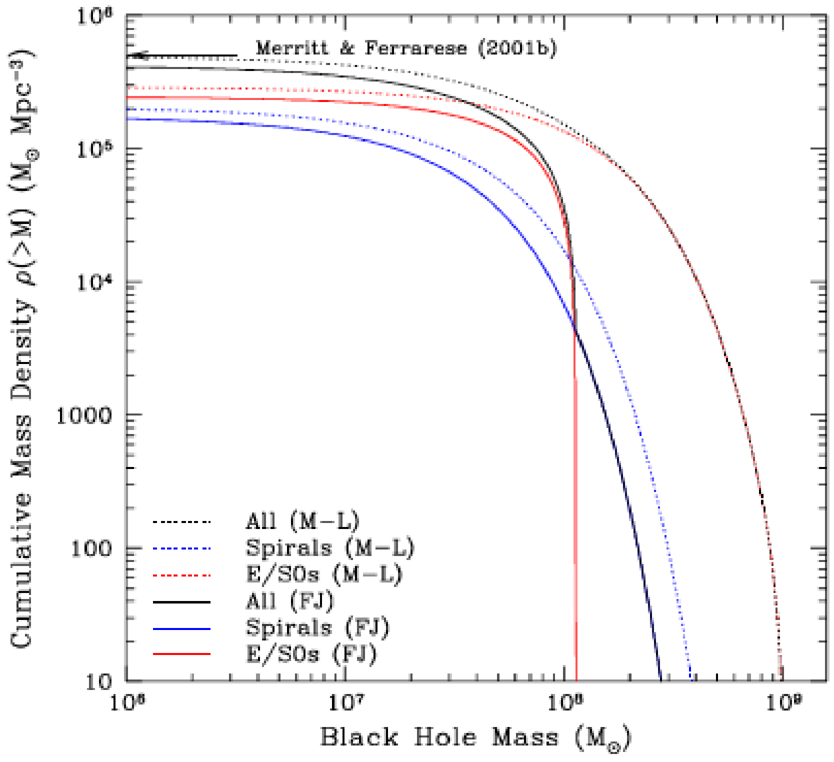


Figure I.6: Cumulative mass density for local black holes."M-L" indicates the mass of black hole is obtained from $M_\bullet - L$ relation, while "F-J" indicates the mass of black hole obtained from $M_\bullet - \sigma$ combined with the Faber-Jackson relation. Figure adapted from Figure 4 of Ferrarese (2002).

Natarajan & Treister (2009) argued that there is a black hole mass upper limit at roughly $10^{10} M_\odot$ due to the extension of the $M - L$ relation at the high luminosity end.

I.2.2 Spin of black holes

As we stated before, spin, as one of three intrinsic parameters of a black hole, plays a significant role in the interaction between a black hole and the surrounding gas or stars or other black holes. However, to date there is no direct evidence showing the existence of black hole spin, so the models of rotating black holes are compared with observations. In a recent review Gammie, Shapiro, & McKinney (2004) note that while there are several observations that are consistent with theoretical predictions of SMBH spin, no observation is ironclad. For example the fact that Fe K_α broad lines have red wings to some SMBHs, eg., MCG -6-30-15, Cyg X-1 and XTE J1650-500, requires the black hole models with spin to interpret it. Quasi-periodic oscillations at a frequency up to 450 Hz in GRO J1655-40 (Strohmayer, 2001) implies $j > 0.15$, where $j = \frac{\mathrm{L_{BH}}}{GM/c^2}$, where L_{BH} is the angular momentum of the black hole. Zhang, Cui, & Chen (1997) compared the observational features of flux from X-ray binaries GRO J1655-40 and GRS 1915+105 with the theoretical prediction of a thin disk around a spinning black hole, showing it is very possible that there is a spinning black hole in these X-ray binaries. Generally speaking, it is predicted that a black hole will increase its spin when the spin is aligned with the gas disk around it, and decrease its spin when misaligned. We will talk about how the spin changes during black hole mergers in section I.3.6.3.

I.3 Theoretical background

I.3.1 Dark matter halo

Flat galaxy rotation curves (Rubin & Ford, 1970), such as in Figure I.7 indicate that there is an additional dark matter component to the galaxy.

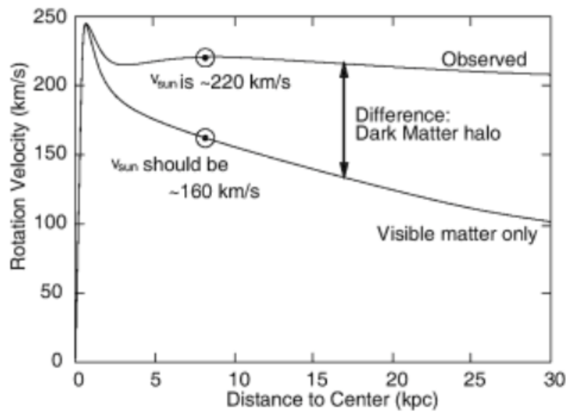


Figure I.7: Galaxy rotation curve for the Milky Way. Horizontal axis is distance from the galactic center. Vertical axis is speed of rotation about the galactic center. The sun is marked with $\odot$. The top curve is the observed curve of speed of rotation. The bottom is the predicted curve based upon stellar mass and gas in the Milky Way. The difference is due to existence of dark matter or perhaps a modification of the law of gravity. Figure adapted from Figure 1.4 of Schneider (2006).

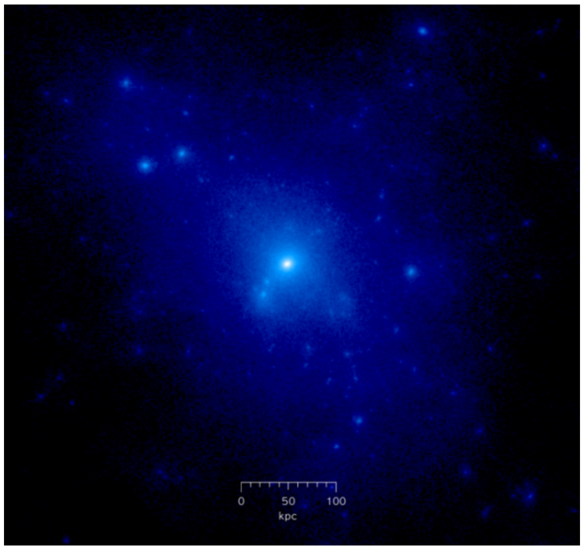


Figure I.8: Simulated dark matter halo from a cosmological N-body simulation. The figure is colored by the projected density of the dark matter. The unit of length is kpc. There is a central halo and several satellite subhalos, each with a galaxy inside.

Dark matter halos (see Figure I.8) typically follow a Navarro-Frenk-White profile (Navarro, Frenk, & White, 1996)

$$\rho(r) = \frac{\rho_{crit}\delta_c}{\frac{r}{r_s}\left(1+\frac{r}{r_s}\right)^2} \tag{I.7}$$

where ρ_{crit} is the critical density, equal to $\frac{3H^2}{8\pi G}$ (H is the Hubble constant), $r_s = r_{200}/C$ (r_{200} is the radius at which the average density of the halo reaches $200\rho_{crit}$, and C is called concentration), δ_c and C are dimensionless parameters, both of which vary from halo to halo. Given this density profile, the mass profile is

$$M = \int_0^{r_{200}} 4\pi r^2 \rho(r) dr = 4\pi\rho_0 r_s^3 \left[\ln(1+\mathrm{C}) - \frac{\mathrm{C}}{1+\mathrm{C}}\right] \tag{I.8}$$

for the halo of Milky Way, where C may take from 10 to 15, for other halos, C may range from 4 to 40 depending on the size of the halo. The mass of the dark matter halo in the Milky Way is believed to be $\sim 10^{12}$ $M\odot$ (Xue et al., 2008). It is believed that 23% of the universe is made of dark matter according to the height ratios of different peaks in Cosmic Microwave Background (CMB) power spectrum (Spergel et al., 2003).

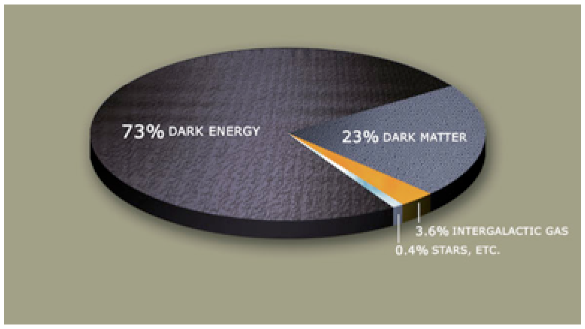


Figure I.9: The components of the universe

I.3.2 Galaxy merger

When two galaxies interact with each other, either by fly-by or merger, the stars in them do not actually collide with each other since stars are so far away from each other and so small. But the gas clouds sometimes in the merger case will collide with each other and fall into the center of the remnant galaxy, which will end up with a fast star formation at the center of the remnant galaxy. At the same time, the orbits of stars will be disturbed and completely change. When two spiral galaxies merge, the remnant galaxy is an elliptical one. Since the fast giving birth of stars during a merger consumes most gas, the remnant elliptical galaxy does not generate stars easily. Also when two galaxies merge, the black holes at the center of each galaxy will also merge, the falling gas will also be accreted by the remnant black hole, which often forms an AGN. Therefore, galaxy interactions are considered to be responsible for many phenomena including star formation, AGN/quasars and galactic morphology(Barnes & Hernquist, 1992)—merging spirals may give rise to ellipticals. Some observational evidence of the signature of the interaction are bridges and tails (see Figure I.10), polar rings, shells, nuclear disks and kinematically decoupled cores.

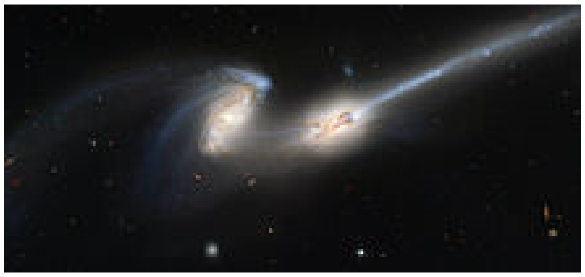


Figure I.10: The Mice Galaxies (NGC 4676 A & B) in the process of merging. The galaxies were photographed in 2002 by the Hubble Space Telescope

I.3.2.1 Final parsec problem

When two galaxies are merging, the black holes at the center of each galaxy will also coalesce. At the first stage the black holes just get closer and closer simply due to the gravity force between them until they form a hard binary at the distance $a_{hard} = G\mu/4\sigma^2$, where $\mu = M_1M_2/(M_1+M_2)$, M_1 and M_2 are the mass of the two black holes respectively, σ is the central velocity dispersion of the new formed galaxy. If taking $M_1 = M_2 = 2\times10^7 M_\odot$, a_{hard} is roughly 1 pc. “Hard” means a star being scattered super-elastically when passing by the binary makes the binary increase its binding energy shrinking its semi-axis. The stars which can be used are in a phase space called the loss cone. The geometry of the loss cone is depending on the morphology of the central part of the combined galaxy. By scattering more and more stars the binary finally gets close as to

$$a_{gr} = \left\{ \frac{64}{5} \frac{G^3 M_1 M_2 (M_1+M_2) F(e)}{c^5} t_{gr} \right\}^{1/4}, \tag{I.9}$$

less than which the binary can eventually merge within t_{gr} by emitting gravitational waves. Here c is the speed of light, F(e) is a factor depending on the eccentricity of the binary's orbit, equal to unity in circular case. The binary must shrink

$$\frac{a_{hard}}{a_{gr}} \approx 34 \times e^{9.2/\alpha} \left(\frac{M_1+M_2}{10^6 M_\odot} \right)^{1/4-2/\alpha} p^{3/4}(1+p)^{-3/2} \left(\frac{t_{gr}}{10^9 yrs} \right)^{-1/4} \tag{I.10}$$

in the scattering process, where $p = M_2/M_1 \leq 1$ and $\alpha = \frac{dlogM_{bh}}{dlog\sigma} \approx 4-5$. t_{gr} could not be longer than Hubble time, since indirect evidence shows that there are rare black hole binaries, so the binary must shrink by a factor of 100, which requires scattering the stellar mass of 10-20μ. For spherical galaxies, once deleted by the binary, the loss cone cannot be replenished fast enough, which is called the final parsec problem (Milosavljević & Merritt , 2003). Figure I.11 adapted from Berczik et al. (2006) shows that for a triaxial galaxy, 1/a, where a is the semi-major axis of the binary, goes up as time is increasing and is nearly independent on the total number of stars in the system, which implies that the result is converging and the real underlying physics is tracked. While for a spherical galaxy, the binary becomes further as the number of particles is increasing, not convergent at all. For asymmetrical galaxies, the problem is still a debate. Khan et al. (2013) says there are enough particles in the loss cone in a flattened galaxy with axes ratio $c/a = 0.75$ for the SMBH binary to fully merge, while Vasiliev, Antonini, & Merritt (2014) says the loss cone filling highly depends on N, the number of particles, when N is between 10^5 and 10^6. Therefore we plan to classify stellar orbits in an axisymmetric galaxy (the same one used in Khan et al. (2013)) with a SMBH embedded at the center to determine what orbits can potentially interact with a SMBH binary and how full the loss cone is.

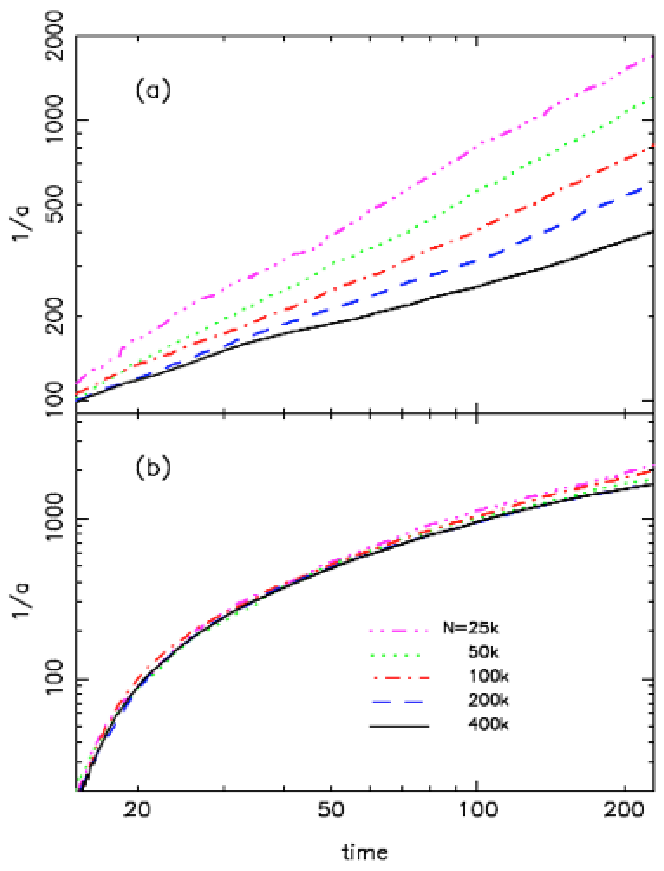


Figure I.11: The evolution of semi-major axis of black hole binary in spherical galaxy model (upper panel) and triaxial galaxy model (lower panel) with different particle number N. Figure adapted from Berczik et al. (2006).

I.3.3 Orbit types

Stellar orbits can be defined by the frequency ratios obtained by Fourier transforming the trajectory in each physical dimension (x(t), y(t), z(t)), which essentially tracks the star as it passes through each principle plane of the galaxy, to frequency domain. If the frequency ratio is stable, i.e. does not change with time, then the stellar orbit is called regular orbit. If the ratio is irreducible, the orbit is a resonant orbit (see Figure I.12, I.13, I.14, and I.15, which are some orbit samples from the simulation described in Chapter II), otherwise is a subfamily. For instance, 10:5 is a subfamily of 2:1 (see Figure I.16). If the frequency ratio is not stable, i.e. sensitive to initial conditions, then the orbit is chaotic (see Figure I.17).

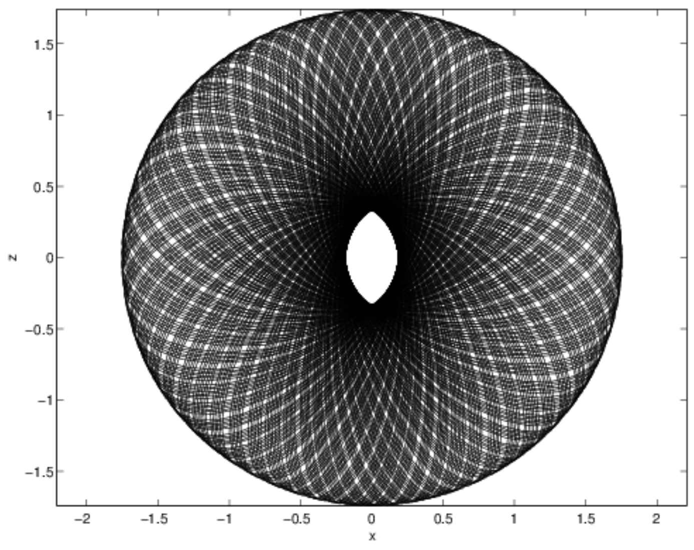


Figure I.12: 2-dimensional resonant orbit with a 1:1 frequency ratio

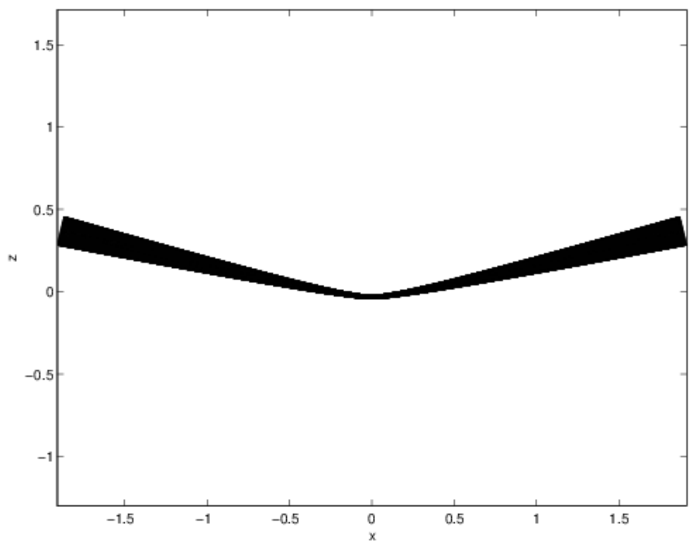


Figure I.13: 2-dimensional resonant orbit with a 2:1 frequency ratio

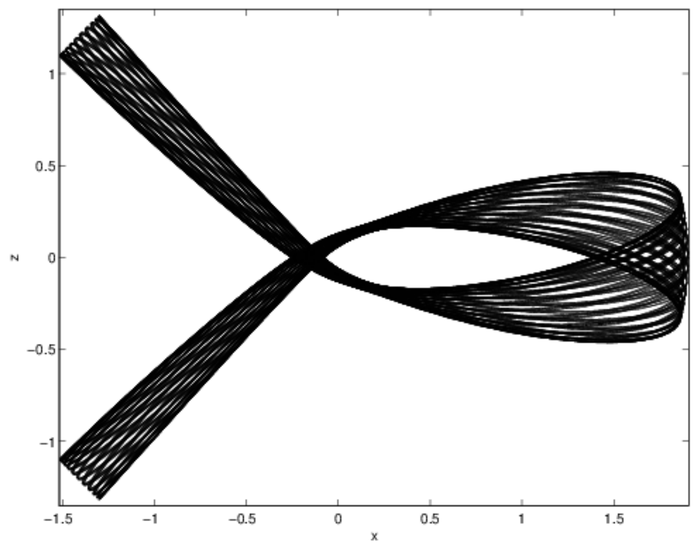


Figure I.14: 2-dimensional resonant orbit with a 3:2 frequency ratio

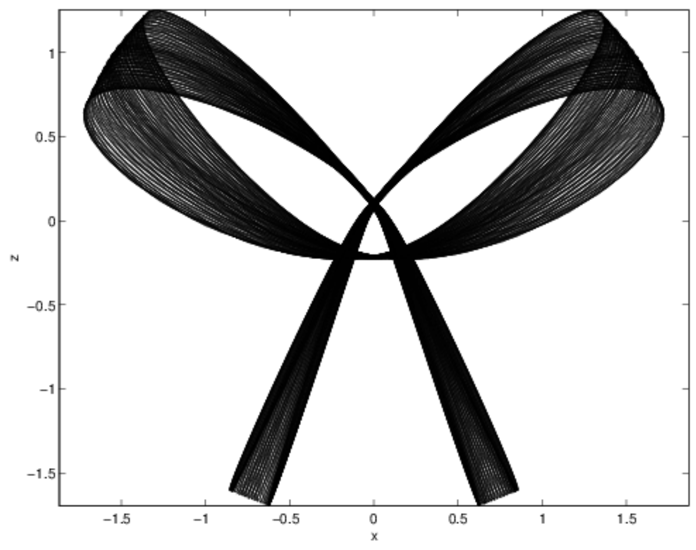


Figure I.15: 2-dimensional resonant orbit with a 4:3 frequency ratio

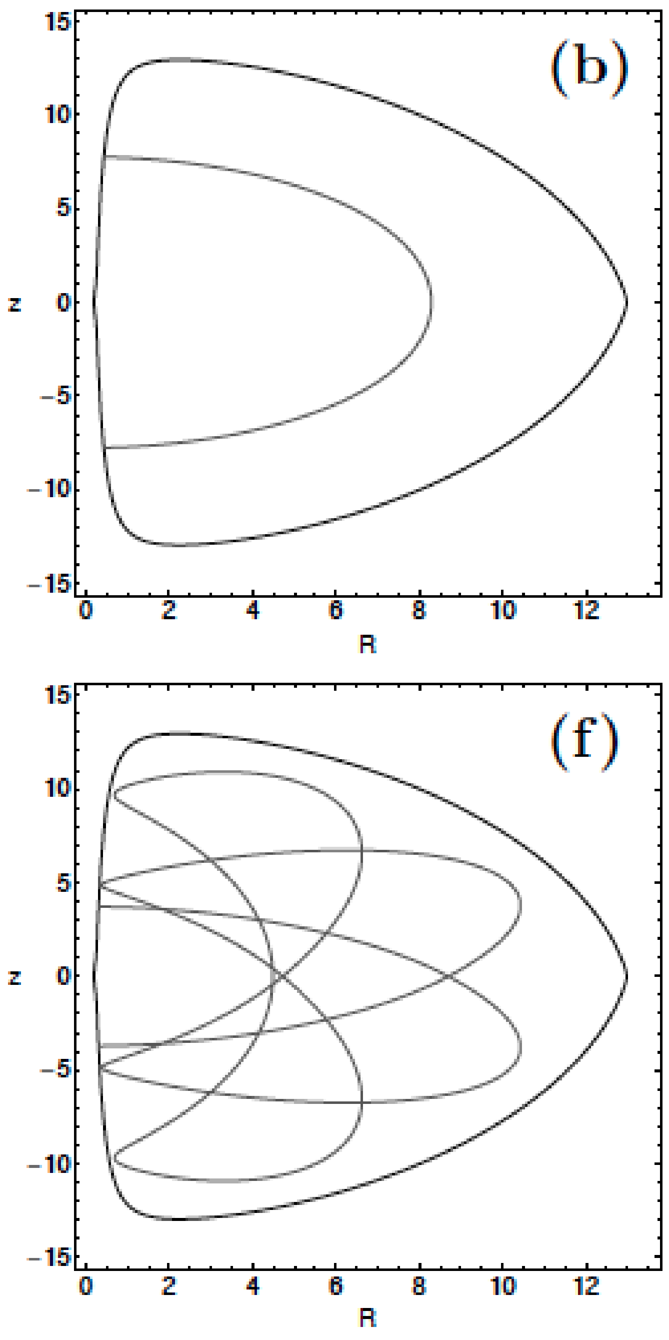


Figure I.16: The upper panel is a 2:1 resonant orbit, and the lower panel is a 10:5 resonant orbit. The contour shows the area that an orbit can spread under a given energy of a particle. Figure adapted from Zotos (2014).

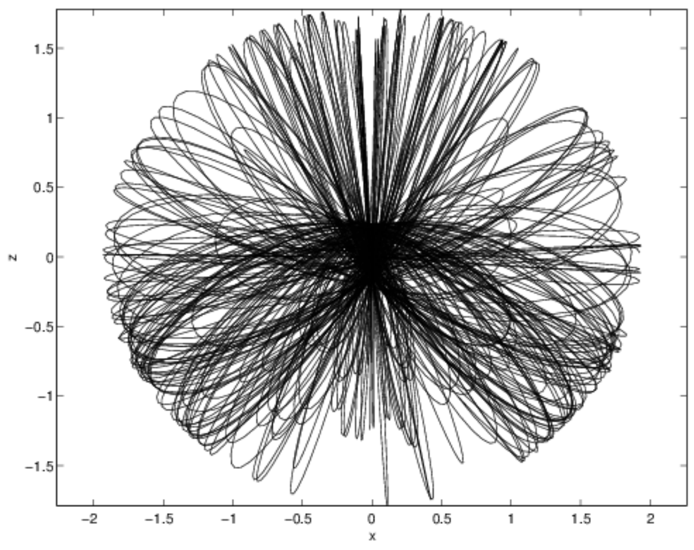


Figure I.17: Chaotic orbit

I.3.4 Gravitational recoil

As we stated in section I.3.2.1, when binary black holes reach to the last stage of their merger, they will release energy through gravitational waves to finally finish coalescence. During this process, both the total energy and angular momentum of the system consisting of the binary black holes and the gravitational waves are conserved, so that the angular momentum of the gravitational radiation is equal to that of the binary. If the radiation is anisotropic, due to the fact that the coalescing black holes are not equal massive or have misaligned spin with their orbit spin, the center of mass of the binary will be recoiled (see Volonteri, Gültekin, & Dotti, 2010, and references therein) (see Figure I.19). The recoil speed can be $v_{recoil,max} \approx 200$ km s^{-1} in non-spinning case and $v_{recoil,max} = 4000$ km s^{-1} in spinning case. This recoil velocity is sufficient for the binary to escape, since for most galaxies the escape velocity is less than 1000 km/s (see Figure I.18). This would probably explain the absence of massive black holes in dwarf galaxies and globular clusters. However, for giant elliptical galaxies, the binary will fall back in roughly half-mass crossing time (Merritt et al., 2004).

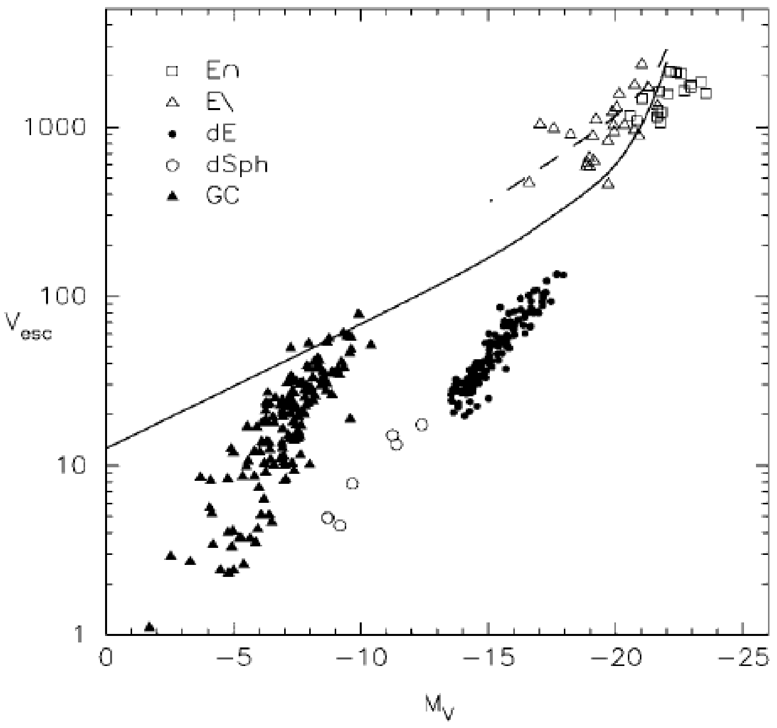


Figure I.18: Estimated central escape velocities of black holes V_{esp} in unit of km/s for different type of galaxies. The solid line is the mean estimated V_{esp} of black holes in different galaxies caused only by the dark matter halo associated with the luminous matter accordingly. The dashed line is the estimated V_{esp} for elliptical galaxies when taking into account both dark matter halo and luminous matter of the galaxies. The rest symbles show the V_{esp} estimated only from luminous matter in galaxies. Elliptical galaxies with core density profile are showed in open squares, elliptical galaxies with power law density profile in open triangles, dwarf elliptical galaxies in solid circles, dwarf spherical galaxies in open circles, and globular clusters in solid triangles. Figure adapted from Figure 2 of Merritt et al. (2004), see the references therein.

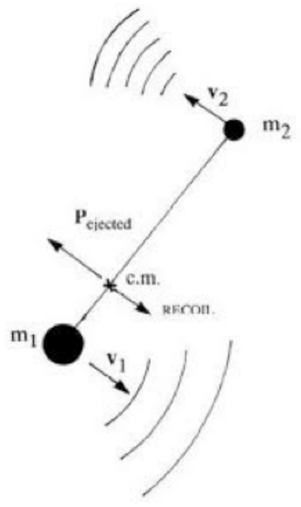


Figure I.19: Diagram of gravitational recoil. Diagram adapted from http://www.astro.cornell.edu/ favata/research.html

I.3.5 Post-Newtonian dynamics

Einstein 's general relativity equations describe the spacetime of the universe, which can explain many effects, such as frame-dragging and Mercury perihelion precession. Frame-dragging means a massive spinning object drags the space with it, making the particle orbiting it feel the distortion of the space. Perihelion precession is a phenomenon that the major axis of an object's orbit moves due to some perturbation, which can be caused by the distortion of the space. Mercury perihelion precession cannot be explained well with Newtonian dynamics, but is consistent with the prediction of general relativity, which proofs the general relativity theory. However, Einstein 's equations are nonlinear and do not have analytic solutions. To find an approximate solution, the post-Newtonian expansion is developed, which is expanding the equation as an exponential function of v/c, having the form of summation of Newtonian term and the deviations of it, where v is the velocity of a particle moving in this field. According to the Post-Newtonian dynamical equation, we can calculate the total energy of the system, $E = E_0 + E_1 + E_2 + E_{2.5} + E_3 + E_{3.5}$, where half order terms show the energy emitted by gravitational waves.

I added Post-Newtonian terms of up to 3.5 order to the self consistent field (SCF) code (see Appendix). Note that we did not publish this code , nor did we use it for the papers in Chapters II or III. The SCF code is an N-body field code to evolve a gravitationally interacting system with or without a black hole at the center, which does not calculate the force between two bodies directly, but first generates the global potential of the whole system by applying ultra spherical harmonics expansion given the positions of all particles, then calculates the force exerted on each particle under this global potential.

I.3.6 Black hole growth

I.3.6.1 Seed black hole formation

A black hole can increase its mass by accreting gas and/or merging with other black holes during its growth, but it still needs an original object as a base to grow on, which is called the seed black hole. There are four formalisms of seed black hole formation (see Johnson et al., 2012; Bellovary et al., 2011, and references therein). The seed black hole with mass of $100 \sim 300\ M_\odot$ at redshift $z \sim 20$ can be formed through the collapse of the first stars (Pop III stars), which are extremely metallicity poor. 10^4 to $10^6\ M_\odot$ black holes can be formed at redshift $z \sim 10$ via direct collapse of very metal-poor, low-angular momentum gas. Another way to form 10^4 to $10^6\ M_\odot$ black holes is the collapse of dense primeval star clusters. The primordial black holes also can be formed through the collapse of the overdensity region right after the Big Bang.

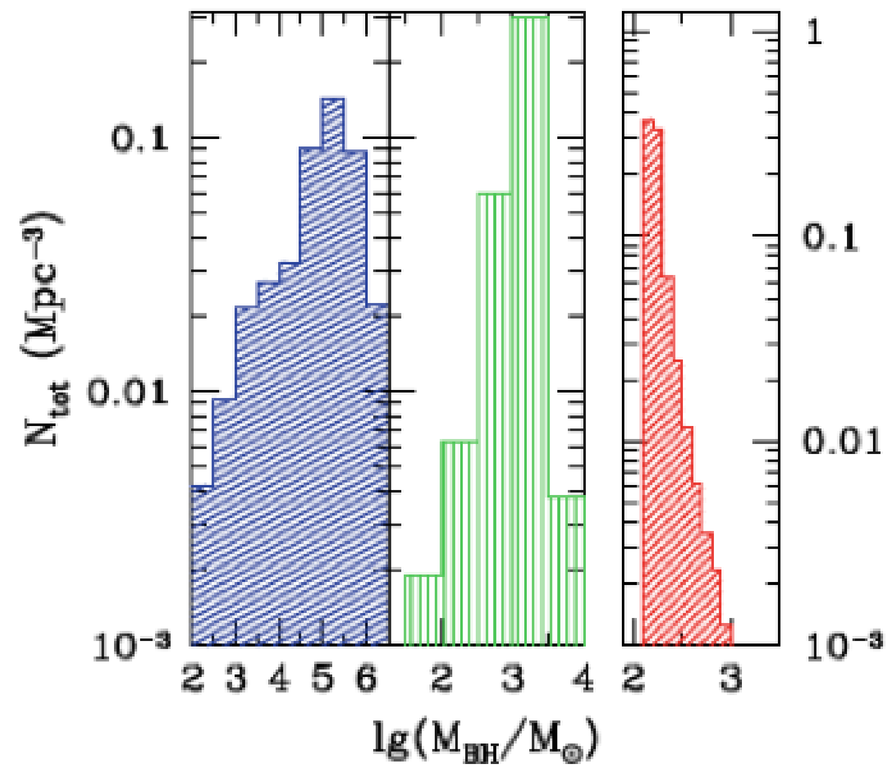


Figure I.20: Number density of seed massive black holes for three different formation scenarios. From left to right the three scenarios are direct gas collapse, runaway stellar mergers in high-redshift clusters and Population III remnants, respectively. Figure adapted from Figure 5 of Volonteri (2010) (see Volonteri, 2010, and references therein).

I.3.6.2 Accretion disk

One way for a black hole to grow in a gas rich environment is to accrete gas through an accretion disk around it (see Figure I.21). For a thin disk, the Navier-Stokes equations in cylindrical symmetry (Shu, 1992) are

$$\frac{\partial \Sigma}{\partial t} + \frac{1}{R}\frac{\partial}{\partial R}(R\Sigma V_R) = 0 \tag{I.11}$$

$$\frac{\partial}{\partial t}\left(\Sigma R^2 \Omega\right) + \frac{1}{R}\frac{\partial}{\partial R}\left(R \cdot \Sigma R^2 \Omega \cdot V_R\right) = \frac{1}{R}\frac{\partial}{\partial R}\left(R \cdot \Sigma R \cdot \nu R \frac{\partial \Omega}{\partial R}\right) \tag{I.12}$$

which basically show the mass conservation and angular momentum conservation, where σ is the surface density, V_R is the radial velocity, Ω is the angular velocity and ν is the kinetic viscosity. If consider the black hole as a mass point, then $\Omega = \sqrt{(GM/R^3)}$, thus combining the two equations yields

$$\frac{\partial \Sigma}{\partial t} = \frac{3}{R}\frac{\partial}{\partial R}\left[R^{1/2}\frac{\partial}{\partial R}\left(\nu \Sigma R^{1/2}\right)\right] \tag{I.13}$$

By dimension the evolution time $T \sim \frac{R^2}{\nu}$. If the accretion is steady, i.e., $\frac{\partial}{\partial t} = 0$, then the two conservation equations above become

$$M \dot{=} -2\pi R \Sigma V_R = constant \tag{I.14}$$

which means the flow is constant regardless of radius;

$$\nu\Sigma = \frac{\dot{M}}{3\pi}\left[1 - \left(\frac{R_\star}{R}\right)^{1/2}\right] \tag{I.15}$$

where $R_\star$ is the inner radius of the disk. A supermassive black hole with mass of $4 \times 10^6 M_\odot$, like the one in the Milky Way, is estimated to have an accretion disk with mass of several percent of it and with size of O(0.01 pc) (Dotti et al., 2007).

