Simulating the universe with GPU-accelerated supercomputers: *N*-body methods, tests, and examples

Benjamin D. Wibking

Adviser: Andreas A. Berlind

Department of Physics and Astronomy, Vanderbilt University, Nashville, TN 37231

ben@wibking.com

Submitted to fulfill, in part, the requirements for honors in physics, B.A.

ABSTRACT

We demonstrate the acceleration obtained from using GPU/CPU hybrid clusters and supercomputers for N-body simulations of gravity based in part on the author's new code development. Validation tests are shown for cosmological simulations and for galaxy simulations, along with their respective speedups compared to traditional simulations. Potential new applications for science enabled by this advance are highlighted.

Subject headings: n-body: general — n-body: gravitation – computational methods – large-scale structure and cosmology

1. Introduction

N-body simulations are the tool of choice for theoretical predictions of galaxy clustering, large-scale structure, and galaxy formation (Springel et al. 2005; Navarro et al. 1997). They are important for the interpretation of extragalactic observational surveys, such as the simulations and analysis surrounding Sloan Digital Sky Survey (e.g. Abazajian et al. 2005), the Dark Energy Survey (e.g. Busha et al. 2013), and the upcoming efforts for the Large Synoptic Survey Telescope (Ivezic et al. 2008). These simulations also play an important role in understanding galaxy dynamics (Holmberg 1941; Binney & Tremaine 1987), star clusters (Aarseth et al. 1974), and planetary systems.

In particular, the Large Synoptic Survey Telescope (LSST) science goals will require significantly more accurate theoretical predictions about the universe in order for the observations beginning in this decade to be interpreted and used for constraining, for example, cosmological parameters of dark energy. Recent work has suggested that the theoretically-predicted nonlinear power spectrum of the density distribution of the universe will need to be accurate to less than 1% relative error down to scales of 1 h Mpc^{-1} (Heitmann et al. 2010; Reed et al. 2013). For these predictions, N-body simulations will be needed for unprecedented physical scales and numbers of particles and in many distinct realizations (such as the simulations for interpretations of galaxy clustering data from the Sloan Digital Sky Survey, e.g. McBride et al. 2013). All of these factors increase the computational requirements of such a set of simulations by orders of magnitude compared to individual state-of-the-art simulations from just a few years ago.

Recent work on N-body codes has involved scaling to larger simulations ($\approx 10^{12}$ particles) (e.g. Habib et al. 2012; Ishiyama et al. 2012) and taking advantage of newer computer architectures. As noted in Yokota & Barba (2011), the computer architectures of the late 1990s and early 2000s have been abnormal in the history of high-performance

computing, as they required only coarse-grained parallelism. In contrast, the fine-grained parallelism required for high-performance vector supercomputers of the late 1980s and early 1990s (e.g. Makino & Hut 1989) is much closer to the programming model required by the fastest supercomputers in the world today, which use highly thread-parallel graphics processing units as computational accelerators.[1]

Graphics processors have been used for science applications prior to the advent of official APIs, but only in 2007 was an official programming interface released by NVIDIA for its graphics processing (GPU) cards. Previous work with GPU-based N-body simulations has involved direct forces ('all-pairs') calculations (Hamada & Iitaka 2007; Nyland et al. 2007), Barnes-Hut codes for non-cosmological integrations (Burtscher & Pingali 2011; N-Body Shop 2011; Bédorf et al. 2012), and one report of a cosmological CUDA and Gadget-2–based code with Barnes-Hut forces and Ewald periodic image forces (T 2012). (There is one cosmological mesh-based gravity code, described in Schive et al. (2010), but this is not, strictly-speaking, an N-body code.)

In this work, we run simulations that are an order of magnitude larger than T (2012) and provide the first published GPU N-body code with a hybrid short-range Barnes-Hut force calculation (running fully on the GPU) and a long-range particle-mesh calculation (running on the CPU) that can run in parallel (using MPI) on distributed-memory clusters. Additionally, unlike some previous codes, this code is made publicly available.[2]

Our code, CUDA-Gadget, can in principle simulate all of the above astrophysical systems of interest. It is based on Gadget-2 (Springel 2005) and runs on individual desktop computers with graphics processors (GPUs) as well as GPU-based hybrid clusters

[1] "Oak Ridge Claims No. 1 Position on Latest TOP500 List with Titan." http://www.top500.org/blog/lists/2012/11/press-release/

[2] http://code.google.com/p/cuda-gadget/

and supercomputers, including the development version of the NSF flagship GPU-based computing resource 'Keeneland' (Vetter et al. 2011). The design and initial development of this version of the code is due to the G2X project (Frigaard 2009). The final code development, with extensions and modifications for the current version of NVIDIA's CUDA programming interface for GPUs (nvi 2007), modifications to work with the latest (and significantly faster) FFTW library (FFTW3.3 with MPI support, instead of the older, API-incompatible FFTW2, Frigo & Johnson 1998), extension to full double-precision arithmetic on the GPU, extension to non periodic boundary conditions and unequal gravitational softening lengths (present in Gadget-2 but not in G2X), and numerical correctness fixes are due to the author.

