Journal of Molecular Graphics and Modelling 28 (2010) 904–910

Contents lists available at ScienceDirect

Journal of Molecular Graphics and Modelling

journal homepage: www.elsevier.com/locate/JMGM

Accelerating electrostatic surface potential calculation with multi-scale approximation on graphics processing units

Ramu Anandakrishnan[a,+], Tom R.W. Scogland[b,+], Andrew T. Fenley[c], John C. Gordon[d], Wu-chun Feng[e,*], Alexey V. Onufriev[f,*]

[a] Department of Computer Science, Virginia Tech, 2050 Torgersen Hall (0106), Blacksburg, VA 24061, United States
[b] Department of Computer Science, Virginia Tech, 2209 KnowledgeWorks II Building (0902), Blacksburg, VA 24060, United States
[c] Department of Physics, Virginia Tech, 2050 Torgersen Hall (0106), Blacksburg, VA 24061, United States
[d] Microsoft, 1 Microsoft Way, Redmond, WA 98052, United States
[e] Departments of Computer Science and Electrical & Computer Engineering, Virginia Tech, 2209 KnowledgeWorks II Building (0902), Blacksburg, VA 24060, United States
[f] Departments of Computer Science and Physics, Virginia Tech, 2050 Torgersen Hall (0106), Blacksburg, VA 24061, United States

ARTICLE INFO

Article history:
Received 14 November 2009
Received in revised form 3 April 2010
Accepted 7 April 2010

Biomolecular electrostatics
Multi-scale modeling
Graphical processing unit (GPU)
Implicit solvent model

ABSTRACT

Tools that compute and visualize biomolecular electrostatic surface potential have been used extensively for studying biomolecular function. However, determining the surface potential for large biomolecules on a typical desktop computer can take days or longer using currently available tools and methods. Two commonly used techniques to speed-up these types of electrostatic computations are approximations based on multi-scale coarse-graining and parallelization across multiple processors. This paper demonstrates that for the computation of electrostatic surface potential, these two techniques can be combined to deliver significantly greater speed-up than either one separately, something that is in general not always possible. Specifically, the electrostatic potential computation, using an analytical linearized Poisson–Boltzmann (ALPB) method, is approximated using the hierarchical charge partitioning (HCP) multi-scale method, and parallelized on an ATI Radeon 4870 graphical processing unit (GPU). The implementation delivers a combined 934-fold speed-up for a 476,040 atom viral capsid, compared to an equivalent non-parallel implementation on an Intel E6550 CPU without the approximation. This speed-up is significantly greater than the 42-fold speed-up for the HCP approximation alone or the 182-fold speed-up for the GPU alone.

1. Introduction

Electrostatic interactions are critical for biomolecular function. [1–12] At the same time, the long-range nature of these interactions often makes their estimation a computational bottleneck in current atomistic molecular modeling [13,14]. Approaches to speed-up these computations can generally (though not always cleanly) be subdivided into two very broad categories: (1) those that seek to gain speed by making computationally effective approximations to the underlying physical model and (2) those that do *not* affect the accuracy of the physical model but strive to accelerate the computation at the software or hardware levels. Examples in the first category include the spherical cut-off method [15,16], the fast multipole approximation [17–19], and the current "industry standard" for explicit solvent simulations—the particle mesh Ewald (PME) method [20–23]. Among the most prominent practical approaches in the second category is parallel computation on multiple processing cores. Each of these approaches has its advantages and limitations [24,25]. For example, the PME method can be very accurate but requires an artificial periodicity to be imposed on the molecular system. In addition, the method is currently not suitable for implicit solvent simulations [26]. Parallel computation on multiple processing cores may lead to spectacular speed-ups for certain types of problems [27], but not for others—for example, specific PME implementations may not scale all that well on large parallel machines. Furthermore, access to such expensive resources may be limited.

Within either of the above general categories of approaches, estimating the long-range electrostatic interactions is still computationally intensive. Even a single state computation which involves long-range electrostatics, may require an extraordinary amount of computational resources for larger structures. For example, a 2006 study of electrostatic properties of viral capsids [28] using

* Corresponding authors.
E-mail addresses: ramu@cs.vt.edu (R. Anandakrishnan), tom.scogland@vt.edu (T.R.W. Scogland), afenley@vt.edu (A.T. Fenley), john.gordon@microsoft.com (J.C. Gordon), feng@cs.vt.edu (W.-c. Feng), alexey@cs.vt.edu (A.V. Onufriev).
+ First two authors contributed equally to this work.

1093-3263/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jmgm.2010.04.001

the adaptive Poisson–Boltzmann solver [29], required 1000 processors on the Blue Horizon and Data Star supercomputers for each run.

Computational requirements for modern molecular dynamics, which may require millions of such single state estimates, are even more demanding. [30–34].

In the past, in addition to algorithmic advances, scientists could also rely on Moore's Law [35] to continually accelerate these computations.[1] The rapid hardware advances from Moore's Law gave software a "free ride" to better performance, but this free ride is now over. With clock speeds stalling out and computational horsepower instead increasing due to the rapid doubling of the number of cores per processor, serial computing is now moribund in many areas of natural science, and the vision for parallel computing, which started over 40 years ago, is a revolution that is now upon us.

