HHS Public Access

Author manuscript

J Chem Theory Comput. Author manuscript; available in PMC 2017 December 13.

Published in final edited form as:

J Chem Theory Comput. 2017 July 11; 13(7): 3378–3387. doi:10.1021/acs.jctc.7b00336.

Acceleration of Linear Finite-Difference Poisson-Boltzmann Methods on Graphics Processing Units

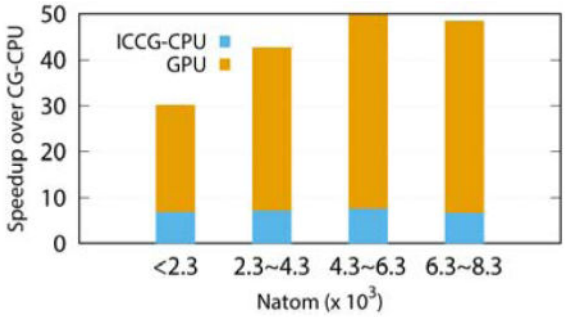
Ruxi Qi, **Wesley M. Botello-Smith**, and **Ray Luo***

Department of Molecular Biology and Biochemistry, University of California, Irvine, CA 92697-3900

Abstract

Electrostatic interactions play crucial roles in biophysical processes such as protein folding and molecular recognition. Poisson-Boltzmann equation (PBE)-based models have emerged as widely used in modeling these important processes. Though great efforts have been put into developing efficient PBE numerical models, challenges still remain due to the high dimensionality of typical biomolecular systems. In this study, we implemented and analyzed commonly used linear PBE solvers for the ever-improving graphics processing units (GPU) for biomolecular simulations, including both standard and preconditioned conjugate gradient (CG) solvers with several alternative preconditioners. Our implementation utilizes standard Nvidia® CUDA libraries cuSPARSE, cuBLAS, and CUSP. Extensive tests show that good numerical accuracy can be achieved given that the single precision is often used for numerical applications on GPU platforms. The optimal GPU performance was observed with the Jacobi-preconditioned CG solver, with a significant speedup over standard CG solver on CPU in our diversified test cases. Our analysis further shows that different matrix storage formats also considerably affect the efficiency of different linear PBE solvers on GPU, with the diagonal format best suited for our standard finite-difference linear systems. Further efficiency may be possible with matrix-free operations and integrated grid stencil setup specifically tailored for the banded matrices in PBE-specific linear systems.

Graphical Abstract



*Please send correspondence to R. Luo. ray.luo@uci.edu; fax: (949) 824-9551.

1. Introduction

In recent years Poisson-Boltzmann equation (PBE)-based electrostatics modeling has gained wide acceptance in biomolecular applications, given the crucial roles played by the electrostatic interactions in biophysical processes such as protein-protein and protein-ligand interactions.[1] Due to the high dimensionalities of typical biomolecular systems, it is extremely important to increase the accuracy and efficiency of PBE models.[2]

For biomolecular applications, the PBE is impossible to be solved analytically, so that only numerical solutions are possible. Traditional numerical schemes include the finite difference method (FDM)[3] where difference grids are used to discretize the space and build up a set of linear/nonlinear equations from the PBE, and the finite-element method[4] where arbitrarily shaped biomolecules are discretized by using elements with a set of associated basis functions. The boundary element method is another alternative approach, in which only the surfaces of the molecules are discretized.[5] Numerical PBE methods have been applied to the prediction of pKa values for ionizable groups in biomolecules,[6] solvation free energies,[7] binding free energies,[8] and protein folding and design.[9]

Among these approaches, the FDM is most widely adopted and has been incorporated in programs such as DelPhi,[3a, 3c, 3j] UHBD,[3b, 3d] APBS,[3e, 3g] CHARMM/PBEQ,[3c, 3i] and Amber/PBSA.[2h, 3l−n, 10] The resulting algebraic systems are often solved by using conjugate gradient methods with or without preconditioners.[3b, 3k, 11] As computational studies shift to larger and more complex biomolecular systems, both the data storage and convergence rate become more challenging to address on traditional CPU platforms. These challenges are more pronounced when incorporating the PBE in typical molecular simulations involving thousands to millions of snapshots.

Recently, graphics processing units (GPU) have been used in a wide range of computational chemistry problems, including MD simulations[12] and ab initio quantum mechanical (QM) calculations[13] with impressive speedup. Different from CPUs that are designed for sequential execution, GPUs have a parallel architecture that is suited for high-performance computation with dense data parallelism, and have enjoyed rapid adoption over the last decade. A number of publications have also shown the use of GPUs to accelerate PBE linear systems for biomolecular systems and reported impressive speedup.[14] However, different from MD or QM simulations, various PBE solvers perform with markedly different efficiency.[3l, 3m] Simpler algorithms may be straightforward to be ported onto GPU platforms, but they may not be robust or efficient enough to begin with (i.e. they may be very slow to converge or need very high number of floating operation counts to achieve a given convergence criterion), particularly on very complex or large biomolecular systems. Therefore, a thorough analysis of existing algorithms on GPUs is a necessary step to realize markedly improved overall efficiency in numerical PBE solutions for biomolecular applications.

To date only the relatively simple successive over-relaxation (SOR) method was implemented on GPUs.[14] However, our prior algorithm analysis of SOR and other algorithms have shown its convergence rate is not among the best on CPU for large systems

or tight convergence criterion even if it is a simple algorithm to implement.[3l, 3m] Furthermore, there are two additional disadvantages when porting the SOR method to GPUs. Firstly, a parallel SOR, such as red-black SOR, has to be used to utilize the parallel GPUs. However the red-black SOR has worse convergence rate than the original SOR due to its altered updating approach. Secondly, for most consumer-grade GPU cards, single precision operations are widely supported with high efficiency. Double precision operations are possible, but are at a significant performance disadvantage. Unfortunately use of single precision further deteriorates the convergence of red-black SOR whether it is on GPUs or on CPUs as our in-house testing has shown.