2. Physics

N-body simulations predict the matter phase space distribution throughout large volumes of the observable universe through the gravitational interactions of matter. For many astrophysical systems, non-gravitational interactions can be neglected. Further, the resolution of these simulations of large-scale systems is such that they are usually *collisionless*, i.e. assuming a continuous 'matter fluid' that does not collide with itself. In the N-body picture, this means that we attempt to neglect two-body interactions, as two-body interactions of particles only physically happen at scales much smaller than those we can resolve in the simulation (unless we are concerned with stellar dynamics, but this is a different story). Two-body interactions can be reduced somewhat with an appropriate choice of force softening (Dehnen 2001), but the only way to substantially reduce two-body interactions is to increase the number of particles in a simulation, thus pushing the two-body relaxation time further away from the simulated time scales (Diemand et al. 2004).

2.1. Vlasov-Poisson equation

Formally, the N-body method solves the Vlasov-Poisson equation by discretely sampling the distribution function of the system (i.e. into the initial configuration of particles used in the simulation). The differential equation to be solved for the action of gravity on particles in the universe with no other forces is the Poisson equation (eq. 1 in Springel 2005):

$$\nabla^2\phi = 4\pi G(\rho(\vec{x}) - \bar{\rho}) \tag{1}$$

$$\ddot{x}_{i,j} = -\frac{1}{m_i}\frac{\partial\phi}{\partial x_{i,j}} \tag{2}$$

with the density field ρ created by the particles, for all particles i (where $\bar{\rho}$ is the mean density of the universe and j are the spatial coordinates). The idealized problem to solve is that where the number of particles i tends to infinity and the mass per particle m_i tends to an infinitesimal mass, creating a continuous density field. In any real simulation, the number of particles is very large ($\approx 10^{10}$ or higher for the largest simulations) but the particles are nonetheless discrete. For particles that are very close to each other, spherical shells of mass are assumed to exist about each particle, which softens the forces and prevents unphysically-large scattering interactions (eq. 4 in Springel 2005). The discreteness of simulations has been a concern regarding their numerical convergence to the continuum solution of the Poisson equation/collisionless Boltzmann equation, but this issue has been resolved to the satisfaction of simulators (Quinlan & Tremaine 1992).

3. Algorithms

The gravity force algorithms used in CUDA-Gadget are Barnes-Hut trees (Barnes & Hut 1986) and the particle-mesh (PM) method (Hockney & Eastwood 1981), introduced in combined fashion by Bode et al. (2000). The Barnes-Hut approximation algorithm and

the hybrid TreePM force approximation scheme have been extensively tested numerically (Barnes 1990; Springel 2005; Heitmann et al. 2008).

The other numerical method to be concerned with here is the choice of ODE integration methods (which solve eq. 2). Gadget-2 uses a modified Verlet algorithm, which is preferable to lower-order Runge-Kutta methods (for details, see Springel 2005). CUDA-Gadget is unmodified with respect to Gadget-2 in its integration code.

3.1. Barnes-Hut tree

As described in Barnes and Hut's original paper, the basic idea of the tree-force approximation algorithm is to approximate the gravitational forces between particles in very large simulations by adaptively changing the resolution of forces between particles based on the distance between them, i.e. particles farther away contribute much less to the gravitational potential so their contribution to the potential can be less precise without significant loss of integration accuracy. Particles are grouped into an *oct-tree* cell structure prior to computing forces for any particles, thus providing hierarchical spatial groupings of particles (or 'trees') which can then be 'walked' to the desired depth in order to approximate the force on a given particle.

The oct-tree is a data structure that splits the simulation box into octants (splitting it in half along each axis), and then recursively splits each octant into sub-octants, stopping when there are only one or zero particles per octant (fig. 1). The tree structure is thus adaptive to regions of higher particle density, an important feature of the approximation scheme, as we will see.

The forces are computed on a given particle by 'walking' the tree from the top-down: the eight cells of the highest-level subdivision is considered, 'opening' a subdivision cell if

that cell is sufficiently close to given particle (there are other, similar criteria considered by Gadget-2, as detailed in the code paper). If the cell is opened, then its subdivided constituents are considered. If the cell is not opened, then the force on given particle is computed due to the monopole (or, in some versions of the Barnes-Hut algorithm, but not considered here, mono- and quadrupole) moment of the mass in the cell (this can be efficiently precomputed for all cells during the construction of the tree, which is the key computation-saving feature of the approximation). This process is repeated for all particles.

In the Gadget-2 implementation, the tree is constructed in such a fashion consistently across all MPI processes, using available distributed memory. However, each process only 'knows' about the sub-cells of the tree that have particles stored in the local process memory, so the cells that happen to have sub-cells and their respective particles residing in other processes' memory are specially flagged in the local copy of the tree as virtual particles ('pseudo particles'), so that the force calculation routine can keep track of the information that needs to be communicated to other processes.

4. GPU implementation

The GPU implementation had to face the challenge of distributing the computational tasks normally executed on one CPU core with one thread onto dozens of GPU cores with thousands of threads. The most low-level parts of this task are handled by the CUDA programming model developed by NVIDIA for its GPU devices. However, the logical division of computational work as well as memory synchronization between CPU and GPU device memory must be manually handled by the programmer. Only the Barnes-Hut tree walk and force summation is computed on the GPU. The rest of the computations are done as normal on the CPU with minimal code modification. This latter part of the code handles the MPI parallel communication between CPU cores or nodes when compiled with MPI