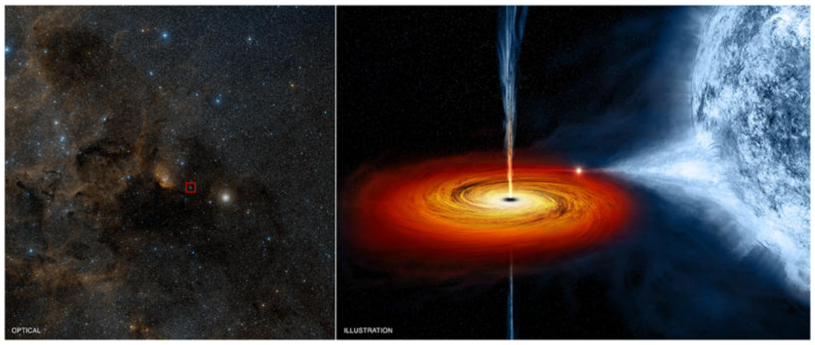


Figure I.21: An artist's conception of Cygnus X-1, another stellar-mass black hole located 6070 ly away. Image credits: NASA/CXC/M.Weiss

I.3.6.3 Black hole mergers

Black holes mainly increase their mass by accretion during both quiescent and active stages according to Soltan argument, while merger with other black holes (see Figure I.22) also contributes to the mass growth in a modest extent (see Centrella et al., 2010, and references therein). The year 2005 was a memorable year in the black hole simulation history, since in that year several groups realized the black hole binary merger simulations, overcoming many difficulties including singularity, grid structure, initial condition and math formulation(Pretorius, 2005a,b; Gundlach et al., 2005; Pretorius, 2006; Campanelli et al., 2006; Baker et al., 2006).

The energy emitted by the gravitational waves at the final coalescence stage of the binary subtracts the total mass of the system by several percent, which also can cause the sudden change of the gravitational potential, therefore influences the motion of the surrounding gas, altering the electric-magnetic field accordingly. The ground-based gravitational wave detectors, LIGO and VIRGO, can measure the gravitational waves of merging black holes with mass of $10 \sim 100 M_{\odot}$, while the space-based gravitational wave detector LISA can measure the waves from black holes with mass of $10^4 \sim 10^6 M_{\odot}$.

The final spin of the black hole remnant is highly dependent on the initial magnitude of the spins and the spins' orientation relative to the orbit angular momentum of the binary. In non-spinning case, the final formed black hole has the spin as a function of mass ratio $(\frac{a}{M})_{final} \sim \frac{q}{(1+q)^2}$, where $(\frac{a}{M})_{final} \approx 0.48$ for $q = 4$ and $(\frac{a}{M})_{final} \approx 0.26$ for $q = 10$. In spinning case, the remnant can either spin up or down.

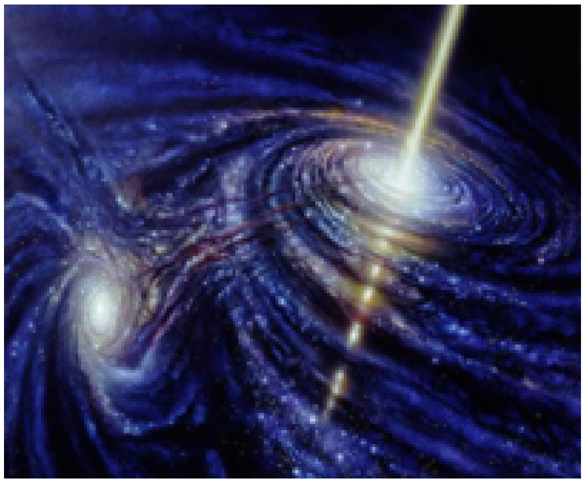


Figure I.22: An artist's conception of merging black holes. Figure adapted from http://geeked.gsfc.nasa.gov/?p=435.

I.3.6.4 Bondi-Hoyle accretion

Bondi-Hoyle accretion is a spherical accretion when a compact object, such as a neutron or black hole, is passing through interstellar medium (Bondi, 1952). The accretion rate has a form of

$$\dot{M} = \pi R^2 \rho v \tag{I.16}$$

where ρ is the density of the surrounding medium, V is the larger one of the object's velocity and sound speed C_s, and R is the effective radius obtained by equating the escape velocity of the object to the sound speed $\sqrt{\frac{2GM}{R}} = C_s$. If V is less than C_s, the accretion rate can be expressed as

$$\dot{M} = \frac{4\pi\rho G^2 M^2}{{C_s}^3} \tag{I.17}$$

For the super massive black hole in our Milky Way this rate could be $\dot{M} = \pi R^2 \rho v = \pi R^2 m_p n_w v \sim 10^{22} g s^{-1}$ with a capture radius $R \sim 0.02 pc$, number density $n_w \sim 5.5 \times 10^3 cm^{-3}$, and $v \sim 700 km s^{-1}$. (Coker & Melia, 1997).

I.4 Summary

From previous sections we know that galaxy merger plays an important role to the galaxy morphology and evolution, in which black hole merger has a strong effect on the formation of stars and AGN at the center of the host galaxy. In Chapter 2 we investigate the type of stellar orbits that can potentially interact with the black hole by classifying stellar orbits in an middle flattened axisymmetric galaxy with an SMBH embedded in. To check if the stars with orbit of interest studied in Chapter 2 really interact with the black hole binary, in Chapter 3 we have exact the same galaxy as in Chapter 2 but add another equal mass black hole into the galaxy. Then we evolve the whole system and check the energy change of particles with orbit of interest.

CHAPTER II

Classification of Stellar Orbits in Axisymmetric Galaxies

[A paper with content of this chapter was submitted to ApJ (Li, Holley-Bockelmann, & Khan, 2014).]

It is known that two supermassive black holes (SMBHs) cannot merge in a spherical galaxy within a Hubble time; an emerging picture is that galaxy geometry, rotation, and large potential perturbations may usher the SMBH binary through the critical three-body scattering phase and ultimately drive the SMBH to coalesce. We explore the orbital content within an N-body model of a mildly-flattened, non-rotating, SMBH-embedded elliptical galaxy. When used as the foundation for a study on the SMBH binary coalescence, the black holes bypassed the binary stalling often seen within spherical galaxies and merged on Gyr timescales (Khan et al., 2013).Using both frequency-mapping and angular momentum criteria, we identify a wealth of resonant orbits in the axisymmetric model, including saucers, that are absent from an otherwise identical spherical system and that can potentially interact with the binary. We quantified the set of orbits that could be scattered by the SMBH binary, and found that the axisymmetric model contained nearly seven times the number of these potential loss cone orbits compared to our equivalent spherical model. In this flattened model, the mass of these orbits is roughly 3 times of that of the SMBH, which is consistent with what the SMBH binary needs to scatter to transition into the gravitational wave regime.

II.1 Introduction

In Nature, a perfectly smooth and spherical galaxy is extremely rare – and arguably may not exist at all. Nearly every galaxy has some degree of non-sphericity, be it axisymmetry, triaxiality, warps, or flares, and it is often the case that the galaxy shape varies with radius. The global shape of the galaxy potential, however, governs the motions of the stars and dark matter throughout.

Galaxies can be grouped according to their shape as spherical, axisymmetric or triaxial. As the degree of symmetry in a galaxy decreases, there is more freedom in the orbit because there are fewer formal isolating integrals of motion. In a triaxial galaxy, for example, orbits do not have to conserve angular momentum, which admits a rich variety of regular resonant orbits (Norman & Silk, 1983; Gerhard & Binney, 1985; Magorrian & Tremaine, 1999; Merritt & Poon, 2004; Merritt & Vasiliev, 2011), such as bananas, pretzels and boxes, that can veer close to the supermassive black hole (SMBH) at the galactic center. The growth of SMBH can change the shape of a galaxy from triaxial to axisymmetric (Gerhard & Binney, 1985; Norman, May, & van Albada, 1985; Merritt & Quinlan, 1998; Wachlin & Ferraz-Mello, 1998; Valluri & Merritt, 1998;

Holley-Bockelmann et al., 2002). SMBH can keep the shape of axisymmetric galaxies by inducing chaos and constraining the shape of regular orbits (Poon & Merritt, 2001).

The orbital content of a galaxy is important because it is the skeleton that defines its shape, structure, and evolution with time. In fact, it is believed that a solution to how black holes merge together and grow may lie with stellar orbits. Theory suggests that when galaxies merge, their two SMBHs sink to the center of the remnant and form a binary, whose orbit slowly shrinks by scattering stars away, but early simulations of the process show that the binary 's orbit stalls before the black holes plunge toward merger. This is "the final parsec problem" (Milosavljević & Merritt , 2003), which has been solved recently by properly simulating SMBH binary evolution in galaxy mergers(Khan, Just, & Merritt, 2011; Preto et al., 2011), triaxial models (Berczik et al., 2006; Holley-Bockelmann & Sigurdsson, 2006), and most recently in an axisymmetric galaxy(Khan et al., 2013). However, Vasiliev, Antonini, & Merritt (2014) argued that the rates of binary hardening within their own axisymmetric model highly depend on N, the number of particles in the simulations, in the range of $10^5 \leq N \leq 10^6$. While Khan et al. (2013) find binary hardening rates consistent with a full loss cone, Vasiliev, Antonini, & Merritt (2014) argue that their own models are far from the full loss cone regime and that the apparent binary evolution is dominated by collisional processes set by numerical relaxation. The apparent disagreement between these axisymmetric results may indicate that interpreting the coalescence time of SMBH binaries within N-body simulations of this sort must be done in conjunction with an analysis of the underlying orbit structure of the model. In this paper, we analyzed the orbit content of an N-body generated black hole embedded axisymmetric galaxy model (Khan et al., 2013) to understand which orbits could enable the binary black holes to pass through the final parsec to the gravitational radiation regime.

We focused on a census of the stars with pericenters well within the radius of influence of the SMBH, though we also take stock of the resonant orbits that populate the model in general as well. The paper is organized as follows. Section II.2 describes our simulation method. Section II.3 presents our results. We conclude with a discussion and conclusion in section II.4.

II.2 Method

We begin with a spherical galaxy model with a Hernquist density profile (Hernquist, 1990), populated with 10^6 equal-mass collisionless particles and a supermassive black hole of mass 0.005 fixed at the center. Then we "adiabatically squeeze" (Holley-Bockelmann et al., 2001) this spherical galaxy to generate an axisymmetric model with axis ratios $\frac{b}{a} = 1, \frac{c}{a} = 0.75$. This model was used in Khan et al. (2013) as the background galaxy to study black hole binary coalescence.

We construct our model with the galactic center in broadly in mind, so the mass of the SMBH in system units, 0.005, maps to $4 \times 10^6 M_\odot$. To pin down the length scale, we find the radius in the model where the enclosed stellar mass is roughly twice of that of the SMBH; in system units this radius is 0.05, while in the Milky Way, this radius is roughly 1 parsec (Genzel et al., 2000; Schödel, Eckart, & Alexander, 2007; Ghez et al., 2008; Oh, Kim, & Figer, 2009). Given the mass and length scaling, the system unit velocities should be scaled by ~ 450 km/s and the system time can by scaled by $\sim 4 \times 10^4$ years. The highest velocity particle is only 1 % the speed of light, so we do not apply post-Newtonian corrections in our simulation.

In general, the technique of orbital analysis involves following the particles within a fixed background galaxy potential. Ideally, the galaxy potential should be as smooth as possible to mitigate numerically-induced diffusion in the particle trajectories; this two-body relaxation will artificially induce chaotic orbit and can scatter particles out of resonant orbits (Hernquist & Weinberg, 1992; Kandrup, 1995; Sellwood, 2003; Holley-Bockelmann, Weinberg, & Katz, 2005; Weinberg & Katz, 2007a,b). To obtain a smoother potential we '8-fold' the model, reflecting each particle position about the principle axes (Holley-Bockelmann et al., 2001). Further, we use a self consistent field (SCF) code(Hernquist & Ostriker, 1992) to evolve the orbits in all six simulation series, which will be discussed in detail in the following. The SCF code is a particle-field code, where the particles do not interact with each other directly, but are accelerated by the global potential of the black hole-embedded galaxy. Here, the stellar potential and density are expressed as series expansion of ultraspherical harmonic functions. Here we employ nmax=10, lmax=6, though the results are largely unchanged for nmax=2-20, lmax=0-15.

We run each orbit for 100 dynamical times of an circular orbit with the same energy within the combined fixed potential from the supermassive black hole and the axisymmetric stellar model(Carpintero & Aguilar, 1998). We adjust the time step of each particle to ensure that fractional energy loss from integration errors is less than 10^{-7} for each orbit. Typical fractional energy loss is less than 10^{-11} over a time span much larger than Hubble time when the model is scaled to the Milky Way.

To analyze each orbit, we evenly sample the positions and Fourier transform the trajectory to obtain the principle frequencies that characterize the motion of the particle with respect to the x, y, and z axes. We can classify the orbit type according to the frequency ratio fx/fz and fy/fz (Laskar, 1993; Carpintero & Aguilar, 1998). To distinguish a chaotic orbit from a stable one, we analyze the orbit in two time slices of 50 dynamical times; the frequency ratio of a chaotic orbit will vary between the two time slices. To record the full information of the orbit, we also keep track of the pericenter distance, the minimum angular momentum and the minimum of the z component of the angular momentum for each particle. The resonances are marked by (u,v,w), which correspond to integers and are coefficients of the equation $u \cdot fx + v \cdot fy + w \cdot fz = 0$.

To fully map the orbital structure of this potential, we conduct 3 experiments, and each experiment is comprised of the axisymmetric model and its spherical counterpart. The “Galaxy” series simply analyzes the orbits of the particles directly within the original spherical N-body model and the final adiabatically-squeezed flat model. The advantage of the Galaxy set is that it directly probes the orbits that could eventually interact with the binary black hole in the Khan et al. (2013) N-body simulation; since we use the axisymmetric model that results in a successful binary black hole coalescence, it is important to take stock of the orbits within. The disadvantage of this set is that it is merely one realization of the potential, and since a galaxy model is constructed from stars over a continuum of energies, it is difficult to compare our results to orbit analyses in the literature, where it is traditional to map out the orbital structure at a fixed energy. For this reason, we run "3D" and "2D" models that sample the phase space much more finely within 8 fixed energy slices. The "2D" series only maps orbits within the x-z plane, but this allows us a straightforward visualization of the resonant orbits, and allows us to construct a meaningful surface of section as well. See table 1 for more detail on each run.

For the 8 energy slices of the "3D" and "2D" models, the energy E=-2.5, -2, -1, -0.5, -0.4, -0.3, -0.2, -0.1, respectively. The stellar mass of particles with energy less than each E in the axisymmetric galaxy is respectively 1×10^{-4}, 6×10^{-4}, 10%, 40%, 45%, 55%, 70%, 80% of the total stellar mass. The corresponding radius for a particle to run on a circular orbit with each E in the axisymmetric galaxy is respectively 0.0015, 0.0025, 0.025, 0.45, 0.7, 1, 2, 4. The units shown in all figures are model units unless otherwise indicated.

Since this model was constructed non-analytically by dragging particles in velocity using an N-body simulation, it is not guaranteed to be a precisely homologous figure. We characterize the global shape by the axes ratios at the half mass radius, $b/a = 1$ and $c/a = 0.75$, and as can be seen in Figure II.4, the shape is fairly stable throughout the bulk of the model. The one big exception is at large radius, where the orbital time of the particles is so long that the squeezing technique is non-adiabatic and therefore the orbits of these outermost particles are largely unaffected by the applied velocity drag; this affects about 20% of the particles on the outer edge of the system. We should therefore expect that the orbital content of the outskirts of this galaxy model should mimic the spherical model and that we are missing the axisymmetric orbit families that would lie out there if the model were perfectly homologous.

We note one other seemingly small detail: at the innermost part of the model, within the central 0.5 parsec, which is within the radius of influence of the SMBH, c/a is less flattened, around 0.85, and b/a trends below 1.0, around 0.96. Here, the model is actually triaxial with T=0.28. The mass fraction involved in the triaxial portion is small – less than 0.25% – about half the SMBH mass. The finding of a technically triaxial shape inside the radius of influence of the SMBH may seem counter to previous work (Valluri & Merritt, 1998; Holley-Bockelmann et al., 2002), which finds that the presence of a SMBH will act to sphericalize a triaxial shape. However, our finding is not inconsistent for several reasons. First, our model was embedded with a SMBH in place at its full mass before we morphed the galaxy shape, while most previous work focused on how galaxy models adjusts to a SMBH that starts at zero mass and grows. Second, earlier work may only

Table II.1: Model detail

model name	particles' initial condition	potential	model dimension	number of particles
Galaxy	axisymmetric galaxy	axisymmetric	3D	1 million
3D	random	axisymmetric	3D	0.8 million
2D	random in xz plane	axisymmetric	2D	0.8 million
Galaxy-sp	spherical galaxy	spherical	3D	1 million
3D-sp	random	spherical	3D	0.8 million
2D-sp	random in xz plane	spherical	2D	0.8 million

have noted the trend toward a spherical figure (which we are in fact seeing – note the axis ratios are both trending toward 0.9); they may have counted such minor triaxiality that we observe as essentially spherical. Finally, previous work followed the evolution of triaxial models over hundreds of dynamical times, while it is not clear how long the figure shape we observe will persist.

II.3 Results

II.3.1 Prominent orbit families

In theory, axisymmetric potentials are thought to harbor resonant, centrophilic orbits (Vasiliev, 2014) that bear some broad similarity to those in triaxial systems (Sridhar & Touma, 1999; Sambhus & Sridhar, 2000); the main difference is that since the degree of symmetry is higher in axisymmetric systems, they should admit fewer chaotic orbits and, naturally, more 1:1 resonances within the symmetry plane (Poon & Merritt, 2001). Of particular note in an axisymmetric model is the saucer orbit, predicted within an analytical potential (Richstone, 1982; Lees & Schwarzschild, 1992). We identified saucer orbits within our N-body model of an axisymmetric galaxy, even though the potential is neither a homologous figure nor an analytic potential. Figure II.1 shows the projection on x-y plane, x-z plane and R-z plane of the saucer orbit within our 3D run. This orbit traversed the inner 0.7 parsec of the model, with pericenter passes only 0.05 parsec from the SMBH. These orbits are thought to be key in interacting with and being scattered by binary SMBHs.

Unexpectedly we also found pyramid orbits(Sridhar & Touma, 1997; Merritt & Valluri, 1999; Sridhar & Touma, 1999; Poon & Merritt, 2001), which are thought to exit only in triaxial galaxies, as they originate from breaking the symmetry axis of a saucer parent orbit (Merritt & Valluri, 1999). Figure II.2 displays a pyramid orbit from our simulation. From the x-y plane projection, it is clear that the pyramid passes through the center of the galaxy, while the saucer does not. These are also ideal orbits to comprise the loss cone for binary black hole coalescence. In our model, these pyramid orbits only exist in the part of the model that exhibits slight triaxiality within 2 parsecs of the SMBH. With such a minor degree of triaxiality, it is perhaps surprising that these orbits exist at all; indeed, it is not clear how small the deviation from non-axisymmetry must be to admit these formally triaxial orbits.

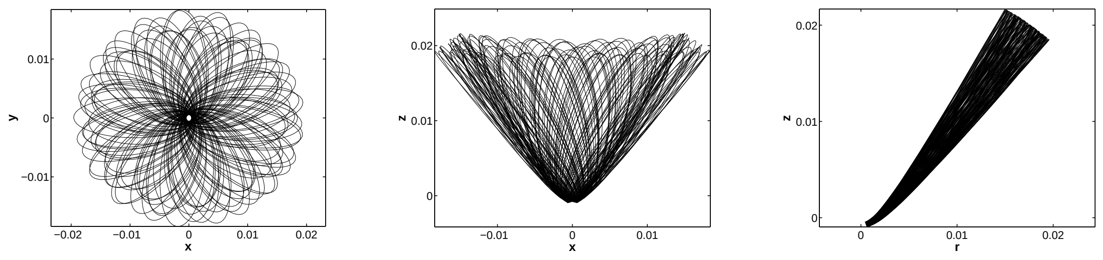


Figure II.1: A typical saucer orbit that emerged in the Galaxy and 3D simulations. The three panels from left to right show the orbit projection in the x-y, x-z and r-z plane respectively, where $r = \sqrt{x^2+y^2}$.

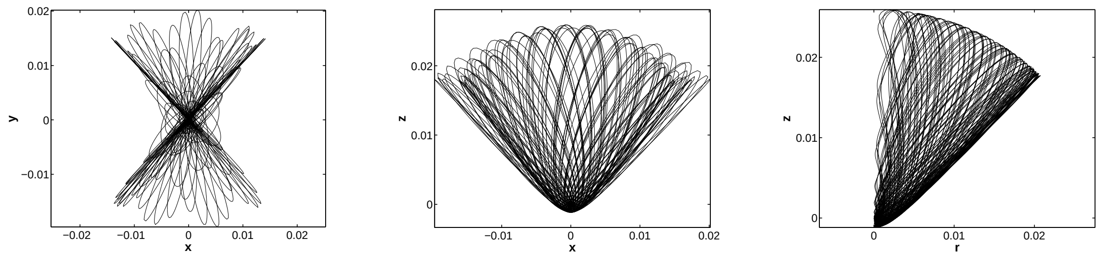


Figure II.2: A typical pyramid orbit emerged in the Galaxy and 3D simulations. The three panels from left to right show the projection of the pyramid orbit in the x-y, x-z and r-z plane respectively, where $r = \sqrt{x^2+y^2}$. Note the hole in the x-y plane projection of the saucer orbit, which is not present in the pyramid orbit.

We found these distinctive orbits in all our axisymmetric runs although note that in two-dimensions, pyramids and saucers do not distinguish from each other (Merritt & Vasiliev, 2011). Through the observation of hundreds of orbits, we defined the criteria to separate saucers from pyramids within our Galaxy run, where the orbits are directly from the N-body model. Saucers satisfy $-2.2 < E < -1.7$, $fx/fz < 1$, $fy/fz < 1$, and $1.74 \times 10^{-4} < L_{min} < 0.0035$. In our particular potential, pyramids satisfy $-2.2 < E < -1.7$, $fx/fz < 1$, $fy/fz < 1$, and $L_{min} <= 1.75 \times 10^{-4}$. There are approximately 600 saucers and 150 pyramids in the Galaxy run.

Since we are motivated to examine orbits to better understand how they promote rapid SMBH coalescence, we search for "orbits of interest" within our Galaxy model(Vasiliev, 2014). These orbits could potentially lie within the binary SMBH loss cone, and are a composite of formally centrophilic orbit families such as boxes or pyramids, as well as those orbits with pericenters roughly that of the separation between SMBHs in Khan et al. (2013), including chaotic orbits. In our axisymmetric model, there are over 14000 such orbits, with a total mass of 0.014 in system units, which is 3 times larger than the SMBH, while the spherical model only hosts about 2000 of these orbits.

In our 3D simulation, where we can more finely-sample the orbit content based on the initial energy of the orbit, saucers and pyramids are primarily evident in the deep energy slice at E=-2. In this region, they are also separately distributed in frequency and angular momentum space. Figure II.5 shows fy/fz versus fx/fz. The red dots are pyramids, green ones are saucers, others are in blue. We can easily see from this figure that the saucers mainly lie on the fx=fy diagonal line, while pyramids spread around the line fy/fz=0.5. Saucers and pyramids are also easily separable in angular momentum at this fixed energy slice; saucer orbits comprise 15 percent of the total mass of this energy slice, while pyramids are 6 percent.

Figure II.3 shows the surface of section of the E=-2 slice in the 2D simulation, the green dots are saucers, the blue ones are others. Saucers are those with a minimum angular momentum less than 0.0035, while other orbits have larger angular momenta, and recall that in 2D simulations, saucers and pyramids are the same (Merritt & Vasiliev, 2011).

II.3.2 Global orbital structure

Though we concentrated on the orbits that could encounter and interact with a binary SMBH in each model, there are a rich variety of resonant orbits throughout the system, and we discuss the global orbital content here.

Figure II.6 shows the surface of section of two energy slices in the 2D run, colored by the fz/fx ratio to denote different orbit families. It is readily apparent from the large area occupied by the 1:1 loop orbit that it is the dominant family; in the spherical model it is the only regular, non-chaotic, orbit family. Fish, pretzels

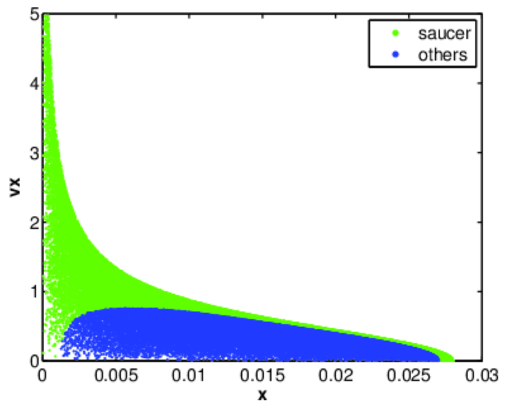


Figure II.3: Surface of section of vx versus x of E=-2 slice in the 2D simulations. The saucers are in green and others in blue. Saucers have the angular momentum $L_{min} < 0.0035$. It is seen that in the upper part of the figure, the angular momentum of the particles are smaller.

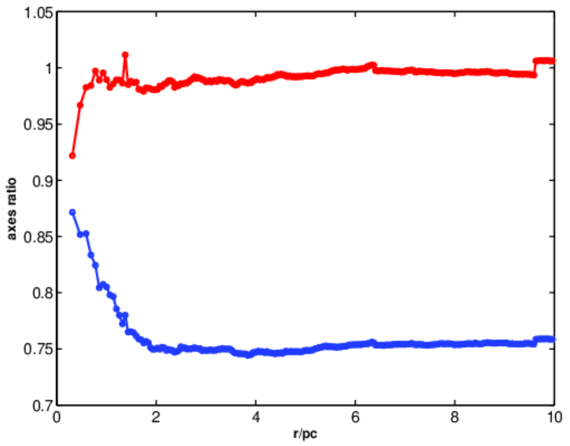


Figure II.4: Axes ratio b/a (red) and c/a (blue) in the inner 10 parsecs of the axisymmetric model. Though not plotted, the axis ratios are stable and the system is axisymmetric within 100 parsecs; at larger distances, the system becomes more spherical because the timescale for the applied velocity drag is non-adiabatic compared to typical orbital timescales there.

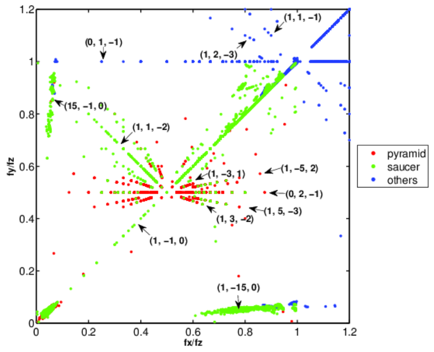


Figure II.5: fy/fz versus fx/fz for 0.1 million particles with E=-2 in the 3D simulation. Pyramids are denoted by red dots, saucers by green dots and others by blue dots. Both saucers and pyramids have $fy/fz < 1$ and $fx/fz < 1$ and small angular momentum. The angular momentum of saucers are $1.75 \times 10^{-4} < L_{min} < 0.0035$, while that of the pyramids are $L_{min} <= 1.75 \times 10^{-4}$. In this energy slice, the saucers are 15% and the pyramids are 6%. Since these are resonant orbits, they will lie on a distinct line in this frequency map. We mark notable resonance lines by (u,v,w), which correspond to integers and are coefficients of the equation $u \cdot fx + v \cdot fy + w \cdot fz = 0$. Pyramids mainly lie on lines (1, -3, 1) (1, -5, 2), (0, 2, -1), (1, 5, -3) and (1, 3, -2), while saucers are present on (15, -1, 0), (1, -15, 0), (1, 1, -2) and (1,-1,0).

also feature significantly in these energy slices, and though the fraction of chaotic orbits are small, they are present peppered throughout the region occupied by high-order resonances.

Figure II.7 presents the percentage of different orbit types as a function of energy. It can be seen that 1:1 loop orbits are the dominant orbit family at nearly every energy; “other resonant” orbits begin to dominate only at the slice that is most highly-bound to the SMBH. At the least-bound energy slice the percentage of 1:1 loops is higher than 85% and this may be partially due to the fact that this slice contains some orbits near the physical outskirts of the system, where adiabatic squeezing is less effective at transforming the shape. The fraction of low-order resonant orbits increases for more tightly-bound orbits. Aside from the loop orbit, the 3:2 fish orbit family is the most prominent of the ones we tracked. The percentages of 4:3 pretzels, 2:1 bananas higher-order resonances and chaotic orbits are always below 10%.

The left and right panels of Figure II.8 show fy/fz vs fx/fz of the Galaxy-sp and Galaxy simulation respectively. In the spherical galaxy model, 88% particles lie around the (1,1) point, which means they have the fx:fy:fz=1:1:1, while in the axisymmetric model this percentage is 33%. However in the axisymmetric model the percentage of particles lying on the line fx=fy is 98%; these are short-axis tubes. The resonance orbits lying on line $u \cdot fx + v \cdot fy + w \cdot fz = 0$ in the axisymmetric model are also marked by the line coefficient (u,v,w) as showed in the figure. It is seen that comparing with the Galaxy-sp model showed in the left panel, the Galaxy model has a rich variety of orbits such as (1, 1, -2), (3, 3, -4), (0, 3, -2), (0, 2, -1) and (1,1,-1), etc..

The left panel of Figure II.9 displays the mean pericenter distance of the particles in each bin as a function of mass fraction for the Galaxy (red) and Galaxy-sp (blue) run. There are 100 bins in each simulation,

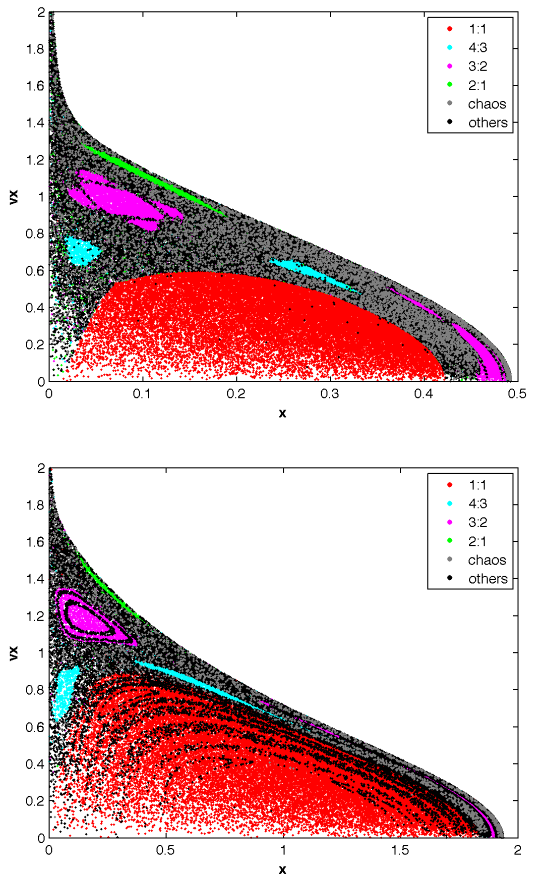


Figure II.6: Surface of section in the 2D simulations. The two panels from left to right show the surface of section of E=-1,-0.4, slice respectively in the ax-2D-random simulations. The stellar mass of particles with energy less than each E in the axisymmetric galaxy is respectively 10% and 45% of the total stellar mass. The dots are colored by fz/fx, in which 1:1 loops are denoted by red dots, 4:3 pretzels by cyan dots, 3:2 fishes by magenta dots, 2:1 bananas by green dots, chaos by grey dots and other resonances by black dots. The 1:1 loop is always dominant. Chaotic orbits always occupy the lower angular momentum part of the figure, as they interact with the SMBH.

with 10000 particles per bin. It is clear that the mean pericentric distances are smaller in the axisymmetric galaxy out to an enclosed mass of 70%, and at that point the model is more nearly spherical. Note that we calculate the pericentric distance in ellipsoidal coordinates so that we are not biased by the more compact vertical dimension in the flattened model; in other words, $r_{min} = \sqrt{(x/a)^2 + (y/b)^2 + (z/c)^2}$. To quantify the difference between the pericentric distances more explicitly, the right panel is the difference between the Galaxy and Galaxy-sp models, weighted by the axisymmetric model. Orbits delve 50% deeper into the center at mass fraction of 2%. For the most part, the difference is over 10%.

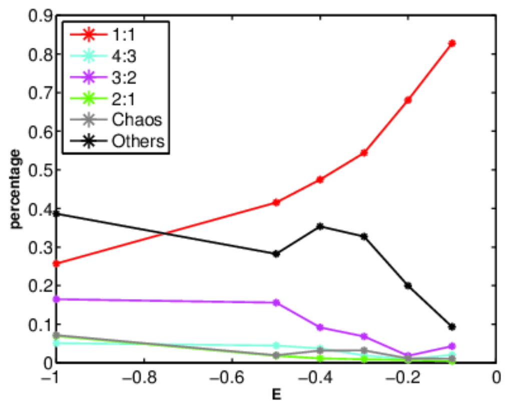


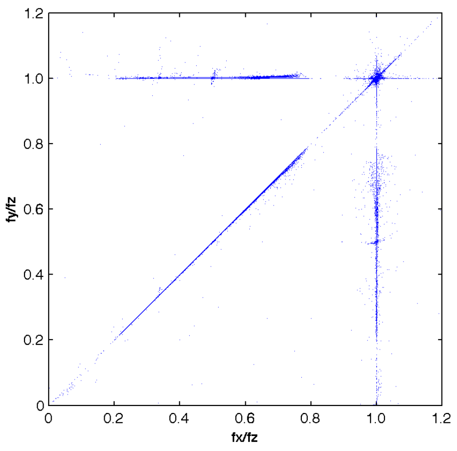
Figure II.7: The percentage of different type of orbits as a function of energy in 2D simulations. This plot shows the percentage of each type of orbits presented in Figure II.6 with the same legend. The trend is the rate of 1:1 loop keeps increasing as the energy rises, while the rates of nearly all the other types decrease, of which only the 3:2 fishes and "other resonances" are ever over 10%.

II.4 Conclusion and Discussion

We identified several major orbits families in our axisymmetric galaxy model such as saucers, bananas, fishes, and short-axis tubes. These orbits are present despite the relatively minor flattening (c/a=0.75) compared to a spherical model. Due to a very slight deviation from an oblate spheroid at the center of the axisymmetric model (T=0.28), pyramid orbits are also present, making up 6% of mass within the inner 0.5 parsec. It is not clear how much a system needs to deviate from axisymmetry to generate pyramids.

Since we are primarily interested in whether the orbital content in the axisymmetric model is sufficient to drive binary black holes to coalesce, we took a census of those particles that would reside in the loss cone of a binary black hole. The total mass of particles with orbits that could interact with a hard binary black hole in the axisymmetric galaxy simulation is roughly three times that of the SMBH, and about seven times of that in the spherical galaxy simulation. According to three-body scattering experiments, the SMBH binary needs to scatter $1.2 \sim 1.5$ times its mass to transition to the gravitational wave regime (Quinlan & Hernquist, 1997; Sesana, Haardt, & Madau, 2007), and this is consistent with the mass of stars on potential loss cone orbits in our axisymmetric model. In a separate work, we will track which of these orbits are actually scattered by the SMBH binary as the system evolves, but it appears that the orbital content in our axisymmetric model is more than enough to drive the SMBHs to merge in less than a Hubble time.

There may be several reasons why the hardening rates in Vasiliev, Antonini, & Merritt (2014) and Khan et al. (2013) differ. A suggestion has been made that numerical relaxation may have artificially enhanced SMBH binary scattering in Khan et al. (2013), while another idea posed is that the system in Khan et al. (2013) is more perturbed from virial equilibrium, and it is this time-dependent perturbation that refills the



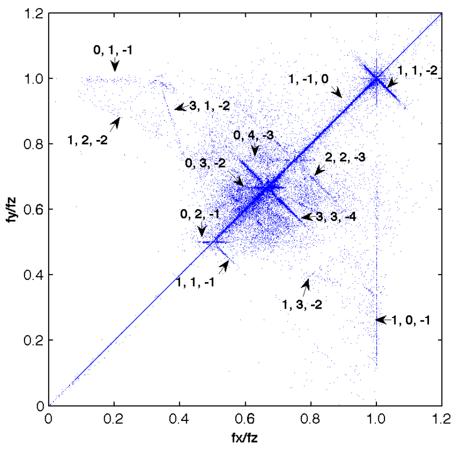


Figure II.8: Left: fy/fz versus fx/fz for Galaxy-sp simulations. 88% particles are lying at the point (1,1), which means they have fx=fy=fz and are 1:1:1 tubes, and nearly all the rest particles reside on lines fx=fy, fx=fz and fy=fz; there are no complex orbit types in this model. Right: fy/fz versus fx/fz for Galaxy simulations. In contrast to the Galaxy-sp model, the Galaxy model has a rich variety of orbits such as (1, 1, -2) and (3, 3, -4), (0, 3, -2), (0, 2, -1) and (1,1,-1), etc. However 98% particles are still short-axis tubes.

