Accelerating Electrostatic Surface Potential Calculation with Multiscale Approximation on Graphics Processing Units

Ramu Anandakrishnan
Department of Computer Science, Virginia Tech
2050 Torgersen Hall (0106), Blacksburg, VA 24061 USA
ramu@cs.vt.edu

Tom Scogland

Andrew T. Fenley

John Gordon

Wuchun Feng[*]
Department of Computer Science, Virginia Tech

Alexey Onufriev[†]
Departments of Computer Science and Physics, Virginia Tech
2050 Torgersen Hall (0106), Blacksburg, VA 24061
alexey@cs.vt.edu

June 6, 2009

[*]corresponding author
[†]corresponding author

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. This paper demonstrates how one can take advantage of graphic processing units (GPUs) available in today's typical desktop computer, together with a multiscale approximation method, to significantly speedup such computations. Specifically, the electrostatic potential computation, using an analytical linearized Poisson Boltzmann (ALPB) method, is implemented on an ATI Radeon 4870 GPU in combination with the hierarchical charge partitioning (HCP) multiscale approximation. This implementation delivers a combined 1800-fold speedup for a 476,040 atom viral capsid.

Keywords: GPU; Biomolecular Electrostatics; Multiscale Approximation.

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 simulations.[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 method (PME).[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 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. Besides, access to such resources may be limited.

Within either of the above general categories, estimating the long-range nature of electrostatic interactions is still computationally intensive. In the past, in addition to algorithmic advances, scientists could also rely on Moore's Law[28] to significantly accelerate these computations.[*] 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 forty years ago, is a revolution that is now upon us.

With the advent of multi-core chips – from the traditional AMD and Intel multi-core to the more exotic hybrid multi-core of IBM Cell and many-core of AMD/ATi and NVIDIA graphics processing units (GPUs) — parallel computing across multiple cores on a single chip has become a necessity. However, parallel computing on a large-scale supercomputer is a challenging endeavor from the perspectives of ease of access, ease of programming, and cost, e.g., the fastest supercomputer in the world, Roadrunner at Los Alamos National Laboratory, cost $133M to build.[29] A significantly more cost-effective solution is general-purpose computation on graphics processing units (GPGPU), also known as video cards. With the peak floating-point performance of a GPU now exceeding a teraflop (tera floating-point op-erations 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 1,457 teraflops at a cost of $133M for a mere performance-price ratio of 11 megaflops per dollar and performance-space ratio of 243 teraflops

[*]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.

per square foot.

A number of different biomolecular modeling applications have recently been implemented on GPGPUs,[30–32] including the computation of long range electrostatic potential in the context of molecular dynamics.[33–36] This implementation focuses on the computation of electrostatic surface potential using a realistic solvation model, to demonstrate that the GPGPU can be combined with approximation schemes, such as the hierarchical charge partitioning (HCP) algorithm[25] to achieve significantly greater multiplicative speedup. The paper seeks to algorithmically map and transform electrostatic potential algorithms onto the GPU to deliver nearly a 2000-fold speed-up of a specific electrostatic potential code known as GEM. In the process of transforming the code, we uncover several general optimizations for GPU computing.

One can gain significant speed-up by using parallel computing with the exact all-atom computation for long range interactions, i.e. without the approximations described above. However, speed-ups achievable by parallel computing alone, even with today's massively parallel supercomputers, falls far short of what is needed for realistic biomolecular simulations. Even with parallel computing, exact all-atom simulations of small biomolecular systems (tens of thousands of atoms) is limited to nanosecond timescales.[37] Therefore parallel computing needs to be combined with approximations of the physical model to achieve biologically meaningful simulation timescales of microseconds and longer.[38–40] However, the current programming model for GPUs is only amenable to data-parallel applications; efficient GPU mappings for non data-parallel applications are extraordinarily difficult to realize.[41] Unlike supercomputer clusters consisting of general purpose processors, 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.

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

2 Methods

This section describes the specific methods used to compute long-range interactions, and an implementation of the entire computational process on the GPGPU. 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 the use GPGPUs.