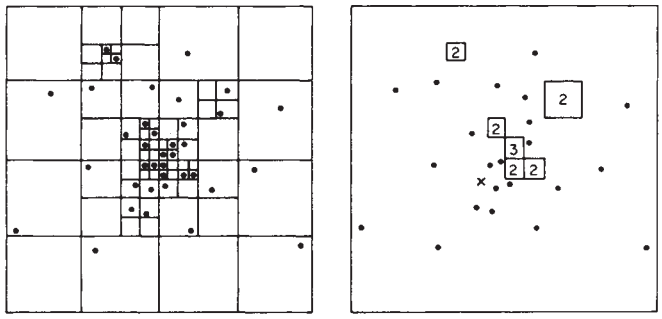

Fig. 1.— Figure 1 from Barnes & Hut (1986). "Hierarchical boxing and force calculation, presented for simplicity in two dimensions. On the left, a system of particles and the recursive subdivision of system space induced by these particles. Our algorithm makes the minimum number of subdivisions necessary to isolate each particle. On the right, how the force on particle x is calculated. Fitted cells contain particles that have been lumped together by our 'opening angle' criterion; each such cell represents a single term in the force summation."

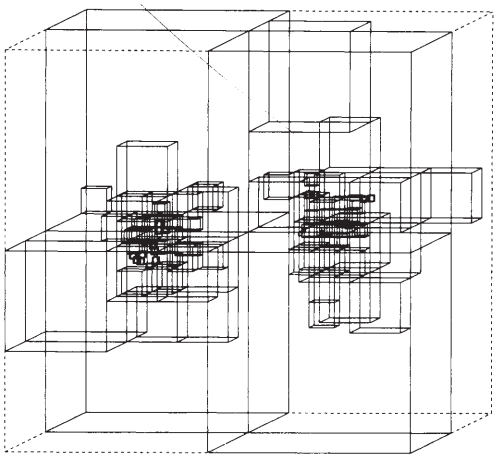

Fig. 2.— Figure 2 from Barnes & Hut (1986). "Box structure induced by a three-dimensional particle distribution. This example was taken from the early stages of an encounter of two $N = 64$ systems, and shows how the boxing algorithm can accommodate systems with arbitrarily complex geometry."

in a multicore or cluster environment with identical code (with one important exception, discussed below) to the public version of Gadget-2.

Initial testing revealed that the tree-code computation on the CPU took up between 90 and 95 percent of total simulation time. Since this was the only part of the computation coded to run on the GPU, Amdahl's law limits the total speedup of the simulation due to the use of the GPU (regardless of how fast the GPU can do computations) to a factor of 10–20, a limit that is approached by the code for certain types of simulations.

4.1. Threading and memory model

The original developer of the G2X code took the strategy of transferring all of the particle data and tree structure to the GPU device memory before starting to compute the Barnes-Hut tree code on the GPU, and transferring the results back to CPU main memory after the whole force calculation. This introduces the problem of keeping track of the particles ('export particles' in the Gadget code paper) in the local tree which experience significant forces from particles residing on other nodes (represented in the local tree by 'pseudo particles'); hence, other MPI processes must compute particle-particle forces between the target particle and the nonlocal particles. This introduces additional logic into the MPI communication, as there is now a buffer of 'export particles' which are sent for force calculation with respect to the particles on other nodes.[3] This mechanism is expected to need to be changed for Gadget-3 (Springel 2012).

The force on each particle is computed independently, with a global tree walk for each particle run on a GPU thread. This has the advantage of producing (in principle) numerically identical forces to the standard version of Gadget-2 (avoiding numerical

[3]The correctness of this piece of the code is the only remaining obstacle to code release.

differences that arise in parallel sum reductions, see Higham 1993). The forces for many thousands of particles are computed simultaneously on each GPU card. (The number of threads is a tunable parameter which should require some adjusting for simulations with order-of-magnitude different particle loads.) This consideration is not trivial, since one of the goals of this project was to see if it were possible to reconstruct the exact accuracy of the standard Gadget-2 code.

As noted above, the number of threads is a manually-tuned parameter. The GPU hardware schedules synchronous groups of threads in units called 'blocks.' Therefore there are two parameters: threads per block, and total number of blocks. NVIDIA recommends setting the number of threads to a multiple of 32 for hardware-specific reasons and setting the number of blocks to a number in the thousands, in order to fully take advantage of future advances in hardware. However, in our implementation, this number can be adjusted by the user through a configuration file. We choose, as a reasonable default, threads-per-block of 64 and a block count of 1024. These values maximize the hardware 'occupancy' for our tree code, which is limited by the number of registers available in GPU hardware to one-third of all execution units on the GPU. In principle, three times as many threads could execute simultaneously on the GPU if additional hardware registers were available (as they are on the new Kepler-class GPUs). The actual performance gain would depend on memory contention and cache efficiency, however.

This threading model, however, is not necessarily ideal for memory accesses. As implemented in the code, each tree walk (i.e. each thread) requires accesses to global memory (in the absence of hardware caching; however, we enable the larger L1 cache configuration and find L1 cache efficiency to be ≈ 75 percent for the galaxy simulation detailed in section 5.2). However, threads that execute in the same hardware unit (warp) coalesce memory accesses, so if a group of threads all access very similar data, this penalty

is reduced significantly. In this implementation, particles are assigned by particle ID to threads, so that threads nearby in ID-space are nearby in thread-space. For cosmological initial conditions appropriately created, close particle IDs should indicate spatial proximity (for most particles)[4]. Re-ordering particles as part of the tree construction may improve tree-walking efficiency (although for the galaxy simulation below we find that branching operations for a given warp only diverge for ≈ 10 percent of conditional branches), but this is left for future work.