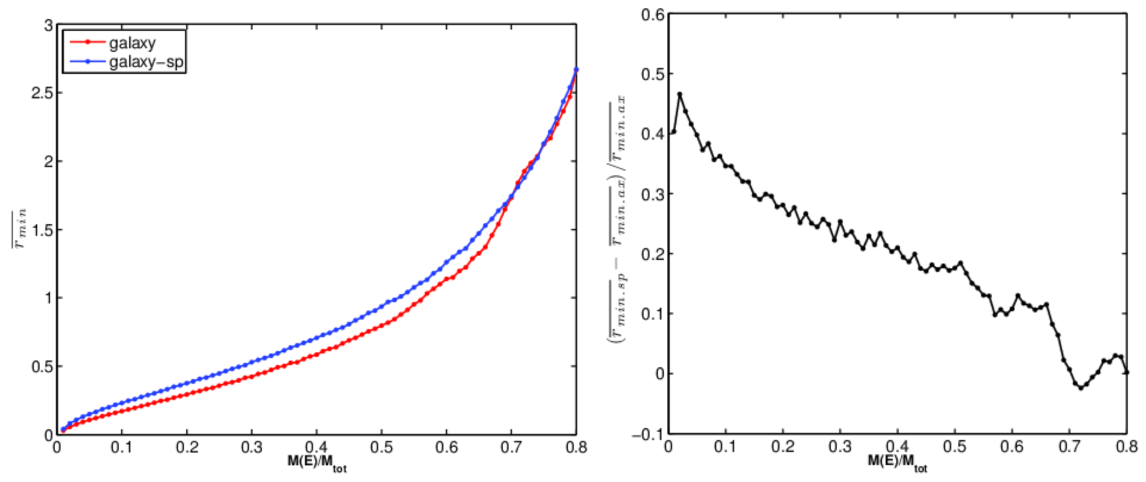


Figure II.9: The left panel shows $\overline{r_{min}}$, the average pericentric distance in each mass bin, as a function of mass fraction. The blue line is for Galaxy-sp model and the red line for Galaxy model. The right panel quantifies the difference between the pericentric distance in each model: $(\overline{r_{min,sp}} - \overline{r_{min,ax}})/\overline{r_{min,ax}}$, as a function of mass fraction. Inside around 70% mass fraction, this difference is always over 10%, reaching nearly 50% at mass fraction of 2%.

loss cone (R. Spurzem, private communication). The results of our work imply that the slight triaxiality in our model inside the radius of influence of the black hole may be the key in explaining the apparent difference between the two results. The triaxial center in our model increased the number of potential loss cone orbits near the black hole, spawning formally centrophilic orbit families – like pyramids – to appear, and allowing for a wide diffusion of orbits in angular momentum. If it is true that the central shape is a major factor in the differing SMBH coalescence times in these two papers, we are left with several interesting and related questions: what is the orbital content for more realistically-flattened models?; how small a deviation from pure axisymmetry is required to gain enough centrophilic orbits to drive SMBH binaries to coalesce?; and are any real galaxies perfectly axisymmetric enough to pose a final parsec problem?

CHAPTER III

Supermassive Black Hole Binary Mergers within Axisymmetric Galaxies

We analyze the time-dependent structure of an axisymmetric galaxy with an initially wide SMBH binary at the center. As the SMBH binary coalesces, we investigate the orbits and origin of stars that are scattered by the binary, and explore the effect of the binary merger on the orbital content of the host galaxy.

The initial conditions and particle ids of the axisymmetric galaxy here are exactly the same as in Chapter II. The only difference is that we add another equal mass black hole at an position x=0.5 with a initial velocity v=vy=0.45 (both are in system unit, see Chapter II for scaling to physical unit). Then we let the whole system composed of two black holes and 1 million particles evolve under Newtonian gravity until the distance between the black hole binary reaches around 0.01 pc. To see if the particles with orbit of interest (see Chapter II for the definition) really interact with the black hole binary, we plot the histogram of energy change for all particles and particles with orbit of interest in Figure III.1, in which the horizontal axis is the energy change in percentage, $(E_f - E_i)/E_i$, where E_i and E_f are the initial energy and final energy of a particle respectively; the vertical axis is number of particles N in each bin over total N in log space, where N_{tot} for all particles is 1 million, N_{tot} for particles with orbit of interest is 14122. From this Figure, we can notice that particles with orbit of interest nearly always have higher percentage of particles at the same energy change rate than all particles have. This is a preliminary result.

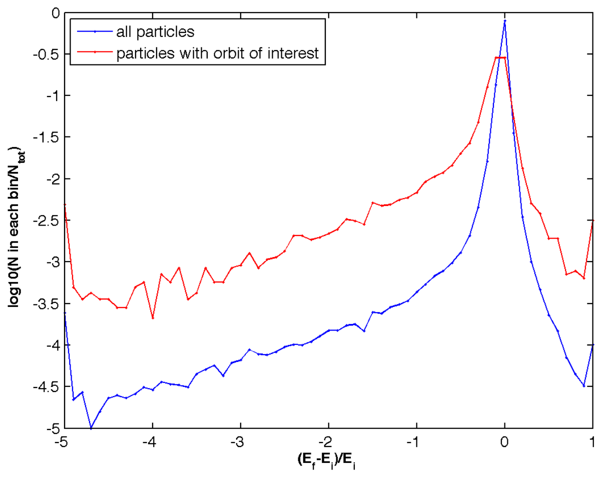


Figure III.1: Comparison of energy change of all particles and particles with orbit of interest. The particle ids are the same with project in Chapter II.

CHAPTER IV

Expansion Techniques for Collisionless Stellar Dynamical Simulations

[A version of this chapter was published as Meiron, Li, Holley-Bockelmann, & Spurzem (2014).]

We present GPU implementations of two fast force calculation methods, based on series expansions of the Poisson equation. One is the Self-Consistent Field (SCF) method, which is a Fourier-like expansion of the density field in some basis set; the other 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 method and optimized for collisionless galactic dynamics, but while SCF is a "pure" expansion, MEX is an expansion in just the angular part; it is thus capable of capturing radial structure easily, where 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 ~ 0.1 seconds (depending on expansion cutoff), making simulations with as many as 10^8 particles fast on a comparatively small number of nodes.

IV.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 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 yields 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 in gravity too this simplification can cause problems, as some dynamical effects depend on number density rather than just 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 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 similar order of magnitude to 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 stellar interactions, this is cheaper computationally. 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 these facts. These 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 of 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 radius r from the center is simply $GM(r)/r^2$, towards the center, where $M(r)$ is the enclosed (internal) mass. The gravitational constant, G, will be omitted in the following text. 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 this idea is that when spherical symmetry breaks, corrections to the force can be expressed by summing higher multiples (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 a 1 000 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 conducted simulations of also up to $N = 1\,000$ 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 Secion IV.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 2D 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. Cf. a related method by Schaefer et al. (1973). Clutton-Brock (1973) introduced a 3D 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 IV.2.5.

To exploit recent developments in the world of general purpose computing on GPUs, we implemented both SCF and MEX routines in a code called *ETICS* (acronym for *Expansion Techniques in Collisionless Systems*). In Section IV.2 we explain the mathematical formalism of both methods and highlight the differences between them. In Section 3 we explain the unique challenges in a 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.

IV.1.1 Glossary

Here we clarify some terms used throughout this work:

Expansion methods a way to get potential and force by summing a series of terms; in this paper 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 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 is also used for general-purpose computing. It often lies on a video card[1] that can be inserted into an expansion slot on a computer motherboard.

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

IV.2 Formalism

IV.2.1 Series Expansions

Both MEX and SCF methods are ways to solve the Poisson equation:

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

the formal solution of which is given by the integral:

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

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

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

where $\delta(\mathbf{r})$ is the 3D Dirac delta function. Direct N-body techniques evaluate integral (IV.2) *directly*:

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

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 in 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 (IV.2) is expanded as a series of terms, each of which 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 particle to sample the phase space.

To demonstrate the difference between the two approaches in the following Section, let us consider a 1D version of integral (IV.2); let us further assume that the density exists in the interval $0 \le x \le 1$:

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

Note that this is not a solution for a 1D Poisson equation (hence the notation I instead of Φ), but just a simplification we will use to illustrate the properties of each method. We will conveniently ignore the fact that this integral is generally divergent in 1D, as it does not affect the 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 the 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.

IV.2.2 MEX

Let us define $z \equiv x'/x$ and expand the Green's function equivalent in equation (IV.5) around $z = 0$, we get that for $z < 1$ or $x' < x$:

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

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

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

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

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

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

and thus

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

While seemingly we 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

$$\begin{aligned} q_l(x_i) &= \sum_{j<i} m_j x_j^l && \text{(IV.11)} \\ p_l(x_i) &= \sum_{j>i} m_j x_j^{-(l+1)} && \text{(IV.12)} \end{aligned}$$

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.

IV.2.3 SCF

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

$$\begin{aligned} \rho(x) &= \sum_{n=0}^{\infty} a_n \rho_n(x) && \text{(IV.13)} \\ a_n &= \int_0^1 \rho(x')\rho_n^*(x')\mathrm{d}x' \end{aligned}$$

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

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

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 that the following:

1. The functions $\rho_n(x)$ and $I_n(x)$ are easy to evaluate numerically.

2. The sum (IV.13) convergence quickly, or in other words, $\rho_0(x)$ is already close to $\rho(x)$.

IV.2.4 Properties in Three Dimensions

The standard form of MEX in 3D is

$$\begin{aligned}\Phi(\mathbf{r}) &= -\sum_{l=0}^{\infty}\frac{4\pi}{2l+1}\sum_{m=-l}^{l}\left[q_{lm}(r)r^{-(l+1)}+p_{lm}(r)r^{l}\right]Y_{lm}(\theta,\phi) && \text{(IV.15)}\\ q_{lm}(r) &= \int_{r'<r} r'^{l}\rho(\mathbf{r}')Y^{*}_{lm}(\theta',\phi')\mathrm{d}^3r' && \text{(IV.16)}\\ p_{lm}(r) &= \int_{r<r'} r'^{-(l+1)}\rho(\mathbf{r}')Y^{*}_{lm}(\theta',\phi')\mathrm{d}^3r'. && \text{(IV.17)}\end{aligned}$$

All together there are $\frac{1}{2}(l_{\max}+1)(l_{\max}+2)$ 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:

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

All together there are $\frac{1}{2}(n_{\max}+1)(l_{\max}+1)(l_{\max}+2)$ 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 relation between the two methods:

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

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, and they are related to each other by complex conjugation.

If one decompose 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 term 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 the zeroth radial basis function, and is usually determined by trial and error (see Section IV.4.1).

It is interesting to note a nontrivial mathematical difference between the two methods. One can show that the Laplacian of equation (IV.15) is zero when substituting the appropriate expressions for q_{lm} and p_{lm} from the equations (IV.16) and (IV.17); the proof is mathematically cumbersome and will not be brought 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 a 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 no information about the density at $\mathbf{r}$ itself. This is the case also when the potential is constructed by a direct-summation of all gravitational point sources, so one may say MEX is similar to direct methods in this sense. In SCF, by construction, taking the Laplacian of equation (IV.18) leads right back to the density field (equation IV.1). One can thus use the coefficients A_{nlm} to represent a smoothened 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.

IV.2.5 Radial Basis

A key difference between MEX and SCF is the freedom of choice of *radial* basis. There are in fact two function sets: the radial densities $\{\rho_{nl}(r)\}$ and the radial potentials $\{\Phi_{nl}(r)\}$; they 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 (cf. 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) which zeroth order was a Hernquist (1990) model; this is the basis set we adopt in *ETICS*. They argued that this basis was more well 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 as set which 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), Weinberg (1999) describe 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 which zeroth order is the perfect sphere of de Zeeuw (1985).

IV.3 Implementation

IV.3.1 Parallelism

There are several levels of task parallelism available when writing computer code. 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 they are generally managed within one "process" of the program. A higher level of parallelism is called *distributed memory* scheme, where tasks are performed on different units (often called "nodes"), but each unit has access only 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 faster and more easily cooperate. 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.

Graphics processing 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 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, thus it is a shared memory device[2]. 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 to smaller chunks (each could be on a different node), from each chunk partial A_{nlm}-s are calculated. Then the partial coefficients summed up and the result communicated to all the nodes. This was done by (Hernquist, Sigurdsson, & Bryan, 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 IV.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.

[2]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 private at the thread or block level. Thus, GPUs has also distributed memory characteristics.

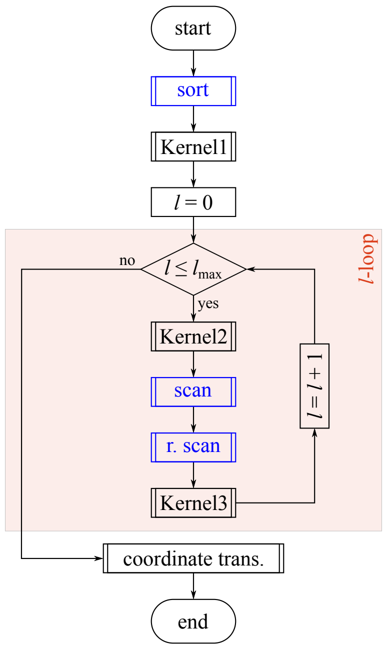


Figure IV.1: Flowchart of the MEX routine. Boxes with double-struck vertical edges indicate a GPU-accelerated operation. Blue color represents call to the represent *Thrust* library.

IV.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 choice of algorithm is performed by the library. *Thrust* provides a sorting routine that 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 Fig. IV.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 ones are regular CUDA kernel calls. When a GPU operation is in progress, the CPU flow is paused. Fig. IV.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 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

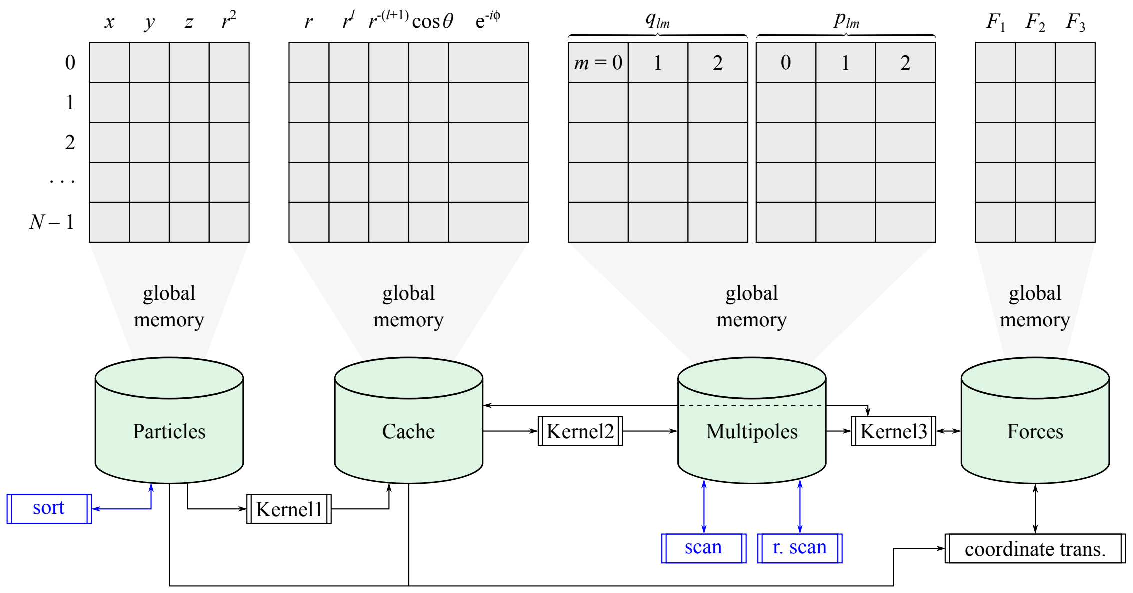


Figure IV.2: The main memory structures in the MEX routine, and a scheme of the action of the *Thrust* subroutines (blue) and kernels on them. The wider boxes for the exponent (in the cache structure) and the multipoles represent complex numbers (require twice the memory). The layout of the multipole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

the cache structure.

Kernel2 calculates the spherical harmonics at the current l-level and from that the contribution 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 both in fact 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. Fig. IV.6 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 Kernel3) and stored at another memory structure (not shown in Fig. IV.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 done with C++ templates.

IV.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 3D 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 necessary special functions were calculated using recursion relations. Step (2a) was identical but additionally, the derivatives of those functions were 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 it was achieved using just one command, `MPI_Allreduce`. There are two main reasons why this algorithm could not be used effectively on a GPU, 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 by far exceed the number of thread processors on the GPU chip), and every partial A_{nlm} is 5 kilobyte 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 complex exponent (as is done in the HO92 code), 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*[3]. 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 kilobytes of shared memory per block, while they have 5 gigabytes 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

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

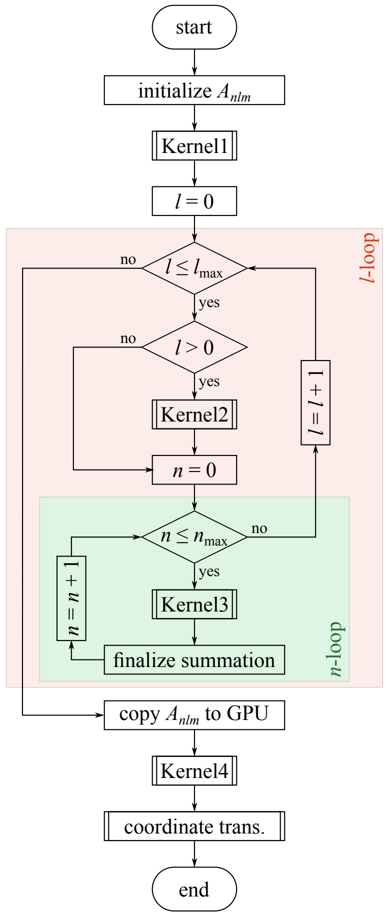


Figure IV.3: Same as Fig. IV.1 but for the SCF routine.

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 threads 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 Fig. IV.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. Fig. IV.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 particle coordinates (in practice it contains additional data such as ID and velocity, but this is not used by the SCF routine); the cache structure contains 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 structure. Kernel2 only

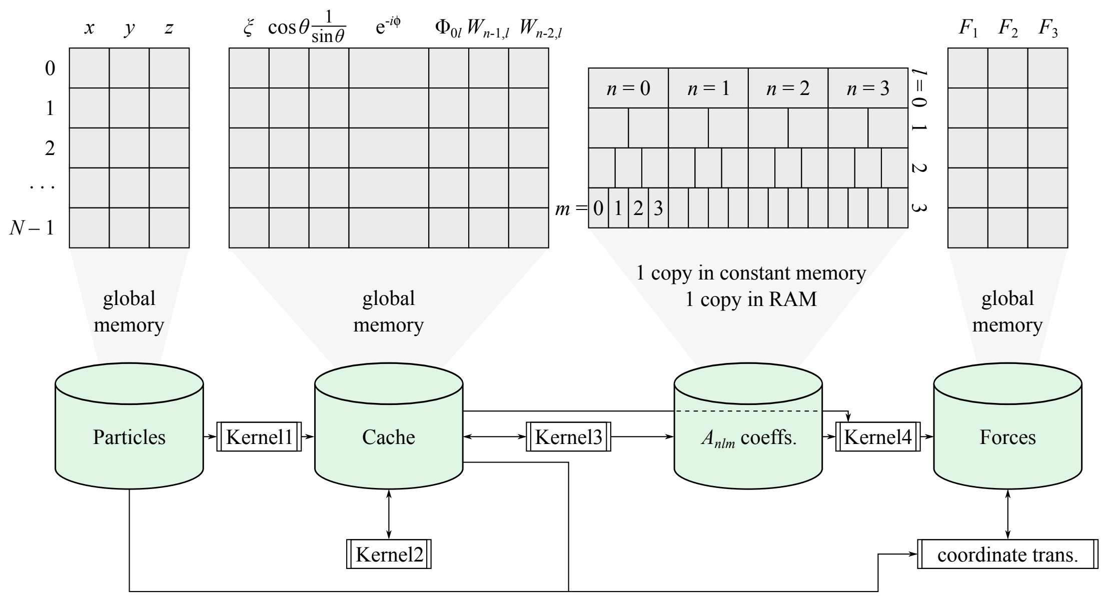


Figure IV.4: Same as Fig. IV.2 but for the SCF routine. The main memory structures in the SCF routine, and a scheme of the action of the kernels on them. Every cell in the coefficient structure (shown here for $(n_{\max}, l_{\max}) = (3,3)$) is a complex number.

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 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 transfered to the host machine's memory and the CPU finishes the summation process.

For the force calculation, just a reading the A_{nlm}-s is required. The GPU has yet another type of memory which is ideal for storing 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 kilobytes 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

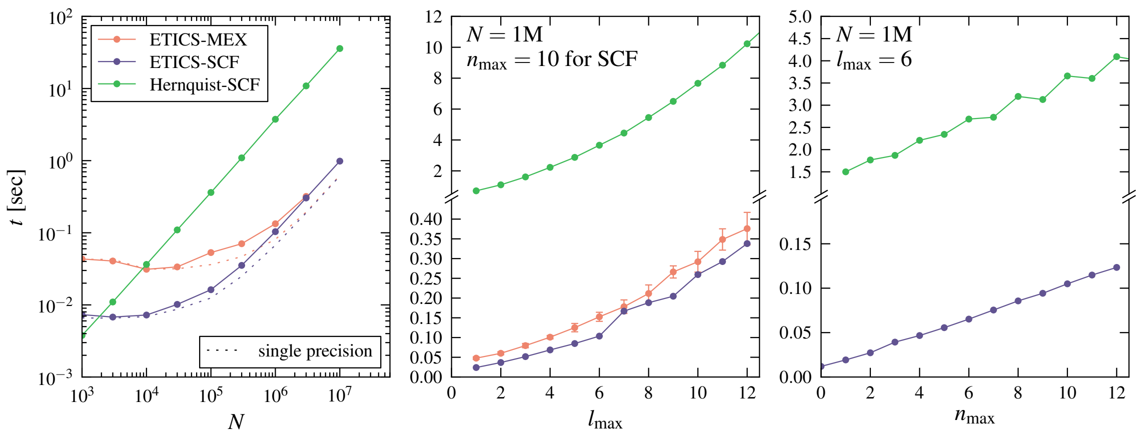


Figure IV.5: Scaling of one full force calculation time. Hernquist's SCF code (in green) is a CPU code and was tested on Intel Xeon E5520 CPU (one core). *ETICS* is a GPU code with both MEX (red) and SCF (blue) methods, and was tested with Nvidia Tesla K20 GPU; for the GPU codes, dotted lines show the performance in single-precision mode. For the scaling with N we set $l_{\max} = 6$, and for the SCF codes also $n_{\max} = 10$. 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.

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. Fig. IV.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 Fig. IV.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 done with C++ templates.

IV.3.4 Performance

We tested the performance of *ETICS* (both MEX and SCF) on a single Nvidia Tesla K20 GPUs 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 Intel Xeon E5520 CPU). If the initial conditions are not sorted by r in advance, the first MEX force calculation is more costly than all 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. Fig. IV.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 $l_{\max} \gtrsim 6$; only for which we 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 IV.4.2. The speedup factor could be very different for different GPUs.

All codes should scale quadratically with $l_{\max}$, but as the middle panel of Fig. IV.5 shows, this behavior is not so 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 $n_{\max}$. A strange behavior of the CPU code is noted: 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 it 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 help 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 hardware we tested, *ETICS*-GPU outperforms Hernquist's code by a factor of about 20 (which depends little on all parameters). However, Hernquist's code can utilize a multicore CPU (using MPI). The Xeon CPU we used has 4 cores, and two such CPUs are mounted on a single ACCRE node. We could use the Fortran code in MPI mode on all 8 effective cores with almost no overhead, and the calculation is accelerated by a factor of 8. Also, Hernquist's code calculate the jerk (force derivative), which *ETICS*-GPU does not; this takes ~ 14 percent of the total time.

Figs. IV.6 and IV.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 $N = 10^6$, $l_{\max} = 6$ and for SCF, $n_{\max} = 10$. 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 IV.3.2). The most costly operations are the ones we entrust to *Thrust*, namely the sorting and cumulative sum. In Fig. IV.7 the internal structure of

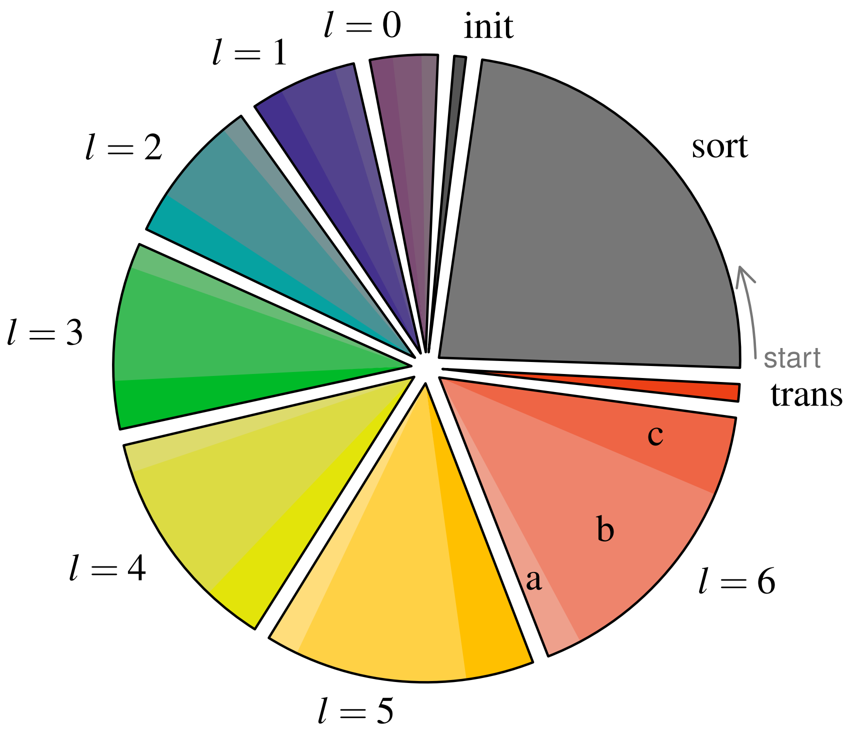


Figure IV.6: Pie chart showing the relative time of each operation required to perform one full MEX force calculation with *ETICS* (double-precision, $N = 10^6$ and $l_{\max} = 6$); the total time is 0.15 sec on Nvidia Tesla K20 GPU. The results may differ significantly on different hardware and if single-precision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

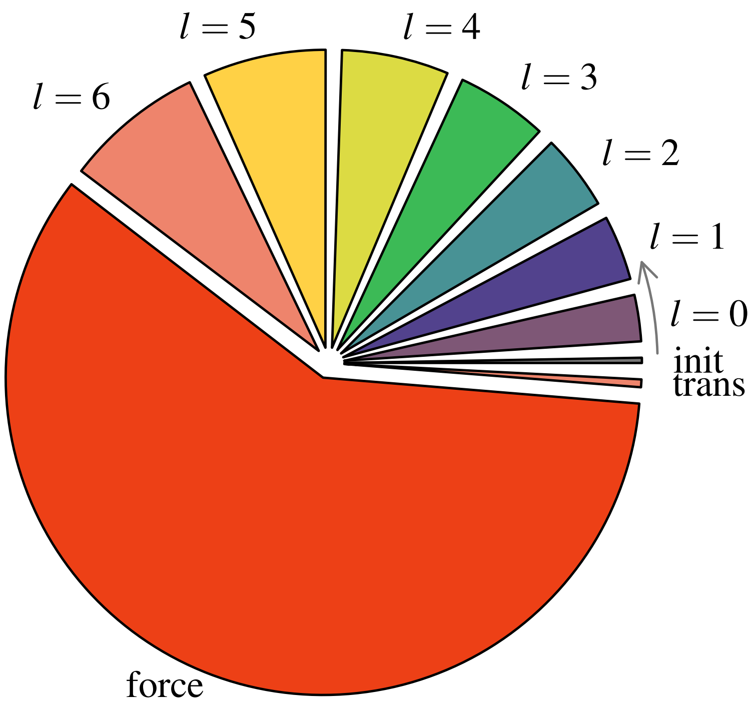


Figure IV.7: Same as Fig. IV.6 for a full SCF force calculation with *ETICS* ($N = 10^6$ particles, $l_{\text{max}} = 6$ and $n_{\text{max}} = 10$); the total time is 0.16 sec on Nvidia Tesla K20 GPU. The first operation is initialization, followed counter-clockwise by the l-loop (in which the n-loop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

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).

IV.4 Expansion Accuracy

IV.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 these questions depend not only on the expansion cutoff and N, but on the stellar distribution as well (e.g. global shape, central concentration, fractality, etc.); this will not be fully explored 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, this 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 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 particle 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 accuracy of SCF by showing 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 convergence of the coefficients does not give information about the force error. The bias and the 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 our models are Ferrers ellipsoids (Ferrers, 1877) (often called Ferrers bars) with index[4] $n = 1$: `model1` is a mildly oblate spheroid with axis ratio of 10:10:9, `model2` is triaxial with axis ratio of 3:2:1. Ferrers ellipsoids are often used in stellar dynamics, especially in the modeling of bars (e.g. Athanassoula 1992).

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

They have a very simple mass density:

$$\rho(\mu^2) = \begin{cases} \rho_0 \left[1-(\mu/a)^2\right]^n & \mu \leq a \\ 0 & \mu > a \end{cases} \tag{IV.20}$$

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

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

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 1D integral (Binney & Tremaine, 2008, Chapter 2.5). For both 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 realization of these two models, ranging from just 100 particles to 10^6. The goal is to compare for each realization the force calculated using MEX, SCF and direct-summation (no softening), with the real force. All calculations performed using double-precision, and the direct-summation force is not softened. For each realization we get a distribution of N values of the relative force error,

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

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

The left panel of Fig. IV.8 shows the mean relative force error $\bar{\varepsilon}$ 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 very small. For this model, $\bar{\varepsilon}$ is decreasing with N for all cases but $l_{\max} = 0$ (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 make 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 stacking of 10^3 realizations, so 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 1 000, in all cases the error distribution shifted down by just a factor of ~ 10.

Fig. IV.9 is the same but for the triaxial `model2`. While in the N-body cases the ε distributions are much the same, MEX shows a different behavior. The most prominent feature is the bump on 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 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-$l_{\max}$ cases would outperform $l_{\max} = 2$.

We repeat this exercise for SCF, which has an additional source of bias due to the radial expansion cutoff. Fig. IV.10 illustrates that 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 same angular cutoff ($l_{\max} = 10$). With increasing $n_{\max}$, the SCF error distribution approaches that of MEX, demonstrating the point made in Section IV.2.4 that MEX is equivalent to SCF with $n_{\max} \to \infty$. It must 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 $n_{\max} = 10$ 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 $0 \leq \gamma \leq 1$ 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, there is a large bias; if too many terms are used, the particle noise dominates. Vasiliev (2013) addressed this issue by calculating the variance of each SCF coefficient among several realizations of the same triaxial Dehnen model, found that for $N = 10^6$ particles, 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). Vasiliev (2014, in prep.) examined energy diffusion in a Plummer

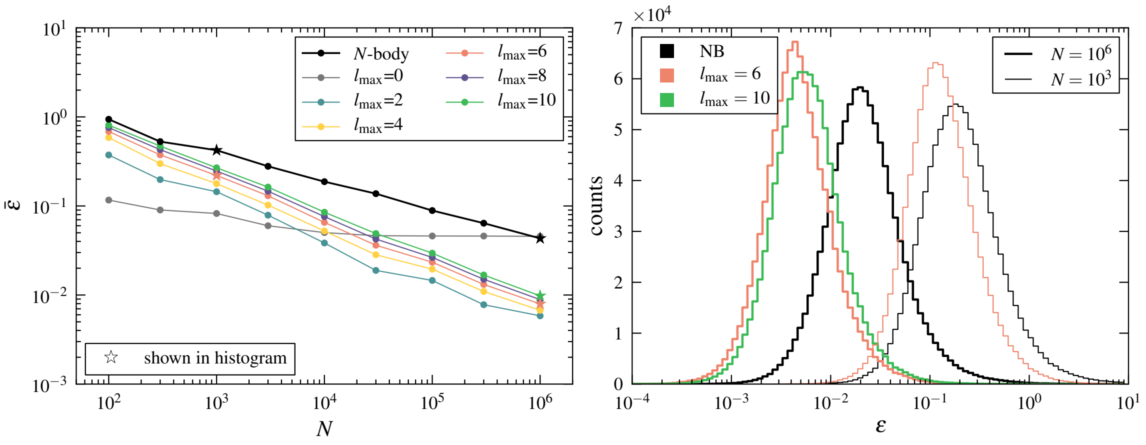


Figure IV.8: For the mildly oblate `model1`, the left panel shows, as a function of N, the mean relative force error ε (defined in equation IV.22) in direct-summation ("N-body"; thick black line) as well as MEX expansions with even $l_{\max}$ between 0 and 10 (odd terms have almost no effect). Each point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

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 of 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; this procedure however introduces a global energy errors unless some measures are taken to amend this.

IV.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

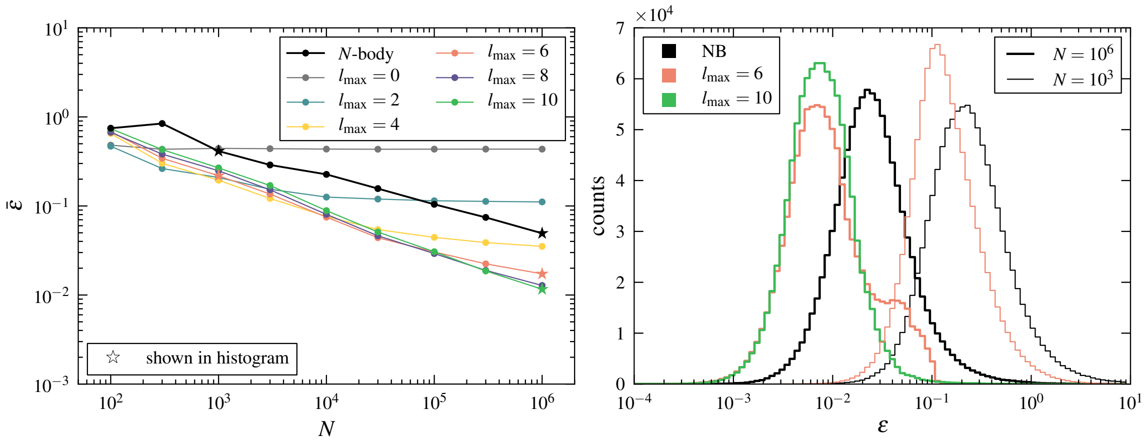


Figure IV.9: Same as Fig. IV.8 but for the triaxial `model2`. For this model, the behavior of the mean error seems different for the MEX (but nearly identical for the N-body): the low-multipoles outperform (have smaller mean error than) the higher ones only at low-N, but get saturated (increasing N does not improve accuracy). This could be understood from the right panel: while the error distributions are very similar to the previous model, the $l_{\max}=6$, $N=10^6$ 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 $l_{\max}=10$).

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.

Fig. IV.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:

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

We testes an $N=10^6$ realization of a Hernquist sphere with 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 IV.4.1, its nature is numeric and it could hinder energy conservation.