2.1 Computation of Biomolecular Electrostatic Potential

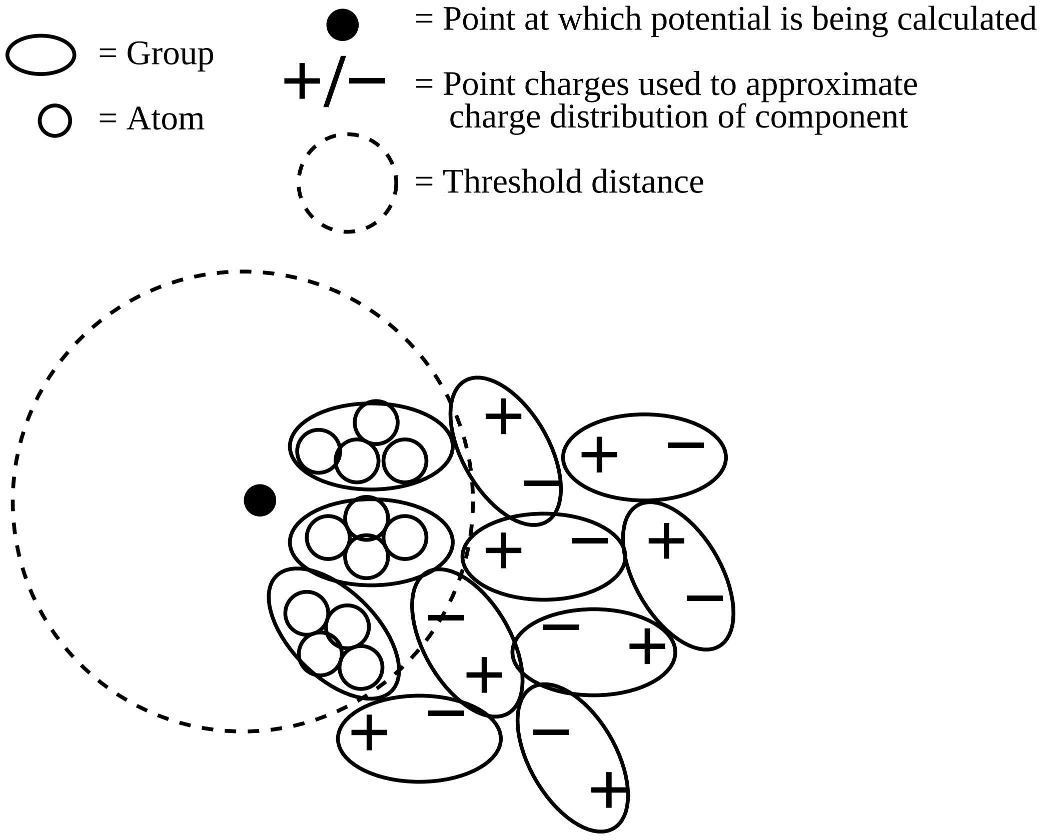
All of the electrostatic calculations presented in this paper were done using the analytic, linearized Poisson-Boltzmann (ALPB) model.[42] The ALPB model is based on a closed form, analytical approximation to the Poisson equation for an arbitrary distribution of point charges and a spherical dielectric boundary. The analytical approximation was extended to include the effects of mobile ions at the Debye-Hückel level along with a strategy for handling realistic biomolecular shapes. The ALPB model analytically defines physically admissible electrostatic potential everywhere in space. This allows for the calculation of the potential at any point in space without relying on the knowledge of 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 and its applicability to computing biomolecular electrostatic potential is presented elswhere.[43]

To assess the speedup resulting from the combination of the HCP and the GPGPU we selected the GEM software package.[43] 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 approximation

The hierarchical charge partitioning (HCP) approximation exploits the natural partitioning of biomolecules into constituent components 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 figure 1, 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 are 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 a component, the fewer charges are used to represent the component. The actual speedup 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 Ref.[25] For the purpose of this analysis we use the simplest 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, for the purpose of this analysis we use the simplest 1 level of partitioning for