More explicit management of the GPU memory hierarchy can create problems unless the programmer is very careful about correctness. Caching strategies and parallel reduction threading models will almost certainly introduce bugs in the first coding attempt unless one is an an expert shared-memory parallel programmer, since there is (almost) no compiler or hardware protection against race conditions or write-after-read memory inconsistencies (there is no hardware cache coherence, for instance). Debugging tools have improved greatly with the advent of versions four and five of CUDA, but are still substandard compared to the robustness and richness of tools available for on-CPU debugging (especially problematic for hybrid CUDA-MPI codes). There are hardware atomic operations present in newer GPUs that protect against non-cache memory inconsistencies, but performance seems to be still too slow for practical use in N-body codes.

4.2. Numerical precision and rounding issues

In principle, the GPU and CPU versions of Gadget-2 should yield the same numerical results. However, we discovered that single-precision computations are not, in fact, identical across architectures. As noted in a technical document, the details of the rounding in

[4](as noted by Frigaard in the release notes)

arithmetic operations in GPUs is not identical to that in CPUs, although they should meet a minimal level of accuracy established by the IEEE standard (Whitehead & Fit-Florea 2011). Hence double precision arithmetic and data structures were implemented in the code and enabled by default. An additional source of numerical differences is that of *fused multiply-add* instructions, which combine multiply and add operations without an intermediate rounding step. (Among mainstream architectures, this combined operation without intermediate rounding currently only exists on GPUs, but will be introduced to Intel CPUs in 2013.) In principle, these arithmetic operations should be more precise and result in a more accurate simulation. However, rounding differences at this level should be subdominant to all other sources of error.

A more problematic source of error that has usually required fine-tuning of simulation parameters in order to control has been the scale cut-off between short- and long-range force calculations (see sections 3 and 6.3 in Heitmann et al. (2010), figure 2 in Springel (2005)). The smoothing between calculational methods requires the evaluation of the complementary error function for each force term, a difficult function to approximate numerically. Since this is extremely expensive computationally, Gadget-2 uses a look-up table to approximate this factor. On the GPU, we can in principle take advantage of the CUDA library function `erfc`, which should be considerably faster than on the CPU due to the faster floating-point arithmetic and additional intrinsics. However, we leave careful numerical tests of this for future work.

For a fair comparison of precision between CPU and GPU codes, we enabled compiler options for strict IEEE compliance and strict double precision in the x86 code (no extended-precision allowed on x86, since there is no such hardware equivalent for GPUs).[5]

[5]A summary of our compiler options (`nvcc` is for CUDA-Gadget only, of course):

```
nvcc -m64 -arch sm_20 -ftz=false -prec-div=true -prec-sqrt=true -fmad=false -Xptxas -dlcm=ca
```

We chose to use `gcc` since it is the most commonly used compiler for scientific work and is the only officially-supported compiler for CUDA.[6]

4.3. Current state of the code

There is one outstanding bug involved with multiple MPI processes and tree-walking on the GPU. On single-process GPU runs, the results of all tests we could run are numerically identical to the accuracy of Gadget-2 snapshot outputs (i.e. single precision). For multiple-process runs, there remain some unresolved discrepancies (see fig. 8 and discussion below).

5. Astrophysical examples

5.1. The Universe

We ran cosmological simulations of varying particle number in a comoving volume of size $(130Mpc)^3$ using standard cold dark matter initial conditions, with parameters $\Omega_m = 0.25$, $\Omega_\Lambda = 0.75$, and $\sigma_8 = 0.8$. For evolution to the starting redshift $z_{start} = 126$, we used 2LPT evolution from the primordial CMB power spectrum (Crocce et al. 2006).

As an illustration, for the 384^3-particle simulation, we show halo profiles (fig. 3; fig. 4) mass functions (fig. 5), mass function residuals (fig. 6), and a ray-traced visualization (fig. 7).

For the 512^3-particle simulation running on 36 cores and 36 GPUs, the total runtime of

---

`mpicc -m64 -O3 -mmmx -msse -msse2 -msse3 -mfpmath=sse`

[6]This does not mean that you should use it, if you have a choice.

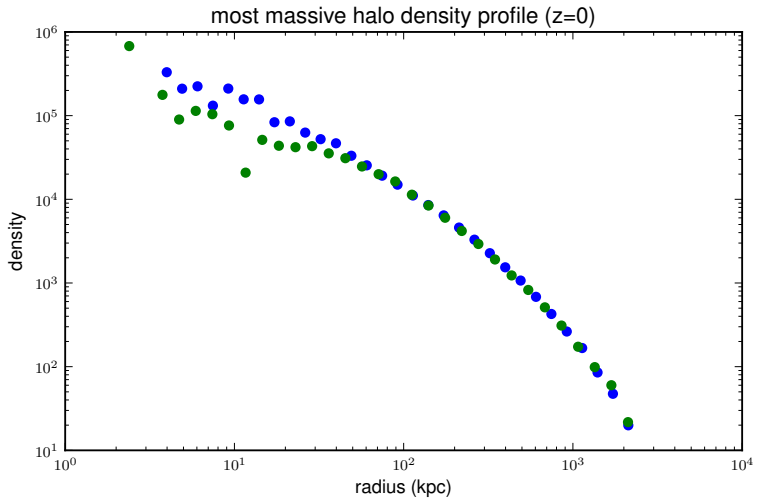

Fig. 3.— A plot of the most massive halo in a simulation, with radial density profiles at $z = 0$ computed with AHF (Knollmann & Knebe 2009) from the outputs of CUDA-Gadget and standard Gadget-2. There are small discrepancies, as noted in the text, due to a remaining bug in the interaction between MPI and CUDA code.