In this paper, we present the implementation and systemic assessment of four types of linear PBE solvers on GPUs using the Nvidia CUDA (Version 7.5) libraries. In the following the underlining linear systems solvers are first reviewed. This is followed by an assessment of the accuracy and efficiency observed for different implementations. The impact of matrix storage formats upon the computation efficiency is then discussed. Finally the memory usage on the GPUs is briefly addressed.

2. Methods

2.1 Poisson-Boltzmann Equation

In implicit solvent models, the solvent is treated as high dielectric continuum and the solute is approximated as low dielectric continuum with charges embedded inside. The PBE is then introduced to describe the electrostatic interactions in the heterogeneous dielectric environment, with the Boltzmann term describing the salt effect of a dissolved electrolyte. This gives the well-known non-linear PBE

$$\nabla \cdot \varepsilon(\mathbf{r})\nabla\phi(\mathbf{r}) + \lambda(\mathbf{r})\sum_i n_i q_i \exp\left[-q_i\phi(\mathbf{r})/kT\right] = -\rho(\mathbf{r}), \quad (1)$$

where ρ is the charge density, ϕ is the electrostatic potential, ε is the dielectric constant, and λ is a masking function for the Stern layer. All variables are functions of the spatial vector $\mathbf{r}$. In the salt related term, n_i is the number density of ion of type i in the bulk solution, q_i is the charge of the ion of type i, k is the Boltzmann constant and T is the temperature. When the term $q_i\phi(\mathbf{r}) / kT$ is small, the PBE can be linearized into

$$\nabla \cdot \varepsilon(\mathbf{r})\nabla\phi(\mathbf{r}) - \lambda(\mathbf{r})\sum_i n_i q_i^2 \phi(\mathbf{r})/kT = -\rho(\mathbf{r}) \quad (2)$$

For biomolecules of arbitrary shape, the solution of equation (1) or (2) can only be obtained numerically, typically through finite-difference procedures. In this scheme, the PBE is discretized as follows

$$
\begin{aligned}
&-h^{-2}\varepsilon_i(i-1,j,k)[\phi(i-1,j,k)-\phi(i,j,k)]\\
&-h^{-2}\varepsilon_i(i,j,k)\quad[\phi(i+1,j,k)-\phi(i,j,k)]\\
&-h^{-2}\varepsilon_j(i,j-1,k)[\phi(i,j-1,k)-\phi(i,j,k)]\\
&-h^{-2}\varepsilon_j(i,j,k)\quad[\phi(i,j+1,k)-\phi(i,j,k)]\\
&-h^{-2}\varepsilon_k(i,j,k-1)[\phi(i,j,k-1)-\phi(i,j,k)]\\
&-h^{-2}\varepsilon_k(i,j,k)\quad[\phi(i,j,k+1)-\phi(i,j,k)]\\
&+\quad\kappa^2\quad\phi(i,j,k)\quad=h^{-3}q(i,j,k),
\end{aligned}
\tag{3}
$$

where h is the grid spacing in each dimension, i, j, and k are the grid indexes along x, y and z axes, respectively. $\varepsilon_i(i, j, k)$ is the dielectric constant between grid points (i, j, k) and (i+1, j, k). $\varepsilon_j(i, j, k)$ and $\varepsilon_k(i, j, k)$ are defined similarly. All the related coefficients in Boltzmann term are absorbed into κ^2, and $q(i, j, k)$ is the charge within the cubic volume centered at (i, j, k). The linear system can be conveniently written as

$$\mathbf{A}\phi=b, \tag{4}$$

where **A** is the coefficient matrix of dielectric constants and the Boltzmann term, and b is the constant vector of charges on the grids.

To solve equation (4), various solvers have been developed for biomolecular applications, such as successive over-relaxation (SOR),[15] conjugate gradient (CG),[15] (modified) incomplete Cholesky conjugate gradient ((M)ICCG),[11] geometric multigrid (GMG),[16] and algebraic multigrid (AMG).[17] All solvers proceed from an initial guess of $\phi(i, j, k)$ to generate a sequence of improving solutions iteratively.

2.2 Conjugate Gradient Solvers

Symmetric and positive-definite linear systems are often solved with the CG solvers. The CG method searches for the exact solution along a series of conjugate directions, and is implemented as an iterative procedure as follows:

1. set $l = 0$, $p_0 = r_0$
2. compute the norm of $||r_l||$. If $||r_l||/||b|| < \delta$, output ϕ_l. Otherwise go to the next step.
3. compute

$$\alpha_l=\frac{r_l^T p_l}{p_l^T \mathbf{A} p_l},\qquad \phi_{l+1}=\phi_l+\alpha_l p_l$$

4. compute

$$r_{l+1}=b-\mathbf{A}\phi_{l+1},\quad \beta_l=\frac{r_{l+1}^T\,\mathbf{A}p_l}{p_l^T\,\mathbf{A}p_l},\quad p_{l+1}=r_{l+1}+\beta_l p_l$$