The traditional approach to parallel computing has made use of large-scale supercomputers, oftentimes referred to as *big iron*. However, such supercomputers present significant challenges with respect to ease of access and use, and cost, e.g., the fastest supercomputer in the world in 2009 cost $133M to build [36]. In contrast, the growing proliferation of many-core processors, like the graphics processing unit (GPU) on a video card, promises to deliver supercomputing horsepower to the desktop while simultaneously enhancing ease of access as well as dramatically reducing cost. With the peak floating-point performance of a GPU now at a teraflop (10^{12} floating-point operations per second), the GPU delivers supercomputing in a small and economical package. For example, a high-end server with the latest GPU card costs a mere $1,500, resulting in an astounding performance-price ratio of 667 megaflops per dollar and performance–space ratio of 500 teraflops per square foot. In contrast, the world's fastest supercomputer, Roadrunner, has a peak of 1457 teraflops at a cost of $133M for a mere performance-price ratio of 11 megaflops per dollar and performance–space ratio of 243 teraflops per square foot. However, the current programming model for GPUs is only amenable to highly data-parallel applications; efficient GPU mappings for less data-parallel applications are extraordinarily difficult to realize [37]. Unlike supercomputer clusters consisting of general-purpose processors and direct support for interprocessor communication, the GPU has limited interprocessor communication capabilities and limited data cache. Therefore the GPU is most effective when performing stream processing, i.e., performing a similar set of computations against a large set of data. This paper discusses the techniques used to address limitations of the GPU while taking advantage of its multiprocessing capabilities in the context of electrostatic computations.

A number of different biomolecular modeling applications have recently been implemented on GPUs [38–40], including the computation of long-range electrostatic potential in the context of molecular dynamics [41–44]. Our implementation focuses on the computation of electrostatic surface potential, using a realistic solvation model, in order to demonstrate that multi-scale approximation schemes, such as the hierarchical charge partitioning (HCP) algorithm [25], can be combined with the GPU to achieve significantly greater speed-up than the HCP approximation or the GPU alone. Specifically, we algorithmically map and transform electrostatic potential calculation [45] along with the HCP approximation onto the GPU.

The remainder of this paper is organized as follows. In the next section, we briefly describe the specific application considered here (i.e., computation of molecular surface potential), the HCP, and the GPU implementation. Then we examine the speed and accuracy of this implementation for a set of representative biomolecular structures. In conclusion, we summarize our findings and discuss the future potential for the approach presented here.

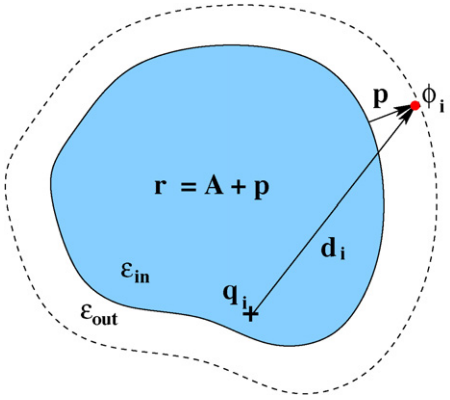

Fig. 1. Definition of the geometric parameters that enter Eq. (1) used to compute the electrostatic potential ϕ_i due to a single charge located inside an arbitrary biomolecule (in the absence of mobile ions). Here $\boldsymbol{d_i}$ is the distance from the source charge q_i to the point of observation where ϕ_i needs to be computed. The molecular surface (solid line) separates the low dielectric interior (ϵ_{in} blue region) from the high dielectric solvent space, ϵ_{out}. We project the molecular surface defined by pre-computed vertex points a small distance, **p**, outwards into the solvent space along the surface normals. The projected surface is shown as the dashed line. The so-called effective electrostatic size of the molecule, **A**, characterizes its global shape and is computed analytically [46]. The distance from the point of observation to the "center" is then defined as $r = \mathbf{A} + \mathbf{p}$.(For interpretation of the references to color in this figure legend, the reader is referred to the web version of the article.).

2. Methods

This section describes the specific methods used to compute long-range interactions and an implementation of the entire computational process on the GPU. Specifically, we investigate the extent to which the speed-up resulting from approximating the electrostatic potential via a multi-scale approach (HCP, described below) can be combined with the speed-up resulting from mapping and transforming the electrostatic potential calculation described below, onto the GPU.

The implementation of GEM with one level of HCP for the CPU, with full graphics functionality, can be downloaded from http://people.cs.vt.edu/~onufriev/software. The platform-specific computational core for running GEM with one level of HCP on the GPU, as described below, is available from the authors upon request.

2.1. Computation of biomolecular electrostatic potential

All of the electrostatic calculations presented in this paper were done using the analytic, linearized Poisson–Boltzmann (ALPB) model [45–47] as implemented in the GEM package. The functional form of the ALPB electrostatic potential ϕ_i varies depending on the region of space where the potential is computed; two of the regions – the interior of the molecule and the solvent space – are depicted in Fig. 1 as the two dielectrics. The specific functional forms of ϕ_i for all the regions are given in Gordon et al. [45]. Eq. (1) is the simplest solution for the calculation of physically admissible ϕ_i anywhere in the solvent, including the molecular surface, see Fig. 1. The constant $\alpha = 0.580127$, in Eq. (1) minimizes the error in the solvation energy of a random charge distribution inside a sphere [48]. Adding the effects of salt in the Debye-Hückel (linear) limit is outlined in Gordon et al. [45]. The potential at any single point in

[1] Moore's Law states that the number of transistors that can be inexpensively placed on an integrated circuit doubles every 24 months. This doubling in transistors typically translated into a corresponding performance improvement.