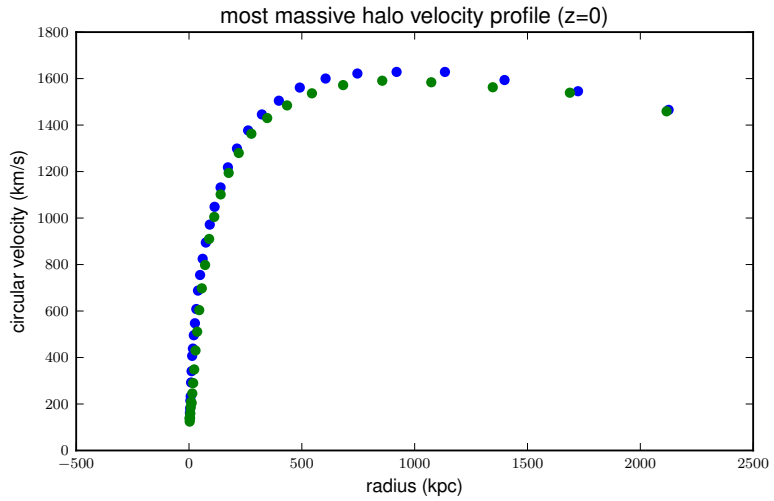

Fig. 4.— As above, but for radial velocity.

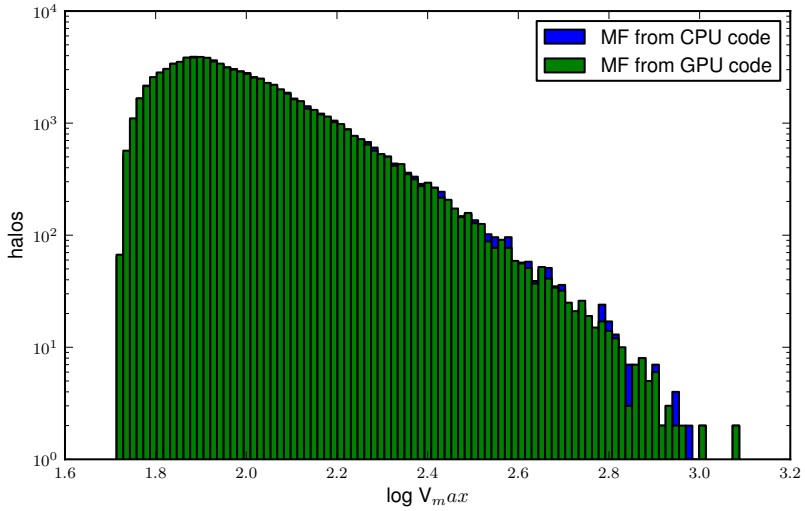

Fig. 5.— The halo mass function for the GPU results and CPU results at $z = 0$ (also with AHF halos).

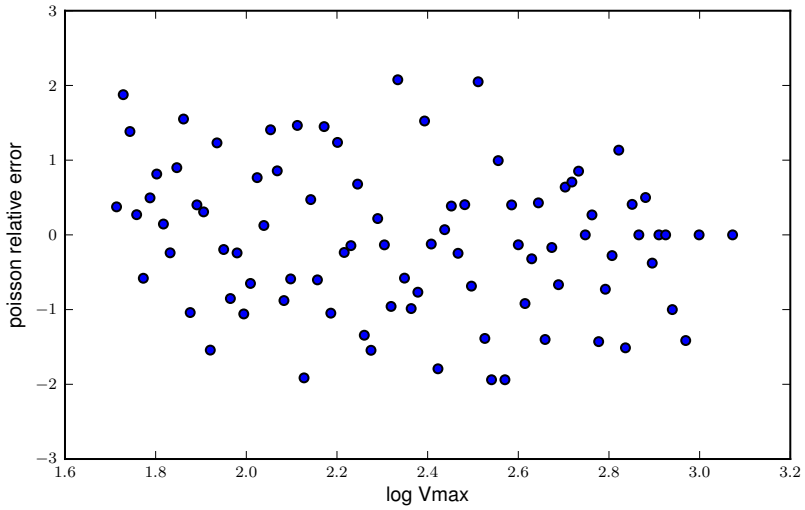

Fig. 6.— The residuals between the mass function from the GPU results and the CPU results normalized relative to Poisson error in halo counts in the CPU results).