= Point at which potential is being calculated
= Group
= Point charges used to approximate charge distribution of component
= Atom
= Threshold distance
+
−

Figure 1: Illustration of the Hierarchical Charge Partitioning (HCP) approximation. In this illustration a biomolecular structure is partitioned into its constituent groups, each of which consist of multiple atoms. The charge distribution of each group is approximated by a small set of point charges: two (+ and -) in this illustration. The threshold distance determines when to use the approximate charge distribution in electrostatic computations, instead of individual atomic charges.

the HCP algorithms. Increasing the number of levels used by the HCP algorithm would reduce accuracy and computational cost.

2.3 GPGPU implementation

Our initial exploration of mapping GEM to the GPU was conducted through the high level abstraction of AMD's Stream SDK, also known as Brook+,[44] on a machine with a Radeon 4870 GPU running Ubuntu 8.10 Linux. Brook+ is based on the brook stream computing language from Stanford,[45] and specifically optimized to compile and run on AMD stream capable GPUs through their CAL and CTM layers. It has the distinct advantage among the three available layers of being the first to appear equivalent to a higher level language, where the other two are a domain specific syntax highly influenced by assembler.

The calculation of electrostatic potential via an analytical approach implemented in GEM code described above has the very useful property of being embarrassingly parallel across all the points of reference, or vertices at which the electrostatic potential is calculated. In addition to this, each vertex can be computed as a reduction, a sum to be specific, of a set of completely independent calculations on each atom allowing for multiple dimensions of parallelism. For the initial implementation, we decided to treat each vertex as an individual unit of execution. All of the vertices are computed simultaneously until a point is reached where they no longer fit on the card simultaneously. When this pointed is reached, multiple kernels are launched with a subset of the total vertices producing comparable parallelism.

This alone is too simple to achieve the true potential of the GPU, or at least it still retains some features of the original code which do not translate favorably into GPU performance.

The first such issue we came across was a necessity to alter our data arrangement to move the data to the card. All the arrays of structures had to be flattened into arrays of primitives in the version of Brook+ in use. Even so, the change was an optimization in that it allows the fetching of data on the card to be performed far more efficiently than would otherwise be possible without significant code changes in the kernel. We intend to investigate this further as the support for structures in Brook+ improves sufficiently for us to implement the code with the original structure of the data intact. What we can test of it now gives the impression that this is an important optimization for this architecture.

Aside from the memory we also found a major procedural bottleneck in the form of a large number of conditionals deciding what equations to use and how to behave. Conditionals are very common in all code, we rely on them for nearly everything, and it has been made clear in previous literature on the subject of programming GPUs that divergent branches are expensive. Divergent branches being conditionals which cause some threads executing on the GPU in a set to take different paths than others, causing the SIMD units on the card to spend extra cycles. Even so, most literature ignores the issue of non-divergent branching, which we argue is nearly as large a problem as divergent branching is if the non-divergent branch is hit a sufficient number of times. We decided to find a way to remove as many of these as possible, we refer to the process as kernel splitting. In GEM's computational core, the majority of conditionals are to decide between alternatives which are predetermined at the time of the function invocation, and or constant across all items within an input set. This allowed us to create several different versions of the Brook+ kernel, 15 total, which are selected by a set of conditionals run on the CPU, each of which now contains only the conditionals which must be evaluated on every input item individually, cutting the total number of branches in most kernels to 0, and at the most three.