5. set $l = l + 1$ and go to step 2

The convergence of CG is optimal when the eigenvalues of the coefficient matrix are similar to each other.[11a] Thus preconditioner is often used in the CG method to achieve this goal. Specifically a preconditioner matrix **M** is introduced into equation (4)

$$(\mathbf{M}^{-1}\mathbf{A}\mathbf{M}^{-1})(\mathbf{M}\phi)=\mathbf{M}^{-1}b, \quad (5)$$

so that the new linear system becomes

$$\tilde{\mathbf{A}}\tilde{\phi}=\tilde{b} \quad (6)$$

By directly incorporating preconditioning into CG iteration, the resulting algorithm can be summarized as follows:

1. set $l = 0$, $r_0 = b - A\phi_0$
2. solve $Mz_0 = r_0$ for z_0, let $p_0 = z_0$
3. calculate the norm of residue $||r_l||$. If $||r_l||/||b|| < \delta$, output ϕ_l. Otherwise go to the next step.
4. set $l = l + 1$
5. calculate

$$\alpha_l=(r_{l-1}^T\,z_{l-1})/(p_{l-1}^T A p_{l-1}),\quad \phi_l=\phi_{l-1}+\alpha_l p_{l-1},\quad r_l=r_{l-1}-\alpha_l A p_{l-1}$$

6. solve $Mz_l = r_l$ for z_l
7. calculate

$$\beta_l=(r_l^T\,z_l)/(r_{l-1}^T\,z_{l-1}),\quad p_l=z_l+\beta_l p_{l-1}$$

8. go to step 3

We can see that the preconditioned CG algorithm involves an additional operation at each iteration to solve the linear system $Mz_l = r_l$.

2.3 Incomplete Cholesky Preconditioners

A commonly used type of preconditioners is based on the incomplete $\mathbf{LDL}^{\mathrm{T}}$ factorization

$$\mathbf{M}=(\tilde{\mathbf{D}}+\mathbf{L})\tilde{\mathbf{D}}^{-1}(\tilde{\mathbf{D}}+\mathbf{L}^{T}). \quad (7)$$

Here the matrices are related to the original coefficient matrix **A** as **A** = **L** + **D** + $\mathbf{L}^T$ with **L** as the strictly lower triangular matrix of **A** and **D** as the positive diagonal matrix of **A**. Finally $\tilde{\mathbf{D}}$ is an undetermined positive diagonal matrix. If the diagonal of **M** is defined as **D**, the preconditioned conjugate gradient is termed ICCG. In MICCG, the diagonal elements of $\tilde{\mathbf{D}}$ are optimized to further improve the convergence.[8] The MICCG method is our default CPU implementation for our PBSA program in the Amber and AmberTools releases.

2.4 Jacobi Preconditioner

The Jacobi preconditioner (aka diagonal preconditioner) simply extracts the main diagonal **D** of **A** as **M**. Jacobi preconditioning is very inexpensive to use and is reasonably efficient for diagonally dominant matrices, though its reduction in the iteration number is modest. However, for the GPU implementation, the Jacobi preconditioner is advantageous because it is completely lack of row dependency, leading to great parallel efficiency. Additionally, the Jacobi preconditioner needs very little storage as to be discussed below.

2.5 Smoothed-aggregation-based Algebraic Multigrid Preconditioner

Multigrid methods are highly efficient techniques to solve linear or nonlinear equations. Typically there are two classes of multigrid methods: geometric multigrid (GMG) and algebraic multigrid (AMG).[18] GMG methods require prior physical/mathematical knowledge of the underlying discretization and grid hierarchy, whereas AMG methods only require the coefficient matrix. Classical AMG methods involve the construction of a hierarchy of grids using the original coefficient matrix. The hierarchical grids are obtained by partitioning the grid nodes into coarse and fine grid nodes. The coarse grid nodes form a coarse level, and an interpolation operator, via a weighted sum of the coarse grid nodes, is used to interpolate a coarse level solution to a fine level. The restriction operator, usually taken as the transpose of the interpolation operator, is used to restrict a fine level solution to a coarse level.[19] Aggregation AMG methods obtain the hierarchical grids by aggregating a few fine grid nodes to form a coarse grid node. The interpolation operator uses a piecewise constant interpolation to obtain a fine level solution from a coarse level solution. This leads to rather sparse interpolation. The restriction operator is similar to that of the classical AMG methods. The aggregation scheme reduces the memory requirement and improves the interpolation efficiency, but it does not provide grid independent convergence.[19b] Therefore smooth interpolation or smooth aggregation is often used to improve the convergence.[20]

Unlike classical AMG, smoothed-aggregation-based AMG (SA-AMG) is not robust for various applications.[19b] Thus SA-AMG is often used as a preconditioner for generalized minimal residual and conjugate gradient methods. In this study, we tested the use of SA-AMG method to build a preconditioner (**M**) to the conjugate gradient method as implemented in CUSP.

2.6 GPU Implementation

The latest generations of GPU cards and Nvidia CUDA provide mature computing platforms for scientific applications. CUDA gives developers direct access to parallel computational elements (GPUs) and enables code to run concurrently in CPUs. Several CUDA-compatible libraries were utilized to implement a GPU-ready Amber/PBSA program. The CUDA Basic Linear Algebra Subroutines (cuBLAS) library is a GPU-accelerated BLAS library that are "6× to 17× faster" than the latest MKL in GEMM (GEneral Matrix Multiplication) performance measurement.[21] The Nvidia CUDA Sparse Matrix (cuSPARSE) library provides basic linear algebra procedures for sparse matrix operations that are "up to 8× faster" than the latest MKL.[22] The cuSPARSE library is designed to interface with C or C++ functions. It supports multiple sparse matrix storage formats, such as Coordinate (COO), Compressed Sparse Row (CSR), Compressed Sparse Column (CSC), ELLPACK (ELL), Hybrid ELL+COO (HYB), and Blocked CSR. Finally CUSP is an open source C++ library based on Thrust. It can also provide sparse matrix operations in the CUDA environment.[23] CUSP supports COO, CSR, Diagonal (DIA), ELL, and HYB matrix formats.