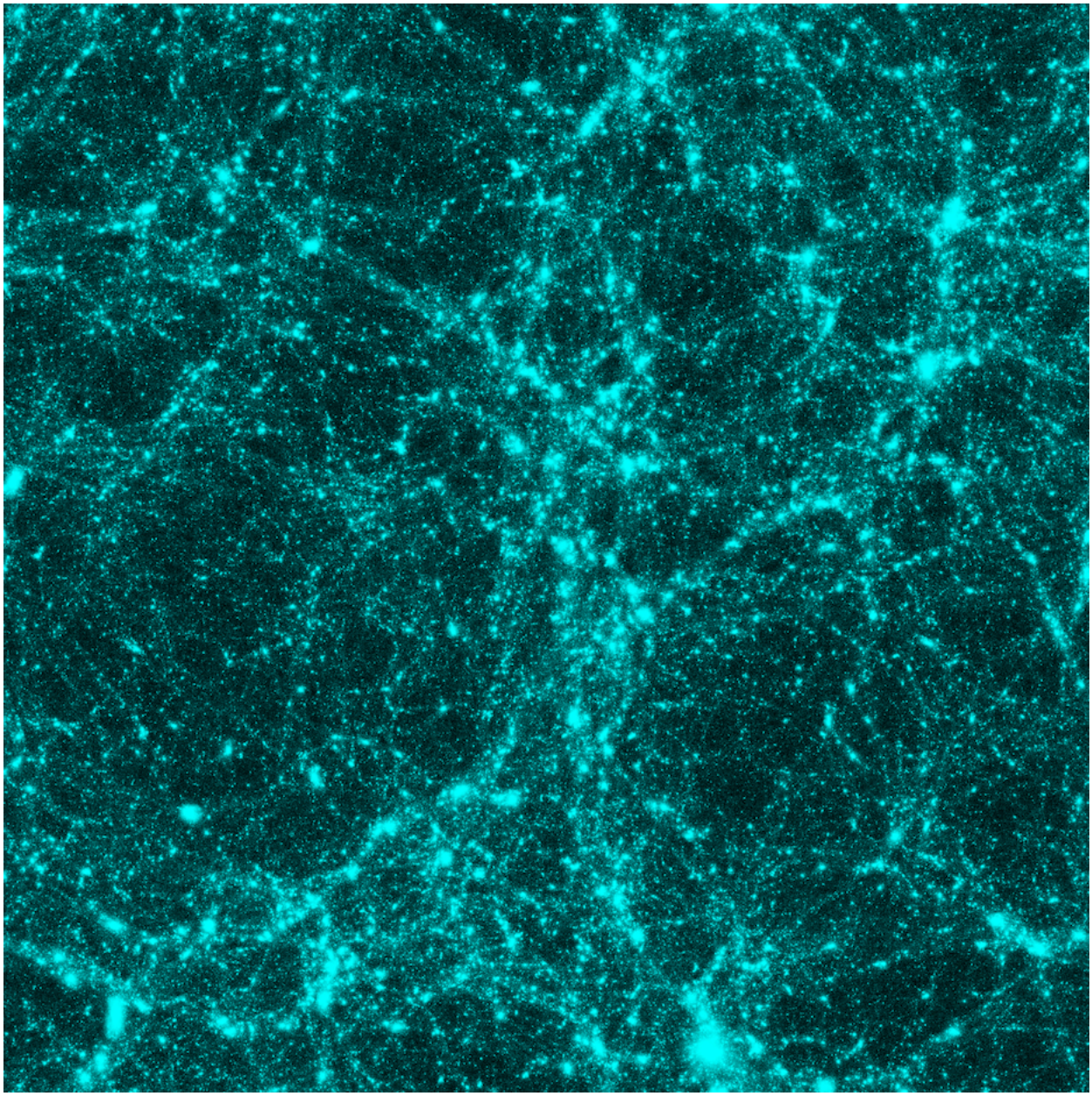

Fig. 7.— A 3D ray-traced visualization of particles from a cosmological simulation run with CUDA-Gadget at $z = 0$. Rendered with Splotch (Dolag et al. 2008).

the simulation using only CPU cores was 35.6 hours, compared to a GPU-enabled runtime of 12.7 hours, yielding a speedup factor of 2.8. The speedup is less impressive than that for the galaxy simulation (see below); this is likely due to the larger size of the tree in memory causing cache problems. The cache hit rate measured for this cosmological simulation is 65 percent, meaning that this simulation is about 10 percent less cache-efficient than the galaxy simulation (as discussed in section 4.1). Since the latency for GPU memory access is on the order of 20 to 40 times worse than due to cache access, this yields a worst-case scenario slowdown of 3.6, which is close to the observed relative slowdown.

5.2. Galaxies: M81

We simulated a M81-type galaxy in isolation (realized as 17.5 million total particles in the halo, bulge, and disc) as a test of galaxy-size simulation problems, which live on a smaller physical scale and where small-scale forces are comparatively more important. Even in this case, the discrepancies between CUDA-Gadget and Gadget-2 are apparent only with a detailed analysis (fig. 8).

With the CPU Gadget-2 running just under 24 hours and CUDA-Gadget running for a total of 2.6 hours for identical numbers of time steps (5742), the speedup between CPU and GPU simulations was a factor of 9.4. Due to current limitations of the code, the GPU cards must be matched to the CPU cores in a 1:1 ratio (in this case, 12 GPU cards and 12 CPU cores), so this speedup is on a per-card/per-core basis, even though the recent sharp increase in cores per socket in computer systems may affect whether this is the most practical comparison to make, for instance, for hardware acquisitions. However, this is the fairest way to evaluate the effect of the GPU on speeding up the code; this should also tell us to what extent the code approaches the theoretical maximum speedup which is limited by the amount of code remaining on the CPU, as discussed above. Since the

forces are shorter-range and higher accuracy is needed and periodic boundary conditions are inadvisable, galaxy simulations such as this one use a Barnes-Hut tree-only force computation (no PM hybrid).