The relatively large error is not remedied by usual methods to improve accuracy of floating point arithmetic such as Kahan summation algorithm (Kahan, 1965), because the error does not come from 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 additionally they contribute erroneously to *all* the coefficient or

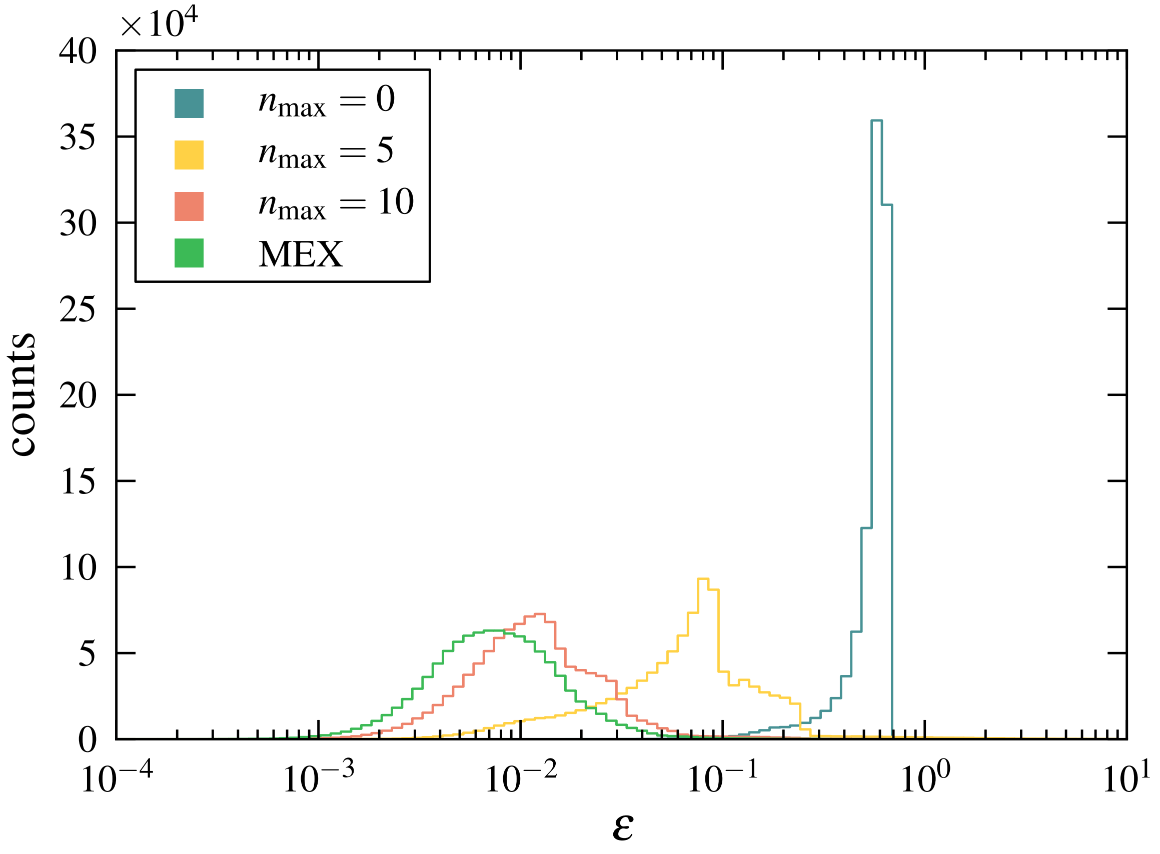


Figure IV.10: The relative force error distribution for the triaxial `model2` with $N = 10^6$. The green histogram is also shown in the right panel of Fig. IV.9 and represent a MEX expansion with $l_{\max} = 10$. The other histograms represent SCF expansions with $l_{\max} = 10$ and varying $n_{\max}$ 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.

multipoles, thus causing some error in the force calculation of all 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 well as their contribution to the coefficients or multipoles. The latter group causes large error because the recursion relation used to calculate the associated Legendre polynomials:

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

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 Fig. IV.11 may vary significantly depending on the model. For example, Ferrers ellipsoids are finite and flat at the center, 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 at 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.

IV.5 Discussion

IV.5.1 The Methodology

Expansion techniques, on their own, 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 has

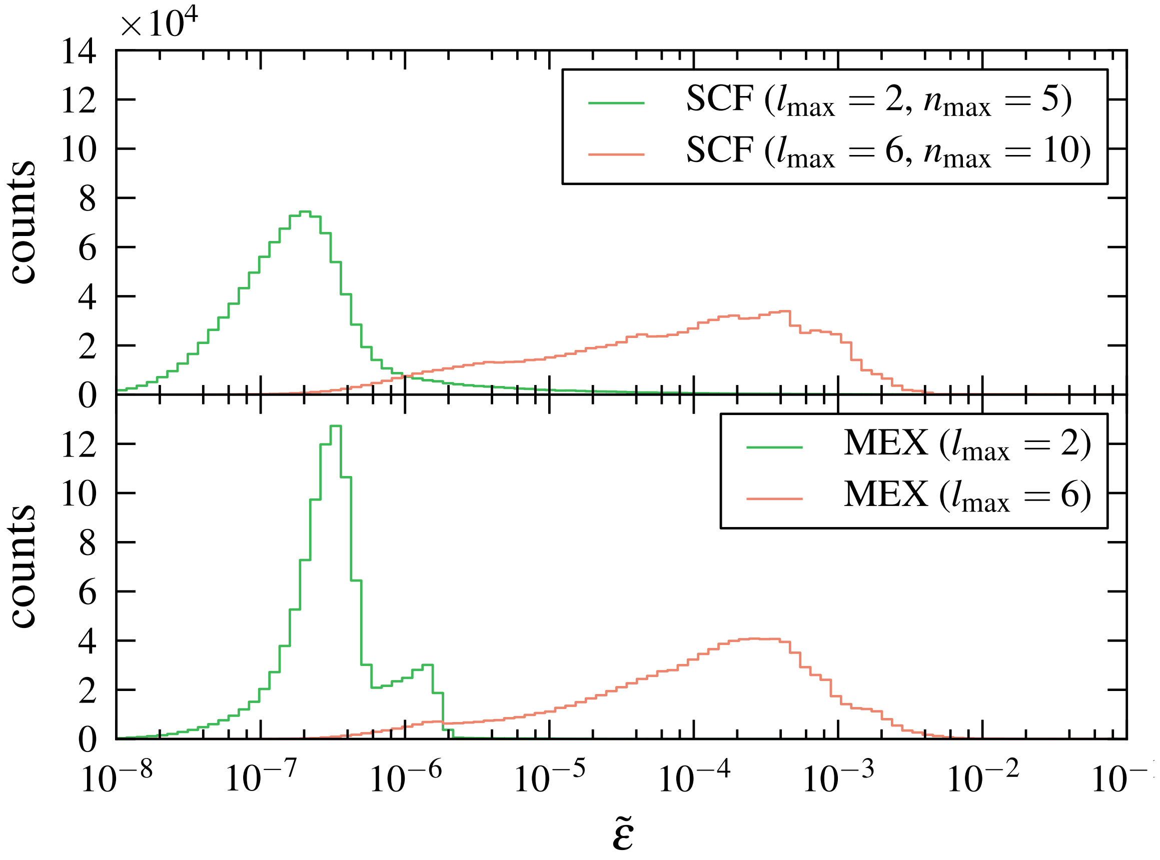


Figure IV.11: Relative force error distributions of single-precision force expansions compared to double-precision (defined in equation IV.23). The model used is a Hernquist sphere with $N = 10^6$. The top and bottom panels shows SCF and MEX force calculations, respectively. In both cases the left (green)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

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 scale length $a = 1$.

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 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 function we use are completely regular at the center).

IV.5.2 MEX

It is clear from the literature that SCF has been by far more popular. But despite the above, we do not think that most authors intentionally avoided MEX, and that 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 extends codes like MOCCA (Giersz et al., 2008) to non-spherical cases using our version of MEX[5]. 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 to 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 is by Aarseth (1967), who divided the simulation volume into thick shells, and the force on a particle was calculated by summing the multipoles of all shells except its own (own-shell correction was added). Similarly, Fry & Peebles (1980) used a MEX code with $l_{\max} = 3$ to explore galaxy correlation functions; in their version each shell had six particles, and softened Newtonian interaction was used within a shell. As noted in the introduction, 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 a follow up work (van Albada, 1982; Bontekoe & van Albada, 1987; Bertin & Stiavelli, 1989; Merritt & Stiavelli, 1990) the method was extended to 3D geometry. Finally, McGlynn (1984) used a grid in r only, with logarithmic spacing. He argues that softening sacrifices the higher resolution 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 (IV.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 (IV.15). This treatment was expanded to $l_{\max} = 5$ by Schneider et al. (2011).

IV.5.3 SCF

As noted in the previous section, SCF gained much more popularity than MEX. The SCF formalism has had wide use on galaxy-scale problems. It has been used to model the effect of black hole growth or adiabatic

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

contraction on the structure (density profile) of the dark matter halo (e.g. Sigurdsson 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, Weinberg, & Katz, 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 that need 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 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.

IV.5.4 Implementation

Our SCF implementation on GPU outperformed the serial Hernquist CPU version by a factor of ~ 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 $N \geq 10^8$, and to perform them fast enough so that many could be performed, exploring 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 IV.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*, by no means 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 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 if 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 the particle list to be sorted and a cumulative sum performed over the multiples, 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 higher order integrator instead of the leapfrog. Hernquist's SCF code already contains a 4th 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.

IV.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 system, 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).