In this study we implemented four types of FDM solvers, i.e. CG, ICCG, Jacobi-CG and SA-AMG-CG using cuBLAS, cuSPARSE, and CUSP libraries. We also tested these implementations with five different matrix formats DIA, CSR, COO, ELL, and HYB to analyze the impact of matrix formats upon efficiency. A total of 15 GPU combinations are possible as summarized in Table 1 with CG-CPU and ICCG-CPU also listed for comparison. Apparently not every combination is available, e.g. the cuSPARSE library only works with the CSR format; Jacobi-CG only works with the CUSP library, and SA-AMG-CG only works with the CSR, COO and HYB formats in the CUSP library.

2.7 Computational Details

All CUDA solvers were implemented in the single precision within the Amber/PBSA program of the Amber 16 package,[24] while the system setup and the energy/force calculation were still implemented in the double precision. In contrast, all implementations in the CPU solvers use the double precision. A total of 573 biomolecular structures including proteins, short peptides, and nucleic acids in the Amber benchmark suite were used in our test.[3l] These biomolecules consist of atoms ranging from 247 to 8,254 and have quite different geometries, and they were assigned charges of Cornell *et al*[25] and the modified Bondi radii.

All testings were performed with the following conditions unless specified otherwise. The convergence criteria of 10^{-3} and 10^{-6} were used for performance comparisons for low and high-precision applications, respectively. The default grid spacing of 0.5 Å was used. The ratio of the grid dimension over the solute dimension was set to 1.5. No electrostatic focusing was applied for easy timing analysis. The potential values on all grids were initialized to zero. The dielectric constants were set to 80 and 1 for solvent and solute, respectively. The weighted harmonic average of the solvent and solute dielectric constants was used as the boundary dielectric constants. Therefore, the symmetric and positive-definite coefficient matrices were obtained and suitable for all tested linear solvers. In addition, the FDM matrix was initialized into CSR format and transformed into other

formats when needed. Finally both the free space boundary condition (FBC) and periodic boundary condition (PBC) were tested. In PBC applications, we filled the matrix elements on the additional 6 bands into the original 7 bands and stored their column index non-consecutively in the CSR index arrays, thus we managed to use the same space as the original 7 bands in free boundary condition. All other parameters were set as default in the PBSA program in the Amber 16 package.[24]

We performed all measurements on a hybrid node with two NVIDIA GeForce GTX 980 Ti GPU cards and one Intel Xeon E5-1620 v3 CPU and 16GB main memory. The test platform is one of our standard GPU nodes with Intel X99 chipset, LGA2011-v3 CPU socket, DDR4-2133 memory, and PCIE 3.0 ×16 interconnection between the host CPU and the two GPU cards. The Intel Xeon E5-1620 v3 CPU was set as four threads, though all test runs were performed on a single thread. The Operating System is CentOS 6.6 as distributed in the ROCK 6.2 release. The CPU timing measurements include all execution time of the core routine, i.e. time elapsed on both GPU and CPU, as well as time for transferring data between GPU and CPU.

3. Results and Discussion

3.1 Accuracy of GPU implementations

It is important to guarantee that the GPU implementations achieve consistent numerical results with existing CPU implementations within specified convergence criterion. As shown in Figure 1 for calculations in the free boundary condition, the electrostatic solvation energies on GPU (Jacobi-CG) and on CPU (CG) correlate quite well with both 10^{-3} and 10^{-6} convergence criteria. The linear regression slopes are 0.999931 and 0.999996, respectively, and the correlation coefficients are 1.0 for both. The maximum relative energy errors are 3.0×10^{-3} and 6.3×10^{-6}, which are in agreement with the convergence criteria chosen.

Similar agreements were also observed between the two implementations in the periodic boundary condition as shown in Figure 2. The linear regression slopes are 0.999639 and 1.0 for 10^{-3} and 10^{-6} convergence criteria, respectively, and the correlation coefficients are 1.0 for both. The maximum relative errors are 3.9×10^{-3} and 5.8×10^{-6}, respectively, in agreement with the preset convergence criteria. For all other GPU implementations of which the data are not shown here, similar agreements were also observed.

3.2 Efficiency of GPU implementations

To compare the efficiency of GPU and CPU implementations, we first selected eight representative proteins and measured their solver CPU times with 10^{-3} and 10^{-6} convergence criteria, respectively. The standard CG solver as implemented on GPU and CPU was first analyzed. Table 2 shows that the GPU/CG solver overall performs better than the CPU/CG solver, with a speedup of ~6 to ~10 for the low convergence criterion and a speedup of ~8 to ~13 for the high convergence criterion. This is encouraging given that the tested GPU/CG solver is an unconditioned CG from the standard library without any change.

We next compared the GPU/CG solver with our default CPU solver CPU/ICCG, which was hand-optimized in the matrix-free fashion for modern CPUs. It is interesting to note that GPU/CG still performs better than CPU/ICCG for the larger proteins with the number of grid nodes over 2 million at the low convergence criterion. Furthermore, it performs better than CPU/ICCG for all tested proteins at the high convergence criterion.