6. Future work

The current work is only a beginning in exploring the opportunities provided by GPU clusters for large astrophysical simulations, but should indicate that doing so is both possible and practical, due to the code's public availability, for computational astronomers. Further optimizations, such as temporally overlapping the computation of Fast Fourier Transforms for the long-range PM forces (which run efficiently on the CPU) and the Barnes-Hut tree forces (which run efficiently on the GPU), will only increase the speedup gained with GPUs. (Another strategy might be to use Ewald periodic image summation for periodic forces instead of using a hybrid TreePM scheme, as T (2012) explored.)

Some algorithmic modifications are also suggested by the hybrid CPU-GPU architecture, and work is needed to test their efficiency as well as the accuracy of simulations with such modifications. One such optimization is to group nearby particles into 'buckets' and walk the Barnes-Hut tree for each bucket, rather than for each particle, thus reducing the total number of tree walks and thus improving cache usage, as done in T (2012) and Barnes (1990). In another area, a static domain decomposition strategy such as the 'slab' decomposition used in FFTW, rather than the adaptive one used by Gadget (Springel 2005), might suffice for very large cosmological simulations and would reduce the CPU overhead that limits the maximum speedup available from the GPU (suggested by Nikhil Padmanabhan).

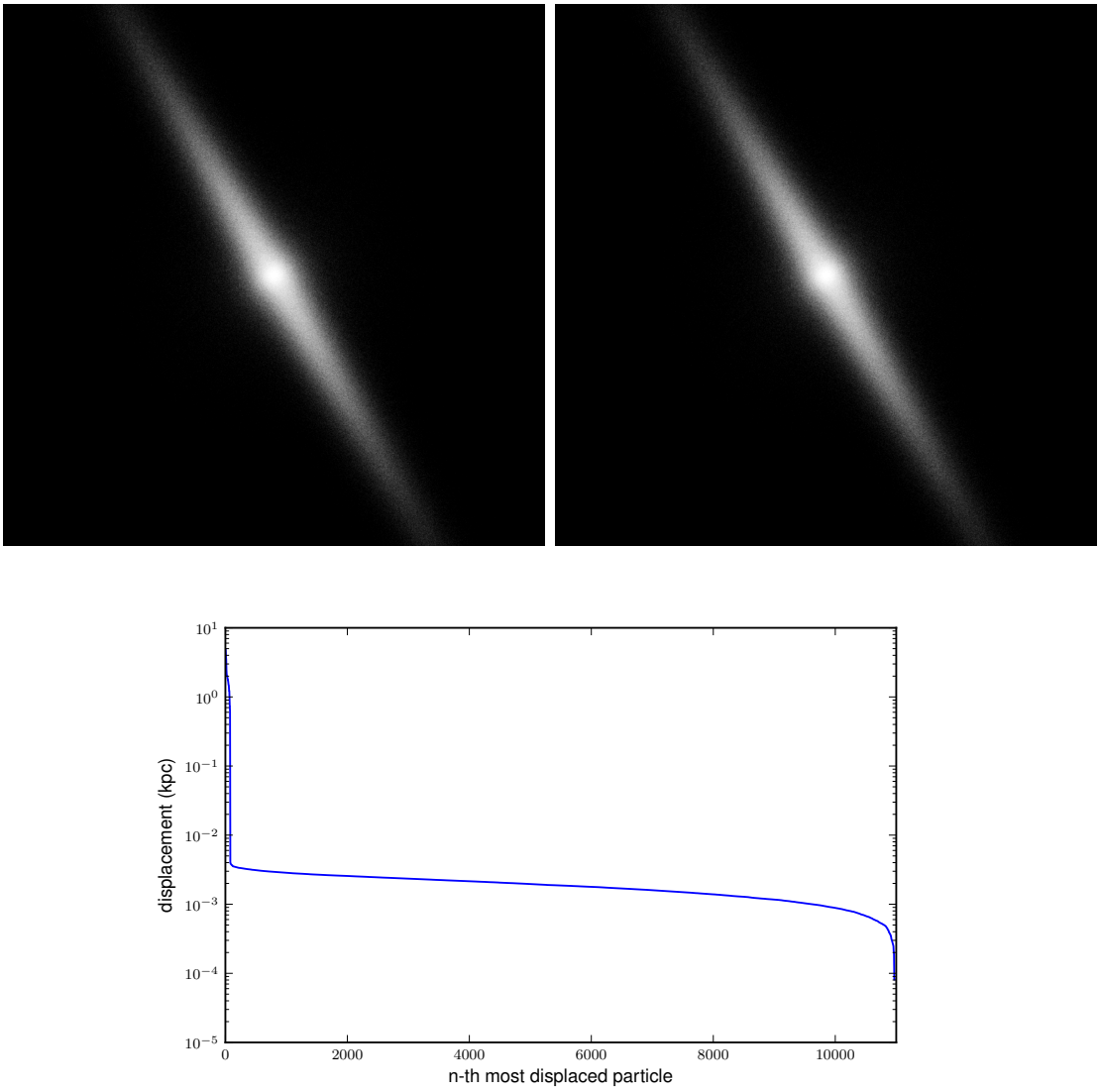

Fig. 8.— A ray-traced density projection of the stars in an M81-type galaxy dynamics simulation. Rendered with Splotch (Dolag et al. 2008). The top left panel shows the system evolved with CUDA-Gadget; the top right shows the evolution with Gadget-2. The initial conditions are identical in both simulation runs. The bottom panel shows the particle displacement between the GPU and CPU simulations as a function of the n-th most displaced particle. The maximum displacement is ≈ 5 kpc, however, the mean displacement was of order 10^{-5} kpc. Of a total of 17.4 million particles, only 10790 were displaced with respect to the particles in the reference simulation. This suggests that there remains a subtle bug in the code for multi-GPU runs, rather than a numerical defect of the GPU code in general.