APPENDIX

Here we show the Self Consistent Field code with post-Newtonian terms added, called mpiscf.f. Then following that is the head file tmhscf.h, input mode file scfmod, input parameter file scfpar and the compile file run.pl. To run the code, one still needs an initial condition file scfbi, which can be given by the users.

```
C*********************************************************************
C
C
                          PROGRAM tmhscf
C                          MPI -version
C
C*********************************************************************
C
C
C    A code to evolve self-gravitating systems using a self-consistent
C    field approach.  This version has been optimized for scalar workstations
C    and parallel computers.  The code is written in standard FORTRAN.
C
C    The computational system of units is determined by the input data.
C    No explicit assumtions about the value of the gravitational
C    constant have been made; it is read in as a parameter.
C    Particles are not required to have identical masses.
C
C
C                       Version 4: December1, 1995
C
C
C                    Steinn Sigurdsson: IoA, Cambridge
C                       From original by Lars Hernquist, UCSC
C
C    Apr 15 1997: option to multistep the SCF algorithm upgraded
C
```

```
C                   Chris Mihos, Johns Hopkins University
C
C  Jan 15, 1999 : various optimizations and conversion to MPI
C                   Bob Leary, San Diego Supercomputer Center
C
C
C       Feb, 2012 : Post-Newtonian up to 3.5 order added
C                   Baile Li, Vanderbilt University
C
C
C=======================================================================
C
C
C     This is the high-level evolution subroutine ascf.  Its tasks are:
C
C          1) to input parameters and the initial system state;
C          2) to perform a diagnostic analysis of the system at
C             each time step (energy, angular momentum, etc.);
C          3) initialise a-dot on first call
C          4) to provide an array of accelerations and a-dots
C           for a master hermite integrator.
C          5) to advance the SCF particles
C
C
C
C=======================================================================
C
C
C     Basic global variables/parameters:
C
C          ax,ay,az      : accelerations of bodies.
C          clm, dlm,     : radial functions used to evaluate expansions.
```

```
C           elm, flm
C           cputime     : cpu time (secs) used during the simulation.
C           cputime0    : cumulative cpu time at start of run.
C           cputime1    : cumulative cpu time at end of run.
C           dtime       : the timestep.
C           fixacc      : option to force conservation of linear
C                         momentum by setting acceleration of c.o.m.=0.
C           G           : the gravitational constant.
C           headline    : identification string for the run.
C           inptcoef    : option to read in expansion coefficients.
C           lmax        : number of angular eigenfunctions.
C           mass        : masses of bodies.
C           nbodies     : total number of bodies.
C           nbodsmax    : maximum number of bodies.
C           nmax        : number of radial eigenfunctions.
C           noutbod     : frequency of system record outputs.
C           noutlog     : frequency of outputs to log file.
C           nsteps      : number of time-steps to integrate the system.
C           one         : the constant 1.
C           onesixth    : the constant 1/6.
C           outpcoef    : option to write out expansion coefficients.
C           pi          : the constant pi.
C           pot         : potentials of bodies (self-gravity).
C           potext      : potentials of bodies (external field).
C           selfgrav    : option to turn off (.FALSE.) system self-
C                         gravity.
C           tnow        : current system time.
C           tpos        : current position time.
C           tvel        : current velocity time.
C           twoopi      : the constant 2./pi.
C           vx,vy,vz    : velocities of bodies.
C           x,y,z       : positions of bodies.
```

```
C          zeroeven     : option to zero out all even terms in the
C                         basis function expansions.
C          zeroodd      : option to zero out all odd terms in the
C                         basis function expansions.
C
C
C------------------------------------------------------------------------
C
C   Definitions specific to input/output.
C
C          uterm, upars, ulog, ubodsin,    : logical i/o unit numbers.
C            ubodsout,utermfil,uoutcoef,
C            uincoef,ubodsel
C          parsfile, logfile, ibodfile,    : character names of files.
C            obodfile,termfile,outcfile,
C            incfile,elfile
C
C------------------------------------------------------------------------
C
C   Definitions specific to timing MPI code
C
C          t0mpi,t1mpi,t2mpi,tmpi - real*8 timing variables
C========================================================================
        include 'mpif.h'
        INCLUDE 'tmhscf.h'


C   Declaration of local variables.
C   --------------------------------

        INTEGER n
        LOGICAL firstc
```

```
      DATA firstc/.TRUE./

      real*8 t0mpi,t1mpi,t2mpi,tmpi
      SAVE firstc,n

C=======================================================================
C   Initialize state of the system.
C   ---------------------------------
     call initmpi
     t0mpi=mpi_wtime()
C
      IF(firstc) THEN

        firstc=.FALSE.
        n=0
        CALL initsys(n)
C                ------
      ENDIF
     t1mpi=mpi_wtime()
     tmpi=t1mpi-t0mpi
     if(me.eq.0)then
       write(utermfil,*) 'initsys time = ',tmpi,' on proc ',me

     end if

C
C   Advance system state
C   --------------------
C
 101  IF (fixedn) THEN
        DO 100 n=1,nsteps
      if(me.eq.0)then
         if(mod(n,100).eq.0)then
         write(utermfil,*) 'running_step', n
         endif
      endif
            CALL stepsys(n)
C                   ————

 100         CONTINUE
          ELSE IF (tnow.lt.tfinal) THEN
             n=n+1
c           write(*,*)'tnow,n',tnow,n
             CALL stepsys(n)
C                 ————
             GOTO 101

          ENDIF

C   Terminate the simulation.
C   ———————————————

      t2mpi=mpi_wtime()
      tmpi=t2mpi-t1mpi



      call mpi_finalize(ier)
          STOP
          END
*****************************************************************************
C
C
          SUBROUTINE accp_bh(n)
```

```
C
C
C*****************************************************************
C
C
C      Subroutine to compute accelerations, .adot for a BH.
C
C
C=================================================================

          INCLUDE 'tmhscf.h'
          include 'mpif.h'

C    Declaration of local variables.
C    ---------------------------------

          LOGICAL firstc
          INTEGER n,p
          REAL*8 del,acci,del3,bhold,vrd,dtp,
     &bholdsmall,bholdbig,daxbhf,daybhf,dazbhf,r

          DATA firstc/.TRUE./
          SAVE firstc,bholdsmall,bholdbig
c         initialized dabh by Baile
          axbh=0.0
          aybh=0.0
          azbh=0.0
          daxbh=0.0
          daybh=0.0
          dazbh=0.0

C=================================================================
          IF (tnow.lt.tstartbh) THEN
            firstc=.TRUE.
           RETURN
          ENDIF
          IF (tnow.ge.tstartbh) THEN
             IF (firstc) THEN
                bholdbig=bhmasst
                bholdsmall=bhmasst
                firstc=.FALSE.
             ENDIF
          ENDIF

          IF((tnow-tstartbh.GE.0.0) .AND. (tnow-tstartbh.LT.tgrowbh))
     &       fac=bhmass*(3.*((tnow-tstartbh)/tgrowbh)**2 -
     &                       2.*((tnow-tstartbh)/tgrowbh)**3)

          IF((tnow-tstartbh.GT.tlivebh+tgrowbh) .AND.
     &       (tnow-tstartbh.LE.tdiebh+tlivebh+tgrowbh))
     &      fac=bhmass*(1.-3.*((tnow-tstartbh-tlivebh-tgrowbh)/tdiebh)**2+
     &                      2.*((tnow-tstartbh-tlivebh-tgrowbh)/tdiebh)**3)

          IF(tnow-tstartbh.GT.tdiebh+tlivebh+tgrowbh) fac=0.0

          IF (tgrowbh.LT.0) THEN
             fac=bhmass
             bholdbig=bhmass
             bholdsmall=bhmass
          ENDIF

          fac=fac*G
          DO 100 p=1,nbodies

          IF ((mstpflag.AND.(im(p).eq.rtag)).or. n .eq. 0) then
            dtp=dtbin(rtag)

c         bhold=bholdsmall*(1-im(p))+bholdbig*im(p)
c       write(*,*)'bhold',bhold
c         dtp=dtime*(1-im(p))+dtbig*im(p)
c       write(*,*)'dtp,dtbin(rtag),rtag',dtp,dtbin(rtag),rtag

c       del=SQRT((x(p)-xbh)**2+(y(p)-ybh)**2+(z(p)-zbh)**2)+epsbh
 888       del=sqrt(x(p)**2+y(p)**2+z(p)**2+epsbh**2)
c       write(*,*)'p,x,y,z',p,x(p),y(p),z(p)
          del3 = 1./(del*del*del)
             potext(p) = - fac/del

          call   rel(vx(p),vy(p),vz(p),vxbh,vybh,vzbh,
     &v1jx(p),v1jy(p),v1jz(p))
          rlength(p)=r(xbh,ybh,zbh,x(p),y(p),z(p))+epsbh
          vlength(p)=sqrt(v1jx(p)**2+v1jy(p)**2+v1jz(p)**2)

          E0(p)=(1./2.*vlength(p)**2-1./rlength(p))*mass(p)
c     write(*,*)1./2.*vlength(p)**2,1./rlength(p)
c     write(*,*)E0(p)
c   here E0 does not multiply miu, will do later
             acci=fac*del3
c       write(*,*)'p,del3',p,del3
c         write(*,*)'ax,ay,az',ax(p),ay(p),az(p)
             ax(p)=ax(p)-x(p)*acci
             ay(p)=ay(p)-y(p)*acci
             az(p)=az(p)-z(p)*acci
c       write(*,*)'accpbh,p,ax,x*acci,acci',p,ax(p),x(p)*acci,acci
```

```
c          antx(p)=ax(p)
c          anty(p)=ay(p)
c          antz(p)=az(p)

           axbh=axbh+x(p)*acci/fac*G*mass(p)
           aybh=aybh+y(p)*acci/fac*G*mass(p)
           azbh=azbh+z(p)*acci/fac*G*mass(p)



           daxbhf = -vx(p)*acci
           daybhf = -vy(p)*acci
           dazbhf = -vz(p)*acci
c         write(*,*)'p,daxbhf,vx,acci',p,daxbhf,vx(p),acci
           daxbhf = daxbhf - x(p)*acci*(fac - bhold)/dtp
           daybhf = daybhf - y(p)*acci*(fac - bhold)/dtp
           dazbhf = dazbhf - z(p)*acci*(fac - bhold)/dtp
c         write(*,*)'p,daxbhf,acci,fac,bhold,dtp',
c     &p,daxbhf,acci,fac,bhold,dtp
c          write(*,*)'p,daxbhf,daybhf,dazbhf',p,daxbhf,daybhf,dazbhf
           vrd = (vx(p)*x(p)+vy(p)*y(p)+vz(p)*z(p))/(del*del)
c         write(*,*)
           daxbhf = daxbhf + 3.*x(p)*vrd*acci
           daybhf = daybhf + 3.*y(p)*vrd*acci
           dazbhf = dazbhf + 3.*z(p)*vrd*acci
c         write(*,*)'p,dax,day,daz',p,dax(p),day(p),daz(p)
           dax(p)=dax(p)+daxbhf
           day(p)=day(p)+daybhf
           daz(p)=daz(p)+dazbhf
c         write(*,*)'p,ax,ay,az,dax,day,daz',p,ax(p),ay(p),az(p),
c     &dax(p),day(p),daz(p)
```

```
          daxbh=daxbh+daxbhf/fac*G*mass(p)
          daybh=daybh+daybhf/fac*G*mass(p)
          dazbh=dazbh+dazbhf/fac*G*mass(p)
        end if
 100    CONTINUE




        if(firstpn) call pn1(n) !added by Baile
        if(secondpn) call pn2(n)
        if(secondhpn) call pn2h(n)
        if(thirdpn) call pn3(n)
        if(thirdhpn) call pn3h(n)


        IF (.not.multistep.OR..NOT.mstpflag) THEN
          bholdbig=fac
        ELSE
          bholdsmall=fac
        ENDIF

        RETURN
        END
C
C***********************************************************************
C
C
        SUBROUTINE accp_LH(nstep)
C
C
C***********************************************************************
```

```
C
C
C      Subroutine to compute accelerations, potential, and density.
C
C
C=====================================================================

         INCLUDE 'tmhscf.h'
         INCLUDE 'mpif.h'
         INTEGER k,l,m,n,lmin,lskip,nstep
         LOGICAL firstc
         REAL*8 anltilde,knl,sinth,sinmphi,cosmphi,phinltil,deltam0,
     &          gammln,arggam,sinsum,cossum,coeflm,factrl,
     &          dblfact,ttemp5,ar,ath,aphi,temp2,temp3,temp4,
     &          temp5,temp6,plm,dplm,ultrasp,ultrasp1,ultraspt,clm,
     &          dlm,elm,flm,xi,costh,phi,r,twoalpha,c1,c2,c3,un,unm1,
     &          plm1m,plm2m,rhonltil,alm,blm,cosmphi1,sinmphi1
        real*8 lmm,lplusm,ratio

c note tmpsum must be large enough to hold sinsum1,cossum1,sinsum2,cossum2
c each array is of the form sinsum1(0:nmax,0:lmax,0:lmax)
c with element sinsum1(n,l,m).  But for a given l, m ranges from 0:l.

ccompaq     tmpsum replaced with tmpsum_snd and tmpsum_rcv so that distinct
ccompaq     buffers are used in MPI_Allreduce. Change courtesy of Compaq

         REAL*8 tmpsum_snd((nmax+1)*(lmax+1)*(lmax+2)),
     &          tmpsum_rcv((nmax+1)*(lmax+1)*(lmax+2))
         DIMENSION ultrasp(0:nmax,0:lmax),
     &             ultraspt(0:nmax,0:lmax),ultrasp1(0:nmax,0:lmax),
     &             anltilde(0:nmax,0:lmax),dblfact(lmax+1),
     &             coeflm(0:lmax,0:lmax),sinsum(0:nmax,0:lmax,0:lmax),
     &                cossum(0:nmax,0:lmax,0:lmax),
     &                twoalpha(0:lmax),c1(1:nmax,0:lmax),c2(1:nmax,0:lmax),
     &                c3(1:nmax),cosmphi(0:lmax),sinmphi(0:lmax),
     &                plm(0:lmax,0:lmax),dplm(0:lmax,0:lmax),
     &                knl(0:nmax,0:lmax),twolp1(0:lmax),twolm1(0:lmax),
     &                lplusm(0:lmax,0:lmax),lmm(0:lmax,0:lmax)
        DATA firstc/.TRUE./

        SAVE firstc,dblfact,anltilde,coeflm,lmin,
     &       lskip,twoalpha,c1,c2,c3,knl,twolp1,twolm1,
     &       lmm,lplusm


crss    New declarations (RSS)

        real*8 rxy2,rxyinv,arxyinv(nbodsper)
        real*8 rxyz2,rxyzinv,arxyzinv(nbodsper)


C=============================================================================

        IF(firstc) THEN

           firstc=.FALSE.

           dblfact(1)=1.

           DO 5 l=2,lmax
              dblfact(l)=dblfact(l-1)*(2.*l-1.)
 5         CONTINUE

           DO 20 n=0,nmax
```

```
              DO 10  l=0,lmax
                 knl(n,l)=0.5*n*(n+4.*l+3.)+(l+1.)*(2.*l+1.)
                 anltilde(n,l)=-2.**(8.*l+6.)*FACTRL(n)*(n+2.*l+1.5)
                 arggam=2.*l+1.5
                 anltilde(n,l)=anltilde(n,l)*(EXP(GAMMLN(arggam)))**2
            anltilde(n,l)=anltilde(n,l)/(4.*pi*knl(n,l)*FACTRL(n+4*l+2))
10            CONTINUE
20          CONTINUE

            DO 25  l=0,lmax


              twoalpha(l)=2.0*(2.*l+1.5)
              twolp1(l)=2.*l+1.
              twolm1(l)=2.*l-1.


              DO 23  m=0,l
                 deltam0=2.
                 IF(m.EQ.0)  deltam0=1.
                 coeflm(l,m)=(2.*l+1.)*deltam0*FACTRL(l-m)/FACTRL(l+m)
                 lmm(l,m)=l-m
      if(l.ne.m)  lmm(l,m)=1./lmm(l,m)
                 lplusm(l,m)=l+m-1.
23            CONTINUE
25          CONTINUE

            DO 30  n=1,nmax
              c3(n)=1.0/(n+1.0)


              DO 27  l=0,lmax
                 c1(n,l)=2.0*n+twoalpha(l)
                 c2(n,l)=n-1.0+twoalpha(l)
27            CONTINUE
```

```
 30         CONTINUE

      ENDIF

      lskip=1
      IF(zeroodd .OR. zeroeven) lskip=2
      lmin=0
      IF(zeroeven) lmin=1



      DO 60 l=0,lmax
         DO 50 m=0,l
            DO 40 n=0,nmax
               sinsum(n,l,m)=0.0
               cossum(n,l,m)=0.0
               sinsum1(n,l,m)=0.0
               cossum1(n,l,m)=0.0
               IF (.NOT. mstpflag) THEN
                 sinsum2(n,l,m)=0.0
                 cossum2(n,l,m)=0.0
               ENDIF
 40         CONTINUE
 50      CONTINUE
 60   CONTINUE

      DO 120 k=1,nbodies

      IF((mstpflag.AND.im(k).EQ.rtag) .or. nstep .eq. 0) THEN
c       write(*,*)'accp_LH,k,im(k)',k,im(k)
crss    Following lines added for more efficient calculation of
crss    r, sin(theta), cos(theta), sin(phi), and cos(phi)
crss      Intermediate results saved in arrays arxyinv and arxyzinv

            rxyz2 = x(k)*x(k) + y(k)*y(k) + z(k)*z(k)
            rxy2 =  x(k)*x(k) + y(k)*y(k)
            arxyzinv(k) = 1.0/sqrt(rxyz2)
            arxyinv(k)  = 1.0/sqrt(rxy2)
            rxyzinv = arxyzinv(k)
            rxyinv = arxyinv(k)
            r = rxyz2 * rxyzinv
            costh=z(k)*rxyzinv
            sinth=rxy2*rxyinv*rxyzinv
            cosmphi(1) = x(k)*rxyinv
            sinmphi(1) = y(k)*rxyinv

crss      End new code

crss      Following lines commented out and replaced by more efficient
crss      calculations above
crss              r=SQRT(x(k)**2+y(k)**2+z(k)**2)
crss              costh=z(k)/r
crss              sinth=SQRT(1.-costh*costh)
crss              phi=ATAN2(y(k),x(k))
crss              cosmphi(1)=COS(phi)
crss              sinmphi(1)=SIN(phi)

            xi=(r-1.)/(r+1.)
            cosmphi(0)=1.
            sinmphi(0)=0.
            DO 105 m=2,lmax
                cosmphi(m)=cosmphi(1)*cosmphi(m-1)-sinmphi(1)*sinmphi(m-1)
                sinmphi(m)=cosmphi(1)*sinmphi(m-1)+sinmphi(1)*cosmphi(m-1)
 105        CONTINUE
```

```
          DO 113 l=0,lmax

            ultrasp(0,l)=1.0
            ultrasp(1,l)=twoalpha(l)*xi

            un=ultrasp(1,l)
            unm1=1.0

            DO 111 n=1,nmax-1
              ultrasp(n+1,l)=(c1(n,l)*xi*un-c2(n,l)*unm1)*c3(n)
              unm1=un
              un=ultrasp(n+1,l)
 111        CONTINUE

crss   Calculation of ultraspt commented out. Multiplication by anltilde
crss   now performed outside of loop over particles
crss            DO 112 n=0,nmax
crss              ultraspt(n,l)=ultrasp(n,l)*anltilde(n,l)
crss 112        CONTINUE

 113      CONTINUE

        ratio=1.

crss   Calculation of plm modified so as to avoid unnecessary logical
crss   tests for m.eq.0 inside loop over l, special case m=0 split off

          m = 0
          plm(m,m)=1.0
          plm1m=plm(m,m)
          plm2m=0.0
          DO l=m+1,lmax
```

```
                plm(l,m)=(costh*twolm1(l)*plm1m-
     %                   lplusm(l,m)*plm2m)*lmm(l,m)
                plm2m=plm1m
                plm1m=plm(l,m)
             enddo

             DO m=1,lmax
                plm(m,m)=1.0
                ratio=-sinth*ratio
                plm(m,m)=dblfact(m)*ratio
                plm1m=plm(m,m)
                plm2m=0.0
                DO l=m+1,lmax
                   plm(l,m)=(costh*twolm1(l)*plm1m-
     %                      lplusm(l,m)*plm2m)*lmm(l,m)
                   plm2m=plm1m
                   plm1m=plm(l,m)
                enddo
             enddo

crss      End modified loops for plm calculation

             if(lmin.eq.0) then
             temp5=mass(k)/(1.+r)
             else
             temp5=mass(k)*r/(1.+r)**3
             endif
             ratio=r/(1.+r)**2
             if(lskip.eq.2) ratio=ratio*ratio

crss      Loops used in accumulation of sinsum1, sinsum2, cossum1, cosssum2
crss      modified so that:
```

```
crss     (i)   multiplication by coeflm moved outside loop over particles
crss     (ii)  logical test on im(k) moved outside nested loops
crss     (iii) ultraspt changed to ultrasp

           if((im(k).eq.rtag) .or. nstep .eq. 0) then
             DO l=lmin,lmax,lskip
               DO m=0,l
                 ttemp5=temp5*plm(l,m)
                 temp3=ttemp5*sinmphi(m)
                 temp4=ttemp5*cosmphi(m)
                 DO n=0,nmax
                   sinsuml(n,l,m)=sinsuml(n,l,m) +
     &                    temp3*ultrasp(n,l)
                   cossuml(n,l,m)=cossuml(n,l,m) +
     &                    temp4*ultrasp(n,l)
                 enddo
               enddo
               temp5=temp5*ratio
             enddo
           endif



crss      End modified loops for accumulating sinsuml, sinsum2, etc.
         end if
 120     CONTINUE

crss      Following loops added to multiply sinsuml, sinsum2, etc.
crss      by loop invariants anltilde and coeflm. These operations
crss      were formerly done inside loop over particles

         do l=0,lmax
```

```
              do m=0,lmax
                 do n=0,nmax
                    sinsum1(n,l,m)=sinsum1(n,l,m)*anltilde(n,l)*coeflm(l,m)
                    sinsum2(n,l,m)=sinsum2(n,l,m)*anltilde(n,l)*coeflm(l,m)
                    cossum1(n,l,m)=cossum1(n,l,m)*anltilde(n,l)*coeflm(l,m)
                    cossum2(n,l,m)=cossum2(n,l,m)*anltilde(n,l)*coeflm(l,m)
c                write(*,*)'n,l,m,sinsum1(n,l,m),cossum1(n,l,m),
c␣␣␣␣␣&anltilde(n,l),coeflm(l,m)',n,l,m,sinsum1(n,l,m),cossum1(n,l,m),
c     &anltilde(n,l),coeflm(l,m)
                enddo
              enddo
            enddo

crss    End modified loops

            icount=0
            DO 1181 l=lmin,lmax,lskip
              DO 1161 m=0,l
                DO 1141 n=0,nmax
                icount=icount+2

ccompaq      tmpsum replaced with tmpsum_snd

                 tmpsum_snd(icount-1) = sinsum1(n,l,m)
                 tmpsum_snd(icount) = cossum1(n,l,m)

c note in this version tmpsum(icount-1) = sinsum,
c and tmpsum(icount) = cossum are globally summed,
c rather than globally summing sinsum1. sinsum2, cossum1, and
c cossum2 and then computing sinsum=sinsum1+sinsum2,
c cossum=cossum1+cossum2 as was previously done.  This saves
c half the message length in the non-multistepping case.
```

```
                if (.NOT. mstpflag) then

ccompaq      tmpsum replaced with tmpsum_snd

                  tmpsum_snd(icount-1) = tmpsum_snd(icount-1)
     &                                    + sinsum2(n,l,m)
                  tmpsum_snd(icount) = tmpsum_snd(icount)
     &                                  + cossum2(n,l,m)
                endif

 1141        CONTINUE
 1161      CONTINUE
 1181    CONTINUE

ccompaq      tmpsum replaced with tmpsum_snd and tmpsum_rcv in MPI call

        call MPI_Allreduce (tmpsum_snd,tmpsum_rcv, icount,
     &                      mpi_double_precision,
     &                      mpi_sum, mpi_comm_world,ierr)
         icount=0
         DO 1182 l=lmin,lmax,lskip
           DO 1162 m=0,l
             DO 1142 n=0,nmax
               icount=icount+2

c see note in DO 1181 loop - we recover sinsum directly rather
c than sinsum1 and sinsum2 individually; similarly
c for cossum.
ccompaq      tmpsum replaced with tmpsum_rcv

                sinsum(n,l,m) = tmpsum_rcv(icount-1)
```

```
                 cossum(n,l,m) = tmpsum_rcv(icount)
c          write(*,*)'k,n,l,m,sinsum(n,l,m),cossum(n,l,m)',
c      &k,n,l,m,sinsum(n,l,m),cossum(n,l,m)
 1142         CONTINUE
 1162       CONTINUE
 1182    CONTINUE

         IF(MOD(nstep,noutlog).EQ.0.AND..NOT.mstpflag) THEN
          IF(outpcoef) CALL iocoef(sinsum,cossum)
C                                         -------
         ENDIF
         IF(inptcoef) CALL iocoef(sinsum,cossum)

         DO 200 k=1,nbodies

         IF((mstpflag.AND.im(k).EQ.rtag ).or. nstep .eq. 0) THEN
c         write(*,*)'houbu,k,im(k)',k,im(k)
crss      Following lines added for more efficient calculation of
crss      r, sin(theta), cos(theta), sin(phi), and cos(phi)

            rxyz2 = x(k)*x(k) + y(k)*y(k) + z(k)*z(k)
            rxy2 =  x(k)*x(k) + y(k)*y(k)
            rxyzinv = arxyzinv(k)
            rxyinv  = arxyinv(k)
            r = rxyz2 * rxyzinv
            costh=z(k)*rxyzinv
            sinth=rxy2*rxyinv*rxyzinv
            cosmphi1 = x(k)*rxyinv
            sinmphi1 = y(k)*rxyinv

crss      End new code
crss      Following lines commented out and replaced by more efficient
crss      calculations above
crss            r=SQRT(x(k)**2+y(k)**2+z(k)**2)
crss            costh=z(k)/r
crss            sinth=SQRT(1.-costh*costh)
crss            phi=ATAN2(y(k),x(k))
crss            cosmphi1=COS(phi)
crss            sinmphi1=SIN(phi)

           xi=(r-1.)/(r+1.)

           cosmphi(0)=1.0
           sinmphi(0)=0.
           cosmphi(1)=cosmphi1
           sinmphi(1)=sinmphi1

           DO 130 m=2,lmax
              cosmphi(m)=cosmphi1*cosmphi(m-1)-sinmphi1*sinmphi(m-1)
              sinmphi(m)=cosmphi1*sinmphi(m-1)+sinmphi1*cosmphi(m-1)
 130       CONTINUE
           adens(k)=0.0d0
           pot(k)=0.0d0
           ar=0.0d0
           ath=0.0d0
           aphi=0.0d0

           DO 148 l=0,lmax

              ultrasp(0,l)=1.0
              ultrasp(1,l)=twoalpha(l)*xi
              ultrasp1(0,l)=0.0
              ultrasp1(1,l)=1.0
```

```
          un=ultrasp(1,l)
          unm1=1.0

          DO 144 n=1,nmax-1
            ultrasp(n+1,l)=(c1(n,l)*xi*un-c2(n,l)*unm1)*c3(n)
            unm1=un
            un=ultrasp(n+1,l)
            ultrasp1(n+1,l)=((twoalpha(l)+n)*unm1-(n+1)*xi*
     &                  ultrasp(n+1,l))/(twoalpha(l)*(1.-xi*xi))
 144      CONTINUE

 148    CONTINUE

      ratio=1.

crss    Calculation of plm modified so as to avoid unnecessary logical
crss    tests for m.eq.0 inside loop over l, special case m=0 split off

        m = 0
        plm(m,m)=1.0
        plm1m=plm(m,m)
        plm2m=0.0
        DO l=m+1,lmax
          plm(l,m)=(costh*twolm1(l)*plm1m-
     %          lplusm(l,m)*plm2m)*lmm(l,m)
          plm2m=plm1m
          plm1m=plm(l,m)
        enddo

        DO m=1,lmax
          plm(m,m)=1.0
```

```
              ratio=-sinth*ratio
              plm(m,m)=dblfact(m)*ratio
              plm1m=plm(m,m)
              plm2m=0.0
              DO l=m+1,lmax
                 plm(l,m)=(costh*twolm1(l)*plm1m-
     %                 lplusm(l,m)*plm2m)*lmm(l,m)
                 plm2m=plm1m
                 plm1m=plm(l,m)
              enddo
           enddo
crss    End modified loops for plm calculation

           dplm(0,0)=0.0

crss    Calculation of dplm modified so as to avoid unnecessary logical
crss    tests for l.eq.m inside loop over m, special case l=m split off

           DO l=1,lmax
              DO m=0,l-1
                 dplm(l,m)=(l*costh*plm(l,m)-(l+m)*plm(l-1,m))/
     &                 (costh*costh-1.0)
              enddo
              dplm(l,l)=l*costh*plm(l,m)/(costh*costh-1.0)
           enddo

crss    End modified loops for dplm calculation

           if(lmin.eq.0) then
           phinltil = 1./(1+r)
           else
           phinltil=r/(1.+r**3)
```

```
            endif
            ratio=r/(1.+r)**2
            if(lskip.eq.2) ratio=ratio*ratio
            DO 190 l=lmin,lmax,lskip

               temp2=0.0
               temp3=0.0
               temp4=0.0
               temp5=0.0
               temp6=0.0

               DO 180 m=0,l

                  alm=0.0
                  blm=0.0
                  clm=0.0
                  dlm=0.0
                  elm=0.0
                  flm=0.0

                  DO 150 n=0,nmax
                     alm=alm+knl(n,l)*ultrasp(n,l)*cossum(n,l,m)
                     blm=blm+knl(n,l)*ultrasp(n,l)*sinsum(n,l,m)
                     clm=clm+ultrasp(n,l)*cossum(n,l,m)
                     dlm=dlm+ultrasp(n,l)*sinsum(n,l,m)
                     elm=elm+ultrasp1(n,l)*cossum(n,l,m)
                     flm=flm+ultrasp1(n,l)*sinsum(n,l,m)
c           write(*,*)'k,im(k),n,l,m,sinsum(n,l,m),cossum(n,l,m)',
c     &k,im(k),n,l,m,sinsum(n,l,m),cossum(n,l,m)
 150              CONTINUE
                  temp2=temp2+plm(l,m)*(alm*cosmphi(m)+blm*sinmphi(m))
                  temp3=temp3+plm(l,m)*(clm*cosmphi(m)+dlm*sinmphi(m))
```

```
                     temp4=temp4-plm(l,m)*(elm*cosmphi(m)+flm*sinmphi(m))
                     temp5=temp5-dplm(l,m)*(clm*cosmphi(m)+dlm*sinmphi(m))
                     temp6=temp6-m*plm(l,m)*(dlm*cosmphi(m)-clm*sinmphi(m))
c           write(*,*)'k,n,l,m,alm,blm,clm,dlm,sinmphi(m),cosmphi(m)',
c        &k,n,l,m,alm,blm,clm,dlm,sinmphi(m),cosmphi(m)
180              CONTINUE


                   rhonltil=phinltil/(r*(1.+r)*(1.+r)*2.0d0*pi)
CC
                   pot(k)=pot(k)+temp3*phinltil
c           write(*,*)'k,temp2,temp3',k,temp2,temp3
CC
CC
                   adens(k)=adens(k)-temp2*rhonltil
c           write(*,*)'k,adens(k),temp2,rhonltil',k,adens(k),temp2,rhonltil
CC
                   ar=ar+phinltil*(-temp3*(l/r-twolp1(l)/
      &                    (1.+r))+temp4*2.*twoalpha(l)/(1.+r)**2)
                   ath=ath+temp5*phinltil
                   aphi=aphi+temp6*phinltil
                 phinltil = phinltil*ratio
 190            CONTINUE

                 ath= -sinth*ath/r
                 aphi=aphi/(r*sinth)
                 ax(k)=G*(sinth*cosmphi1*ar+costh*cosmphi1*ath-
      &                sinmphi1*aphi)
                 ay(k)=G*(sinth*sinmphi1*ar+costh*sinmphi1*ath+
      &                cosmphi1*aphi)
                 az(k)=G*(costh*ar-sinth*ath)
                 pot(k)=pot(k)*G
```

```
c          write(*,*)'k,im(k),adens',k,im(k),adens(k)
C
CC
C     NB no convective derivative!
CC
            dax(k)=-4.*pi*G*adens(k)*vx(k)
            day(k)=-4.*pi*G*adens(k)*vy(k)
            daz(k)=-4.*pi*G*adens(k)*vz(k)
cccccccccccccc rescale ax dax pot
          ax(k)=ax(k)*(2.2751e-7**2)
          ay(k)=ay(k)*(2.2751e-7**2)
          az(k)=az(k)*(2.2751e-7**2)
          pot(k)=pot(k)*2.2751e-7
          dax(k)=dax(k)*(2.2751e-7**2)
          day(k)=day(k)*(2.2751e-7**2)
          daz(k)=daz(k)*(2.2751e-7**2)

c            write(*,*)'k,dax,day,daz',k,dax(k),day(k),daz(k)
c****************************************
          END IF
 200      CONTINUE

          RETURN
          END
C**********************************************************************
C
C
          SUBROUTINE accpot(n)
C
C
C**********************************************************************
```

```
C
C
C     Subroutine to compute accelerations, potential, and density.
C
C
C=========================================================================

      INCLUDE 'tmhscf.h'
      include 'mpif.h'

      INTEGER n,i

C=========================================================================
c     call rtidal

c     if (me.eq.0) then
c     open(278,file='number',status='unknown')
c     write(278,*) 'nbodies,n_pes,stat0'
c     write(278,*) nbodies,n_pes,star0
c     end if
c     close(278)


       IF(selfgrav) THEN
C
c     do i=1,nbodies
c     if(starlive(i).eq.1) mass(i)=1.0/(nbodies*n_pes-star0)
c     end do

      do i=1,nbodies
      x(i)=x(i)*2.2751e-7
      y(i)=y(i)*2.2751e-7
      z(i)=z(i)*2.2751e-7
      end do

        CALL accp_LH(n)
      do i=1,nbodies
      x(i)=x(i)/2.2751e-7
      y(i)=y(i)/2.2751e-7
      z(i)=z(i)/2.2751e-7
      end do



c     do i=1,nbodies
c     if(starlive(i).eq.1) mass(i)=1.0/(nbodies*n_pes)
c      end do



C               -------
      ENDIF
C=====================================================================
C
      IF(bhgrav) CALL accp_bh(n)
C                         ----------
C
C=====================================================================
C
      RETURN
      END
C*********************************************************************
C
C
                  SUBROUTINE checkinp
C
```

```
C
C***********************************************************************
C
C
C     Subroutine to check consistency of input parameters and data,
C     output warnings to the terminal and/or log file, and terminate
C     the simulation if necessary.
C
C
C=======================================================================

        INCLUDE 'tmhscf.h'
        INCLUDE 'mpif.h'

C=======================================================================

        IF(nsteps.LT.0.OR.nsteps.GT.10000000)
     &     CALL terror(' input error for parameter nsteps ')
C              -------

        IF(noutbod.LT.0)
     &     CALL terror(' input error for parameter noutbod ')
C              -------

        IF(noutlog.LT.0)
     &     CALL terror(' input error for parameter noutlog ')
C              -------

        IF(dtime.LE.-1.e20.OR.dtime.GT.1.e20)
     &     CALL terror(' input error for parameter dtime ')
C              -------
```

```
      IF(G.LE.0.0.OR.G.GT.1.e20)
     &      CALL terror(' input error for parameter G ')
C                 -------

      RETURN
      END
C***********************************************************************
C
C
                       SUBROUTINE corracc
C
C
C***********************************************************************
C
C
C     Subroutine to correct accelerations so that the center of
C     mass remains fixed at the origin.
C
C
C=======================================================================

      INCLUDE 'tmhscf.h'
      INCLUDE 'mpif.h'

C   Declaration of local variables.
C   -------------------------------

      INTEGER i
ccompaq     tmpsum replaced with tmpsum_snd and tmpsum_rcv
      REAL*8 axcm,aycm,azcm,mtot,tmpsum_snd(4), tmpsum_rcv(4)

C=======================================================================
          axcm=0.0
          aycm=0.0
          azcm=0.0
          mtot=0.0

          DO 10 i=1,nbodies
             IF ((.NOT.multistep).OR.(multistep.AND.im(i).EQ.rtag)) THEN
                mtot=mtot+mass(i)
                axcm=axcm+mass(i)*ax(i)
                aycm=aycm+mass(i)*ay(i)
                azcm=azcm+mass(i)*az(i)
             ENDIF
 10       CONTINUE

ccompaq      tmpsum replaced with tmpsum_snd

          tmpsum_snd(1) = mtot
          tmpsum_snd(2) = axcm
          tmpsum_snd(3) = aycm
          tmpsum_snd(4) = azcm
ccompaq      tmpsum replaced with tmpsum_snd and tmpsum_rcv in MPI call

        call MPI_Allreduce (tmpsum_snd,tmpsum_rcv, 4,
     &                       mpi_double_precision,
     &                       mpi_sum, mpi_comm_world,ierr)

ccompaq      tmpsum replaced with tmpsum_rcv

          mtot = tmpsum_rcv(1)
          axcm = tmpsum_rcv(2)
          aycm = tmpsum_rcv(3)
```

```
      azcm = tmpsum_rcv(4)

      axcm=axcm/mtot
      aycm=aycm/mtot
      azcm=azcm/mtot

      DO 20 i=1,nbodies
        IF ((mstpflag.AND.im(i).EQ.rtag)) THEN
          ax(i)=ax(i)-axcm
          ay(i)=ay(i)-aycm
          az(i)=az(i)-azcm
          oax(i)=oax(i)-axcm
          oay(i)=oay(i)-aycm
          oaz(i)=oaz(i)-azcm
        ENDIF
 20   CONTINUE

      RETURN
      END

C***********************************************************************
C
C
      FUNCTION FACTRL(N)
C
C
C***********************************************************************
C
C
C     A function to compute factorials. (From numerical recipes.)
C
C
```

```
C=====================================================================

      INTEGER n, ntop, j
      REAL*8 factrl, a, gammln, arggam

      DIMENSION A(33)

      DATA NTOP,A(1)/0,1./

      IF (N.LT.0) THEN
        PAUSE 'negative factorial'
      ELSE IF (N.LE.NTOP) THEN
        FACTRL=A(N+1)
      ELSE IF (N.LE.32) THEN
        DO 11 J=NTOP+1,N
          A(J+1)=J*A(J)
11      CONTINUE
        NTOP=N
        FACTRL=A(N+1)
      ELSE
        arggam=n+1.
        FACTRL=EXP(GAMMLN(arggam))
      ENDIF

      RETURN
      END
C*********************************************************************
C
C
      FUNCTION GAMMLN(XX)
C
C
```

```
C*********************************************************************
C
C
C     A routine to compute the natural logarithm of the gamma
C     function.  (Taken from numerical recipes.)
C
C
C=====================================================================

      INTEGER j

      REAL*8 COF(6),STP,HALF,ONE,FPF,X,TMP,SER,gammln,xx

      DATA COF,STP/76.18009173D0,-86.50532033D0,24.01409822D0,
     &     -1.231739516D0,.120858003D-2,-.536382D-5,2.50662827465D0/
      DATA HALF,ONE,FPF/0.5D0,1.0D0,5.5D0/

      X=XX-ONE
      TMP=X+FPF
      TMP=(X+HALF)*LOG(TMP)-TMP
      SER=ONE

      DO 11 J=1,6
        X=X+ONE
        SER=SER+COF(J)/X
11    CONTINUE

      GAMMLN=TMP+LOG(STP*SER)

      RETURN
      END
C*********************************************************************
```

```
C
C
          SUBROUTINE inbods
C
C
C***********************************************************************
C
C
C       Subroutine to read phase coordinates.
C
C
C=======================================================================

          INCLUDE 'tmhscf.h'
          include 'mpif.h'

          INTEGER i, j, junk

          IF (me .eq. 0) THEN
             OPEN(ubodsin,FILE=ibodfile,STATUS='OLD')
c              READ(ubodsin,*) nbodies,tnow, bhmass
             READ(ubodsin,*) nbodies,tnow
          ENDIF
          IF (nbodies .gt. nbodsmax) THEN
             IF (me .eq. 0) CLOSE(ubodsin)
             CALL terror(' Number of bodies exceeds the upper bound. ')
          ENDIF

        call shareint(nbodies,1)
        call share(tnow,1)

c         IF (MOD(nbodies, n_pes) .ne. 0) THEN
```

```
c            IF (me .eq. 0) CLOSE(ubodsin)
c            CALL terror('␣Bodies␣cannot␣be␣distributed␣to␣PEs␣evenly.␣')
c         ENDIF

        nbodies = nbodies/n_pes
c new - for acceptance test only - put nbodies back to orignal file length
c - ie. we will have nbodies/processor, where the file contains
c nbodies and data is siply duplicated on all processors via broadcast
c         nbodies = nbodies*n_pes

         IF (nbodies .gt. nbodsper) THEN
            IF (me .eq. 0) CLOSE(ubodsin)
            CALL terror('␣Number␣of␣bodies␣per␣proc␣exceeds␣bound.␣')
         ENDIF

         IF (me .eq. 0) THEN
            DO 10 i=1,nbodies
               READ(ubodsin,*) mass(i),x(i),y(i),z(i),vx(i),vy(i),vz(i)
 10         CONTINUE
         ENDIF
c       wait here until initial read is completed by proc 0
c         call mpi_barrier(mpi_comm_world,ier)
c now broadcast so everyone has same data
c       call mpi_bcast(mass,nbodies,mpi_double_precision,
c     &       0,mpi_comm_world,ierr)
c       call mpi_bcast(x,nbodies,mpi_double_precision,
c     &       0,mpi_comm_world,ierr)
c       call mpi_bcast(y,nbodies,mpi_double_precision,
c     &       0,mpi_comm_world,ierr)
c       call mpi_bcast(z,nbodies,mpi_double_precision,
c     &       0,mpi_comm_world,ierr)
c       call mpi_bcast(vx,nbodies,mpi_double_precision,
c        &         0,mpi_comm_world,ierr)
c          call  mpi_bcast(vy,nbodies,mpi_double_precision,
c        &         0,mpi_comm_world,ierr)
c          call  mpi_bcast(vz,nbodies,mpi_double_precision,
c        &         0,mpi_comm_world,ierr)
          DO 30 j = 1, n_pes-1
              IF (me .eq. 0) THEN
                  DO 20 i=1,nbodies
                      READ(ubodsin,*) temp01(i),temp02(i),temp03(i),
     &                                 temp04(i),temp05(i),temp06(i),
     &                                 temp07(i)
20                CONTINUE
              ENDIF

                  CALL mpiget(mass, temp01, nbodies,j,0,me)
                  CALL mpiget(x,    temp02, nbodies,j,0,me)
                  CALL mpiget(y,    temp03, nbodies,j,0,me)
                  CALL mpiget(z,    temp04, nbodies,j,0,me)
                  CALL mpiget(vx,   temp05, nbodies,j,0,me)
                  CALL mpiget(vy,   temp06, nbodies,j,0,me)
                  CALL mpiget(vz,   temp07, nbodies,j,0,me)
              IF (me .eq. j) THEN
              ENDIF
30        CONTINUE

          IF (me .eq. 0) CLOSE(ubodsin)

          DO 40 i=1,nbodies
          rlength(i)=sqrt(x(i)**2+y(i)**2+z(i)**2)



 40       CONTINUE
```

```
      RETURN
      END
C***********************************************************************
C
C
                        SUBROUTINE initpars
C
C
C***********************************************************************
C
C
C     Subroutine to initialize system parameters that depend on
C     either the input data or defined PARAMETERS.
C
C
C=======================================================================

      INCLUDE 'tmhscf.h'
      include 'mpif.h'

C=======================================================================

C   Initialize misc. useful numbers.
C   ------------------------------------
      integer i
      one=1.0
      two=2.0
      pi=4.0*ATAN(one)
      twoopi=2./pi
      onesixth=1./6.
      tiny=1.e-30
      zero=0.0

      tpos=tnow
      tvel=tnow

      dtime = 1.e-6
      dtsmall = 1.e-6
c       dtbig = 1.e-6
      data dtbin/1.0,10,100,1000/
c       data dtbin/1.0,2.0,4.0,8.0/
      data numb/0,0,0,0/
      data nbin/0,0,0,0/
      data nbin0/0/
c       call share(nbin,4)
c       call share(nbin0,1)



c       do i=1,3
c       dtbin(i)=2**(i-1)
c       end do
c       call share(dtbin,3)
      rtag=1
c       call share(rtag,1)
      RETURN
      END
C*******************************************************************
C
C
                SUBROUTINE initsys(n)
C
C
```

```
C*******************************************************************
C
C
C      Subroutine to initialize the state of the system.
C
C
C===================================================================

         INCLUDE 'tmhscf.h'
         include 'mpif.h'
         INTEGER n,p
         real*8 tmpstar0_snd,tmpstar0_rcv,rtid

C    Declaration of local variables.
C    ----------------------------------

C===================================================================
C
C    Begin timing.
C    -------------
         totime0=mpi_wtime()
         totime0=mpi_wtime()
         totime=totime0

C                 --------
         CALL startout
C                 --------
         CALL inparams
C                 --------
         CALL inbods
C                 ------
         CALL checkinp
```

```
c the following is to initial the bh position and velocity by Baile
      xbh=0.0
      ybh=0.0
      zbh=0.0
      vxbh=0.0
      vybh=0.0
      vzbh=0.0
      kix=0.5
      kiy=0.0
      kiz=0.0
      star0=0
      sx=kix*G*bhmass**2/C
      sy=kiy*G*bhmass**2/C
      sz=kiz*G*bhmass**2/C
      do n=1,nbodies
      er(n)=0.0
      et(n)=0.0
      ephi(n)=0.0
      maxratio(n)=0.0
      rlength(n)=sqrt(x(n)**2+y(n)**2+z(n)**2)
c     rtid=(0.844**2*2.e6/0.42*bhmass)**(1./3.)*2.254e-8/2.2751e-7
      end do


C          ---------
      CALL initpars
C          ---------
      if (multistep) CALL setim
c       do n=1,nbodies
c        if (im(n).eq.0) then
c          bhmass=bhmass+mass(n)
c         mass(n)=0.0
```

```
c            star0=star0+1
c          end if
c          end do
c          tmpstar0_snd=star0
c          call MPI_Allreduce (tmpstar0_snd,tmpstar0_rcv, 1,
c     &                          mpi_double_precision,
c     &                          mpi_sum, mpi_comm_world,ierr)
c          star0=tmpstar0_rcv

C                                 ———



         cputime0 = mpi_wtime()
         cputime = cputime0
c        call torb
        CALL accpot(0)

C                 ———

        IF(fixacc) CALL corracc
C                              ————



       CALL outstate(0)
C                ————



         n=1
C
        RETURN
        END
```

```
C**********************************************************************
C
C
                         SUBROUTINE initmpi
C
C
C**********************************************************************
C
C
C     Subroutine to initialize the MPI variables.
C
C
C======================================================================
       INCLUDE 'mpif.h'
       INCLUDE 'tmhscf.h'


CCCCCCCCCCC  INITIALIZE MPI ENVIRONMENT (BELOW)  CCCCCCCCCCCCCCCC

      CALL MPI_INIT( IERR )

C  GET THE PROCESS NUMBER AND ASSIGN IT TO THE VARIABLE MYID

      CALL MPI_COMM_RANK( MPI_COMM_WORLD, MYID, IERR )
      me=MYID
C  DETERMINE HOW MANY PROCESSES THE PROGRAM WILL RUN ON AND
C  ASSIGN THAT NUMBER TO NUMPROCS

      CALL MPI_COMM_SIZE( MPI_COMM_WORLD, NUMPROCS, IERR )
      n_pes=numprocs

CCCCCCCCCC  INITIALIZE MPI ENVIRONMENT (ABOVE)  CCCCCCCCCCCCCCCC
```

```
      RETURN
      END
C***********************************************************************
C
C
                        SUBROUTINE inparams
C
C
C***********************************************************************
C
C
C     Subroutine to read in parameters.
C
C     Input parameters:
C
C        headline  : identification string for the run.
C        nsteps    : number of timesteps.
C        noutbod   : output system state once every nsteps/noutbod
C                    steps.
C        noutlog   : output logfile data once every nsteps/noutlog
C                    steps.
C        dtime     : the timestep.
C        G         : value of gravitational constant, in appropriate
C                    units.
C        selfgrav  : option to turn off (.FALSE.) system self-gravity.
C        inptcoef  : option to read-in expansion coefficients.
C        outpcoef  : option to write-out expansion coefficients.
C        zeroodd   : option to zero all odd terms in the expansion.
C        zeroeven  : option to zero all even terms in the expansion.
C        fixacc    : option to force conservation of linear
C                    momentum by subtracting acceleration of c.o.m.
```

```
C
C       ecrit       : energy below which we want particle to be n^2 integrated
C       rcrit       : radius below which we want particle to be multistepped
C                     as multiple of radbh (typically 0.5-1)
C=======================================================================

        INCLUDE 'tmhscf.h'
        include 'mpif.h'

        CHARACTER *1 pcomment

C=======================================================================

        IF (me .eq. 0) THEN
           OPEN(UNIT=upars,FILE=parsfile,STATUS='OLD')

C   Read parameters, close the file.
C   --------------------------------

           READ(upars,'(a)') pcomment

           READ(upars,'(a)') headline
           READ(upars,*) nsteps
           READ(upars,*) noutbod
           READ(upars,*) noutlog
           READ(upars,*) dteps
           READ(upars,*) G
           REAd(upars,*) C !added by Baile
           READ(upars,*) firstpn !added by Baile
           READ(upars,*) secondpn
           READ(upars,*) secondhpn
           READ(upars,*) thirdpn
           READ(upars,*) thirdhpn
           READ(upars,*) tfinal
           READ(upars,*) multistep
           READ(upars,*) fixedn
           READ(upars,*) selfgrav
           READ(upars,*) inptcoef
           READ(upars,*) outpcoef
           READ(upars,*) zeroodd
           READ(upars,*) zeroeven
           READ(upars,*) fixacc
           READ(upars,*) rcrit
           READ(upars,*) ecrit
           READ(upars,*) lilout
           READ(upars,*) nlilout

           CLOSE(UNIT=upars)

   C***********************************************************************
   C
   C          inparams now reads in SCFMOD with other data
   C          Steinn Sigurdsson, Dec. 1993
   C
   C***********************************************************************
   C
   C
   C      Subroutine to read in parameters.
   C
   C      Input parameters:
   C
   C          pcomment      : first line
   C          iseed         : initialises random number generator
   C        msys          : mass of system to be generated
```

```
C         rsys          : radius of system
C         r0            : scale radius
C         bhmass        : mass of black hole
C         epsbh         : softening length for black hole
C         tstartbh      : time to start bh growth
C         tgrowbh       : time to grow bh
C         tlivebh       : time bh lives for
C         tdiebh        : time to shrink bh
C         xdrag         : drag coeff for vx
C         ydrag         : drag coeff for vy
C         zdrag         : drag coeff for vz
C         tstartd   : time drag starts
C          tgrowdrag    : time drag lasts
C          tdiedrag     : time drag dies down
C          mkmod        : make internal model or read in data?
C         bhgrav        : do we grow a black hole
C         bhgrav        : do we grow a black hole
C         usedrag       : is there drag
C         stellev       : is there stellar evolution
C
C=====================================================================
              OPEN(UNIT=umods,FILE=modsfile,STATUS='OLD')

              READ(umods,'(a)')
              READ(umods,*) iseed
              READ(umods,*) bhmass
              READ(umods,*) epsbh
              READ(umods,*) tstartbh
              READ(umods,*) tgrowbh
              READ(umods,*) tlivebh
              READ(umods,*) tdiebh
              READ(umods,*) xdrag
```

```
          READ(umods,*) ydrag
          READ(umods,*) zdrag
          READ(umods,*) tstartdrag
          READ(umods,*) tgrowdrag
          READ(umods,*) tlivedrag
          READ(umods,*) tdiedrag
          READ(umods,*) bhgrav
          READ(umods,*) usedrag
          READ(umods,*) stellev
C
          CLOSE(UNIT=umods)
C
       ENDIF

          CALL  shareint(nsteps,1)
          CALL  shareint(noutbod,1)
          CALL  shareint(noutlog,1)
          CALL  share(dteps,1)
          CALL  share(G,1)
          call  share(C,1)
          call  sharelog(firstpn,1)
          call  sharelog(secondpn,1)
          call  sharelog(secondhpn,1)
          call  sharelog(thirdpn,1)
          call  sharelog(thirdhpn,1)
          CALL  share(tfinal,1)
          CALL  sharelog(multistep,1)
          CALL  sharelog(fixedn,1)
          CALL  sharelog(selfgrav,1)
          CALL  sharelog(inptcoef,1)
          CALL  sharelog(outpcoef,1)
          CALL  sharelog(zeroodd,1)
          CALL sharelog(zeroeven,1)
          CALL sharelog(fixacc,1)
          CALL share(rcrit,1)
          CALL share(ecrit,1)
          CALL sharelog(lilout,1)
          CALL shareint(nlilout,1)
C    ------------------------------------
          CALL shareint(iseed,1)
          CALL share(bhmass,1)
          CALL share(epsbh,1)
          CALL share(tstartbh,1)
          CALL share(tgrowbh,1)
          CALL share(tlivebh,1)
          CALL share(tdiebh,1)
          CALL share(xdrag,1)
          CALL share(ydrag,1)
          CALL share(zdrag,1)
          CALL share(tstartdrag,1)
          CALL share(tgrowdrag,1)
          CALL share(tlivedrag,1)
          CALL share(tdiedrag,1)
          CALL sharelog(bhgrav,1)
          CALL sharelog(usedrag,1)
          CALL sharelog(stellev,1)


c          dtime = dteps
        nfrac = 1

        RETURN
        END
C*********************************************************************
```

```
C
C
                    SUBROUTINE iocoef(sinsum,cossum)
C
C
C***********************************************************************
C
C
C     Subroutine to input and output expansion coefficients.
C
C
C=======================================================================

      INCLUDE 'tmhscf.h'
      INCLUDE 'mpif.h'

      INTEGER n,l,m
      LOGICAL firstc
      REAL*8 sinsum,cossum,tt

      DIMENSION sinsum(0:nmax,0:lmax,0:lmax),
     &          cossum(0:nmax,0:lmax,0:lmax)

      DATA firstc/.TRUE./

      SAVE firstc

C=======================================================================

      tmptime = mpi_wtime()

      IF(firstc) THEN
```

```
         firstc=.FALSE.

         IF(outpcoef .and. me.eq.0)
     &        OPEN(uoutcoef,FILE=outcfile,STATUS='UNKNOWN')
         IF(inptcoef .and. me.eq.0)
     &        OPEN(uincoef,FILE=incfile,STATUS='OLD')

      ENDIF

      IF(outpcoef .and. me.eq.0) THEN

         WRITE(uoutcoef,100) tnow

         DO 30 n=0,nmax
            DO 20 l=0,lmax
               DO 10 m=0,l
                  WRITE(uoutcoef,100) sinsum(n,l,m),cossum(n,l,m)
 10            CONTINUE
 20         CONTINUE
 30      CONTINUE

 100     FORMAT(1x,10(1pe22.13))

      ENDIF

      IF(inptcoef) THEN

         if (me .eq. 0) then
              READ(uincoef,*) tt
              tmpsum01 = tt
         endif
         call share(tt,1)

         IF(tt.NE.tnow) CALL terror('␣input␣error␣in␣iocoef␣')
C                              -------

         DO 130 n=0,nmax
           DO 120 l=0,lmax
             DO 110 m=0,l
               if (me .eq. 0) then
                 READ(uoutcoef,*) sinsum(n,l,m),cossum(n,l,m)
               end if
               call share(sinsum(n,l,m),1)
               call share(cossum(n,l,m),1)
 110         CONTINUE
 120       CONTINUE
 130     CONTINUE

      ENDIF

      tmptime = mpi_wtime()-tmptime
      cputime0 = cputime0 + tmptime
      cputime = cputime + tmptime

      RETURN
      END
C***********************************************************************
C
C
      SUBROUTINE liloutb
C
C
C***********************************************************************
```

```
C
C
C       Subroutine to output phase coordinates.
C
C
C=======================================================================

        INCLUDE 'tmhscf.h'
        INCLUDE 'mpif.h'

        CHARACTER*3 sstring
        CHARACTER*7 filename
        CHARACTER*8 filepar
        CHARACTER*10 nstring

        INTEGER i,j,istring,insnap

        SAVE insnap,nstring

        DATA insnap/0/,nstring/'0123456789'/

C=======================================================================

        tmptime = mpi_wtime()

        insnap=insnap+1

        sstring(1:1)=nstring(1+insnap/100:1+insnap/100)
        istring=1+MOD(insnap,100)/10
        sstring(2:2)=nstring(istring:istring)
        istring=1+MOD(insnap,10)
        sstring(3:3)=nstring(istring:istring)
```

```
      filepar=olilfile
      filename=filepar(1:4)//sstring(1:3)

      if (me .eq. 0)
     &     OPEN(UNIT=ubodslil,FILE=filename,STATUS='UNKNOWN')



      if (me .eq. 0) then
          WRITE(ubodslil,20) nbodies*n_pes,tnow
 20       FORMAT(1x,1i8,2(1pe14.6))
      endif

      DO 30 i=1,nbodies
      if (me .eq. 0) then
            WRITE(ubodslil,111) mass(i),x(i),y(i),z(i),vx(i),
     &                           vy(i),vz(i),pot(i)+potext(i),
     &                           adens(i),dti(i),im(i)
      end if
 30   CONTINUE


 111      FORMAT(1x,10(1pe14.6),1i2)

      if (me .eq. 0) then
          CLOSE(ubodslil)
      end if

      tmptime = mpi_wtime()-tmptime
      cputime0 = cputime0 + tmptime
      cputime = cputime + tmptime
      RETURN
      END

C***************************************************************
C
C
      SUBROUTINE outbods
C
C
C***************************************************************
C
C
C     Subroutine to output phase coordinates.
C
C
C===============================================================

      INCLUDE 'tmhscf.h'
      include 'mpif.h'

      CHARACTER*3 sstring
      CHARACTER*7 filename
      CHARACTER*8 filepar
      CHARACTER*10 nstring

      INTEGER i,j,istring,nsnap,p,k

      SAVE nsnap,nstring

      DATA nsnap/0/,nstring/'0123456789'/

C===============================================================
        tmptime = mpi_wtime()

        nsnap=nsnap+1

        sstring(1:1)=nstring(1+nsnap/100:1+nsnap/100)
        istring=1+MOD(nsnap,100)/10
        sstring(2:2)=nstring(istring:istring)
        istring=1+MOD(nsnap,10)
        sstring(3:3)=nstring(istring:istring)
        filepar=obodfile
        filename=filepar(1:4)//sstring(1:3)


        if (me .eq. 0)
     &     OPEN(UNIT=ubodsout,FILE=filename,STATUS='UNKNOWN')


        if (me .eq. 0) then
           WRITE(ubodsout,20) nbodies*n_pes,tnow,bhmass,xbh,ybh,
     &zbh,
     &vxbh,vybh,vzbh,sx/(G*bhmass**2/C),
     &sy/(G*bhmass**2/C),sz/(G*bhmass**2/C)

 20        FORMAT(1x,1i10,1x,11(1pe20.10))
        end if




        DO 30 i=1,nbodies
c       write(*,*)'outbods,me,i,E3(i),x(i)',me,i,E3(i),x(i)
           IF (me .eq. 0)
     &           WRITE(ubodsout,111) id(i),mass(i),x(i),y(i),z(i),
     &vx(i),vy(i),vz(i),
     &E(i)/mass(i),
     &E0(i)/mass(i),
     &E1(i)/mass(i),E2(i)/mass(i),E3(i)/mass(i),
     &0.5*(vx(i)**2+vy(i)**2+vz(i)**2)+potext(i),
     &E(i)+mass(i)*pot(i)-0.5*mass(i)*(vx(i)**2+vy(i)**2+vz(i)**2),
     &-1.*mass(i)/sqrt(x(i)**2+y(i)**2+z(i)**2),mass(i)*pot(i),
     &mass(i)*potext(i),
     &Jx(i),Jy(i),Jz(i),
     &er(i),
     &et(i),ephi(i),
     &adens(i),dti(i),im(i),ax(i),ay(i),az(i),
     &maxratio(i),rlength(i),vlength(i),J2x(i),J2y(i),J2z(i)!output a added by Baile
 30        CONTINUE


              DO 50 j=1, n_pes-1
c          do k=1,nbodies
c          write(*,*)'b4_mpiget:me,j,k,E3(k),x(k)',me,j,k,E3(k),x(k)
c          end do


                 CALL mpiget(temp01, mass,    nbodies, 0,j,me)
c                CALL mpigetint(temp18, starlive, nbodies, 0, j,me)
                 CALL mpiget(temp02, x,       nbodies, 0,j,me)
                 CALL mpiget(temp03, y,       nbodies,0,j,me)
                 CALL mpiget(temp04, z,       nbodies,0,j,me)
                 CALL mpiget(temp05, vx,      nbodies,0,j,me)
                 CALL mpiget(temp06, vy,      nbodies,0,j,me)
                 CALL mpiget(temp07, vz,      nbodies,0,j,me)
                 CALL mpiget(temp19, E,      nbodies,0,j,me)
```

```
                    CALL mpiget(temp20, Jx,      nbodies,0,j,me)
                    CALL mpiget(temp21, Jy,      nbodies,0,j,me)
                    CALL mpiget(temp22, Jz,      nbodies,0,j,me)
                    CALL mpiget(temp23, er,      nbodies,0,j,me)
                    CALL mpiget(temp24, et,      nbodies,0,j,me)
                    CALL mpiget(temp25, ephi,     nbodies,0,j,me)
                    CALL mpiget(temp08, pot,     nbodies,0,j,me)
                    CALL mpiget(temp09, potext, nbodies,0,j,me)
                    CALL mpiget(temp10, adens,  nbodies,0,j,me)
                    CALL mpiget(temp11, dti,     nbodies,0,j,me)
                    CALL mpiget(temp12, ax,       nbodies,0,j,me)
                    CALL mpiget(temp13, ay,       nbodies,0,j,me)
                    CALL mpiget(temp14, az,       nbodies,0,j,me)
                    CALL mpiget(temp15, maxratio,    nbodies,0,j,me)
                    CALL mpiget(temp16, rlength, nbodies,0,j,me)
                    CALL mpiget(temp17, vlength, nbodies,0,j,me)
                    CALL mpigetint(itemp01, im,      nbodies,0,j,me)
                    CALL mpigetint(itemp02, id,      nbodies,0,j,me)
                    CALL mpiget(tempE0,E0,nbodies,0,j,me)
                    CALL mpiget(tempE1,E1,nbodies,0,j,me)
                    CALL mpiget(tempE2,E2,nbodies,0,j,me)
                    CALL mpiget(tempE3,E3,nbodies,0,j,me)
                    CALL mpiget(tempJ2x,J2x,nbodies,0,j,me)
                    CALL mpiget(tempJ2y,J2y,nbodies,0,j,me)
                    CALL mpiget(tempJ2z,J2z,nbodies,0,j,me)
c              do k=1,nbodies
c              write(*,*)'afmpi:me,j,k,E2(k),tempE2(k),x(k),temp02(k)',
c         &me,j,k,E2(k),tempE2(k),x(k),temp02(k)
c              end do


                 IF (me .eq. 0) THEN
                 DO 40 i=1,nbodies
                    WRITE(ubodsout,111) itemp02(i),temp01(i),temp02(i),
     &temp03(i),temp04(i),temp05(i),temp06(i),