A natural direction to go is to port ICCG to GPU, and this was implemented using the cuSPARSE library in this study. Unfortunately, our test shows that ICCG performs poorly on GPU platforms. This is consistent with the widely known fact that ICCG is not suitable for parallel platforms. Indeed, the inefficiency of the GPU/ICCG solver is significant: over 20 times slower than the standard CPU/CG solver. It should be pointed out that the the poor efficiency of GPU/ICCG is observed even with Nvidia's in-house optimization.[26] The specialized solver intends to find any independence in the sparse matrix during the analysis phase to solve the linear system in a parallel fashion.[26] In the case of linear PBE systems, however, this strategy fails to find any significant data independence in the seven-banded matrix.

Nevertheless, there are other solvers that are more suitable for the parallel GPU platforms. It appears that several are available. Our comprehensive analysis shows that Jacobi-CG is quite attractive. It was implemented with the CUSP library in the DIA matrix format. As shown in Table 2, the GPU/Jacobi-CG solver is about 95 to 283 times faster than GPU/ICCG for the low convergence criterion; and 194 to 407 times faster than GPU/ICCG for the high convergence criterion. The dramatically better performance of GPU/Jacobi-CG over GPU/ICCG lies in the simple utilization of the diagonal matrix as a preconditioner, which is completely without row dependency, so that it greatly facilitates parallel execution.

Another interesting solver that can take advantage of GPU platforms is the SA-AMG-CG solver. We implemented the SA-AMG-CG solver in the CUSP library and observed reasonable speedup. Different from ICCG, the GPU/SA-AMG-CG implementation is observed to perform similarly among the best GPU implementations: slightly slower than GPU/CG and GPU/Jacobi-CG, but more efficient than both CPU/CG and CPU/ICCG at the high convergence criterion as shown in Table 2. It is less efficient than CPU/ICCG at the low convergence criterion, but clearly better than the standard CPU/CG implementation. These data indicate the potential to further implement multigrid types of linear solvers, such as geometric multi-grid solvers, for GPU platforms.

Given the above detailed comparison of multiple implementations on selected proteins, it is clear that the GPU/Jacobi-CG is overall the most efficient implementation at both testing conditions (low and high convergence criteria). To properly gauge the overall speedups for typical applications for both free space boundary condition and periodic boundary condition, we plotted the speedup ratios of the GPU/Jacobi-CG implementation over the CPU/CG implementation using all test cases. As shown in Figure 3 for the free space boundary condition and Figure 4 for the periodic boundary condition, a speedup ratio of about 5 to 50 can be observed. The actual values clearly depend on the size/structure and of a given system. An interesting observation is that the speedup is not influenced much by the boundary conditions by comparing the trends in Figure 3 and Figure 4. However, it is clear

that the CPU/ICCG implementation is more efficient in the free space boundary condition with speedup ratios up to 18 versus speedup ratios up to 6 in the periodic boundary condition in the low convergence criterion. This is because the difficulties in implementing periodic boundary condition in the highly optimized ICCG solver that prevent certainly data management ideas, i.e. array padding, to be used on CPU platforms as discussed previously.[3n]

As in other computational sciences, the sparse matrix structure is a typical feature when a partial differential equation, such as PBE, is discretized. As a result sparse matrix-vector multiplications (SpMV) are critical operations in PBE solvers and represents the dominant cost in many iterative methods. In our CPU/CG and CPU/ICCG implementations, hand tuning was employed to fully utilize the banded structure of the matrix for efficient SpMV operations. For example, the SpMV operation between boundary elements and the potential grids in the PBC linear solvers is carried out by directly shifting the column index into the array index, avoiding extra matrix column manipulation.[3n] In addition, with only several extra indices to mark the columns, seven arrays are enough to store the non-zero elements of the banded coefficient matrix, i.e. no extra row or column index is needed. Compared to the CSR format storage, this can save as much as 53% of the memory usage, which also leads to dramatically reduced memory load and store operations. However, these improvements in our CPU implementations are not fully available in the existing CUDA libraries. These features will be adopted when developing hand-optimized GPU solvers in our next step.

3.3 Other issues of GPU implementations

Efficiency of a GPU solver is also significantly affected by the matrix storage format. There are a number of sparse matrix representations with different storage requirements, computational characteristics, and methods of accessing and manipulating matrix elements as summarized in the Methods section. The DIA format is tailored for highly specific classes of matrices and is the most computationally attractive for the banded matrices in our linear PBE systems.[27] This is apparent in Figure 5 with Jacobi-DIA and CG-DIA being the best. However, the current cuSPARSE library does not support the DIA format, so that only the CSR format, a general-purpose format, was used for performance evaluation. Finally it should be pointed out none of the GPU/ICCG implementations are shown in Figure 5 due to their extremely long execution times.

Given all the issues addressed, it is instructive to compare all GPU solver implementations as shown in Figure 5. This comparison also provides an opportunity to study the robustness of all GPU implementations. We examined all 573 test cases using both free space boundary condition and periodic boundary condition at both convergence criteria, to analyze the overall scaling of all GPU implementations. Indeed not every GPU implementation is robust enough to function properly for all tested conditions and molecules. There are five failures (failed to converge) and two unstable runs (converged but with unusually long time) for GPU/SA-AMG-CG in three out of the five tested formats (CSR/COO/HYB). The method fails in all test conditions and all molecules in the other two tested formats (DIA and ELL). Most failures were due to bad memory allocation, and others were due to unknown internal failures that lead to incorrect numerical solutions. This comprehensive scaling test confirms

that GPU/Jacobi-CG from the CUSP library outperforms all its GPU and CPU counterparts significantly, and is also noticeably faster than our default CPU/ICCG implementation. As a reference, the CPU/ICCG is on average 10 times faster than CPU/CG as shown in Figure 3 and 4, consistent with the findings of Wang et al.[31]