I am grateful to Andreas Berlind, Kelly Holley-Bockelmann, and Manodeep Sinha for continual discussions and help throughout the project. I thank Prof. Holley-Bockelmann for providing initial conditions for the galaxy simulation. I also acknowledge Carsten Frigaard's work in providing the initial CUDA code base for CUDA-Gadget, and of course Volcker Springel for Gadget itself. Financial support for the author was provided by the College of Arts and Science, Vanderbilt University, and the William and Nancy McMinn Scholarship in the Natural Sciences at Vanderbilt University. Computational resources were provided by ACCRE at Vanderbilt and Keeneland/KIDS at Georgia Tech through a discretionary allocation. We made use of pynbody (http://code.google.com/p/pynbody) and AHF (Knollmann & Knebe 2009) in our analysis for this paper.

More information on CUDA-Gadget is available
at http://code.google.com/p/cuda-gadget.

REFERENCES

2007, Compute unified device architecture programming guide, Tech. rep., Nvidia

Aarseth, S., Henon, M., & Wielen, R. 1974, Astronomy and Astrophysics, 37, 183

Abazajian, K., Zheng, Z., Zehavi, I., et al. 2005, The Astrophysical Journal, 625, 613

Barnes, J., & Hut, P. 1986, Nature, 324, 446

Barnes, J. E. 1990, Journal of Computational Physics, 87, 161

Bédorf, J., Gaburov, E., & Portegies Zwart, S. 2012, Journal of Computational Physics, 231, 2825

Binney, J., & Tremaine, S. 1987, Galactic dynamics (Princeton, NJ: Princeton University Press)

Bode, P., Ostriker, J. P., & Xu, G. 2000, ApJS, 128, 561

Burtscher, M., & Pingali, K. 2011, in GPU Computing Gems Emerald Edition (Morgan Kaufmann), 75–92

Busha, M. T., Wechsler, R. H., Becker, M. R., Erickson, B., & Evrard, A. E. 2013, in American Astronomical Society Meeting Abstracts, Vol. 221, American Astronomical Society Meeting Abstracts, no. 341.07

Crocce, M., Pueblas, S., & Scoccimarro, R. 2006, Monthly Notices of the Royal Astronomical Society, 373, 369

Dehnen, W. 2001, Monthly Notices of the Royal Astronomical Society, 324, 273

Diemand, J., Moore, B., Stadel, J., & Kazantzidis, S. 2004, Monthly Notices of the Royal Astronomical Society, 348, 977

Dolag, K., Reinecke, M., Gheller, C., & Imboden, S. 2008, New Journal of Physics, 10, 125006

Frigaard, C. 2009, unpublished source code.

Frigo, M., & Johnson, S. G. 1998, in Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on, Vol. 3, IEEE, 1381–1384

Habib, S., Morozov, V., Finkel, H., et al. 2012, ArXiv e-prints, arXiv:1211.4864

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

Heitmann, K., White, M., Wagner, C., Habib, S., & Higdon, D. 2010, ApJ, 715, 104

Heitmann, K., Lukić, Z., Fasel, P., et al. 2008, Computational Science and Discovery, 1, 015003

Higham, N. J. 1993, SIAM Journal on Scientific Computing, 14, 783

Hockney, R. W., & Eastwood, J. J. W. 1981, Computer simulation using particles (Taylor & Francis Group)

Holmberg, E. 1941, The Astrophysical Journal, 94, 385

Ishiyama, T., Nitadori, K., & Makino, J. 2012, ArXiv e-prints, arXiv:1211.4406

Ivezic, Z., Tyson, J. A., Acosta, E., et al. 2008, ArXiv e-prints, arXiv:0805.2366

Knollmann, S. R., & Knebe, A. 2009, ApJS, 182, 608

Makino, J., & Hut, P. 1989, Celestial Mechanics, 45, 141

McBride, C., et al. 2013, in preparation

N-Body Shop. 2011, ChaNGa: Charm N-body GrAvity solver, astrophysics Source Code Library, ascl:1105.005

Navarro, J. F., Frenk, C. S., & White, S. D. 1997, The Astrophysical Journal, 490, 493

Nyland, L., Harris, M., & Prins, J. 2007, GPU Gems 3 (Addison-Wesley Professional), 677–695

Quinlan, G. D., & Tremaine, S. 1992, MNRAS, 259, 505

Reed, D. S., Smith, R. E., Potter, D., et al. 2013, MNRAS, arXiv:1206.5302

Schive, H.-Y., Tsai, Y.-C., & Chiueh, T. 2010, ApJS, 186, 457

Springel, V. 2005, MNRAS, 364, 1105

Springel, V. 2012, personal communication

Springel, V., White, S. D., Jenkins, A., et al. 2005, Nature, 435, 629

T, S. 2012, Master's thesis, Indian Institute of Science, Bangalore, India

Vetter, J. S., Glassbrook, R., Dongarra, J., et al. 2011, Computing in Science & Engineering, 13, 90

Whitehead, N., & Fit-Florea, A. 2011, Precision and Performance: Floating Point and IEEE-754 Compliance for GPUs, Tech. rep., NVIDIA

Yokota, R., & Barba, L. A. 2011, 1108.5815