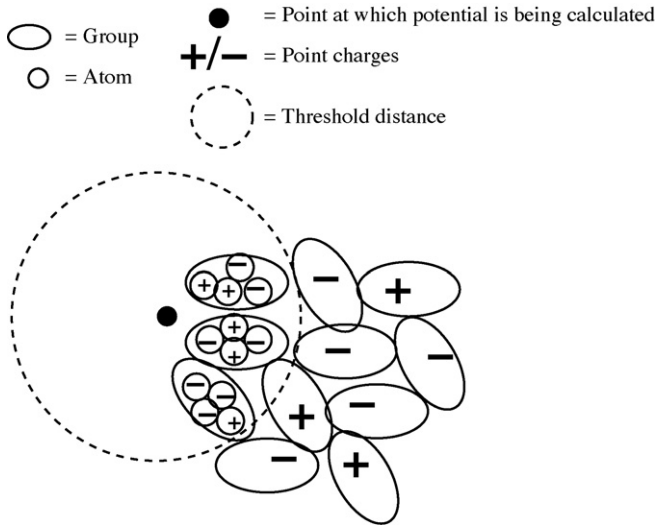

Fig. 2. Illustration of the hierarchical charge partitioning (HCP) approximation. In this illustration a biomolecular structure is partitioned into its constituent groups, each of which consists of multiple atoms. The charge distribution of each group is approximated by a single charge. The threshold distance determines when to use the approximate charge distribution in electrostatic computations, instead of individual atomic charges.

space, for example a vertex point on the molecular surface, is computed as a sum of contributions from individual atomic charges in the solute. Such a calculation can be done without the need to also compute the potential at any other point(s) – a computational freedom that the numerical Poisson–Boltzmann solvers are missing. An extensive analysis of the accuracy of the ALPB model in computing biomolecular electrostatic potential is presented elsewhere [45].

$$\phi_i^{\text{solvent}} = \frac{q_i}{\epsilon_{\text{out}}} \frac{1}{(1+\alpha(\epsilon_{\text{in}}/\epsilon_{\text{out}}))} \left[\frac{(1+\alpha)}{d_i} - \frac{\alpha(1-(\epsilon_{\text{in}}/\epsilon_{\text{out}}))}{r} \right] \quad (1)$$

To assess the speed-up resulting from the combination of the HCP and the GPU, we selected the GEM software package [45]. GEM is an open-source implementation of the ALPB model. GEM was selected as a platform for experimentation, in this case, due to the facile nature of the algorithm and the flexibility of the platform for innovation and modification to rapidly generate prototypes.

2.2. The hierarchical charge partitioning (HCP) approximation

The hierarchical charge partitioning (HCP) approximation [25] exploits the natural partitioning of biomolecules into constituent structural components in order to speed-up the computation of electrostatic interactions with limited and controllable impact on accuracy. Biomolecules can be systematically partitioned into multiple molecular *complexes*, which consist of multiple polymer chains or *subunits*, which in turn are made up of multiple amino acid or nucleotide *groups*. These components form a hierarchical set with, for example, complexes consisting of multiple subunits, subunits consisting of multiple groups, and groups consisting of multiple atoms. Atoms represent the lowest level in the hierarchy while the highest level depends on the problem. Briefly, HCP works as follows. As illustrated in Fig. 2, the charge distribution of components, other than at the atomic level, is approximated by a small set of point charges. The electrostatic effect of distant components is calculated using the smaller set of point charges, while the full set of atomic charges is used for computing electrostatic interactions within nearby components. The distribution of charges for each component used in the computation varies depending on distance from the point in question: the farther away the component, the fewer charges are used to represent the component. The actual speed-up from using HCP depends on the specific hierarchical organization of the biomolecular structure under consideration. Under conditions consistent with the hierarchical organization of realistic biomolecular structures, the HCP algorithm scales as $O(N \log N)$, where N is the number of atoms in the structure. For large structures, the HCP can be several orders of magnitude faster than the exact $O(N^2)$ all-atom computation. A detailed description of the HCP algorithm can be found in Anandakrishnan and Onufriev [25]. For the purpose of the analysis presented in this work, we use the 1-charge approximation, where the charge distribution of components is approximated by a single charge. Increasing the number of charges used in the approximation would increase accuracy and computational cost. The HCP can also use multiple hierarchical levels of partitioning of biomolecular structures, as described above. However, in the current work we use the first level of partitioning for the HCP algorithms. Increasing the number of levels used by the HCP algorithm reduces the computational cost further.

2.3. Mapping GEM and HCP onto the GPU

The initial exploration of mapping GEM and HCP to the GPU was conducted using the ATI Stream Software Development Kit (SDK) from AMD. The SDK provides a high-level programming language called ATI Brook+ [49] to ease the development of applications for the GPU, and it includes a compiler and run-time layer that handles the low-level details necessary to run computations on the GPU.

Brook+ is based on the Brook stream computing language from Stanford [50], which in turn is an extension of standard ANSI C. Brook+ is specifically optimized to compile and run on ATI stream-capable GPUs through the AMD/ATI Compute Abstraction Layer (CAL). Whereas CAL presents a low-level programming interface, Brook+ provides a high-level programming interface, thus easing the development of GPU applications.

The calculation of electrostatic potential via the analytical formula implemented in GEM, mentioned above, can be decomposed as a data-parallel computation across all points on the surface, or vertices, at which the electrostatic potential is calculated. In addition, the potential at each vertex can be computed as a reduction (or more specifically, a sum) of a set of independent calculations on each atom, thus allowing for multiple dimensions of parallelism.

The basic mapping of GEM to the GPU treated the calculation of potential at each vertex as an individual unit of execution, and thus, computed one vertex per GPU thread. In the initial version, the electrostatic surface potential at all the vertices was computed with a single kernel launch, which is equivalent to an offloaded function call to the GPU device.