     &temp07(i),
     &temp19(i)/temp01(i),
     &tempE0(i)/temp01(i),
     &tempE1(i)/temp01(i),tempE2(i)/temp01(i),tempE3(i)/temp01(i),
     &0.5*(temp05(i)**2+temp06(i)**2+temp07(i)**2)+temp09(i),
     &temp19(i)+temp01(i)*temp08(i)-0.5*temp01(i)*(temp05(i)**2+
     &temp06(i)**2+temp07(i)**2),
     &-temp01(i)/sqrt(temp02(i)**2+temp03(i)**2+temp04(i)**2),
     &temp01(i)*temp08(i),temp01(i)*temp09(i),
     &temp20(i),temp21(i),
     &temp22(i),temp23(i),
     &temp24(i),temp25(i),
     &temp10(i),temp11(i),itemp01(i),temp12(i),temp13(i),temp14(i),
     &temp15(i),temp16(i),temp17(i),tempJ2x(i),tempJ2y(i),tempJ2z(i)
40               CONTINUE
              ENDIF
50          CONTINUE

111         FORMAT(1x,1i7,1(1pe20.10),1x,24(1pe20.10),1i2,9(1pe20.10)) !format editec


            if(me.eq.0) CLOSE(ubodsout)

         tmptime = mpi_wtime()-tmptime
         cputime0 = cputime0 + tmptime
         cputime = cputime + tmptime

         RETURN
         END
```

```
C*********************************************************************
C
C
      SUBROUTINE outlog
C
C
C*********************************************************************
C
C
C     Subroutine to output phase coordinates.
C
C
C=====================================================================

      INCLUDE 'tmhscf.h'
      INCLUDE 'mpif.h'

      INTEGER i,j
      LOGICAL firstc
      REAL*8 lxtot,lytot,lztot,mtot,vxcm,vycm,vzcm,etot,ektot,eptot,
   &        m2tw,t1,clausius,m2claus,cpux,xcm,ycm,zcm,epselfg
ccompaq    tmpsum replaced with tmpsum_snd and tmpsum_rcv
     REAL*8 tmpsum_snd(14), tmpsum_rcv(14)
     real*8 cumcpu,cumtot

      DATA firstc/.TRUE./

      SAVE firstc
     save cumcpu,cumtot
     data cumcpu,cumtot/0.d0,0.d0/

C=====================================================================
      cpux=mpi_wtime()

      IF (me.eq.0) THEN
        IF(firstc) THEN
          OPEN(UNIT=ulog,FILE=logfile,STATUS='NEW')
     write(ulog,*) 'numprocs= ',n_pes, 'nbodies/proc= ',nbodies
          firstc=.FALSE.
        ELSE
          OPEN(UNIT=ulog,FILE=logfile,STATUS='OLD',POSITION='APPEND')
c           OPEN(UNIT=ulog,FILE=logfile,STATUS='OLD',ACCESS='APPEND')
        ENDIF
      ENDIF

      mtot=0.0
      xcm=0.0
      ycm=0.0
      zcm=0.0
      vxcm=0.0
      vycm=0.0
      vzcm=0.0
      lxtot=0.0
      lytot=0.0
      lztot=0.0
      etot=0.0
      ektot=0.0
      eptot=0.0
      epselfg=0.0
      clausius=0.0

      DO 30 i=1,nbodies
         mtot=mtot+mass(i)
```

```
             xcm=xcm+mass(i)*x(i)
             ycm=ycm+mass(i)*y(i)
             zcm=zcm+mass(i)*z(i)
             vxcm=vxcm+mass(i)*vx(i)
             vycm=vycm+mass(i)*vy(i)
             vzcm=vzcm+mass(i)*vz(i)
             lxtot=lxtot+mass(i)*(y(i)*vz(i)-z(i)*vy(i))
             lytot=lytot+mass(i)*(z(i)*vx(i)-x(i)*vz(i))
             lztot=lztot+mass(i)*(x(i)*vy(i)-y(i)*vx(i))
             eptot=eptot+0.5*mass(i)*pot(i)+mass(i)*potext(i)
             epselfg=epselfg+0.5*mass(i)*pot(i)
             ektot=ektot+0.5*mass(i)*(vx(i)**2+vy(i)**2+vz(i)**2)+
     &E1(i)+E2(i)+E3(i)
             clausius=clausius+mass(i)*(x(i)*ax(i)+y(i)*ay(i)+z(i)*az(i))
 30      CONTINUE

ccompaq    tmpsum replaced with tmpsum_snd
         tmpsum_snd(1) = mtot
         tmpsum_snd(2) = xcm
         tmpsum_snd(3) = ycm
         tmpsum_snd(4) = zcm
         tmpsum_snd(5) = vxcm
         tmpsum_snd(6) = vycm
         tmpsum_snd(7) = vzcm
         tmpsum_snd(8) = lxtot
         tmpsum_snd(9) = lytot
         tmpsum_snd(10) = lztot
         tmpsum_snd(11) = eptot
         tmpsum_snd(12) = epselfg
         tmpsum_snd(13) = ektot
         tmpsum_snd(14) = clausius
ccompaq    tmpsum replaced with tmpsum_snd and tmpsum_rcv in MPI call
```

```
        call MPI_Allreduce (tmpsum_snd,tmpsum_rcv, 14,
     &                        mpi_double_precision,
     &                        mpi_sum, mpi_comm_world,ierr)
ccompaq    tmpsum replaced with tmpsum_rcv
         mtot     = tmpsum_rcv(1)
         xcm      = tmpsum_rcv(2)
         ycm      = tmpsum_rcv(3)
         zcm      = tmpsum_rcv(4)
         vxcm     = tmpsum_rcv(5)
         vycm     = tmpsum_rcv(6)
         vzcm     = tmpsum_rcv(7)
         lxtot    = tmpsum_rcv(8)
         lytot    = tmpsum_rcv(9)
         lztot    = tmpsum_rcv(10)
         eptot    = tmpsum_rcv(11)
         epselfg  = tmpsum_rcv(12)
         ektot    = tmpsum_rcv(13)
         clausius = tmpsum_rcv(14)

         xcm=xcm/mtot
         ycm=ycm/mtot
         zcm=zcm/mtot
         vxcm=vxcm/mtot
         vycm=vycm/mtot
         vzcm=vzcm/mtot

         etot=ektot+eptot
         m2tw= -2.*ektot/eptot
         m2claus= -2.*ektot/clausius

         IF (me .eq. 0) THEN
            WRITE(ulog,120) tnow
            WRITE(ulog,120) mtot
            WRITE(ulog,120) xcm,ycm,zcm
            WRITE(ulog,120) vxcm,vycm,vzcm
            WRITE(ulog,120) lxtot,lytot,lztot
            WRITE(ulog,120) ektot,eptot,epselfg,etot
            WRITE(ulog,151) nbin0
                do j=1,4
                write(ulog,151)nbin(i)
               end do

            WRITE(ulog,120) m2tw,clausius,m2claus
            WRITE(ulog,140) (cpux-cputime)
            WRITE(ulog,150) (cpux-totime)
      cumcpu=cumcpu+cpux-cputime
        cumtot=cumtot+cpux-totime
        write(ulog,*)   'cumulative step time without I/O =',cumcpu
        write(ulog,*)   'cumulative step time with I/O    =',cumtot
            WRITE(ulog,130)

    120     FORMAT(5(1pe18.10))
    130     FORMAT(/)
    140     FORMAT(10x,1pe18.10, ' (Step time without I/O) ')
    150     FORMAT(10x,1pe18.10, ' (Step time with I/O) ')
    151      format(5(1i7))
         ENDIF

         IF (me.eq.0) CLOSE(ulog)

         totime=cpux
         cputime=mpi_wtime()
         tmptime = cputime-cpux
         cputime0 = cputime0 + tmptime
         RETURN
         END
    C*********************************************************************
    C
    C
                        SUBROUTINE outstate(n)
    C
    C
    C*********************************************************************
    C
    C
    C     Subroutine to output information about the system state to
    C     the log and body data files.
    C
    C
    C=====================================================================

         INCLUDE 'tmhscf.h'
         include 'mpif.h'

    C   Declaration of local variables.
    C   -------------------------------

         INTEGER n,j

    C=====================================================================

         if(mod(n,100).eq.0)then
         CALL outterm(' step completed: ',n)
    C              -------
         end if
         IF(n.EQ.0) THEN


            CALL outbods
    C            --------
            CALL outlog
    C            ------

         ELSE


            IF((MOD(n,noutlog).EQ.0).OR.(MOD(n,noutbod).EQ.0).
         &  OR.(MOD(n,nlilout).EQ.0)) THEN


              IF(MOD(n,noutlog).EQ.0) CALL outlog
    C                                      ------
    c           if(mod(nbin1,noutbod1).eq.0) call outbods
              IF(MOD(n,noutbod).EQ.0) CALL outbods
    C                                      --------
              IF(MOD(n,nlilout).EQ.0) CALL liloutb
    C                                      -------
            ENDIF
    C
    C
         ENDIF

         RETURN
         END
    C*****************************************************************
    C
```

```
C
                    SUBROUTINE outterm(message,n)
C
C
C*****************************************************************
C
C
C     Subroutine to output a message to the terminal and to the
C     terminal emulation file.
C
C
C=================================================================

      include 'mpif.h'
      INCLUDE 'tmhscf.h'

C   Declaration of local variables.
C   -------------------------------

      CHARACTER*(*) message
      INTEGER n

C=================================================================

      tmptime = mpi_wtime()

C   Write the message.
C   ------------------

      IF (me .eq. 0) THEN
         IF(n.GE.0) THEN
```

```
c                 WRITE(uterm,*) message,n
                 WRITE(utermfil,*) message,n

              ELSE
c                 WRITE(uterm,40)
c                 WRITE(uterm,50) message
c                 WRITE(uterm,40)

                 WRITE(utermfil,40)
                 WRITE(utermfil,50) message
                 WRITE(utermfil,40)
              ENDIF

 40       FORMAT(/,1x,72('*'))
 50       FORMAT(/,a)

          ENDIF

          tmptime = mpi_wtime()-tmptime
          cputime0 = cputime0 + tmptime
          cputime = cputime + tmptime

          RETURN
          END

C***********************************************************************
C
C
                         SUBROUTINE setim
C
C
C***********************************************************************
```

```
C currently this subroutine is a bit insane. if ecrit = 0, then the
C multistepping criteria is r < rcrit. if ecrit < 0, then the
C criteria is e < ecrit. and if ecrit > 0, then the criteria is M< ecrit,
C really a critical *MASS*, not energy...
C*********************************************************************
c        include 'tmhscf.h'
c         integer i
c         real*8 rcrit2,r2,energy

c         if (ecrit.eq.0) then
c            rcrit2=rcrit*rcrit
c            do i=1,nbodies
c        if (starlive(i).eq.1) then
c               r2=x(i)*x(i)+y(i)*y(i)+z(i)*z(i)
c               if (r2.lt.rcrit2) then
c                  im(i)=0
c               else
c                  im(i)=1
c               endif
c        end if
c            end do
c        elseif (ecrit.lt.0) then
c            do i=1,nbodies
c        if (starlive(i).eq.1) then
c               energy=0.5*(vx(i)*vx(i)+vy(i)*vy(i)+vz(i)*vz(i)) +
c     &                    pot(i)+potext(i)
c               if (energy.lt.ecrit) then
c                  im(i)=0
c               else
c                  im(i)=1
c               endif
c        end if
```

```
c               end do
c            elseif (ecrit.gt.0) then
c               do i=1,nbodies
c          if (starlive(i).eq.1) then
c                 if (mass(i).lt.ecrit) im(i) = 0
c           end if
c           enddo
c           endif

c           RETURN
c           END

c**********************************************************by baile
          include 'tmhscf.h'
          include 'mpif.h'

          real*8 r0,r1,r2
          integer i
          integer tmpnbin_snd(5),tmpnbin_rcv(5)
          nbin0=0
          do i=1,4
          nbin(i)=0
          end do
c           data nbin/0,0,0,0/
c           data nbin0/0/
c           call shareint(nbin,4)
c           call shareint(nbin0,1)
          r0=(0.844**2*2.e6/0.42*bhmass)**(1./3.)*2.254e-8/2.2751e-7
          r1=1.0e-4/2.2751e-7
          r2=1.0e-3/2.2751e-7
          r3=1.0e-2/2.2751e-7
c           write(*,*)'r0,r1,r2',r0,r1,r2
```

```
c         do i=1,nbodies
c         if(rlength(i).le.r0) then
c         im(i)=0
c         bhmass=bhmass+mass(i)
c         mass(i)=0.0
c         end if

c         if(rlength(i).gt.r0.and.rlength(i).lt.r1) im(i)=1
c         if(rlength(i).ge.r1.and.rlength(i).lt.r2) im(i)=2
c         if(rlength(i).ge.r2) im(i)=3
c         end do
          do i=1,nbodies
          id(i)=me*nbodies+i
          if(rlength(i).ge.r3) then
          im(i)=4
          nbin(4)=nbin(4)+1
          else if(rlength(i).ge.r2.and.rlength(i).lt.r3) then
          im(i)=3
          nbin(3)=nbin(3)+1
          else if(rlength(i).ge.r1.and.rlength(i).lt.r2) then
          im(i)=2
          nbin(2)=nbin(2)+1
          else if(rlength(i).gt.r0.and.rlength(i).lt.r1) then
          im(i)=1
          nbin(1)=nbin(1)+1
          else if(rlength(i).le.r0) then
          im(i)=0
          bhmass=bhmass+mass(i)
          mass(i)=0.0
          nbin0=nbin0+1
          end if
```

```
      end do
      do i=1,4
      tmpnbin_snd(i)=nbin(i)
      end do
c       tmpnbin_snd(2)=nbin(2)
c       tmpnbin_snd(3)=nbin(3)
c       tmpnbin_snd(4)=nbin(4)
      tmpnbin_snd(5)=nbin0
      call MPI_Allreduce (tmpnbin_snd,tmpnbin_rcv, 5,
     &                      mpi_integer,
     &                      mpi_sum, mpi_comm_world,ierr)
      do i=1,4
      nbin(i)=tmpnbin_rcv(i)
      end do
c       nbin(2)=tmpnbin_rcv(2)
c       nbin(3)=tmpnbin_rcv(3)
c       nbin(4)=tmpnbin_rcv(4)
      nbi0=tmpnbin_rcv(5)
      write(*,*)'nbin0',nbin0
      do i=1,4
      write(*,*)'i,nbin(i)',i,nbin(i)
      end do

      return
      end
```

```
C*********************************************************by baile
C*****************************************************************
C
C
                    SUBROUTINE startout
C
C
C*****************************************************************
C
C
C     Subroutine to open disk files for subsequent input/output.
C
C
C=================================================================

      INCLUDE 'tmhscf.h'
      INCLUDE 'mpif.h'

C=================================================================

C   Create terminal emulation file.
C   -------------------------------
      IF (me .eq. 0) THEN
         OPEN(UNIT=utermfil,FILE=termfile,STATUS='UNKNOWN')
         WRITE(utermfil,*) ' Start of output, woohoo! '
      ENDIF

      RETURN
      END
C*****************************************************************
C
C
```

```
                    SUBROUTINE steppos
C
C
C*****************************************************************
C
C
C      Subroutine to advance the positions of the bodies for a timestep.
C
C
C=================================================================

          INCLUDE 'tmhscf.h'

C    Declaration of local variables.
C    ---------------------------

          INTEGER p
          real*8 ax2,ay2,az2,ax3,ay3,az3,dtp,kiabs
          real*8 axbh2,aybh2,azbh2,axbh3,aybh3,azbh3   !Baile
C=================================================================
C
C    Loop over all spatial coordinates for all bodies.
C    -------------------------------------------------
C    Advance correct SCF particles
C    use a hermite integrator.
C
c            IF (.NOT.multistep) dtbig=dtime

          DO 10 p=1,nbodies

          IF (mstpflag.AND.(im(p).eq.rtag)) then
          dtp=dtbin(rtag)
```

```
c         write(*,*)'dtp particle'
c         write(*,*)dtp
c           write(*,*)'steppos,p,oax,dax',p,oax(p),dax(p)
 777        ax2=(-6.*(oax(p)-ax(p))-dtp*(4.*odax(p)+2.*dax(p)))/dtp**2
         ax3=( 12*(oax(p)-ax(p))+6.*dtp*(odax(p)+dax(p))  )/dtp**3
c
         ay2=(-6.*(oay(p)-ay(p))-dtp*(4.*oday(p)+2.*day(p)))/dtp**2
         ay3=( 12*(oay(p)-ay(p))+6.*dtp*(oday(p)+day(p))  )/dtp**3
c
         az2=(-6.*(oaz(p)-az(p))-dtp*(4.*odaz(p)+2.*daz(p)))/dtp**2
         az3=( 12*(oaz(p)-az(p))+6.*dtp*(odaz(p)+daz(p))  )/dtp**3
c
            x(p)=x(p)+(dtp**4)*(ax2/24. +dtp*ax3/120.)
            y(p)=y(p)+(dtp**4)*(ay2/24. +dtp*ay3/120.)
            z(p)=z(p)+(dtp**4)*(az2/24. +dtp*az3/120.)
c           write(*,*)'me,p,x(p),y(p),z(p)',me,p,x(p),y(p),z(p)
c
            vx(p)=vx(p)+(dtp**3)*(ax2*onesixth +dtp*ax3/24.)
            vy(p)=vy(p)+(dtp**3)*(ay2*onesixth +dtp*ay3/24.)
            vz(p)=vz(p)+(dtp**3)*(az2*onesixth +dtp*az3/24.)
c
c         write(*,*)'steppos,p,dtp,x,ax,ax2,ax3',p,dtp,x(p),ax(p),ax2,ax3
          end if
 10       CONTINUE
c         should add ki term        !by Baile

c         axbh2=(-6.*(oaxbh-axbh)-dtp*(4.*odaxbh+2.*daxbh))/dtp**2
c           axbh3=( 12*(oaxbh-axbh)+6.*dtp*(odaxbh+daxbh)  )/dtp**3

c         aybh2=(-6.*(oaybh-aybh)-dtp*(4.*odaybh+2.*daybh))/dtp**2
c           aybh3=( 12*(oaybh-aybh)+6.*dtp*(odaybh+daybh)  )/dtp**3
```

```
c       azbh2=(-6.*(oazbh-azbh)-dtp*(4.*odazbh+2.*dazbh))/dtp**2
c        azbh3=( 12*(oazbh-azbh)+6.*dtp*(odazbh+dazbh) )/dtp**3


c       xbh=xbh+(dtp**4)*(axbh2/24. +dtp*axbh3/120.)
c       ybh=ybh+(dtp**4)*(aybh2/24. +dtp*aybh3/120.)
c       zbh=zbh+(dtp**4)*(azbh2/24. +dtp*azbh3/120.)

c       vxbh=vxbh+(dtp**3)*(axbh2*onesixth +dtp*axbh3/24.)
c       vybh=vybh+(dtp**3)*(aybh2*onesixth +dtp*aybh3/24.)
c       vzbh=vzbh+(dtp**3)*(azbh2*onesixth +dtp*azbh3/24.)
        sx=sx+asx*dtp
        sy=sy+asy*dtp
        sz=sz+asz*dtp

c       akix2=(-6.*(oakix-akix)-dtp*(4.*odakix+2.*dakix))/dtp**2
c        akix3=( 12*(oakix-akix)+6.*dtp*(odakix+dakix))/dtp**3

c       akiy2=(-6.*(oakiy-akiy)-dtp*(4.*odakiy+2.*dakiy))/dtp**2
c        akiy3=( 12*(oakiy-akiy)+6.*dtp*(odakiy+dakiy))/dtp**3

c       akix2=(-6.*(oakiz-akiz)-dtp*(4.*odakiz+2.*dakiz))/dtp**2
c        akiz3=( 12*(oakiz-akiz)+6.*dtp*(odakiz+dakiz))/dtp**3


c       kix=kix+(dtp**3)*(akix2*onesixth +dtp*akix3/24.)
c       kiy=kiy+(dtp**3)*(akiy2*onesixth +dtp*akiy3/24.)
c       kiz=kiz+(dtp**3)*(akiz2*onesixth +dtp*akiz3/24.)
c       write(*,*)'steppos dtp',dtp
c       !end adding
        if(mstpflag.and.rtag.eq.4) then
        tpos=tpos+dtp
```

```
      tnow=tpos
      end if
      RETURN
      END
C***********************************************************************
C
C
                     SUBROUTINE steppred
C
C
C***********************************************************************

      INCLUDE 'tmhscf.h'

C   Declaration of local variables.
C   --------------------------------

      INTEGER i
      REAL*8 dtp,kiabs

C=======================================================================
C
C      advance particles to x_p with old acceleration
C

      DO 11 i=1,nbodies

      IF (mstpflag.AND.(im(i).eq.rtag)) then
       dtp=dtbin(rtag)

 999          x(i)=((dtp**3)*odax(i)*onesixth+0.5*dtp*dtp*oax(i)
     &          +dtp*vx(i))+x(i)
```

```
              y(i)=((dtp**3)*oday(i)*onesixth+0.5*dtp*dtp*oay(i)
     &             +dtp*vy(i))+y(i)
              z(i)=((dtp**3)*odaz(i)*onesixth+0.5*dtp*dtp*oaz(i)
     &             +dtp*vz(i))+z(i)

              vx(i)=(0.5*dtp*dtp*odax(i)  +dtp*oax(i))+vx(i)
              vy(i)=(0.5*dtp*dtp*oday(i)  +dtp*oay(i))+vy(i)
              vz(i)=(0.5*dtp*dtp*odaz(i)  +dtp*oaz(i))+vz(i)
c         write(*,*)'steppred'
c         write(*,*)vx(i),vy(i),vz(i)
          end if
 11       CONTINUE
c !added by Baile
c         xbh=((dtp**3)*odaxbh*onesixth+0.5*dtp*dtp*oaxbh
c    &             +dtp*vxbh)+xbh
c          ybh=((dtp**3)*odaybh*onesixth+0.5*dtp*dtp*oaybh
c    &             +dtp*vybh)+ybh
c         zbh=((dtp**3)*odazbh*onesixth+0.5*dtp*dtp*oazbh
c    &             +dtp*vzbh)+zbh

c               vxbh=(0.5*dtp*dtp*odaxbh  +dtp*oaxbh)+vxbh
c              vybh=(0.5*dtp*dtp*odaybh  +dtp*oaybh)+vybh
c              vzbh=(0.5*dtp*dtp*odazbh  +dtp*oazbh)+vzbh
c         write(*,*)'steppred oakix y z'
c         write(*,*)oakix,oakiy,oakiz
           sx=sx+oasx*dtp
           sy=sy+oasy*dtp
           sz=sz+oasz*dtp

c         kix=(0.5*dtp*dtp*odakix+dtp*oakix)+kix
c         kiy=(0.5*dtp*dtp*odakiy+dtp*oakiy)+kiy
c         kiz=(0.5*dtp*dtp*odakiz+dtp*oakiz)+kiz
```

```
c          write(*,*)'steppred dtp',dtp
           RETURN
           END
C*****************************************************************
C
C
                            SUBROUTINE stepsys(n)
C
C
C*****************************************************************
C
C
C      Subroutine to advance the state of the system by one timestep.
C
C
C=================================================================

           INCLUDE 'tmhscf.h'
           include 'mpif.h'

C    Declaration of local variables.
C    ------------------------------
           INTEGER i,n,stepnum(3),k
           REAL*8 tnext




          IF (multistep) THEN
             mstpflag=.TRUE.
           do i=1,3
           stepnum(i)=dtbin(i+1)/dtbin(i)
          end do
c          do i=1,4
c           write(*,*)'i,nbin(i)',i,nbin(i)
c          end do
          rtag=1
          do while(rtag.gt.0.and.rtag.le.4)
          if(rtag.eq.1) then

            DO j=1,stepnum(1)
c           write(*,*)'rtag,first',rtag

              CALL accpot(n)
           IF (fixacc) CALL corracc
            DO 1020 i = 1,nbodies
                 IF (im(i).EQ.rtag) THEN
                    oax(i) = ax(i)
                    oay(i) = ay(i)
                    oaz(i) = az(i)
                    odax(i) = dax(i)
                    oday(i) = day(i)
                    odaz(i) = daz(i)
                 ENDIF
 1020         CONTINUE
              oasx=asx
              oasy=asy
              oasz=asz
              CALL steppred
              CALL accpot(n)
              IF (fixacc) CALL corracc
              CALL steppos
              CALL timestep(n)
              IF (usedrag) CALL veldrag
              if (multistep) CALL setim

            numb(rtag)=numb(rtag)+1
            end do
            rtag=rtag+1
            end if



c             write(*,*)'rtag,numb(rtag-1),stepnum(rtag-1)',rtag,numb(rtag-1),
c        &stepnum(rtag-1)
            if((rtag.gt.1).and.(numb(rtag-1).gt.0).and.
        &(mod(numb(rtag-1),stepnum(rtag-1)).eq.0)) then
c             write(*,*)'rtag,secondpart',rtag
               CALL accpot(n)
             IF (fixacc) CALL corracc
              DO 1021 i = 1,nbodies
                 IF (im(i).EQ.rtag) THEN
                    oax(i) = ax(i)
                    oay(i) = ay(i)
                    oaz(i) = az(i)
                    odax(i) = dax(i)
                    oday(i) = day(i)
                    odaz(i) = daz(i)
                 ENDIF
 1021          CONTINUE
               oasx=asx
               oasy=asy
               oasz=asz
               CALL steppred
               CALL accpot(n)
               IF (fixacc) CALL corracc
               CALL steppos
```

```
                CALL timestep(n)
                IF (usedrag) CALL veldrag
                if (multistep) CALL setim

            numb(rtag)=numb(rtag)+1
            if(rtag.lt.4) then
            rtag=rtag+1
            else
            goto 999
            end if

            else
            rtag=1
            end if

            end do
            ENDIF
999         CALL outstate(n)
            RETURN
            END

C*********************************************************************
C
C
                          SUBROUTINE stopout
C
C
C*********************************************************************
C
C
C       Subroutine to end output.
C
```

```
C
C==================================================================

      INCLUDE 'tmhscf.h'
      include 'mpif.h'

      INTEGER i
      REAL*8 t1,c1
     data t1,c1/0.0d0,0.0d0/
     save t1,c1

C==================================================================

      IF (me .eq. 0) THEN
     t1=t1+totime1-totime0
     c1=c1+cputime1-cputime0
        WRITE(ulog,30) (cputime1-cputime0)
        WRITE(ulog,40) (totime1-totime0)
 30     FORMAT(' CPU time (without I/O) used =',1pe18.10,'(secs)')
 40     FORMAT(' Total time (with I/O)  used =',1pe18.10,'(secs)')
     write(ulog,*) 'cumulative times=',c1,t1
        CLOSE(ulog)
      ENDIF

      RETURN
      END
C*****************************************************************
C
C
                    SUBROUTINE terror(message)
C
C
```

```
C*********************************************************************
C
C
C      Subroutine to terminate the program as the result of a fatal
C      error, close the output files, and dump timing information.
C
C
C=====================================================================

        INCLUDE 'tmhscf.h'
      include 'mpif.h'

C   Declaration of local variables.
C   ---------------------------------

        CHARACTER*(*) message
        INTEGER ierror

C=====================================================================

C   Write error message to the log file and to the terminal.
C   -----------------------------------------------------------
        ierror=-1

        CALL outterm(message,ierror)
C            -------

C---------------------------------------------------------------------
C   Stop timing, output timing data, close files, terminate the
C   simulation.
C---------------------------------------------------------------------
```

```
      cputime1=mpi_wtime()

      STOP
      END

*********************************************************************
C
C
                         SUBROUTINE timestep(n)
C
C
C*********************************************************************
C
C
C     Subroutine to calculate the minimum time step for each group.
C
C
C=====================================================================

      INCLUDE 'tmhscf.h'
      INCLUDE 'mpif.h'

C    Declaration of local variables.
C    ------------------------------

      INTEGER p
      integer n
      REAL*8 dtsold,dtlilold
      REAL*8 as,das,dtsmin,dtlilmin
ccompaq      tmpmin replaced with tmpmin_snd and tmpmin_rcv
      REAL*8 tmpmin_snd(3), tmpmin_rcv(3)
      integer i
```

```
c          dtsold=1d20
c          dtlilold=1d20
cc         dtsold=1.
cc            dtlilold=1.



cc         DO 10 p=1,nbodies
cc
cc              if (mstpflag.AND.(im(p).eq.1)) go to 10

cc               as = (ax(p)*ax(p)+ay(p)*ay(p)+az(p)*az(p))
cc               das = (dax(p)*dax(p)+day(p)*day(p)+daz(p)*daz(p))

cc            dts = SQRT(as/das)
cc            dti(p) = dts
cc            dtlil=MIN(dts,dtlilold)
cc            dts = dts + (1-im(p))
cc            dts = MIN(dts,dtsold)

cc            dtsold = dts
cc            dtlilold = dtlil

cc 10      CONTINUE

c now compute minimum of dts and dtlil over all processors
ccompaq      tmpmin replaced with tmpmin_snd
cc            tmpmin_snd(1)=dts
cc            tmpmin_snd(2)=dtlil
ccompaq      tmpmin replaced with tmpmin_snd and tmpmin_rcv in MPI call
```

```
cc          call MPI_Allreduce (tmpmin_snd,tmpmin_rcv, 2,
cc     &                     mpi_double_precision,
cc     &                     mpi_min, mpi_comm_world,ierr)
ccompaq     tmpmin replaced with tmpmin_rcv
cc          dts = tmpmin_rcv(1)
cc          dtlil = tmpmin_rcv(2)

C    Update position time, system time.
C    ---------------------------------------
cc          dtsmall = dteps*dtlil
cc          if (.not.mstpflag) then
cc            dtbig    = dteps*dts
cc            dtime    = dtbig
cc            tnextbig= tnow + dtbig
cc          endif
c         if(me.eq.0) write(*,*)dtime
C          if (me.eq.0) then
C             if(mod(n,2000).eq.0)then
C                 write(utermfil,*) tnow, dtbig, dtsmall
C             endif
C          endif
          data dtbin/1.0,10,100,1000/
c          data dtbin/1.0,2.0,4.0,8.0/



c         do i=1,3
c          dtbin(i)=2**(i-1)
c          write(*,*)'i,dtbin',i,dtbin(i)
c          tnextbig(i)=tnow+dtbin(i)
c          end do
c          tmpmin_snd(1)=dtbin(1)
c          tmpmin_snd(2)=dtbin(2)
```

```
c          tmpmin_snd(3)=dtbin(3)
c       tmpmin replaced with tmpmin_snd and tmpmin_rcv in MPI call
c          call MPI_Allreduce (tmpmin_snd,tmpmin_rcv, 3,
c       &                      mpi_double_precision,
c       &                      mpi_min, mpi_comm_world,ierr)
c       tmpmin replaced with tmpmin_rcv
c          dtbin(1) = tmpmin_rcv(1)
c          dtbin(2) = tmpmin_rcv(2)
c          dtbin(3) = tmpmin_rcv(3)
         call share(dtbin,4)

         RETURN
         END

C**************************************************
C***************************************************************
C
C
                    SUBROUTINE veldrag
C
C
C***************************************************************
C
C
C       Subroutine to advance the velocities of the bodies for timestep.
C
C
C===============================================================

         INCLUDE 'tmhscf.h'

C    Declaration of local variables.
```

```
C    ------------------------------------
         INTEGER p
         REAL*8 tnowdrag, facd, temp4, temp5, temp6, dtp
C
C=======================================================================

         facd=0.0

          tnowdrag=tnow-tstartdrag

            IF(tnowdrag.GE.0.0.AND.tnowdrag.LT.tgrowdrag) facd=(3.*(
     &         tnowdrag/tgrowdrag)**2-2.*(tnowdrag/tgrowdrag)**3)

            IF(tnowdrag.GT.tgrowdrag.AND.tnowdrag.LE.tlivedrag+tgrowdrag)
     &        facd = 1.

            IF(tnowdrag.GT.tlivedrag+tgrowdrag.AND.tnowdrag.LE.tdiedrag+
     &         tlivedrag+tgrowdrag) facd=(1.-3.*((tnowdrag-tlivedrag-
     &         tgrowdrag)/tdiedrag)**2+2.*((tnowdrag-tlivedrag-tgrowdrag)
     &         /tdiedrag)**3)

            IF(tnowdrag.GT.tdiedrag+tlivedrag+tgrowdrag) facd=0.0



            temp4=facd*xdrag
            temp5=facd*ydrag
            temp6=facd*zdrag

         IF (.NOT.multistep) dtbig=dtime

         do 2030 p = 1,nbodies
           IF (mstpflag.AND.(im(p).EQ.1)) GO TO 2030
          dtp=dtime*(1-im(p))+dtbig*im(p)

             vx(p)=(vx(p)*(1.0-0.5*dtp*temp4))/
     &                 (1.0+0.5*dtp*temp4)
             vy(p)=(vy(p)*(1.0-0.5*dtp*temp5))/
     &                 (1.0+0.5*dtp*temp5)
             vz(p)=(vz(p)*(1.0-0.5*dtp*temp6))/
     &                 (1.0+0.5*dtp*temp6)

2030   continue



        RETURN
        END

      subroutine share(x,n)
      include 'mpif.h'
      dimension x(n)
      call MPI_BCAST(x,n,MPI_DOUBLE_PRECISION,0,
     $                   MPI_COMM_WORLD,ierr)



      return
      end
      subroutine shareint(x,n)
      include 'mpif.h'
      integer x(n)
      call MPI_BCAST(x,n,MPI_INTEGER,0,
     $                   MPI_COMM_WORLD,ierr)
      return
      end
```

```
  subroutine sharelog(x,n)
  include 'mpif.h'
  logical x(n)
  call MPI_BCAST(x,n,MPI_LOGICAL,0,
$                       MPI_COMM_WORLD,ierr)
  return
  end
  subroutine mpiget(rbuf,sbuf,n,idest,isrc,me)
  include 'mpif.h'
  integer status(MPI_STATUS_SIZE)
  dimension rbuf(n),sbuf(n)
  itag=isrc
  if(me.eq.isrc) then
  call mpi_send(sbuf,n,mpi_double_precision,
&        idest,itag,mpi_comm_world,ierr)
  endif
  if(me.eq.idest) then
  call mpi_recv(rbuf,n,mpi_double_precision,
&        isrc,itag,mpi_comm_world,status,ierr)
  endif
  return
  end
  subroutine mpigetint(rbuf,sbuf,n,idest,isrc,me)
  include 'mpif.h'
  integer status(MPI_STATUS_SIZE)
  integer rbuf(n),sbuf(n)
  itag=isrc
  if(me.eq.isrc) then
  call mpi_send(sbuf,n,mpi_integer,
&        idest,itag,mpi_comm_world,ierr)
  endif
  if(me.eq.idest) then
```

```
      call mpi_recv(rbuf,n,mpi_integer,
     &      isrc,itag,mpi_comm_world,status,ierr)
      endif
      return
      end

c !the following is all added by Baile
        subroutine pn1(n)
      implicit none
c this is a first post newtonian code supposed to be added to mpiscf.f

         INCLUDE 'tmhscf.h'
         include 'mpif.h'
c Declaration of local variables
         integer n
          real*8 tempt1,tempt2,tempt3,tempt4,tempt5,tempt6,tempt7,
     &tempt8,tempt9,tempt10,tempt11,tempt12,tempt13,tempt14,tempt15,
     &tempt16,tempt17,tempt18,tempt19,tempt20,tempt21,tempt22,
     &tempt23,tempt24,tempt25,tempt26,
     &tempt27,tempt28,tempt29,
     &tempt30,tempt31,tempt32,tempt33,tempt34,tempt35,tempt36,
     &tempt37,tempt38,tempt39,tempt40,tempt41,tempt42,tempt43,
     &tempt44,tempt45,tempt46,tempt47,tempt48,tempt49,tempt50

        integer j,i
        real*8 dtp,miu,m,eta,delm,deltax,deltay,
     &deltaz,coef1,coef2,ratio1(nbodsper),co1,co2,co3,co4,coj1

        real*8 r
        data asx,asy,asz/0.0d0,0.0d0,0.0d0/
c       data dasx,dasy,dasz/0.0d0,0.0d0,0.0d0/
```

```
        DO  j=1,nbodies
 143    format(6(1pe20.10))
        IF ((mstpflag.AND.im(j).EQ.rtag) .or. n.eq.0 ) THEN
        dtp=dtbin(rtag)

        call  nij(x(j),y(j),z(j),xbh,ybh,zbh,n1jx(j),n1jy(j),n1jz(j))
c           call  rel(vx(j),vy(j),vz(j),vxbh,vybh,vzbh,
c      &v1jx(j),v1jy(j),v1jz(j))
        call cross(n1jx(j),n1jy(j),n1jz(j),v1jx(j),
     &v1jy(j),v1jz(j),tempt1,tempt2,tempt3)
c       rlength(j)=r(xbh,ybh,zbh,x(j),y(j),z(j))+epsbh
c       vlength(j)=sqrt(v1jx(j)**2+v1jy(j)**2+v1jz(j)**2)
        call dot(n1jx(j),n1jy(j),n1jz(j),v1jx(j),
     &v1jy(j),v1jz(j),rdot(j))

        rcvx(j)=rlength(j)*tempt1 ! x component of r cross v
        rcvy(j)=rlength(j)*tempt2
        rcvz(j)=rlength(j)*tempt3



        miu=mass(j)*bhmass/(mass(j)+bhmass)
        m=mass(j)+bhmass
        eta=miu/m
        delm=bhmass-mass(j)
        deltax=m*(-sx/bhmass)
        deltay=m*(-sy/bhmass)
        deltaz=m*(-sz/bhmass)
        coef1=(1+3*eta)*vlength(j)**2-2*(2+eta)*m/rlength(j)-
     &3./2.*eta*rdot(j)**2
        coef2=2*(2-eta)*rdot(j)
        call rel(coef1*n1jx(j),coef1*n1jy(j),coef1*n1jz(j),
     &coef2*v1jx(j),coef2*v1jy(j),coef2*v1jz(j),tempt4,tempt5,tempt6)
```

```
          a1pnx(j)=-m/rlength(j)**2*tempt4
          a1pny(j)=-m/rlength(j)**2*tempt5
          a1pnz(j)=-m/rlength(j)**2*tempt6

c         write(*,*)'a1pnx,a1pny,a1pnz'
c         write(*,*)a1pnx(j),a1pny(j),a1pnz(j)

c         write(*,*)'apnx,apny,apnz',j,apnx(j),apny(j),apnz(j)

          call dot(tempt1,tempt2,tempt3,2*sx+delm/m*deltax,
     &2*sy+delm/m*deltay,2*sz+delm/m*deltaz,tempt7)
          call cross(v1jx(j),v1jy(j),v1jz(j),7*sx+3*delm/m*deltax,
     &7*sy+3*delm/m*deltay,7*sz+3*delm/m*deltaz,tempt8,tempt9,tempt10)
          call cross(n1jx(j),n1jy(j),n1jz(j),3*sx+delm/m*deltax,
     &3*sy+delm/m*deltay,3*sz+delm/m*deltaz,tempt11,tempt12,tempt13)
        asox(j)=1/rlength(j)**3*(6*tempt7*n1jx(j)-tempt8+
     &3*rdot(j)*tempt11)
        asoy(j)=1/rlength(j)**3*(6*tempt7*n1jy(j)-tempt9+
     &3*rdot(j)*tempt12)
        asoz(j)=1/rlength(j)**3*(6*tempt7*n1jz(j)-tempt10+
     &3*rdot(j)*tempt13)
c          write(*,*)'asox,asoy,asoz',j,asox(j),asoy(j),asoz(j)

c         ax(j)=ax(j)+a1pnx(j)+asox(j)
c         ay(j)=ay(j)+a1pny(j)+asoy(j)
c         az(j)=az(j)+a1pnz(j)+asoz(j)

          ax(j)=ax(j)+a1pnx(j)
          ay(j)=ay(j)+a1pny(j)
         az(j)=az(j)+a1pnz(j)

c********************************* below to calculate da1pn**********
```

```
      call dot(ax(j),ay(j),az(j),vx(j)/vlength(j),vy(j)/vlength(j),
     &vz(j)/vlength(j),vdot(j))
      call rel(vx(j),vy(j),vz(j),rdot(j)*n1jx(j),rdot(j)*n1jy(j),
     &rdot(j)*n1jz(j),tempt17,tempt18,tempt19)
      ndotx(j)=tempt17/rlength(j)
      ndoty(j)=tempt18/rlength(j)
      ndotz(j)=tempt19/rlength(j)

      call dot(ax(j),ay(j),az(j),n1jx(j),n1jy(j),n1jz(j),tempt20)
      call dot(vx(j),vy(j),vz(j),ndotx(j),ndoty(j),ndotz(j),tempt21)
      rddot(j)=tempt20+tempt21

      co1=2*m*rdot(j)/rlength(j)**3*((1+3*eta)*vlength(j)**2-2*(2+eta)
     &*m/rlength(j)-3./2.*eta*rdot(j)**2)-m/rlength(j)**2*((1+3*eta)
     &*2*vlength(j)*vdot(j)+2*(2+eta)*m*rdot(j)/rlength(j)**2
     &-3*eta*rdot(j)*rddot(j))
      co2=2*m/rlength(j)**2*(2-eta)*rddot(j)-2*m/
     &rlength(j)**3*rdot(j)**2*2*(2-eta)
      co3=2*m/rlength(j)**2*(2-eta)*rdot(j)
      co4=-m/rlength(j)**2*((1+3*eta)*vlength(j)**2-2*(2+eta)*
     &m/rlength(j)-3./2.*eta*rdot(j)**2)
      da1pnx(j)=co1*n1jx(j)+co2*vx(j)+co3*ax(j)+co4*ndotx(j)
      da1pny(j)=co1*n1jy(j)+co2*vy(j)+co3*ay(j)+co4*ndoty(j)
      da1pnz(j)=co1*n1jz(j)+co2*vz(j)+co3*az(j)+co4*ndotz(j)

      dax(j)=dax(j)+da1pnx(j)
      day(j)=day(j)+da1pny(j)
      daz(j)=daz(j)+da1pnz(j)
c     write(*,*)'da1pnx,da1pny,da1pnz'
c     write(*,*)da1pnx(j),da1pny(j),da1pnz(j)
      mor(j)=m/rlength(j)
      E1(j)=1.D0/2.D0*mor(j)**2.D0+3.D0/8.D0*(1.D0-3.D0*eta)
     &*vlength(j)**4.D0+
     &1.D0/2.D0*(3.D0+eta)*vlength(j)**2.D0*mor(j)+
     &1.D0/2.D0*eta*mor(j)*rdot(j)**2.D0
        E1(j)=E1(j)*miu
c         write(*,*)'me,j,E1(j),x(j)',me,j,E1(j),x(j)

        coj1=(3.+eta)*mor(j)+1./2.*(1-3.*eta)*vlength(j)**2
        J1x(j)=miu*coj1*rcvx(j)
        J1y(j)=miu*coj1*rcvy(j)
        J1z(j)=miu*coj1*rcvz(j)

        J0x(j)=miu*1.0*rcvx(j)
        J0y(j)=miu*1.0*rcvy(j)
        J0z(j)=miu*1.0*rcvz(j)

c       write(*,*)'J0x(j),J0y(j),J0z(j)'
c         write(*,*)J0x(j),J0y(j),J0z(j)

c*********************end calculating da1pn***************************

c       write(*,*)'ax,ay,az',j,ax(j),ay(j),az(j)
        ratio1(j)=sqrt((a1pnx(j)+asox(j))**2+(a1pny(j)+asoy(j))**2+
     &(a1pnz(j)+asoz(j))**2)/sqrt(ax(j)**2+ay(j)**2+az(j)**2)
c       ratio1(j)=(a1pnx(j)+asox(j))/(ax(j)-a1pnx(j)-asox(j))
        if (abs(ratio1(j)).gt.abs(maxratio(j))) then
        maxratio(j)=ratio1(j)
c       rleng(j)=rlength
c       vleng(j)=vlength
        end if



        call cross(tempt1,tempt2,tempt3,sx,sy,sz,tempt14,tempt15,tempt16)
```

```
c          write(*,*)'j,tempt123tob',j,tempt1,tempt2,tempt3,torbit(j)
c          if(torbit(j).gt.0.0) then
c          asx=asx+1/rlength(j)**2*miu*(2+3./2.*mass(j)/bhmass)*
c        &tempt14*dtp/torbit(j)
c          asy=asy+1/rlength(j)**2*miu*(2+3./2.*mass(j)/bhmass)*
c        &tempt15*dtp/torbit(j)
c          asz=asz+1/rlength(j)**2*miu*(2+3./2.*mass(j)/bhmass)*
c        &tempt16*dtp/torbit(j)
           asx=0.0
           asy=0.0
           asz=0.0
c          end if
c          write(*,*)'asx,asy,asz',j,asx,asy,asz
           end if
            end do

c          write(*,*)'dtp in pn1'
c          write(*,*)dtp

c          write(*,*)'asx asy asz'
c           write(*,*)asx,asy,asz




c          write(*,*)'tnow,kix kiy kiz asx asy asz dasx dasy dasz'
c          write(*,*)tnow,kix,kiy,kiz,asx,asy,asz,dasx,dasy,dasz
                return
                 end


                 subroutine dot(x1,y1,z1,x2,y2,z2,ji)
           include 'mpif.h'
           real*8 x1,y1,z1,x2,y2,z2,ji
```

```
ji=x1*x2+y1*y2+z1*z2
     return
     end

     subroutine cross(x1,y1,z1,x2,y2,z2,xc,yc,zc)
include 'mpif.h'
real*8  x1,y1,z1,x2,y2,z2,xc,yc,zc
xc=y1*z2-y2*z1
yc=x2*z1-x1*z2
zc=x1*y2-x2*y1
     return
     end

     function r(x1,y1,z1,x2,y2,z2)
include 'mpif.h'
real*8 x1,y1,z1,x2,y2,z2,r
r=sqrt((x1-x2)**2+(y1-y2)**2+(z1-z2)**2)
     return
     end

     subroutine nij(x1,y1,z1,x2,y2,z2,x12,y12,z12)
include 'mpif.h'
real*8 x1,y1,z1,x2,y2,z2,x12,y12,z12,r
x12=(x1-x2)/r(x1,y1,z1,x2,y2,z2)
y12=(y1-y2)/r(x1,y1,z1,x2,y2,z2)
z12=(z1-z2)/r(x1,y1,z1,x2,y2,z2)
     return
     end
   subroutine rel(x1,y1,z1,x2,y2,z2,x12,y12,z12)
include 'mpif.h'
real*8 x1,y1,z1,x2,y2,z2,x12,y12,z12
x12=x1-x2
```

```
      y12=y1-y2
      z12=z1-z2
        return
        end



      subroutine pn2h(n)
      INCLUDE 'tmhscf.h'
        include 'mpif.h'
      integer j,i,n
      real*8 dtp,miu,m,eta,delm,deltax,deltay,A2hdot,B2hdot,A2h,B2h,
     &deltaz,coef1,coef2,co1,co2,co3,co4,tempt1,tempt2,tempt3,r

      do j=1,nbodies
      IF((mstpflag.AND.im(j).EQ.rtag) .or. n.eq.0 ) THEN

      miu=mass(j)*bhmass/(mass(j)+bhmass)
      m=mass(j)+bhmass
      eta=miu/m

      coef1=8./5.*eta*m/rlength(j)*rdot(j)*(17./3.*m/rlength(j)
     &+3*vlength(j)**2)
      coef2=8./5.*eta*m/rlength(j)*(3*m/rlength(j)+vlength(j)**2)
      call rel(n1jx(j)/rlength(j)**2*coef1,n1jy(j)/
     &rlength(j)**2*coef1,
     &n1jz(j)/rlength(j)**2*coef1,vx(j)/rlength(j)**2*coef2,
     &vy(j)/rlength(j)**2*coef2,vz(j)/rlength(j)**2*coef2,
     &tempt1,tempt2,tempt3)
      a2hpnx(j)=tempt1*m
      a2hpny(j)=tempt2*m
      a2hpnz(j)=tempt3*m
      ax(j)=ax(j)+a2hpnx(j)
      ay(j)=ay(j)+a2hpny(j)
      az(j)=az(j)+a2hpnz(j)



      A2h=coef1
      B2h=-coef2
      A2hdot=1.6*eta*(17./3.*m/rlength(j)+3*vlength(j)**2)
     &*(m*rddot(j)/rlength(j)-m*rdot(j)**2/rlength(j)**2)
     &+1.6*eta*m*rdot(j)/rlength(j)*(-17*m*rdot(j)/(3*rlength(j)**2)
     &+6*vlength(j)*vdot(j))
      B2hdot=1.6*eta*m/rlength(j)*
     &((3*m/rlength(j)+vlength(j)**2)*rdot(j)/rlength(j)
     &+3*m*rdot(j)/rlength(j)**2-2*vlength(j)*vdot(j))
      co1=A2hdot-2./rlength(j)*A2h*rdot(j)
      co2=B2hdot-2./rlength(j)*B2h*rdot(j)
      co3=A2h
      co4=B2h
      da2hpnx(j)=m/rlength(j)**2*(co1*n1jx(j)+co2*vx(j)+
     &co3*ndotx(j)+co4*ax(j))
      da2hpny(j)=m/rlength(j)**2*(co1*n1jy(j)+co2*vy(j)+
     &co3*ndoty(j)+co4*ay(j))
      da2hpnz(j)=m/rlength(j)**2*(co1*n1jz(j)+co2*vz(j)+
     &co3*ndotz(j)+co4*az(j))



      dax(j)=dax(j)+da2hpnx(j)
      day(j)=day(j)+da2hpny(j)
      daz(j)=daz(j)+da2hpnz(j)

c     write(*,*)'a2hpnx,a2hpny,a2hpnz,da2hpnx,da2hpny,da2hpnz'
c       write(*,*)a2hpnx(j),a2hpny(j),a2hpnz(j),
c    &da2hpnx(j),da2hpny(j),da2hpnz(j)
```

```
      end if
      end do
      return
      end

      subroutine pn3(n)
      INCLUDE 'tmhscf.h'
       include 'mpif.h'
      integer j,i,n
      real*8 dtp,miu,m,eta,delm,deltax,deltay,A3dot,B3dot,A3,B3,
     &deltaz,coef1,coef2,co1,co2,co3,co4,tempt1,tempt2,tempt3,r,coj3

      do j=1,nbodies
      IF((mstpflag.AND.im(j).EQ.rtag) .or. n.eq.0) THEN
c       write(*,*)'pn3,me,j,x(j)',me,j,x(j)
      miu=mass(j)*bhmass/(mass(j)+bhmass)
      m=mass(j)+bhmass
      eta=miu/m



      coef1=-(16.D0+(1399./12.-41/16*pi**2)*eta+71./2.*eta**2)
     &*m**3/rlength(j)**3
     &-eta*(20827./840.+123./64.*pi**2-eta**2)*m**2/
     &rlength(j)**2*vlength(j)**2
     &+(1+(22717./168.+615./64.*pi**2)*eta+
     &11./8.*eta**2-7*eta**3)*m**2/rlength(j)**2*rdot(j)**2+
     &eta/4.*(11-49*eta+52*eta**2)*vlength(j)**6-
     &35/16.*eta*(1-5*eta+5*eta**2)*rdot(j)**6+
     &eta/4.*(75+32*eta-40*eta**2)*m/rlength(j)*vlength(j)**4+
     &eta/2*(158-69*eta-60*eta**2)*m/rlength(j)*rdot(j)**4-
     &eta*(121-16*eta-20*eta**2)*m/rlength(j)*vlength(j)**2*rdot(j)**2-
```

```
&3./8.*eta*(20-79*eta+60*eta**2)*vlength(j)**4*rdot(j)**2+
&15./8.*eta*(4-18*eta+17*eta**2)*vlength(j)**2*rdot(j)**4
   coef2=rdot(j)*((4+(5849./840.+123./32.*pi**2)*eta-
&25*eta**2-8*eta**3)*m**2/rlength(j)**2
&+eta/8*(65-152*eta-48*eta**2)*vlength(j)**4+15./8.*eta*
&(3-8*eta-2*eta**2)*rdot(j)**4
&+eta*(15+27*eta+10*eta**2)*m/rlength(j)*vlength(j)**2-eta/
&6*(329+177*eta+108*eta**2)*m/rlength(j)*rdot(j)**2
& -3./4.*eta*(16-37*eta-16*eta**2)*vlength(j)**2*rdot(j)**2)
   call rel(-n1jx(j)/rlength(j)**2*coef1,
&-n1jy(j)/rlength(j)**2*coef1,
&-n1jz(j)/rlength(j)**2*coef1,-vx(j)/rlength(j)**2*coef2,
&-vy(j)/rlength(j)**2*coef2,-vz(j)/rlength(j)**2*coef2,tempt1,
&tempt2,tempt3)



    a3pnx(j)=tempt1*m
   a3pny(j)=tempt2*m
   a3pnz(j)=tempt3*m
   ax(j)=ax(j)+a3pnx(j)
   ay(j)=ay(j)+a3pny(j)
   az(j)=az(j)+a3pnz(j)



   A3=-coef1
   B3=coef2
   A3dot=
&3.*eta*rdot(j)**5*rddot(j)/8.*(35-175.*eta+175.*eta**2)
&+eta*(2*rdot(j)**3*rddot(j)*vlength(j)**2+
&rdot(j)**4*vlength(j)*vdot(j))*(-15+135./2*eta-255./4*eta**2)
&+eta*(rdot(j)*rddot(j)*vlength(j)**4
&+2*rdot(j)**2*vlength(j)**2*vlength(j)*vdot(j))
&*(15-237./4*eta+45.*eta**2)
&+6*vlength(j)**4*vlength(j)*vdot(j)*eta*(-11/4+49./4*eta-
&13*eta**2)
&+m/rlength(j)*eta*(4*rdot(j)**3*rddot(j)*(-79+69./2*eta+
&30.*eta**2)
&+2*(rdot(j)*rddot(j)*vlength(j)**2+rdot(j)**2*vlength(j)*vdot(j))
&*(121-16.*eta-20.*eta**2)
&+4*vlength(j)**2*vlength(j)*vdot(j)*(-75./4-8.*eta+10.*eta**2))
&-m*rdot(j)/rlength(j)**2*eta*(rdot(j)**4*(-79+69./2*eta+
&30.*eta**2)
&+rdot(j)**2*vlength(j)**2*(121-16.*eta-20.*eta**2)
&+vlength(j)**4*(-75./4-8.*eta+10.*eta**2))
&-2*m**2*rdot(j)/rlength(j)**3*(rdot(j)**2*
&(-1-615.*pi**2/64*eta-22717./168*eta-11./8*eta**2+7*eta**3)
&+eta*vlength(j)**2*(20827./840+123.*pi**2/64-eta**2))
&+m**2/rlength(j)**2*(2*rdot(j)*rddot(j)*
&(-1-615.*pi**2/64*eta-22717./168*eta-11./8*eta**2+7*eta**3)
&+2*eta*vlength(j)*vdot(j)*(20827./840+123.*pi**2/64-eta**2))
&-3*m**3*rdot(j)/rlength(j)**4*
&(16+eta*(1399./12-41.*pi**2/16)+71./2*eta)

  B3dot=75.*rdot(j)**4*rddot(j)*eta*(3./8-eta-1./4*eta**2)
&+eta*(3.*rdot(j)**2*rddot(j)*vlength(j)**2+
&2*rdot(j)**3*vlength(j)*vdot(j))*(-12-111./4*eta+12.*eta**2)
&+eta*(rddot(j)*vlength(j)**4
&+4*rdot(j)*vlength(j)**2*vlength(j)*vdot(j))*
&(65./8-19.*eta-6*eta**2)
&-m*rdot(j)/rlength(j)**2
&*(rdot(j)**3*eta*(-329./6-59./2*eta-18*eta**2)
&+rdot(j)*vlength(j)**2*eta*(15+27.*eta+10.*eta**2))
&+m/rlength(j)*eta*(3*rdot(j)**2*rddot(j)*
     &(-329./6-59./2*eta-18*eta**2)+
     &(rddot(j)*vlength(j)**2
     &+2*rdot(j)*vlength(j)*vdot(j))*(15+27.*eta+10.*eta**2))
     &-2*m**2*rdot(j)**2/rlength(j)**3*
     &(4+123.*pi**2/32*eta+5849./840*eta-25.*eta**2-8.*eta**3)
     &+m**2*rddot(j)/rlength(j)**2*
     &(4+123.*pi**2/32*eta+5849./840*eta-25.*eta**2-8.*eta**3)


        co1=A3dot-2./rlength(j)*A3*rdot(j)
        co2=B3dot-2./rlength(j)*B3*rdot(j)
        co3=A3
        co4=B3
        da3pnx(j)=m/rlength(j)**2*(co1*n1jx(j)+co2*vx(j)+
     &co3*ndotx(j)+co4*ax(j))
        da3pny(j)=m/rlength(j)**2*(co1*n1jy(j)+co2*vy(j)+
     &co3*ndoty(j)+co4*ay(j))
        da3pnz(j)=m/rlength(j)**2*(co1*n1jz(j)+co2*vz(j)+
     &co3*ndotz(j)+co4*az(j))

        dax(j)=dax(j)+da3pnx(j)
        day(j)=day(j)+da3pny(j)
        daz(j)=daz(j)+da3pnz(j)
c       write(*,*)'a3pnx,a3pny,a3pnz,da3pnx,da3pny,da3pnz'
c       write(*,*)a3pnx(j),a3pny(j),a3pnz(j),da3pnx(j),
c     &da3pny(j),da3pnz(j)
        E3(j)=(3.D0/8.D0+18469.D0/840.D0*eta)*mor(j)**4.D0+
     &(5.D0/4.D0-(6747.D0/280.D0
     &-41.D0/64.D0*pi**2.D0)*eta
     &-21.D0/4.D0*eta**2.D0+1.D0/2.D0*eta**3.D0)
     &*mor(j)**3.D0*vlength(j)**2.D0+
     &(3.D0/2.D0+(2321.D0/280.D0-123.D0/64.D0*pi**2.D0)
     &*eta+51.D0/4.D0*eta**2.D0+7.D0/2.D0*eta**3.D0)
     &*mor(j)**3.D0*rdot(j)**2.D0+
     &1.D0/128.D0*(35.D0-413.D0*eta+1666.D0*eta**2.D0-
     &2261.D0*eta**3.D0)*vlength(j)**8.D0+
     &1.D0/16.D0*(135.D0-194.D0*eta+406.D0*eta**2.D0-108.D0*eta**3.D0)
     &*mor(j)**2.D0*vlength(j)**4.D0
     &+1.D0/16.D0*(12.D0+248.D0*eta-815.D0*eta**2.D0-324.D0*eta**3.D0)
     &*mor(j)**2.D0*vlength(j)**2.D0
     &*rdot(j)**2.D0
     &-1.D0/48.D0*eta*(731.D0-492.D0*eta-288.D0*eta**2.D0)*
     &mor(j)**2.D0*rdot(j)**4.D0
     &+1.D0/16.D0*(55.D0-215.D0*eta+116.D0*eta**2.D0+325.D0*
     &eta**3.D0)*mor(j)*vlength(j)**6.D0
     &+1.D0/16.D0*eta*(5.D0-25.D0*eta+25.D0*eta**2.D0)*
     &mor(j)*rdot(j)**6.D0
     &-1.D0/16.D0*eta*(21.D0-75.D0*eta-375.D0*eta**2.D0)*
     &mor(j)*vlength(j)**4.D0*rdot(j)**2.D0
     &-1.D0/16.D0*eta*(9.D0-84.D0*eta+165.D0*eta**2.D0)
     &*mor(j)*vlength(j)**2.D0*rdot(j)**4.D0

        E3(j)=miu*E3(j)
c         write(*,*)'j,E3(j)',j,E3(j)
c         write(*,*)'me,j,E3(j),x(j)',me,j,E3(j),x(j)

        E(j)=E0(j)+E1(j)+E2(j)+E3(j)+mass(j)*pot(j)
c       E(j)=E0(j)
c        write(*,*)E(j),E0(j),E1(j),E2(j),E3(j)
        coj3=(5./2.-(5199./280.-41./32.*pi**2)*eta-
     &7.*eta**2+eta**3)*mor(j)**3+
     &1./16.*(5.-59.*eta+238.*eta**2-323.*eta**3)*vlength(j)**6+
     &1./12.*(135.-322.*eta+315.*eta**2-108.*eta**3)
     &*mor(j)**2*vlength(j)**2+
     &1./24.*(12.-287.*eta-951.*eta**2-324.*eta**3)*mor(j)**2*rdot(j)**2
     &+1./8.*(33.-142.*eta+106.*eta**2+195.*eta**3)*mor(j)*vlength(j)**4
     &-1./4.*eta*(12.-7.*eta-75.*eta**2)*mor(j)*vlength(j)**2*rdot(j)**2
     &+3./8.*eta*(2.-2*eta-11*eta**2)*mor(j)*rdot(j)**4
        J3x(j)=miu*coj3*rcvx(j)
        J3y(j)=miu*coj3*rcvy(j)
        J3z(j)=miu*coj3*rcvz(j)

        Jx(j)=J0x(j)+J1x(j)+J2x(j)+J3x(j)
        Jy(j)=J0x(j)+J1y(j)+J2y(j)+J3y(j)
        Jz(j)=J0x(j)+J1z(j)+J2z(j)+J3z(j)
c       write(*,*)tnow,E(j),Jx(j),Jy(j),Jz(j)

        h(j)=sqrt(Jx(j)**2+Jy(j)**2+Jz(j)**2)/(G*m)
c         write(*,*)'j.h(j),E(j)',j,h(j),E(j)
c       write(*,*)'Jx(j),Jy(j),Jz(j)'
c       write(*,*)Jx(j),Jy(j),Jz(j)

        er(j)=1+2*E(j)*h(j)**2+(-2*E(j))/(4*C**2)*(24.-4.*eta+
     &5*(-3.+eta)*(-2*E(j)*h(j)**2))
     &+(-2*E(j))**2/(8*C**4)*(60.+148.*eta+2.*eta**2-(-2.*E(j)*eta**2)*
     &(80.-45.*eta+4.*eta**2)+32/(-2.*E(j)*h(j)**2)*(4.-7.*eta))+
     &(-2.*E(j))**3/(6720.*C**6)*(-3360.+181264.*eta+8610.*pi**2*eta-
     &67200.*eta**2+105.*(-2.*E(j)*h(j)**2)*(-1488.+1120.*eta-
     &195.*eta**2+
     &4.*eta**3)-80./(-2.*E(j)*h(j)**2)*(1008.-21130.*eta+
     &861.*pi**2*eta+
     &2268.*eta**2)+16./(-2.*E(j)*h(j)**2)**2*
     &(53760.-176024.*eta+4305.*pi**2*eta+15120.*eta**2))
        er(j)=sqrt(er(j))
```

```
      et(j)=1.+2*E(j)*h(j)**2+(-2*E(j))/(4*C**2)*(-8.+8.*eta-
     &(-2.*E(j)*h(j)**2)*
     &(-17.+7.*eta))
     &+(-2.*E(j))**2/(8.*C**4)*(12.+72.*eta+20.*eta**2-24.*
     &sqrt(-2.*E(j)*h(j)**2)
     &*(-5.+2.*eta)-(-2.*E(j)*h(j)**2)*(112.-47.*eta+16.*eta**2)-16./
     &(-2*E(j)*h(j)**2)*(-4.+7.*eta)+24./sqrt(-2.*E(j)*h(j)**2)*
     &(-5.+2.*eta))
     &+(-2.*E(j))**3/(6720.*C**6)*(23520.-464800.*eta+179760.*eta**2+
     &16800.*eta**3-2520.*sqrt(-2.*E(j)*h(j)**2)*(265.-193.*eta
     &+46.*eta**2)
     &-525.*(-2.*E(j)*h(j)**2)*(-528.+200.*eta-77.*eta**2+24.*eta**3)
     &-6./(-2.*E(j)*h(j)**2)*(73920.-260272.*eta+4305.*pi**2*eta+
     &61040.*eta**2)
     &+70./sqrt(-2.*E(j)*h(j)**2)*(16380.-19964.*eta+123.*pi**2*eta
     &+3240.*eta**2)
     &+8./(-2.*E(j)*h(j)**2)**2*(53760.-176024.*eta+4305.*pi**2*eta+
     &15120.*eta**2)-70./(-2.*E(j)*h(j)**2)**(3./2.)*(10080.-
     &13952.*eta+123.*pi**2*eta+1440.*eta**2))
      et(j)=sqrt(et(j))


      ephi(j)=1+2.*E(j)*h(j)**2+(-2.*E(j))/(4*C**2)*
     &(24.+(-2.*E(j)*h(j)**2)*(-15.+eta))+
     &(-2.*E(j))**2/(16.*C**4)*(-40.+34.*eta+18.*eta**2-
     &(-2.*E(j)*h(j)**2)*
     &(160.-31.*eta+3.*eta**2)-1./(-2.*E(j)*h(j)**2)*(-416.+91.*eta+
     &15.*eta**2))+(-2.*E(j))**3/(13440.*C**6)*(-584640.-17482.*eta-
     &4305.*pi**2*eta-7350.*eta**2+8190.*eta**3-420.*(-2.*E(j)*h(j)**2)*
     &(744.-248.*eta+31.*eta**2+3.*eta**3)-14./(-2.*E(j)*h(j)**2)*
     &(36960.-
     &341012.*eta+4305.*pi**2*eta-225.*eta**2+150.*eta**3)-
     &1./(-2.*E(j)*h(j)**2)**2*(-2956800.+5627206.*eta-81795.*pi**2*eta+
     &14490.*eta**2+7350.*eta**3))

      ephi(j)=sqrt(ephi(j))
c     write(*,*)'h,er,et,ephi'
c     write(*,*)h(j),er(j),et(j),ephi(j)
      end if
      end do
      return
      end
      subroutine pn3h(n)
      INCLUDE 'tmhscf.h'
       include 'mpif.h'
      integer j,i,n
      real*8 dtp,miu,m,eta,delm,deltax,deltay,A3hdot,B3hdot,A3h,B3h,
     &deltaz,coef1,coef2,co1,co2,co3,co4,tempt1,tempt2,tempt3,r
      real*8 apnx,apny,apnz,dapnx,dapny,dapnz
      do j=1,nbodies
      IF((mstpflag.AND.im(j).EQ.rtag) .or. n.eq.0) THEN

      miu=mass(j)*bhmass/(mass(j)+bhmass)
      m=mass(j)+bhmass
      eta=miu/m



      coef1=8./5.*eta*m/rlength(j)*rdot(j)*(23./14.*
     &(43+14*eta)*m**2/rlength(j)**2+3./28.
     &*(61+70*eta)*vlength(j)**4+70*rdot(j)**4
     &+1./42.*(519-1267*eta)*m/rlength(j)*vlength(j)**2+
     &1./4.*(147+188*eta)*m/rlength(j)*rdot(j)**2-
     &15./4.*(19+2*eta)*vlength(j)**2*rdot(j)**2)
```

```
      coef2=8./5.*eta*m/rlength(j)*(1./42.*(1325+546*eta)
     &*m**2/rlength(j)**2
     &1./28.*(313+42*eta)*vlength(j)**4+75*rdot(j)**4
     &-1./42.*(205+777*eta)*m/rlength(j)*vlength(j)**2
     &+1./12.*(205+424*eta)*m/rlength(j)*rdot(j)**2
     &-3./4.*(113+2*eta)*vlength(j)**2*rdot(j)**2)

      call rel(-n1jx(j)/rlength(j)**2*coef1,-n1jy(j)/
     &rlength(j)**2*coef1,
     &-n1jz(j)/rlength(j)**2*coef1,-vx(j)/rlength(j)**2*coef2,
     &-vy(j)/rlength(j)**2*coef2,-vz(j)/rlength(j)**2*coef2,
     &tempt1,tempt2,tempt3)



      a3hpnx(j)=tempt1*m
      a3hpny(j)=tempt2*m
      a3hpnz(j)=tempt3*m
      ax(j)=ax(j)+a3hpnx(j)
      ay(j)=ay(j)+a3hpny(j)
      az(j)=az(j)+a3hpnz(j)


      A3h=-coef1
      B3h=coef2
      A3hdot=m/rlength(j)*eta*(4*vlength(j)**2*vlength(j)*vdot(j)
     &*(-366./35-12.*eta)+2*(vlength(j)*vdot(j)*rdot(j)**2+
     &2*rdot(j)/rlength(j)*(vlength(j)**2*(-692./35+724./15*eta)+
     &rdot(j)**2*(-294./5-376./5*eta))-
     &3.*m*rdot(j)/rlength(j)**2*(-3956./35-184./5*eta)))
      B3hdot=m/rlength(j)*eta*
     &(4*vlength(j)**2*vlength(j)*vdot(j)*(616./35+12./5*eta)+
     &2*(vlength(j)*vdot(j)*rdot(j)**2+vlength(j)**2*rdot(j)*rddot(j))
     &*(-678./5-12./5*eta)+480*rdot(j)**3*rddot(j)-rdot(j)/rlength(j)*
     &(vlength(j)**4*(626./35+12./5*eta)+
     &vlength(j)**2*rdot(j)**2*(-678./5-12./5*eta)+120.*rdot(j)**4)+
     &m/rlength(j)*(2*vlength(j)*vdot(j)*(-164./21-148./5*eta)+
     &2*rdot(j)*rddot(j)*(82./3+848./15*eta)-
     &2*rdot(j)/rlength(j)*(vlength(j)**2*(-164./21-148./5*eta)+
     &rdot(j)**2*(82./3+848./15*eta))-
     &3.*m*rdot(j)/rlength(j)**2*(1060./21+104./5*eta)))



      co1=A3hdot-2./rlength(j)*A3h*rdot(j)
      co2=B3hdot-2./rlength(j)*B3h*rdot(j)
      co3=A3h
      co4=B3h
      da3hpnx(j)=m/rlength(j)**2*(co1*n1jx(j)+co2*vx(j)+
     &co3*ndotx(j)+co4*ax(j))
      da3hpny(j)=m/rlength(j)**2*(co1*n1jy(j)+co2*vy(j)+
     &co3*ndoty(j)+co4*ay(j))
      da3hpnz(j)=m/rlength(j)**2*(co1*n1jz(j)+co2*vz(j)+
     &co3*ndotz(j)+co4*az(j))

      dax(j)=dax(j)+da3hpnx(j)
      day(j)=day(j)+da3hpny(j)
      daz(j)=daz(j)+da3hpnz(j)

      apnx=a1pnx(j)+a2pnx(j)+a2hpnx(j)+a3pnx(j)+a3hpnx(j)
      apny=a1pny(j)+a2pny(j)+a2hpny(j)+a3pny(j)+a3hpny(j)
      apnz=a1pnz(j)+a2pnz(j)+a2hpnz(j)+a3pnz(j)+a3hpnz(j)
```

```
      dapnx=da1pnx(j)+da2pnx(j)+da2hpnx(j)+da3pnx(j)+da3hpnx(j)
      dapny=da1pny(j)+da2pny(j)+da2hpny(j)+da3pny(j)+da3hpny(j)
      dapnz=da1pnz(j)+da2pnz(j)+da2hpnz(j)+da3pnz(j)+da3hpnz(j)
c     write(*,*)'a3hpnx,a3hpny,a3hpnz,da3hpnx,da3hpny,da3hpnz'
c       write(*,*)a3hpnx(j),a3hpny(j),a3hpnz(j),
c    &da3hpnx(j),da3hpny(j),da3hpnz(j)


c     write(*,*)'apnx,apny,apnz,dapnx,dapny,dapnz'
c     write(*,*)apnx,apny,apnz,dapnx,dapny,dapnz
      end if
      end do
      return
      end



c     subroutine torb
c      INCLUDE 'tmhscf.h'
c        include 'mpif.h'
c     real*8 miu,energy,aaxis
c     integer i
c     do i=1,nbodies
c     if (starlive(i).eq.1) then
c     energy=1./2.*mass(i)*(vx(i)**2+vy(i)**2+vz(i)**2)-
c    &G*bhmass*mass(i)/(sqrt(x(i)**2+y(i)**2+z(i)**2)+epsbh)

c     aaxis=-G*bhmass*mass(i)/(2*energy)
c     torbit(i)=2*pi*aaxis**(3./2.)*sqrt(1/(G*(bhmass+mass(i))))
c     write(*,*)'i,E,a,t',i,energy,aaxis,torbit(i)
c     end if
c     end do
```

```
c         return
c         end

c         subroutine rtidal
c         INCLUDE 'tmhscf.h'
c           include 'mpif.h'

c         integer i
c         real*8 rtispe,tempstar0_snd,tempstar0_rcv
c         do i=1,nbodies
c         if (starlive(i).eq.1) then
c          rlength(i)=sqrt(x(i)**2+y(i)**2+z(i)**2)
c         rtispe=(0.844**2*2.e6/0.42*bhmass)**(1/3)*2.254e-8/2.2751e-7
c          if (rlength(i).le.rtispe)then
c         write(*,*)"rtidal"
c         write(*,*) 1.434e3*bhmass**(1./3.)
c           bhmass=bhmass+mass(i)
c          mass(i)=0.0
c           star0=star0+1
c         write(*,*)i
c          end if
c         end if
c          end do
c         tempstar0_snd=star0
c         call MPI_Allreduce (tempstar0_snd,tempstar0_rcv, 1,
c      &                      mpi_double_precision,
c      &                      mpi_sum, mpi_comm_world,ierr)
c         star0=tempstar0_rcv
c         return
c         end
            SUBROUTINE pn2(n)
          INCLUDE 'tmhscf.h'
```

```
      INCLUDE 'mpif.h'
      INTEGER j,i,n
      REAL*8 dtp,miu,m,eta,delm,deltax,deltay,A2dot,B2dot,A2,B2,
     &deltaz,coef1,coef2,co1,co2,co3,co4,tempt1,tempt2,tempt3,r,coj2
      DO j=1,nbodies
      IF((mstpflag.AND.im(j).EQ.rtag).or. n.eq.0) THEN


      miu=mass(j)*bhmass/(mass(j)+bhmass)
      m=mass(j)+bhmass
      eta=miu/m
c     write(*,*)'j,x,y,z,xyzbh,r,epsbh,rlength(j)',
c    &j,x(j),y(j),z(j),xbh,ybh,
c    &zbh,r(xbh,ybh,zbh,x(j),y(j),z(j)),epsbh,rlength(j)
     coef1=3./4.*(12+29*eta)*m**2/rlength(j)**2+eta*(3-4*eta)
     &*vlength(j)**4+
     &15./8.*eta*(1-3*eta)*rdot(j)**4-3./2.*eta*(3-4*eta)*vlength(j)**2
     &*rdot(j)**2-1./2.*eta*(13-4*eta)*vlength(j)**2*m/rlength(j)-
     &(2+25*eta+2*eta**2)*rdot(j)**2*m/rlength(j)
c     write(*,*)'coef1,eta,rlength(j)',j,coef1,eta,rlength(j)


      coef2=rdot(j)/2*(eta*(15+4*eta)*vlength(j)**2-
     &(4+41*eta+8*eta**2)
     &*m/rlength(j)-3*eta*(3+2*eta)*rdot(j)**2)
c     write(*,*)'coef2,eta,rlength(j)',j,coef2,eta,rlength(j)
      call rel(-n1jx(j)/rlength(j)**2*coef1,
     &-n1jy(j)/rlength(j)**2*coef1,
     &-n1jz(j)/rlength(j)**2*coef1,-vx(j)/rlength(j)**2*coef2,
     &-vy(j)/rlength(j)**2*coef2,-vz(j)/rlength(j)**2*coef2,
     &tempt1,tempt2,tempt3)
      a2pnx(j)=tempt1*m
      a2pny(j)=tempt2*m
      a2pnz(j)=tempt3*m
      ax(j)=ax(j)+a2pnx(j)
      ay(j)=ay(j)+a2pny(j)
      az(j)=az(j)+a2pnz(j)

      A2=-coef1
      B2=coef2
      A2dot=3.*m**2*rdot(j)/(2*rlength(j)**3)*(12+29.*eta)
     &-4*vlength(j)**2*vlength(j)*vdot(j)*eta*(3.-4.*eta)
     & -15./2.*eta*rddot(j)*(1.-3*eta)
     &-m*rdot(j)/(2*rlength(j)**2)*vlength(j)**2*eta*(13.-4.*eta)
     &+m/rlength(j)*vlength(j)*vdot(j)*eta*(13.-4.*eta)
     & -(2+25*eta+2*eta**2)*m/rlength(j)**2*rdot(j)**3
     &+2*(2+25*eta+2*eta**2)*m/rlength(j)*rdot(j)*rddot(j)
     &+3*vlength(j)*vdot(j)*rdot(j)**2*eta*(3.-4.*eta)
     &+3*vlength(j)**2*rdot(j)*rddot(j)*eta*(3-4.*eta)

      B2dot=-1./2*rddot(j)*(-eta*(15+4.*eta)*vlength(j)**2+
     &(4+41.*eta+8.*eta**2)
     &*m/rlength(j)+3*eta*(3+2.*eta)*rdot(j)**2)
     & -1./2.*rdot(j)*(-(4+41.*eta+8.*eta**2)*m*rdot(j)/rlength(j)**2
     &-2*eta*vlength(j)*vdot(j)*(15+4.*eta)
     &+6*eta*rdot(j)*rddot(j)*(3+2.*eta))
      co1=A2dot-2./rlength(j)*A2*rdot(j)
      co2=B2dot-2./rlength(j)*B2*rdot(j)
      co3=A2
      co4=B2
      da2pnx(j)=m/rlength(j)**2*(co1*n1jx(j)+co2*vx(j)+
     &co3*ndotx(j)+co4*ax(j))
      da2pny(j)=m/rlength(j)**2*(co1*n1jy(j)+co2*vy(j)+
     &co3*ndoty(j)+co4*ay(j))
      da2pnz(j)=m/rlength(j)**2*(co1*n1jz(j)+co2*vz(j)+
     &co3*ndotz(j)+co4*az(j))



          dax(j)=dax(j)+da2pnx(j)
          day(j)=day(j)+da2pny(j)
          daz(j)=daz(j)+da2pnz(j)
c         write(*,*)'a2pnx,a2pny,a2pnz,da2pnx,da2pny,da2pnz'
c         write(*,*)a2pnx(j),a2pny(j),a2pnz(j),da2pnx(j),da2pny(j),da2pnz(j)
          E2(j)=-1.D0/4.D0*(2.D0+15.D0*eta)*mor(j)**3.D0
     &+5.D0/16.D0*(1.D0-7.D0*eta+
     &13.D0*eta**2.D0)*vlength(j)**6.D0
     &+1.D0/8.D0*(14.D0-55.D0*eta+4.D0*eta**2.D0)*mor(j)**2.D0
     &*vlength(j)**2.D0+
     &1.D0/8.D0*(4.D0+69.D0*eta+12.D0*eta**2.D0)*mor(j)**2.D0
     &*rdot(j)**2.D0+
     &1.D0/8.D0*(21.D0-23.D0*eta-27.D0*eta**2.D0)*mor(j)
     &*vlength(j)**4.D0+
     &1.D0/4.D0*eta*(1.D0-15.D0*eta)*mor(j)*
     &vlength(j)**2.D0*rdot(j)**2.D0-
     &3.D0/8.D0*eta*(1.D0-3.D0*eta)*mor(j)*rdot(j)**4.D0
          E2(j)=E2(j)*miu
c          write(*,*)'me,j,E2(j),x(j)',me,j,E2(j),x(j)
          coj2=1./4.*(14.-41.*eta+4.*eta**2)*mor(j)**2+
     &3./8.*(1.-7.*eta+13.*eta**2)*vlength(j)**4
     &+1./2.*(7.-10.*eta-9.*eta**2)*mor(j)*vlength(j)**2
     &-1./2.*eta*(2.+5.*eta)*mor(j)*rdot(j)**2
          J2x(j)=miu*coj2*rcvx(j)
          J2y(j)=miu*coj2*rcvy(j)
          J2z(j)=miu*coj2*rcvz(j)
c          write(*,*)'me,j,J2x(j)',me,j,J2x(j)
          end if
          end do
```

```
 return
end
```