3 Results and Discussion

The HCP implementation on the GPGPU is used to compute electrostatic potential for a representative set of seven biomolecular structures ranging in size from 382 to 476,040 atoms. We have four main versions of the code which merit comparison, see figure 2. The dashed line is the reference non-HCP serial CPU version of the code. The use of the HCP approximation, the red line, speeds up the calculation from about 2x on the smallest structures to roughly 55x on the viral capsid, relative to the reference. The dark blue line represents the brook+ GPU version with all

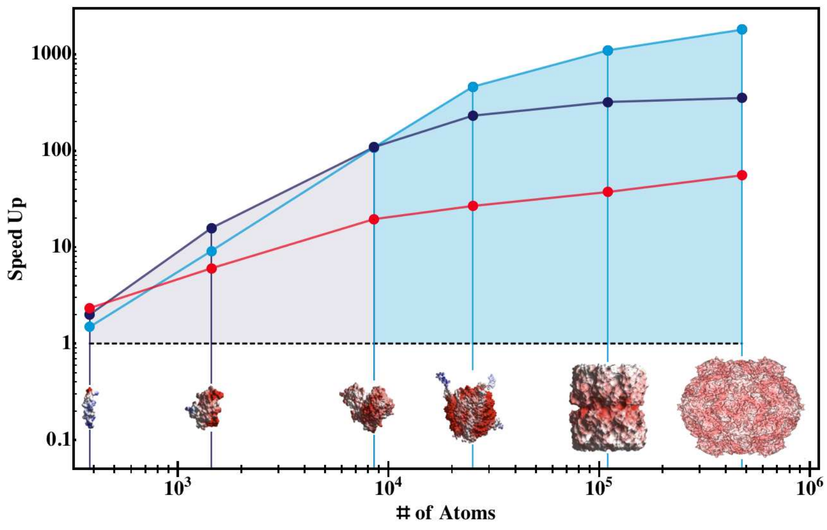


Figure 2: Acceleration of the electrostatic potential computations resulting from various HCP/GPU combinations considered in this work. The black dashed line represents the single CPU implementation of the ALPB model without any additional acceleration, which is used as a reference to compute the relative speed-ups of other methods. The red line shows the relative speed-up of the potential computation resulting from the use of the multi-scale approximation HCP alone. The dark blue line corresponds to the same potential computed on the GPU alone, and the light blue line 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.

optimizations except the HCP applied, and is found to yield substantially higher speed-up than the HCP alone[†], especially for larger structures. The light blue line corresponds to a 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. The interesting point here for GPU developers is that the HCP version on the GPU introduces divergent branches, a procedure which usually causes a significant reduction in performance, but in this case at sufficient sizes it actually improves performance to introduce divergence.

The accuracy of the computations were also evaluated. The errors introduced by the HCP computation are consistent with results from a previous study using a much larger number (600)

[†]in this work we consider only the lowest level-1 HCPḢigher levels of ”multiscale” are achievable within the HCP see Ref. 25 for details.

of biomolecular structures where the relative RMS force error for the 1-charge HCPwas 0.024. The relative RMS error introduced by the GPGPU is 10^{-5} with a relative maximum RMS error of 6×10^{-5}. For the single point computation considered here the error introduced by the single precision GPGPU is much smaller than the error due to the HCP approximation. 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 GPGPU errors on such applications.

4 Conclusions

Electrostatic surface potential can be an important indicator of biomolecular function and activity, thus the ability to compute and visualize surface potential can be a valuable tool for studying biomolecules. However, for large structures, the computation of the surface potential can be computationally demanding, making such computations inaccessible to desktop computing and even to large clusters. We present here a method for accelerating the computation of electrostatic surface potential through a combination of a multiscale approximation and the use of a GPU on a desktop computer. The electrostatic surface potential is computed using the analytical linearized Poisson Boltzmann (ALPB) model which computes potential using an implicit solvent representation. The hierarchical charge partitioning (HCP) method is used to speed-up the calculation by using a multi-scale 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 GPGPU with the HCP approximation can achieve a combined speedup of up to 1800x over the reference computation based on a single processor without the use of the HCP. The combined speedup is larger than the individual speed-ups for structures larger than about 10,000 atoms, and for the largest structures tested is many times larger than what could have been achieved by the GPU (350x) or the HCP (55x) 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 speedup is constrained by the slowest thread. One particular challenge in achieving this speedup was the presence of divergent branching due to the HCP which severely limited the performance gain. Kernel splitting, where each branch is processed by a separate kernel, was used to partially overcome this limitation. This optimization helped improve speedup for large structures (> 10000 atoms) but the associated overhead reduced performance for smaller structures.