3.4 Memory usage of GPU implementations

Memory usage is also crucial for GPU implementations because memory is often limited on most consumer-grade graphics cards. Apparently different solvers and matrix storage methods lead to different memory usages. In implementations with the CUSP library, typical memory usage is 88×Ngrid bytes for GPU/CG and 92×Ngrid for GPU/Jacobi-CG in the CSR matrix format. With the cuSPARSE library, GPU/CG consumes up to 76×Ngrid bytes of GPU memory and GPU/ICCG uses 180×Ngrid bytes in the CSR matrix. Here the estimations are based on the use of four-byte integer and float types. In addition, we managed to use about the same memory for both PBC and FBC applications as mentioned in Methods. This in part contributes to the consistent efficiency between the two boundary conditions for the tested molecules.

The above estimations are only based on those arrays explicitly allocated in the program. Run-time analysis by the NVIDIA hardware manage tool (nvidia-smi), however, shows that the total memory is about twice as much due the hidden buffer space allocated within the CUSP and cuSPARSE libraries. Thus the actual memory limit was underestimated in the estimations. Extensive test of the fastest implementation, GPU/Jacobi-CG, shows that it was able to successfully complete linear PBE calculations with ~29.6 million grid nodes on the NVIDIA GTX 980 Ti cards with ~6GB GPU memory, about twice as smaller as the estimation.

4. Conclusions

In this study, we implemented multiple linear PBE solvers based on the standard CUDA libraries and conducted a systematic analysis on their performance with a large set of realistic biomolecules. We first analyzed the accuracy of the GPU implementation with respect to the CPU implementation in both free boundary condition and periodic boundary condition. The analysis shows that the GPU and CPU implementations agree within specified convergence criteria even if single precision was used in consumer grade graphics cards used in the test.

Many GPU solvers perform better than the standard CPU/CG solver, with various speedup ratios, depending on convergence criterion and size of the linear systems. In the comprehensive scaling test, our data shows that GPU/Jacobi-CG from the CUSP library outperforms all its GPU and CPU counterparts significantly. A speedup ratio of about 5 to 50 can be observed and it is not influenced much by the boundary conditions or convergence criteria. This should be compared with our default CPU implementation – the CPU/ICCG implementation, which is more efficient in the free space boundary condition. The speedup is reduced in the high convergence criterion in both boundary conditions tested. Unfortunately our test shows that the ICCG method performs poorly on GPU platforms. Moreover, we implemented the SA-AMG-CG method and it was found to perform similarly

among the best GPU implementations. These data indicate the potential to further implement multigrid types of linear solvers for GPU platforms.

It is also worth pointing out that the efficiency of a GPU solver is significantly affected by matrix storage formats. The DIA format is tailored for banded matrices and is the most computationally efficient for linear PBE matrices. Furthermore we discussed the memory usage of these solvers. Extensive test of the fastest implementation, GPU/Jacobi-CG, shows that it was able to successfully complete FDPB calculations with ~29.6 million grid points on the NVIDIA GTX 980 Ti cards with 6GB GPU memory, about twice as smaller as the theoretical analysis.

Finally further efficiency gain in GPU implementations is more likely to be achieved with customized matrix-free operations, integrated grid stencil setup on GPU, and also multigrid types of solvers, specifically tailored for our particular linear PBE problems. These developments are currently underway in our group.

Acknowledgments

This work was supported by National Institutes of Health/NIGMS (GM093040 & GM079383).

References

1. (a) Davis ME, McCammon JA. ELECTROSTATICS IN BIOMOLECULAR STRUCTURE AND DYNAMICS. Chem Rev (Washington, DC, U S). 1990; 90(3):509–521.(b) Honig B, Sharp K, Yang AS. MACROSCOPIC MODELS OF AQUEOUS-SOLUTIONS - BIOLOGICAL AND CHEMICAL APPLICATIONS. J Phys Chem. 1993; 97(6):1101–1109.(c) Honig B, Nicholls A. CLASSICAL ELECTROSTATICS IN BIOLOGY AND CHEMISTRY. Science. 1995; 268(5214): 1144–1149. [PubMed: 7761829] (d) Beglov D, Roux B. Solvation of complex molecules in a polar liquid: An integral equation theory. J Chem Phys. 1996; 104(21):8678–8689.(e) Cramer CJ, Truhlar DG. Implicit solvation models: Equilibria, structure, spectra, and dynamics. Chem Rev (Washington, DC, U S). 1999; 99(8):2161–2200.(f) Bashford D, Case DA. Generalized born models of macromolecular solvation effects. Annu Rev Phys Chem. 2000; 51:129–152. [PubMed: 11031278] (g) Baker NA. Improving implicit solvent simulations: a Poisson-centric view. Curr Opin Struct Biol. 2005; 15(2):137–143. [PubMed: 15837170] (h) Chen JH, Im WP, Brooks CL. Balancing solvation and intramolecular interactions: Toward a consistent generalized born force field. J Am Chem Soc. 2006; 128(11):3728–3736. [PubMed: 16536547] (i) Feig M, Chocholousova J, Tanizaki S. Extending the horizon: towards the efficient modeling of large biomolecular complexes in atomic detail. Theor Chem Acc. 2006; 116(1–3):194–205.(j) Koehl P. Electrostatics calculations: latest methodological advances. Curr Opin Struct Biol. 2006; 16(2):142–151. [PubMed: 16540310] (k) Im W, Chen JH, Brooks CL. Peptide and protein folding and conformational equilibria: Theoretical treatment of electrostatics and hydrogen bonding with implicit solvent models. Peptide Solvation and H-Bonds. 2006; 72:173.(l) Lu BZ, Zhou YC, Holst MJ, McCammon JA. Recent progress in numerical methods for the Poisson-Boltzmann equation in biophysical applications. Communications in Computational Physics. 2008; 3(5):973–1009.(m) Wang J, Tan CH, Tan YH, Lu Q, Luo R. Poisson-Boltzmann solvents in molecular dynamics Simulations. Communications in Computational Physics. 2008; 3(5):1010–1031.(n) Altman MD, Bardhan JP, White JK, Tidor B. Accurate Solution of Multi-Region Continuum Biomolecule Electrostatic Problems Using the Linearized Poisson-Boltzmann Equation with Curved Boundary Elements. J Comput Chem. 2009; 30(1):132–153. [PubMed: 18567005] (o) Cai, Q., Wang, J., Hsieh, M-J., Ye, X., Luo, R. Chapter Six - Poisson–Boltzmann Implicit Solvation Models. In: Ralph, AW., editor. Annual Reports in Computational Chemistry. Vol. 8. Elsevier; 2012. p. 149-162. (p) Xiao L, Wang C, Luo R. Recent progress in adapting Poisson–Boltzmann methods to molecular simulations. Journal of Theoretical and Computational Chemistry. 2014; 13(03):1430001.(q)