Below is the head file tmhscf.h

```
C=======================================================================
C
C
C                         INCLUDE FILE tmhscf.h
C
C
C=======================================================================
C
C
C     Parameter declarations, allocation of array storage, common
C     block definitions.
C
C
C=======================================================================

C=======================================================================
C     Definitions I added -- Bohr He, June 1995
C     Further Modifications by Steinn Sigurdsson Sep 1995
C=======================================================================
        integer n_pes, me, lognpes
        real*8 tmpsum01, tmpsum02, tmpsum03, tmpsum04, tmpsum05,
     &         tmpsum06, tmpsum07, tmpsum08, tmpsum09, tmpsum10,
     &         tmpsum11, tmpsum12, tmpsum13, tmpsum14
        real*8 temp01, temp02, temp03, temp04, temp05, temp06, temp07,
     &         temp08, temp09, temp10,temp11,temp12,temp13,temp14,
     &         temp15,temp16,temp17,temp19,temp20,temp21,temp22,temp23,
     &         temp24,temp25,tempE0,tempE1,
     &         tempE2,tempE3,tempJ2x,tempJ2y,tempJ2z

        integer itemp01,temp18
        real*8 totime0,totime1,totime,tmptime,cycletime
```

```
      parameter(cycletime=6.6666666667E-9)
C=====================================================================

      INTEGER nbodsmax,nbodsper,nmax,lmax

      PARAMETER(nbodsmax=2000000,nbodsper=25000,nmax=13,lmax=9)

      CHARACTER*50 headline
      INTEGER nsteps,noutbod,nbodies,noutlog,nfrac,nlilout,iseed
      INTEGER im,star0

      LOGICAL selfgrav,inptcoef,outpcoef,zeroodd,zeroeven,
     &fixacc,firstpn,secondpn,secondhpn,thirdpn,thirdhpn
C       firstpn quadru added by Baile
      LOGICAL bhgrav,multistep,usedrag,stellev,fixedn,lilout
      LOGICAL mstpflag

      REAL*8 tnow,x,y,z,vx,vy,vz,mass,pot,dtime,G,C,ax,ay,az,one,pi,
     &          twoopi,onesixth,tpos,tvel,cputime0,cputime1,cputime,
     &          rcrit,radbh,vel0bh,dteps,
     &          potext,two,zero,tiny,ecrit,oax,oay,oaz,odax,oday,odaz,
     &          dax,day,daz,adens,dti,aqx,aqy,aqz,n1jx,n1jy,n1jz,
     &          v1jx,v1jy,v1jz,odaxbh,odaybh,odazbh,a1pnx,a1pny,a1pnz,
     &          a2pnx,a2pny,a2pnz,a3pnx,a3pny,a3pnz,
     &          a2hpnx,a2hpny,a2hpnz,a3hpnx,a3hpny,a3hpnz,
     &          asox,asoy,asoz,maxratio,rlength,vlength,
     &          h,er,et,ephi
      REAL*8 bhmass,epsbh,tstartbh,tgrowbh,tlivebh,tdiebh,
     &          tstartdrag,tgrowdrag,
     &          xdrag,ydrag,zdrag,tlivedrag,tdiedrag,bhmasst,tfinal,
     &          torbit,vdot,rdot,rddot,ndotx,ndoty,
     &          ndotz,da1pnx,da1pny,da1pnz,da2pnx,da2pny,da2pnz,
     &           da2hpnx , da2hpny , da2hpnz ,
     &           da3pnx , da3pny , da3pnz , da3hpnx , da3hpny , da3hpnz

      REAL*8  sinsum1 , sinsum2 , cossum1 , cossum2
      REAL*8  dts , dt1i1
      REAL*8  dtsmall , dtbig , tnextbig , dtbin
      REAL*8  J1x , J1y , J1z , J2x , J2y , J2z , J3x , J3y , J3z ,
     &E1 , E2 , E3 , rcvx , rcvy , rcvz , mor ,
     &E , E0 , J0x , J0y , J0z , Jx , Jy , Jz

      COMMON/ bodscom / x ( nbodsper ) , y ( nbodsper ) , z ( nbodsper ) ,
     &                   vx ( nbodsper ) , vy ( nbodsper ) , vz ( nbodsper ) ,
     &                   mass ( nbodsper ) ,
     &                   pot ( nbodsper ) , ax ( nbodsper ) , ay ( nbodsper ) ,
     &                   az ( nbodsper ) , potext ( nbodsper ) , adens ( nbodsper ) ,
     &                   dax ( nbodsper ) , day ( nbodsper ) , daz ( nbodsper ) ,
     &                   oax ( nbodsper ) , oay ( nbodsper ) , oaz ( nbodsper ) ,
     &                   odax ( nbodsper ) , oday ( nbodsper ) , odaz ( nbodsper ) ,
     &                   dti ( nbodsper ) , aqx ( nbodsper ) , aqy ( nbodsper ) ,
     &                   aqz ( nbodsper ) ,
     &                   n1jx ( nbodsper ) , n1jy ( nbodsper ) , n1jz ( nbodsper ) ,
     &                   v1jx ( nbodsper ) , v1jy ( nbodsper ) , v1jz ( nbodsper ) ,
     &                   torbit ( nbodsper ) , a1pnx ( nbodsper ) , a1pny ( nbodsper ) ,
     &                   a1pnz ( nbodsper ) , a2pnx ( nbodsper ) , a2pny ( nbodsper ) ,
     &                   a2pnz ( nbodsper ) , a3pnx ( nbodsper ) , a3pny ( nbodsper ) ,
     &                   a3pnz ( nbodsper ) , a2hpnx ( nbodsper ) ,
     &                   a2hpny ( nbodsper ) ,
     &                   a2hpnz ( nbodsper ) , a3hpnx ( nbodsper ) ,
     &                   a3hpny ( nbodsper ) ,
     &                   a3hpnz ( nbodsper ) , asox ( nbodsper ) , asoy ( nbodsper ) ,
     &                   asoz ( nbodsper ) , maxratio ( nbodsper ) ,
     &                   rlength ( nbodsper ) ,
     &                         vlength(nbodsper),
     &vdot(nbodsper),rdot(nbodsper),rddot(nbodsper),ndotx(nbodsper),
     &ndoty(nbodsper),ndotz(nbodsper),da1pnx(nbodsper),
     &da1pny(nbodsper),
     &da1pnz(nbodsper),da2pnx(nbodsper),da2pny(nbodsper),
     &da2pnz(nbodsper),
     &            da2hpnx(nbodsper),da2hpny(nbodsper),da2hpnz(nbodsper),
     &            da3pnx(nbodsper),da3pny(nbodsper),da3pnz(nbodsper),
     &da3hpnx(nbodsper),da3hpny(nbodsper),da3hpnz(nbodsper),
     &J1x(nbodsper),J1y(nbodsper),J1z(nbodsper),J2x(nbodsper),
     &J2y(nbodsper),J2z(nbodsper),J3x(nbodsper),J3y(nbodsper),
     &J3z(nbodsper),E1(nbodsper),
     &E2(nbodsper),E3(nbodsper),rcvx(nbodsper),rcvy(nbodsper),
     &rcvz(nbodsper),mor(nbodsper),
     &h(nbodsper),er(nbodsper),et(nbodsper),ephi(nbodsper),
     &E(nbodsper),E0(nbodsper),J0x(nbodsper),J0y(nbodsper),
     &J0z(nbodsper),Jx(nbodsper),Jy(nbodsper),Jz(nbodsper)


      COMMON/parcomi/nbodies,nsteps,noutbod,
     &noutlog,nlilout,nfrac,star0
      COMMON/parcomr/dtime,G,C,one,pi,twoopi,onesixth,two,tiny,zero
      COMMON/parcomc/headline
      COMMON/parcoml/selfgrav,inptcoef,outpcoef,zeroodd,zeroeven,
     &                        lilout,fixacc,bhgrav,multistep,usedrag,
     &                        stellev,fixedn,firstpn,secondpn,secondhpn,
     &                        thirdpn,thirdhpn
      COMMON/timecom/tpos,tnow,tvel,ecrit,dteps
      COMMON/timecom2/rcrit,radbh,vel0bh,tfinal
      COMMON/tstepcom/dts,dtlil
      COMMON/cpucom/cputime0,cputime1,cputime
      COMMON/bhgcom/bhmass,epsbh,tstartbh,tgrowbh,tlivebh,tdiebh
```

```
      COMMON/dracom/xdrag,ydrag,zdrag,tlivedrag,tdiedrag,
     &               tstartdrag,tgrowdrag,bhmasst
      COMMON/coefcom/sinsum1(0:nmax,0:lmax,0:lmax),
     &                sinsum2(0:nmax,0:lmax,0:lmax),
     &                cossum1(0:nmax,0:lmax,0:lmax),
     &                cossum2(0:nmax,0:lmax,0:lmax)
      COMMON/mstpcomr/dtsmall,dtbig,tnextbig(3),dtbin(4)
      COMMON/mstpcomi/im(nbodsper)
      COMMON/mstpcoml/mstpflag


C=======================================================================
C     Definition I added -- Bohr He, June 1995
C=======================================================================
      common/t3d/n_pes, me, lognpes
      common/tmpo/tmpsum01, tmpsum02, tmpsum03, tmpsum04, tmpsum05,
     &            tmpsum06, tmpsum07, tmpsum08, tmpsum09, tmpsum10,
     &            tmpsum11, tmpsum12, tmpsum13, tmpsum14
      common/tempo/temp01(nbodsper),temp02(nbodsper),temp03(nbodsper),
     &             temp04(nbodsper),temp05(nbodsper),temp06(nbodsper),
     &             temp07(nbodsper),temp08(nbodsper),temp09(nbodsper),
     &             temp10(nbodsper),temp11(nbodsper),temp12(nbodsper),
     &             temp13(nbodsper),temp14(nbodsper),temp15(nbodsper),
     &             temp16(nbodsper),temp17(nbodsper),temp18(nbodsper),
     &             temp19(nbodsper),temp20(nbodsper),temp21(nbodsper),
     &             temp22(nbodsper),temp23(nbodsper),temp24(nbodsper),
     &             temp25(nbodsper),tempE0(nbodsper),tempE1(nbodsper),
     &             tempE2(nbodsper),tempE3(nbodsper),
     &             tempJ2x(nbodsper),
     &             tempJ2y(nbodsper),tempJ2z(nbodsper)
```

```
      common/itempo/itemp01(nbodsper)
      common/cpucom1/totime0,totime1,totime,tmptime

C=============================================================
C    Definitions specific to input/output.
C=============================================================
      INTEGER uterm,upars,ulog,ubodsin,ubodsout,utermfil,uoutcoef,
     &        uincoef,ubodsel,umods,ubodslil,uchkout
      CHARACTER*8 parsfile,logfile,ibodfile,obodfile,termfile,
     &            outcfile,incfile,elfile,modsfile,olilfile,
     &            chkfile

      PARAMETER(uterm=6,upars=10,ulog=11,ubodsin=12,ubodsout=13,
     &          umods=14,utermfil=15,uoutcoef=16,uincoef=17,
     &          ubodsel=18,ubodslil=19,uchkout=20)
      PARAMETER(parsfile='scfpar',logfile='scflog',
     &          modsfile='scfmod',olilfile='slilxxxx',
     &      ibodfile='scfbi',obodfile='snapxxxx',
     &          termfile='scfout',outcfile='scfocoef',
     &          incfile='scficoef',elfile='scfelxxx',
     &          chkfile='scfchkpt')

c=============================================================
c Definitions specific to black holes added by Baile Li Aril 10 2011
c=============================================================
      real*8 xbh,ybh,zbh,vxbh,vybh,vzbh,axbh,aybh,daxbh,daybh,dazbh,
     &azbh,kix,kiy,kiz,dakix,dakiy,dakiz,oaxbh,oaybh,oazbh,akix,
     &akiy,akiz,oakix,oakiy,oakiz,odakix,odakiy,odakiz,kix1,kiy1,
     &kiz1,akix1,akiy1,akiz1,sx,sy,sz,asx,asy,asz,oasx,oasy,oasz

      real*8 fac
      common/massfac/fac
      common/bhlinear/xbh,ybh,zbh,vxbh,vybh,vzbh,axbh,aybh,azbh,
&                         daxbh,daybh,dazbh,oaxbh,oaybh,oazbh,odaxbh,
&                         odaybh,odazbh
      common/bhangular/kix,kiy,kiz,akix,akiy,akiz,dakix,dakiy,dakiz,
&oakix,oakiy,oakiz,odakix,odakiy,odakiz,kix1,kiy1,kiz1,akix1,
&akiy1,akiz1,sx,sy,sz,asx,asy,asz,oasx,oasy,oasz

      integer rtag,numb,nbin,nbin0
      common/particlestatus/rtag,numb(4),nbin(4),nbin0
      integer nbin1,noutbod1
      common/outbodsmulti/nbin1,noutbod1
```