We found that while this approach worked, it did not scale. That is, as the number of threads increased beyond a certain point (in this case, beyond 25,000–30,000 threads), so did the overhead of scheduling them, which in turn, degraded performance. Specifically, our tests showed that less than 25,000 was a reasonable number for the ATI Radeon 4870 card; anything more than 30,000 was materially slower despite the reduced number of kernel launches. In light of this, subsequent versions only ran at most 25,000 threads per kernel launch.

While the above mapping was effective, it still retained many features of the original code that do not translate favorably into GPU performance. Below we describe additional modifications to improve the execution of GEM on the GPU.

2.3.1. Data structures

While Brook+ version 1.4 supports structures, it does not support referencing structure members in an array of structures. As a result, all the arrays of structures in GEM had to be flattened into arrays of primitives in order to get the GEM code to even run on

the GPU. As much as this change was necessary to enable GEM to run on the GPU, the change also made sense to do from a performance standpoint as it aligns the data structures to allow for coalesced memory accesses. We intend to investigate this further when the support for structures in Brook+ improves sufficiently for us to implement the code with the original structure of the data intact.

2.3.2. Conditional performance

Conditional statements, particularly conditionals that cause divergence (or divergent branches), are a common cause of performance degradation in GPU programs, as has been documented in previous work, including the AMD Stream User Guide [51]. Non-divergent branching, on the other hand, is generally not mentioned in connection with performance issues. However, they too can incur a high cost; in the case of GEM, 30% performance degradation on the GPU and 15% on the CPU. Thus, minimizing conditionals in high traffic areas of code is long-standing conventional wisdom, which is often forgotten when working with modern CPUs. (Programmers sometimes rely on the branch prediction or speculative execution capabilities of CPUs to deal with potentially excessive use of conditionals.) For GPUs, the impact of branching is more severe since GPUs lack branch prediction and speculative execution, and other technologies that reduce the cost of branching, as on modern CPUs.

In GEM's original computational core, most conditionals were used to keep the code maintainable, e.g., selecting between code paths even though they were predetermined by the parameters to the program. Specifically, there are two input parameters: (1) the region in which the potential is computed and (2) the type of potential to be computed, which determine the execution path through five conditionals. For example, two of the conditionals select the region being computed, such as the interior of the molecule or the solvent space shown in Fig. 1.

To remove these conditionals, we created several different versions of the GEM function and Brook+ kernel, 15 total, each of which now contains only the conditionals that must be evaluated on every input item individually, cutting the total number of branches in most kernels to zero and at the most three.

The specific kernel to execute is now selected by a set of conditionals *outside* the function/kernel being run and the conditionals are only invoked once. A total of 2–5 conditionals are in stark contrast to what was originally required. The initial version required $\#atoms \times \#vertices \times \#conditionals$ conditional statements, or for the viral capsid, $476,040 \times 593,615 \times 5 = 1,412,922,423,000$ conditionals! Thus, we reduced almost one and a half trillion conditionals down to five.

Hereafter, we refer to the version of the code with conditionals moved outside the kernel as the *split* version of the code. The *unsplit* version uses conditionals in the computational loop so as to avoid code replication. While this may be good practice in terms of readability and maintainability of the code, it adversely impacts performance. Compiler techniques or source-to-source translation could be used to make this solution more maintainable, but such techniques are outside the scope of this paper. All CPU and GPU versions used in the results section have the conditionals moved outside the main computational loop, as described above.

2.3.3. Hierarchical charge partitioning

To further accelerate the computation and to test the performance of GPU-based computation with a multi-scale algorithm, we incorporated the HCP method described in the previous section. The incorporation of the HCP into the GPU-based computation introduced several new issues that had to be addressed. First, the multi-scale charge partitioning introduced additional branching when the HCP threshold distance was less than the diameter of the molecule. As a consequence, some amount of divergent branching could not be avoided in order to efficiently implementing HCP on the GPU. It also added several fields of data, and thus arguments, that did not exist in the non-HCP versions. Due to a limitation in the number of bytes passable as arguments (i.e., 1024 bytes) using the current version of Brook+ (i.e., 1.4), we moved the calculation of many of the arguments from the CPU to the GPU in order to save space in the argument list.

3. Results and discussion

Wall times from four different implementations of GEM, i.e., the computation of electrostatic surface potential, were collected: (1) GEM without HCP, running on a single CPU; (2) GEM with HCP, running on a single CPU; (3) GEM without HCP but running on the GPU, and (4) GEM with HCP but running on the GPU. Times were measured using time-checking calls within the main function in each implementation, which remained unchanged between versions. Only the computational kernel was measured, excluding the file I/O necessary to read and write the data files.

3.1. Experimental set-up

All results were measured on a system with an Intel Core 2 Duo E6550 processor, which contains two computing cores, 2 GB of DDR2-800 memory, and an ATI Radeon HD 4870 GPU on a PCI-E x16 interface running 32-bit Ubuntu Linux version 8.10. The GPU experiments were run with a single CPU process along with a kernel offloaded to the HD 4870. The CPU experiments were run in a single process on one core of the E6550 processor. All times reported in this section are averages of five runs. The execution times for the runs proved to be quite consistent, with variances in execution time at less than 1%. Our base CPU version of GEM represents the performance attainable by an experienced programmer in C, short of inserting inline assembly code. SSE acceleration is applied via compiler optimizations in GCC 4.24, and using arrays of primitives, as discussed earlier. Both of these CPU optimizations keep the baseline as similar to the GPU implementation as possible. The GCC compiler-applied SSE optimizations improved performance by approximately two-fold. All computations were performed with single-precision floating-point arithmetic, both on the CPU and GPU. A representative set of six atom-level molecular structures was used for testing. The structures were selected to span a large range of sizes. The six structures, their protein databank (PDB) IDs [52], and sizes are as follows:

(1) H helix of myoglobin, 1MBO, 382 atoms
(2) chain A of calcium binding protein S100B, 1UWO, 1,441 atoms,
(3) chain A of cytochrome CD1 nitrite reductase, 1QKS, 8,542 atoms,
(4) nucleosome core particle, 1KX5, 25,086 atoms,
(5) chaperonin GroEL, 2EU1, 109,802 atoms, and
(6) tobacco ringspot virus capsid, 1A6C, 476,040 atoms.