Botello-Smith WM, Cai Q, Luo R. Biological applications of classical electrostatics methods. Journal of Theoretical and Computational Chemistry. 2014; 13(03):1440008.

2. (a) Warwicker J, Watson HC. Calculation of the Electric-Potential in the Active-Site Cleft Due to Alpha-Helix Dipoles. J Mol Biol. 1982; 157(4):671–679. [PubMed: 6288964] (b) Bashford D, Karplus M. Pkas Of Ionizable Groups In Proteins - Atomic Detail From A Continuum Electrostatic Model. Biochemistry. 1990; 29(44):10219–10225. [PubMed: 2271649] (c) Jeancharles A, Nicholls A, Sharp K, Honig B, Tempczyk A, Hendrickson TF, Still WC. Electrostatic Contributions To Solvation Energies - Comparison Of Free-Energy Perturbation And Continuum Calculations. J Am Chem Soc. 1991; 113(4):1454–1455.(d) Gilson MK. Theory of Electrostatic Interactions in Macromolecules. Curr Opin Struct Biol. 1995; 5(2):216–223. [PubMed: 7648324] (e) Edinger SR, Cortis C, Shenkin PS, Friesner RA. Solvation free energies of peptides: Comparison of approximate continuum solvation models with accurate solution of the Poisson-Boltzmann equation. J Phys Chem B. 1997; 101(7):1190–1197.(f) Lu Q, Luo R. A Poisson-Boltzmann dynamics method with nonperiodic boundary condition. J Chem Phys. 2003; 119(21):11035–11047.(g) Tan C, Yang L, Luo R. How well does Poisson-Boltzmann implicit solvent agree with explicit solvent? A quantitative analysis. J Phys Chem B. 2006; 110(37):18680–18687. [PubMed: 16970499] (h) Cai Q, Wang J, Zhao HK, Luo R. On removal of charge singularity in Poisson-Boltzmann equation. J Chem Phys. 2009; 130(14)(i) Ye X, Cai Q, Yang W, Luo R. Roles of Boundary Conditions in DNA Simulations: Analysis of Ion Distributions with the Finite-Difference Poisson-Boltzmann Method. Biophys J. 2009; 97(2):554–562. [PubMed: 19619470] (j) Ye X, Wang J, Luo R. A Revised Density Function for Molecular Surface Calculation in Continuum Solvent Models. J Chem Theory Comput. 2010; 6(4):1157–1169. [PubMed: 24723844] (k) Luo R, Moult J, Gilson MK. Dielectric screening treatment of electrostatic solvation. J Phys Chem B. 1997; 101(51):11226–11236.(l) Wang J, Tan C, Chanco E, Luo R. Quantitative analysis of Poisson-Boltzmann implicit solvent in molecular dynamics. Phys Chem Chem Phys. 2010; 12(5):1194–1202. [PubMed: 20094685] (m) Hsieh MJ, Luo R. Exploring a coarse-grained distributive strategy for finite-difference Poisson-Boltzmann calculations. J Mol Model. 2011; 17(8):1985–1996. [PubMed: 21127924] (n) Cai Q, Ye X, Wang J, Luo R. On-the-Fly Numerical Surface Integration for Finite-Difference Poisson-Boltzmann Methods. J Chem Theory Comput. 2011; 7(11):3608–3619. [PubMed: 24772042] (o) Botello-Smith WM, Liu X, Cai Q, Li Z, Zhao H, Luo R. Numerical Poisson-Boltzmann Model for Continuum Membrane Systems. Chem Phys Lett. 2012(p) Liu X, Wang C, Wang J, Li Z, Zhao H, Luo R. Exploring a charge-central strategy in the solution of Poisson's equation for biomolecular applications. Phys Chem Chem Phys. 2013(q) Wang C, Wang J, Cai Q, Li ZL, Zhao H, Luo R. Exploring High Accuracy Poisson-Boltzmann Methods for Biomolecular Simulations. Computational and Theoretical Chemistry. 2013; 1024:34–44. [PubMed: 24443709]