Although this implementation produced impressive speedups, we do not believe this represents the full potential of this approach. For example, only one level of HCP approximation was implemented here. We speculate that implementing additional levels of HCP can result in an additional order of magnitude speedup. The computation of electrostatic potential as implemented here, involves four steps - loading data, computing HCP group charges, determining charges to use in potential computation and the actual potential computation. Only the last of these four steps is executed on the GPU. Additional speedup may be possible by executing the second and third steps on the GPU. These additional performance improvements will be explored in a future study,

References

[1] Perutz, M. *Science* **201**, 1187–1191 (1978).

[2] Madura, J. D., Davis, M. E., Gilson, M. K., Wade, R. C., Luty, B. A., and McCammon, J. A. *Rev. Comp. Chem.* **5**, 229–267 (1994).

[3] Honig, B. and Nicholls, A. *Science* **268**, 1144–1149 (1995).

[4] Davis, M. E. and McCammon, J. A. *Chem. Rev.* **90**, 509–521 (1990).

[5] Baker, N. A. and McCammon, J. A. *Electrostaic Interactions. In Structural Bioinformatics*. John Wiley & Sons, Inc., New York, (2002).

[6] Warshel, A. and Åqvist, J. *Ann. Rev. Biophys. Biophys. Chem.* **20**, 267–298 (1991).

[7] Fersht, A., Shi, J., Knill-Jones, J., Lowe, D., Wilkinson, A., Blow, D., Brick, P., Carter, P., Waye, M., and Winter, G. *Nature.* **314**, 235–8 (1985).

[8] Szabo, G., Eisenman, G., McLaughlin, S., and Krasne, S. *Ann. N.Y. Acad. Sci.* **195**, 273–290 (1972).

[9] Sheinerman, F. B., Norel, R., and Honig, B. *Curr. Opin. Struct. Biol* **10**(2), 153–9 (2000).

[10] Onufriev, A., Smondyrev, A., and Bashford, D. *J. Mol. Biol.* **332**, 1183–1193 (2003).

[11] Yang, A.-S. and Honig, B. *Curr. Opin. Struct. Biol.* **2**, 40–45 (1992).

[12] Whitten, S. and Garcia-Moreno, B. *Biochemistry* **39**, 14292–14304 (2000).

[13] Koehl, P. *Curr Opin Struct Biol* **16**(6), 142–151 March (2006).

[14] Robertson, A., Luttmann, E., and Pande, V. S. *Journal of Computational Chemistry* **29**(5), 694–700 (2007).

[15] Beck, D. A. C., Armen, R. S., and Daggett, V. *Biochemistry* **44**(2), 609–616 January (2005).

[16] Ruvinsky, A. M. and Vakser, I. A. *Proteins: Structure, Function, and Bioinformatics* **70**(4), 1498–1505 (2008).

[17] Carrier, J., Greengard, L., and Rokhlin, V. *SIAM Journal on Scientific and Statistical Computing* **9**(4), 669–686 (1988).

[18] Cai, W., Deng, S., and Jacobs, D. *Journal of Computational Physics* **223**(2), 846–864 May (2007).

[19] Lambert, C. G., Darden, T. A., and Board Jr., J. A. *Journal of Computational Physics* **126**(2), 274–285 July (1996).

[20] Darden, T., York, D., and Pedersen, L. *The Journal of Chemical Physics* **98**(12), 10089–10092 (1993).

[21] Essmann, U., Perera, L., Berkowitz, M. L., Darden, T., Lee, H., and Pedersen, L. G. *The Journal of Chemical Physics* **103**(19), 8577–8593 (1995).

[22] Toukmaji, A. Y. and Board, J. A. *Computer Physics Communications* **95**(2-3), 73–92 June (1996).