The atomic coordinates for these structures were obtained from the protein databank and atomic charges assigned using the H++ system (http://www.cs.vt.edu/biophysics/H++) [53], which uses the standard continuum solvent methodology [54] to compute protonation states of ionizable groups. The surface vertex points at which electrostatic potential is calculated for these structures were obtained using the program MSMS [55]. MSMS was run with a probe radius of 1.5 Å and a triangulation density of 3.0 vertices per Å^2 for all structures except the virus capsid, for which, due to limitations of the MSMS software a probe radius of 3.0 Å and a triangulation density of 1.0 vertices per Å^2 is used.

Table 1
Times, speed-ups, root mean square (RMS) error and relative RMS error of the surface potential calculations performed on the virus capsid structure, averaged over five runs of each version. Speed-up and error are calculated relative to the CPU version. Relative RMS error is calculated as RMS error divided by average absolute potential.

Version	Average time (s)	Speed-up over CPU	RMS error (kcal/mol/|e|)	Relative RMS error
CPU	80690.2	1.00	0	0
CPU with HCP	1442.2	41.68	0.1680	0.0153
GPU	219.2	182.80	0.0001	0.00001
GPU with HCP	35.0	933.59	0.1680	0.0153

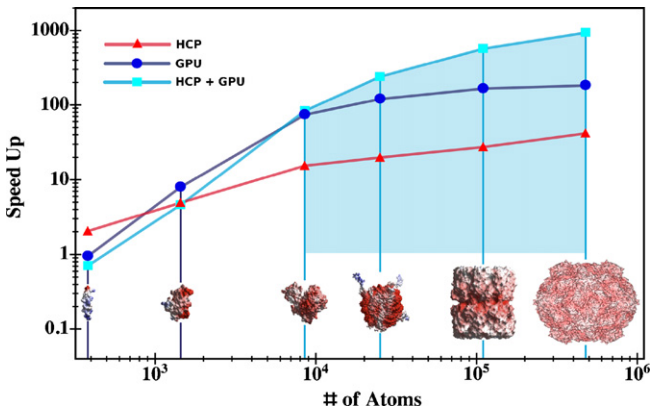

Fig. 3. Acceleration of the electrostatic potential computations resulting from various HCP/GPU combinations considered in this work. The red line with triangle data points shows the speed-up of the potential computation resulting from the use of the multi-scale approximation HCP alone. The dark blue line with circle data points corresponds to the same potential computed on the GPU alone, and the light blue line with square data points represents the combined speed-up of both the HCP and the GPU. The shaded region visually emphasizes the range of structure sizes where the combined speed-up is greater than either the HCP or GPU speed-up by itself. All speed-ups are measured relative to the CPU serial version.

3.2. Experimental results

We have three main versions of the code which merit comparison, see Fig. 3. Each version is compared with the reference non-HCP serial CPU version of the code. The use of the HCP approximation alone, the red line with triangle data points, speeds up the calculation from about 2× on the smallest structures to roughly 42× on the viral capsid, relative to the reference. The dark blue line with circle data points represents the Brook+ GPU version with all optimizations except the HCP applied, and is found to yield substantially higher speed-up than the HCP alone[2], especially for larger structures. The light blue line with square data points corresponds to the use of GPU in combination with the HCP (GPU-HCP). For small structures, the stand alone GPU version performs better than the GPU-HCP version. However, beyond structures of the order of 10^4 atoms, the GPU-HCP version is the clear winner in performance gains. For example, for the largest structure with 476,040 atoms, the speed-up using the GPU only, without HCP, is 182×, the speed-up without the GPU, using HCP only, is 42×, and the speed-up using both the GPU and HCP is 934× as can be seen in Table 1.

In the same table we present the RMS error as a measure of accuracy for each implementation. It is worth noting that the error introduced by switching from running single precision on the CPU to single precision on the GPU is almost immeasurably small. Since most computers are multi-core these days, including the test machine, it is also worth noting that the algorithm scales well across multiple cores, but since the potential combination of GPU acceleration and HCP is the goal of this paper we chose not to include direct results for multi-core CPUs. It is safe to say however that the speed-up is near but not exceeding direct speed-up, 1× per core, or roughly 2× on the test machine, and scaling to as many cores as there may be in a system.

The combined speed-up using both the GPU and HCP is not fully multiplicative due to the following two additional branches in the HCP algorithm. One, for the 1-charge 1-level HCP approximation implemented here, if the distance to a component is beyond a specified threshold distance the HCP algorithm treats the component as a single point-charge, otherwise all atoms within the component are used in the computation. Two, if the net charge of a component is zero, no further computations are performed for that component. However, on the GPU, all threads must execute the same instruction in each cycle. Therefore, even if a component in a given thread is beyond the threshold distance or has a zero net charge, and can complete its computations much faster, the thread can not proceed until other threads have completed.

The accuracy of the computations was also evaluated. For the single-point computation considered here the error introduced by the single precision GPU is much smaller than the error due to the HCP approximation alone [25].