Below is the input mode file scfmod.

```
C**********Basic  input  parameters**********
37563              iseed
1.0           bhmass
0.00001            epsbh
0.0               tstartbh
-10.00             tgrowbh
1.0e12              tlivebh
10.0               tdiebh
0.0                xdrag
0.000              ydrag
0.045              zdrag
0.00               tstartdrag
10.0               tgrowdrag
30.0               tlivedrag
10.0               tdiedrag
.TRUE.            bhgrav
.FALSE.            usedrag
.FALSE.            stellev
C****************************************
```

Below is the input parameter file scfpar.

```
C**********Basic input parameters**********
expansion test headline
10000000        nsteps
1     noutbod
1               noutlog
1.0             dteps
1.0                     G
1.0                   C
.FALSE.            firstpn
.FALSE.           secondpn
.FALSE.           secondhpn
.FALSE.           thirdpn
.FALSE.           thirdhpn
1.0e12              tfinal
.TRUE.            multistep
.FALSE.            fixedn
.TRUE.            selfgrav
.FALSE.           inptcoef
.FALSE.            outpcoef
.FALSE.            zeroodd
.FALSE.           zeroeven
.FALSE.            fixacc
0.00010d0           rcrit
0.0d-4            ecrit
.FALSE.           outlil
10000000         noutlil
C****************************************
```

Below is the compile file run.pl.

```
#!/usr/local/bin/perl -w



system("rm -f snap*");
system("rm -f scflog");
system("rm -f scfout");
system("rm -f out.test");
system("rm -f mpiscf");
system("rm -f sli*");
system("rm -f old*");
system("rm -f fort*");
system("rm -f core*");

system('mpif77 -O3 mpiscf.f -o mpiscf');
#system("./mpiscf");
#system("qsub  squeeze.sript.nbod");
```

BIBLIOGRAPHY

Aarseth, S. 1967, Les Nouvelles Méthodes de la Dynamique Stellaire, 47

Allen, A. J., Palmer, P. L., & Papaloizou, J. 1990, MNRAS, 242, 576

Allen, S. W., Dunn, R. J. H., Fabian, A. C., Taylor, G. B., & Reynolds, C. S. 2006, MNRAS, 372, 21

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

Baker, J.G., Centrella, J.M., Choi, D.I., Koppitz, M. & van Meter, JR. 2006, Phys. Rev. Lett., 96, 111102

Barnes, J. E. & Hernquist, L. 1991, ApJ, 370, L65

Barnes,J.E., & Hernquist,L. 1992, ARA&A, 30,705

Barnes, J. E. & Hernquist, L. 1996, ApJ, 471, 115

Bell, N., & Hoberock, J. 2011, in GPU Computing Gems Jade Edition, ed. W.-M W. Hwu (Waltham, MA: Morgan Kaufmann), 359

Bellovary, J., Volonteri, M., Governato, F., Shen, S., Quinn, T., & Wadsley, J. 2011, ApJ, 742, 13

Berczik, P., Merritt, D., Spurzem, R., & Bischof, H.-P. 2006, ApJ, 642, L21

Bertin, G., & Stiavelli, M. 1989, ApJ, 338, 723

Binney, J., & Tremaine, S. 2008, Galactic Dynamics (2nd ed.; Princeton, NJ: Princeton University Press)

Blandford, R. D., & McKee, C. F. 1982, ApJ, 255, 419

Bondi, H. 1952, MNRAS, 112, 195

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

Campanelli, M., Lousto, C.O., Marronetti, P. & Zlochower, Y. 2006, Phys. Rev. Lett., 96, 111101

Carpintero, D., & Aguilar, L. 1998, MNRAS, 298, 1

Cattaneo, A., et al. 2009. Nature 460, 213

Centrella, J., Baker, J. G., Kelly, B. J., & van Meter, J. R. 2010, Annual Review of Nuclear and Particle Science, 60, 75

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

Coker,R.F. & Melia,F. 1997, ApJL, 488, L149

de Zeeuw, T. 1985, MNRAS, 216, 273

Dehnen, W. 1993, MNRAS, 265, 250

Doeleman SS, Fish VL, Schenck DE, et al. 2012. Science 338, 355

Dotti, M., Colpi, M., Haardt, F., & Mayer, L. 2007, MN- RAS, 379, 956

Earn, D. J. D., & Sellwood, J. A. 1995, ApJ, 451, 533

Earn, D. J. D. 1996, ApJ, 465, 91

Falcke, H., et al. 2004, A&A 414, 895

Ferrarese, L. 2002, in C.-H. Lee & H.-Y. Chang (eds.), Current High-Energy Emission Around Black Holes, pp 3–24

Ferrers N. M., 1877, Q. J. Pure Appl. Math., 14, 1

Fry, J. N., & Peebles, P. J. E. 1980, ApJ, 236, 343

Gammie, C. F., Shapiro, S. L., & McKinney, J. C. 2004, ApJ, 602, 312

Genzel, R., Pichon, C., Eckart, A., Gerhard, O. E., & Ott, T. 2000, MNRAS, 317, 348

Genzel, R., et al. 2003, Nature, 425, 934

Gerhard, O. E., & Binney, J., 1985, MNRAS, 216, 467

Ghez, A. M., Salim, S., Weinberg, N. N., et al. 2008, ApJ, 689, 1044

Giersz, M., Heggie, D. C., & Hurley, J. R. 2008, MNRAS, 388, 429

Gültekin, K., et al. 2009, ApJ, 698, 198

Gundlach, C., et al. 2005, Class. Quantum Grav., 22, 3767

Hamada, T., & Iitaka, T. 2007, arXiv:astro-ph/0703100

Helmi, A., & White, S. D. M. 1999, MNRAS, 307, 495

Hemsendorf, M., Sigurdsson, S., & Spurzem, R. 2002, ApJ, 581, 1256

Hénon, M. 1964, Annales d'Astrophysique, 27, 83

Hénon, M. 1975, Dynamics of the Solar Systems, 69, 133

Hernquist, L. 1989, Nature, 340, 687

Hernquist, L. 1990, ApJ, 356, 359

Hernquist, L., & Ostriker, J. P. 1992, ApJ, 386, 375

Hernquist, L., & Weinberg, M. 1992, ApJ, 400, 80

Hernquist, L. & Mihos, J.C. 1995, ApJ, 448, 41

Hernquist, L., Sigurdsson, S., & Bryan, G. L. 1995, ApJ, 446, 717

Herrnstein, J. R., et al. 1999, Nature, 400, 539

Holley-Bockelmann, K., & Richstone, D. 1999, ApJ, 517, 92

Holley-Bockelmann, K., & Richstone, D. O. 2000, ApJ, 531, 232

Holley-Bockelmann, K., et al. 2001, ApJ, 549, 862

Holley-Bockelmann, K., Mihos, J. C., Sigurdsson, S., Hernquist, L., & Norman, C. 2002, ApJ, 567, 817

Holley-Bockelmann, K., Weinberg, M., & Katz, N. 2005, MNRAS, 363, 991

Holley-Bockelmann, K., & Sigurdsson, S. 2006, arXiv:astro-ph/0601520

Hopkins, P. F., Hernquist, L., Cox, T. J., Robertson, B., Di Matteo, T., Martini, P., & Springel, V. 2005, ApJ, 630, 705

Johnson, J. L., Whalen, D. J., Fryer, C. L., & Li, H. 2012, ApJ, 750, 66

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, Communications of the ACM, 8(1), 40

Kalapotharakos, C., Efthymiopoulos, C., & Voglis, N. 2008, MNRAS, 383, 971

Kandrup, H. E., 1995, A&AT, 7, 225

Khan, F. M., Just, A., & Merritt, D. 2011, ApJ, 732, 89

Khan, F. M., Preto, M., Berczik, P., et al. 2012, ApJ, 749, 147

Khan, F. M., Holley-Bockelmann, K., Berczik, P., & Just, A. 2013, ApJ, 773, 100

King, A. R., & Pringle, J. E. 2006, MNRAS, 373, 90

Kormendy,J. & Richstone,D. 1995, ARA&A, 33,581

Kuo, C. Y., Braatz, J. A., Condon, J. J., et al. 2011, ApJ, 727, 20

Laskar, J. 1993, PhysicaD, 67,257

Lees, J. F., & Schwarzschild, M. 1992, ApJ, 384, 491

Li, B., Holley-Bockelmann, K.,& Khan, F. 2014, arXiv: 1412.2134v1

Lin, D. N. C., & Tremaine, S. 1983, ApJ, 264, 364

Lodato, G. & Natarajan, P. 2006, MNRAS, 371, 1813

Lowing, B., Jenkins, A., Eke, V., & Frenk, C. 2011, MNRAS, 416, 2697

Magorrian J., & Tremaine S. 1999, MNRAS, 309, 447

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

Meiron, Y., Li, B., Holley-Bockelmann, K., & Spurzem, R. 2014, ApJ, 792, 98

Merritt, D., & Stiavelli, M. 1990, ApJ, 358, 399

Merritt, D. 1996, AJ, 111, 2462

Merritt, D., & Fridman, T. 1996, ApJ, 460, 136

Merritt, D., & Quinlan, G. 1998, ApJ, 498, 625

Merritt, D., & Valluri, M. 1999, AJ, 118, 1177

Merritt, D., Milosavljević, M., Favata, M., Hughes, S. A., and Holz, D. E. 2004, ApJL, 607, L9

Merritt, D., & Poon, M. 2004, ApJ, 606, 788

Merritt, D., & Vasiliev, E. 2011, ApJ, 726, 61

Mihos, J. C. & Hernquist, L. 1994, ApJ, 437, 611

Mihos, J. C. & Hernquist, L. 1996, ApJ, 464, 641

Milosavljević, M., & Merritt, D. 2003, AIP Conf. Proc., 686, 201

Miyoshi, M., et al. 1995, Nature, 373, 127

Natarajan, P., & Treister, E. 2009, MNRAS, 393, 838

Navarro, J. F., Frenk, C. S., & White, S. D. M. 1996, ApJ, 462, 563

Nitadori, K. 2009, PhD thesis, Univ. Tokyo

Norman, C. A., May, A., & van Albada, T. S. 1985, ApJ, 296, 20

Norman C., & Silk J. 1983, ApJ, 266, 502

Nusser, A., & Sheth, R. K. 1999, MNRAS, 303, 685

Oh, S., Kim, S. S., & Figer, D. F. 2009, JKAS, 42, 17

Ostriker, J. P., & Mark, J. W.-K. 1968, ApJ, 151, 1075

Pelupessy, F., Di Matteo, T., & Ciardi, B. 2007, ApJ, 665, 107

Pelupessy, F. I., van Elteren, A., de Vries, N., et al. 2013, A&A, 557, A84

Peterson B. M., et al., 2004, ApJ, 613, 682

Peterson, B.M. 2006, Lect. Notes Phys. 693, 77

Plummer, H. C. 1911, MNRAS, 71, 460

Poon, M. Y., & Merritt, D. 2001, ApJ, 549, 192

Portegies Zwart, S. F., Belleman, R. G., & Geldof, P. M. 2007, NewA, 12, 641

Portegies Zwart, S., McMillan, S., Harfst, S., et al. 2009, NewA, 14, 369

Preto, M., Berentzen, I., Berczik, P., & Spurzem, R. 2011, ApJL, 732, L26

Pretorius, F. 2005a, Phys. Rev. Lett., 95, 121101

Pretorius, F. 2005b, Class. Quantum Grav., 22, 425

Pretorius, F. 2006, Class. Quantum Grav., 23, S529

Quinlan, G., & Hernquist, L. 1997, New Astron., 2, 533

Rahmati, A., & Jalali, M. A. 2009, MNRAS, 393, 1459

Richstone, D. O. 1982, ApJ, 252, 496

Rosenbluth, M. N., MacDonald, W. M., & Judd, D. L. 1957, Physical Review, 107, 1

Rubin, V. C., & Ford, Jr., W. K. 1970, ApJ, 159, 379

Saha, P. 1993, MNRAS, 262, 1062

Sambhus, N., & Sridhar, S. 2000, ApJ, 542, 143

Satish, N., Mark, H., Garland, M. 2009, in IEEE International Parallel & Distributed Processing Symposium, ed. (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, P. 2006, Extragalactic Astronomy and Cosmology, Springer, page 4

Schneider, J., Amaro-Seoane, P., & Spurzem, R. 2011, MNRAS, 410, 432

Schödel, R., Eckart, A., Alexander, T., et al. 2007, A&A, 469, 125

Schröder, P., & Sweldens W. 1995, Computer Graphics Proceedings (SIGGRAPH 95), p. 161

Sellwood, J. A. 1987, ARA&A, 25, 151

Sellwood, J. A. 2003, ApJ, 587, 638

Sesana, A., Haardt, F., & Madau, P. 2007, ApJ, 660, 546

Shapiro, S. L. 2005, ApJ, 620, 59

Shu, F. H. 1992, Journal of the British Astronomical Association 102, 230

Sigurdsson, S., Hernquist, L., & Quinlan, G. D. 1995, ApJ, 446, 75

Soltan, A. 1982, MNRAS, 200, 115

Spergel, D. N., et al. 2003, ApJS, 148, 175

Spurzem, R., & Takahashi, K. 1995, MNRAS, 272, 772

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

Sridhar, S., & Touma, J., 1997, MNRAS287, L1-L4

Sridhar, S., & Touma, J. 1999, MNRAS, 303, 483

Strohmayer,T.E. 2001, ApJL, 552, L49

Syer, D. 1995, MNRAS, 276, 1009

Toomre, A. 1963, ApJ, 138, 385

Urry, C. M., & Padovani, P. 1995, PASP, 107, 803

Valluri,M., & Merritt, D. 1998, ApJ, 506, 686

van Albada, T. S., & van Gorkom, J. H. 1977, A&A, 54, 121

van Albada, T. S. 1982, MNRAS, 201, 939

van der Marel, R. P., Sigurdsson, S., & Hernquist, L. 1997, ApJ, 487, 153

Vasiliev, E. 2013, MNRAS, 434, 3174

Vasiliev, E., & Merritt, D. 2013, ApJ, 774, 87

Vasiliev, E. 2014, arXiv: astro-ph/1411.1760

Vasiliev, E., Antonini, F., & Merritt, D. 2014, ApJ, 785, 163

Villumsen, J. V. 1982, MNRAS, 199, 493

Volonteri M., Haardt F., Madau P., 2003, ApJ, 582, 559

Volonteri M., Rees M. J., 2005, ApJ, 633, 624

Volonteri M., Madau P., Quataert E., Rees M. J., 2005, ApJ, 620, 69

Volonteri, M. 2010, A&A Rev., 18, 279

Volonteri, M., Gültekin, K., and Dotti, M. 2010, MNRAS, 404, 2143

Wachlin, F. C., & Ferraz-Mello, S. 1998, MNRAS, 298, 22

Weinberg, M. D. 1996, ApJ, 470, 715

Weinberg, M. D. 1999, AJ, 117, 629

Weinberg, M. D., & Katz, N. 2002, ApJ, 580, 627

Weinberg, M. D., & Katz, N. 2007a, MNRAS, 375, 425

Weinberg, M. D., & Katz, N. 2007b, MNRAS, 375, 460

White, S. D. M. 1983, ApJ, 274, 53

Xue, X. X., Rix, H. W., Zhao, G., Re Fiorentin, P., Naab, T., Steinmetz, M., van den Bosch, F. C., Beers, T. C., Lee, Y. S., Bell, E. F., Rockosi, C., Yanny, B., Newberg, H., Wilhelm, R., Kang, X., Smith, M. C., & Schneider, D. P. 2008, ApJ, 684, 1143

Zhang,S.N., Cui,W., & Chen,W. 1997, ApJL, 482,L155

Zhao, H. 1996, MNRAS, 278, 488

Zotos E.E. 2014, A&A, 563, A19