[23] York, D. and Yang, W. *The Journal of Chemical Physics* **101**(4), 3298–3300 (1994).

[24] Schlick, T. *Molecular modeling and simulation, an interdisciplinary guide*. Springer-Verlag, New York, (2002).

[25] Anandakrishnan, R. and Onufriev, A. Journal of Computational Chemistry, in press, (2009).

[26] Onufriev, A. *Annual Reports in Computational Chemistry* **4**, 125–137 (2008).

[27] Los Alamos National Laboratory. http://mpiblast.lanl.gov/.

[28] G. Moore. *Electronics Magazine* April (1965).

[29] Swaminarayan, S., Kadau, K., Germann, T. C., and Fossum, G. C. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, 1–10 (IEEE Press, Piscataway, NJ, USA, 2008).

[30] Dynerman, D., Butzlaff, E., and Mitchell, J. C. *Journal of Computational Biology* **16**(4), 523–537 (2009).

[31] Narumi, T., Yasuoka, K., Taiji, M., and Höfinger, S. *Journal of Computational Chemistry* (2009).

[32] Ufimtsev, I. S. and Martinez, T. J. *Journal of Chemical Theory and Computation* **4**(2), 222–231 February (2008).

[33] Anderson, J. A., Lorenz, C. D., and Travesset, A. *Journal of Computational Physics* **227**(10), 5342 – 5359 (2008).

[34] Friedrichs, M. S., Eastman, P., Vaidyanathan, V., Houston, M., Legrand, S., Beberg, A. L., Ensign, D. L., Bruns, C. M., and Pande, V. S. *Journal of Computational Chemistry* **30**(6), 864–872 (2009).

[35] Hardy, D. J., Stone, J. E., and Schulten, K. *Parallel Computing* **35**(3), 164 – 177 (2009).

[36] Stone, J. E., Phillips, J. C., Freddolino, P. L., Hardy, D. J., Trabuco, L. G., and Schulten, K. *Journal of Computational Chemistry* **28**(16), 2618–2640 (2007).

[37] Ruscio, J. Z., Kumar, D., Shukla, M., Prisant, M. G., Murali, T. M., and Onufriev, A. V. *Proc. Natl. Acad. Sci. USA* **105**(27), 9204–9209 (2008).

[38] Kumar, S., Huang, C., Zheng, G., Bohm, E., Bhatele, A., Phillips, J. C., Yu, H., and Kalé, L. V. *IBM Journal of Research and Development* **52**(1/2), 177–187 (2008).

[39] Shaw, D. E., Deneroff, M. M., Dror, R. O., Kuskin, J. S., Larson, R. H., Salmon, J. K., Young, C., Batson, B., Bowers, K. J., Chao, J. C., Eastwood, M. P., Gagliardo, J., Grossman, J. P., Ho, R. C., Ierardi, D. J., Kolossváry, I., Klepeis, J. L., Layman, T., Mcleavey, C., Moraes, M. A., Mueller, R., Priest, E. C., Shan, Y., Spengler, J., Theobald, M., Towles, B., and Wang, S. C. *Communications of the ACM* **51**(7), 91–97 (2008).

[40] Zhou, R., Eleftheriou, M., Hon, C. C., Germain, R. S., Royyuru, A. K., and Berne, B. J. *IBM Journal of Research and Development* **52**(1/2), 19 (2008).

[41] Archuleta, J., and Cao, Y., and Feng, W., and Scogland, T. In *23rd IEEE International Parallel and Distributed Processing Symposium*, May (2009).

[42] Fenley, A. T., Gordon, J. C., and Onufriev, A. *The Journal of Chemical Physics* **129**(7), 075101 (2008).

[43] Gordon, J. C., Fenley, A. T., and Onufriev, A. *The Journal of Chemical Physics* **129**(7), 075102 (2008).

[44] AMD/ATI. http://sourceforge.net/projects/brookplus/.

[45] Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M., and Hanrahan, P. *ACM Trans. Graph.* **23**(3), 777–786 (2004).