For example, for the nucleosome the RMS error due to the use of single precision compared to double precision (both on the CPU version) was 0.002916 kcal/mol/|e| and and the error introduced by the GPU compared to the single precision CPU version was 0.000119 kcal/mol/|e|, whereas the error due to the HCP approximation alone was 0.277960 kcal/mol/|e|.

However, this small error could accumulate and grow in applications involving large numbers of cumulative computations, such as molecular dynamics simulations. Further analysis would be required to determine the affect of GPU and HCP errors on such applications.

4. Conclusions

Electrostatic potential can be an important indicator of biomolecular function and activity, thus the ability to compute and visualize potential can be a valuable tool for studying biomolecules. However, for large structures, the estimation of the surface potential can be computationally demanding, making such computations inaccessible to desktop PCs and even to large clusters. One can reduce these computational costs by using coarse-grain (multi-scale) approximations or by parallelization across multiple processors, however it is not in general obvious that the two techniques can be successfully combined. We demonstrate here that for the computation of electrostatic surface potential combining a multi-scale approximation with parallelization on the GPU can deliver significantly greater speed-up than either approach separately. The electrostatic surface potential is computed using the analytical linearized Poisson–Boltzmann (ALPB) model, which use a set of simple formulae within the implicit solvent framework, to compute potential. The hierarchical charge partitioning (HCP) method is used to speed-up the calculation by using a multi-scale

[2] In this work, we consider only the lowest level-1 HCP. Higher levels of "multi-scale" are achievable within the HCP, see Anandakrishnan and Onufriev [25] for details.

approximation for the potential, and further speed-up is achieved by executing the computation on an ATI Radeon 4870 GPU. We find that the errors introduced by the use of single precision GPU implementation are negligible for the purpose of computing single-point biomolecular potential.

We also show that, for large biomolecular structures, combining the power of the GPU with the HCP approximation can achieve a combined speed-up of up to 934× over the reference computation based on a single processor without the use of the HCP. This combined speed-up is larger than the individual speed-ups for structures larger than about 10,000 atoms. For the largest structure tested (half a million atoms) the combined speedup of 934× is many times larger than what was achieved by the GPU (182×) or HCP (42×) alone. However, for structures smaller than about 10,000 atoms the combined performance is less than that of the GPU-based computation alone because of unequal processing times across multiple threads such that the speed-up is constrained by the slowest thread. One particular challenge to achieving this speed-up was the presence of divergent branching due to the multi-scale threshold in HCP, which severely limited the performance gain. This optimization helped improve speed-up for large structures ($> 10,000$ atoms) but the associated overhead reduced performance for smaller structures.

Although this implementation produced impressive speed-ups, we do not believe it represents the full potential of the HCP-GPU approach. For example, only one level of HCP approximation was implemented here. We speculate that implementing additional levels of HCP, despite the additional branching involved, can conservatively result in an additional order of magnitude speed-up. The computation of electrostatic potential as implemented here, involves three steps — computing HCP group charges, determining charges to use in potential computation and the actual potential computation. Only the last of these three steps is executed on the GPU. Additional speed-up may be possible by also executing the first two steps on the GPU. These additional performance improvements will be explored in a future study.

The combined HCP–GPU implementation presented here raises the possibility of making various biomolecular modeling applications accessible to researchers using desktop computers, and in "real time". Examples of applications that can benefit immediately from the methods presented in this work include calculations of surface potential around large structures. Exploration of the changes in the computed potential due to changes in the environment and/or structures, such as changes in pH or mutations, can now be computed virtually immediately, thus greatly facilitating research.

In the longer term, we hope that the main conceptual result of this work – that acceleration of the computation of long-range interactions based on multi-scale ideas can be combined with GPU-based speed-ups – will impact a broad spectrum of applications where the speed of such computations is critical. For example, virtual screening for drug discovery involves screening hundreds of thousands to millions of potential candidate ligands, using "docking" simulations, to identify likely leads for further analysis [56]. Both the ligand and drug target can have multiple conformations and relative orientations resulting in hundreds to thousands of combined degrees of freedom [57]. It may be possible to apply the HCP–GPU approach to accelerate the virtual screening process. Similarly, the all-atom molecular dynamics simulation of even small to medium sized molecular systems ($< 100,000$ atoms), for biologically meaningful time scales (μs), requires the latest massively parallel supercomputers. It may be possible to make such simulations more accessible using the HCP–GPU approach presented here.

Acknowledgments

This work was supported in part by NIH R01 grant GM076121 and by NSF I/UCRC grant IIP-0804155 with support from AMD.

References

[1] M. Perutz, Electrostatic effects in proteins, Science 201 (1978) 1187–1191.
[2] J.D. Madura, M.E. Davis, M.K. Gilson, R.C. Wade, B.A. Luty, J. Andrew, McCammon, Biological applications of electrostatic calculations and Brownian dynamics, Rev. Comp. Chem. 5 (1994) 229–267.
[3] B.H.A. Nicholls, Classical electrostatics in biology and chemistry, Science 268 (1995) 1144–1149.
[4] M.E. Davis, J. Andrew McCammon, Electrostatics in biomolecular structure and dynamics, Chem. Rev. 90 (1990) 509–521.
[5] N.A. Baker, J.A. McCammon, Electrostaic Interactions In Structural Bioinformatics, John Wiley & Sons, Inc, New York, 2002.
[6] A. Warshel, J. Åqvist, Electrostatic energy and macromolecular function, Ann. Rev. Biophys. Biophys. Chem. 20 (1991) 267–298.
[7] A.R. Fersht, J.P. Shi, J. Knill-Jones, D.M. Lowe, A.J. Wilkinson, D.M. Blow, P. Brick, P. Carter, M.M. Waye, G. Winter, Hydrogen bonding and biological specificity analysed by protein engineering, Nature 314 (1985) 235–238.
[8] G. Szabo, G. Eisenman, S.G.A. McLaughlin, S. Krasne, Ionic probes of membrane structures, Ann. N. Y. Acad. Sci. 195 (1972) 273–290, Membrane Structure and Its Biological Applications.
[9] F.B. Sheinerman, R. Norel, B. Honig, Electrostatic aspects of protein-protein interactions, Curr. Opin. Struct. Biol 10 (2) (2000) 153–159.
[10] A. Onufriev, A. Smondyrev, D. Bashford, Proton affinity changes during unidirectional proton transport in the bacteriorhodopsin photocycle, J. Mol. Biol. 332 (2003) 1183–1193.
[11] A.-S. Yang, B. Honig, Electrostatic effects on protein stability, Curr. Opin. Struct. Biol. 2 (1992) 40–45.
[12] S. Whitten, B. Garcia-Moreno, pH dependence of stability of staphyococcal nuclease: evidence of substantial electrostatic interactions in denatured state, Biochemistry 39 (2000) 14292–14304.
[13] P. Koehl, Electrostatics calculations: latest methodological advances, Curr. Opin. Struct. Biol. 16 (6) (2006) 142–151, March.
[14] A. Robertson, E. Luttmann, V.S. Pande, Effects of long-range electrostatic forces on simulated protein folding kinetics, J. Comput. Chem. 29 (5) (2007) 694–700.
[15] D.A.C. Beck, R.S. Armen, V. Daggett, Cutoff size need not strongly influence molecular dynamics results for solvated polypeptides, Biochemistry 44 (2) (2005) 609–616, January.
[16] A.M. Ruvinsky, I.A. Vakser, Interaction cutoff effect on ruggedness of protein–protein energy landscape, Proteins: Struct., Funct. Bioinform. 70 (4) (2008) 1498–1505.
[17] J. Carrier, L. Greengard, V. Rokhlin, A fast adaptive multipole algorithm for particle simulations, SIAM J. Sci. Stat. Comput. 9 (4) (1988) 669–686.
[18] W. Cai, S. Deng, D. Jacobs, Extending the fast multipole method to charges inside or outside a dielectric sphere, J. Comput. Phys. 223 (May (2)) (2007) 846–864.
[19] C.G. Lambert, T.A. Darden, J.A. Board Jr., A multipole-based algorithm for efficient calculation of forces and potentials in macroscopic periodic assemblies of particles, J. Comput. Phys. 126 (July (2)) (1996) 274–285.
[20] T. Darden, D. York, L. Pedersen, Particle mesh Ewald: An N. log(N) method for Ewald sums in large systems, J. Chem. Phys. 98 (12) (1993) 10089–10092.
[21] U. Essmann, L. Perera, M.L. Berkowitz, T. Darden, H. Lee, L.G. Pedersen, A smooth particle mesh Ewald method, J. Chem. Phys. 103 (19) (1995) 8577–8593.
[22] A.Y. Toukmaji, J.A. Board, Ewald summation techniques in perspective: a survey, Comp. Phys. Commun. 95 (June (2–3)) (1996) 73–92.
[23] D. York, W. Yang, The fast fourier Poisson method for calculating Ewald sums, J. Chem. Phys. 101 (4) (1994) 3298–3300.
[24] T. Schlick, Molecular Modeling and Simulation, An Interdisciplinary Guide, Springer-Verlag, New York, 2002.
[25] R. Anandakrishnan, A.V. Onufriev, An $n \log n$ approximation based on the natural organization of biomolecules for speeding up the computation of long range interactions, J. Comput. Chem. 31 (4) (2010) 691–706.
[26] A. Onufriev, Implicit solvent models in molecular dynamics simulations, Annu. Rep. Comput. Chem. 4 (2008) 125–137.
[27] Los Alamos National Laboratory. mpiBLAST. http://mpiblast.lanl.gov/.
[28] R. Konecny, J. Trylska, F. Tama, D. Zhang, N.A. Baker, C.L. Brooks, J.A. McCammon, Electrostatic properties of cowpea chlorotic mottle virus and cucumber mosaic virus capsids, Biopolymers 82 (June (2)) (2006) 106–120.
[29] N.A. Baker, D. Sept, S. Joseph, M.J. Holst, J.A. McCammon, Electrostatics of nanosystems: application to microtubules and the ribosome, Proc. Natl. Acad. Sci. U.S.A. 98 (August (18)) (2001) 10037–10041.
[30] J.L. Klepeis, K. Lindorff-Larsen, R.O. Dror, D.E. Shaw, Long-timescale molecular dynamics simulations of protein structure and function, Curr. Opin. Struct. Biol. 19 (April (2)) (2009) 120–127.
[31] J.Z. Ruscio, D. Kumar, M. Shukla, M.G. Prisant, T.M. Murali, A.V. Onufriev, Atomic level computational identification of ligand migration pathways between solvent and binding site in myoglobin, Proc. Natl. Acad. Sci. U.S.A. 105 (27) (2008) 9204–9209.
[32] S. Kumar, C. Huang, G. Zheng, E. Bohm, A. Bhatele, J.C. Phillips, H. Yu, L.V. Kalé, Scalable molecular dynamics with NAMD on the IBM Blue Gene/L system, IBM J. Res. Dev. 52 (1/2) (2008) 177–187.

[33] D.E. Shaw, M.M. Deneroff, R.O. Dror, J.S. Kuskin, R.H. Larson, J.K. Salmon, C. Young, B. Batson, K.J. Bowers, J.C. Chao, M.P. Eastwood, J. Gagliardo, J.P. Grossman, R.C. Ho, D.J. Ierardi, I. Kolossváry, J.L. Klepeis, T. Layman, C. Mcleavey, M.A. Moraes, R. Mueller, E.C. Priest, Y. Shan, J. Spengler, M. Theobald, B. Towles, S.C. Wang, Anton a special-purpose machine for molecular dynamics simulation, Commun. ACM 51 (7) (2008) 91–97.
[34] R. Zhou, M. Eleftheriou, C.C. Hon, R.S. Germain, A.K. Royyuru, B.J. Berne, Massively parallel molecular dynamics simulations of lysozyme unfolding, IBM J. Res. Dev. 52 (1/2) (2008) 19.
[35] G. Moore, Cramming more components onto integrated circuits, Electron. Mag. (April) (1965).
[36] Swaminarayan. Sriram, Kadau. Kai, C.Germann. Timothy, C. Gordon, Fossum 369 tflop/s molecular dynamics simulations on the roadrunner general-purpose heterogeneous supercomputer, in: SC'08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, IEEE Press, Piscataway, NJ, USA, 2008, pp. 1–10.
[37] J. Archuleta, Y. Cao, W. Feng, T. Scogland, Multi-dimensional characterization of temporal data mining on graphics processors, in: 23rd IEEE International Parallel and Distributed Processing Symposium, May, 2009.
[38] D. Dynerman, E. Butzlaff, J.C. Mitchell, CUSA and CUDE: GPU-accelerated methods for estimating solvent accessible surface area and desolvation, J. Comput. Biol. 16 (4) (2009) 523–537.
[39] T. Narumi, K. Yasuoka, M. Taiji, S. Höfinger, Current performance gains from utilizing the GPU or the ASIC MDGRAPE-3 within an enhanced Poisson Boltzmann approach, J. Comput. Chem. 30 (14) (2009) 2351–2357.
[40] I.S. Ufimtsev, T.J. Martinez, Quantum chemistry on graphical processing units. 1. Strategies for two-electron integral evaluation, J. Chem. Theory Comput. 4 (February (2)) (2008) 222–231.
[41] J.A. Anderson, C.D. Lorenz, A. Travesset, General purpose molecular dynamics simulations fully implemented on graphics processing units, J. Comput. Phys. 227 (10) (2008) 5342–5359.
[42] M.S. Friedrichs, P. Eastman, V. Vaidyanathan, M. Houston, S. Legrand, A.L. Beberg, D.L. Ensign, C.M. Bruns, V.S. Pande, Accelerating molecular dynamic simulation on graphics processing units, J. Comput. Chem. 30 (6) (2009) 864–872.
[43] D.J. Hardy, J.E. Stone, K. Schulten, Multilevel summation of electrostatic potentials using graphics processing units, Parallel Comput. 35 (3) (2009) 164–177.
[44] J.E. Stone, J.C. Phillips, P.L. Freddolino, D.J. Hardy, L.G. Trabuco, K. Schulten, Accelerating molecular modeling applications with graphics processors, J. Comput. Chem. 28 (16) (2007) 2618–2640.
[45] J.C. Gordon, A.T. Fenley, A. Onufriev, An analytical approach to computing biomolecular electrostatic potential. II. Validation and applications, J. Chem. Phys. 129 (7) (2008) 075102.
[46] G. Sigalov, A. Fenley, A. Onufriev, Analytical linearized Poisson-Boltzmann approach: Beyond the generalized Born approximation, J. Chem. Phys. 124 (2006) 124902.
[47] A.T. Fenley, J.C. Gordon, A. Onufriev, An analytical approach to computing biomolecular electrostatic potential. I. Derivation and analysis, J. Chem. Phys. 129 (7) (2008) 075101.
[48] G. Sigalov, P. Scheffel, A. Onufriev, Incorporating variable dielectric environments into the generalized born model, J. Chem. Phys. 122 (Mar (9)) (2005) 094511–194511.
[49] AMD/ATI Brook+. http://sourceforge.net/projects/brookplus/. 2000.
[50] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, P. Hanrahan, Brook for GPUs: stream computing on graphics hardware, ACM Trans. Graph. 23 (3) (2004) 777–786.
[51] Advanced Micro Devices ATI Stream Computing User Guide, March 2009.
[52] H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, P.E. Bourne, The protein data bank, Nucleic Acids Res. 28 (2000) 235–242.
[53] J.C. Gordon, J.B. Myers, T. Folta, V. Shoja, L.S. Heath, A.L. Onufriev, H++: a server for estimating pKa's and adding missing hydrogens to macromolecules, Nucleic Acids Res. 33 (2005) 68–71.
[54] D. Bashford, K. Gerwert, Electrostatic calculations of the pKa values of ionizable groups in bacteriorhodopsin, J. Mol. Biol. 224 (1992) 473–486.
[55] M.F. Sanner, A.J. Olson, J. Claude, Spehner, Reduced surface: an efficient way to compute molecular surfaces, Biopolymers 38 (3) (1996) 305–320.
[56] R. Abagyan, High-throughput docking for lead generation, Curr. Opin. Chem. Biol. 5 (August (4)) (2001) 375–382.
[57] M.L. Teodoro, G.N. Phillips, L.E. Kavraki Jr., Molecular docking: a problem with thousands of degrees of freedom, in: In IEEE International Conference on Robotics and Automation, 2001, pp. 960–966.