3. (a) Klapper I, Hagstrom R, Fine R, Sharp K, Honig B. Focusing of Electric Fields in the Active Site of Copper-Zinc Superoxide Dismutase Effects of Ionic Strength and Amino Acid Modification. Proteins Structure Function and Genetics. 1986; 1(1):47–59.(b) Davis ME, McCammon JA. Solving the Finite-Difference Linearized Poisson-Boltzmann Equation - a Comparison of Relaxation and Conjugate-Gradient Methods. J Comput Chem. 1989; 10(3):386–391.(c) Nicholls A, Honig B. A Rapid Finite-Difference Algorithm, Utilizing Successive over-Relaxation to Solve the Poisson-Boltzmann Equation. J Comput Chem. 1991; 12(4):435–445.(d) Luty BA, Davis ME, McCammon JA. Solving the Finite-Difference Nonlinear Poisson-Boltzmann Equation. J Comput Chem. 1992; 13(9):1114–1118.(e) Holst M, Saied F. Multigrid Solution of the Poisson-Boltzmann Equation. J Comput Chem. 1993; 14(1):105–113.(f) Forsten KE, Kozack RE, Lauffenburger DA, Subramaniam S. Numerical-Solution of the Nonlinear Poisson-Boltzmann Equation for a Membrane-Electrolyte System. J Phys Chem. 1994; 98(21):5580–5586.(g) Holst MJ, Saied F. NUMERICAL-SOLUTION OF THE NONLINEAR POISSON-BOLTZMANN EQUATION - DEVELOPING MORE ROBUST AND EFFICIENT METHODS. J Comput Chem. 1995; 16(3):337–364.(h) Bashford D. An Object-Oriented Programming Suite for Electrostatic Effects in Biological Molecules. Lecture Notes in Computer Science. 1997; 1343:233–240.(i) Im W, Beglov D, Roux B. Continuum Solvation Model: computation of electrostatic forces from numerical solutions to the Poisson-Boltzmann equation. Comput Phys Commun. 1998; 111(1–3):59–75.(j) Rocchia W, Alexov E, Honig B. Extending the applicability of the nonlinear Poisson-Boltzmann equation: Multiple dielectric constants and multivalent ions. J Phys Chem B. 2001; 105(28):6507–6514.(k) Luo R, David L, Gilson MK. Accelerated Poisson-Boltzmann calculations for static and dynamic systems. J Comput Chem. 2002; 23(13):1244–1253. [PubMed: 12210150] (l) Wang J, Luo R. Assessment of Linear Finite-

Difference Poisson-Boltzmann Solvers. J Comput Chem. 2010; 31(8):1689–1698. [PubMed: 20063271] (m) Cai Q, Hsieh MJ, Wang J, Luo R. Performance of Nonlinear Finite-Difference Poisson-Boltzmann Solvers. J Chem Theory Comput. 2010; 6(1):203–211. [PubMed: 24723843] (n) Botello-Smith WM, Luo R. Applications of MMPBSA to Membrane Proteins I: Efficient Numerical Solutions of Periodic Poisson–Boltzmann Equation. J Chem Inf Model. 2015; 55(10):2187–2199. [PubMed: 26389966]

4. (a) Cortis CM, Friesner RA. Numerical solution of the Poisson-Boltzmann equation using tetrahedral finite-element meshes. J Comput Chem. 1997; 18(13):1591–1608.(b) Holst M, Baker N, Wang F. Adaptive multilevel finite element solution of the Poisson-Boltzmann equation I. Algorithms and examples. J Comput Chem. 2000; 21(15):1319–1342.(c) Baker N, Holst M, Wang F. Adaptive multilevel finite element solution of the Poisson-Boltzmann equation II. Refinement at solvent-accessible surfaces in biomolecular systems. J Comput Chem. 2000; 21(15):1343–1352.(d) Shestakov AI, Milovich JL, Noy A. Solution of the nonlinear Poisson-Boltzmann equation using pseudo-transient continuation and the finite element method. J Colloid Interface Sci. 2002; 247(1): 62–79. [PubMed: 16290441] (e) Chen L, Holst MJ, Xu JC. The finite element approximation of the nonlinear Poisson-Boltzmann equation. SIAM Journal on Numerical Analysis. 2007; 45:2298–2320.(f) Xie D, Zhou S. A new minimization protocol for solving nonlinear Poisson–Boltzmann mortar finite element equation. BIT Numerical Mathematics. 2007; 47(4):853–871.(g) Wang J, Cieplak P, Li J, Wang J, Cai Q, Hsieh M, Lei H, Luo R, Duan Y. Development of Polarizable Models for Molecular Mechanical Calculations II: Induced Dipole Models Significantly Improve Accuracy of Intermolecular Interaction Energies. J Phys Chem B. 2011; 115(12):3100–3111. [PubMed: 21391583] (h) Lu B, Holst MJ, McCammon JA, Zhou YC. Poisson-Nernst-Planck equations for simulating biomolecular diffusion-reaction processes I: Finite element solutions. J Comput Phys. 2010; 229(19):6979–6994. [PubMed: 21709855] (i) Bond SD, Chaudhry JH, Cyr EC, Olson LN. A First-Order System Least-Squares Finite Element Method for the Poisson-Boltzmann Equation. J Comput Chem. 2010; 31(8):1625–1635. [PubMed: 19908291]

5. (a) Miertus S, Scrocco E, Tomasi J. Electrostatic Interaction of a Solute with a Continuum - a Direct Utilization of Abinitio Molecular Potentials for the Prevision of Solvent Effects. Chem Phys. 1981; 55(1):117–129.(b) Hoshi H, Sakurai M, Inoue Y, Chujo R. Medium Effects on the Molecular Electronic-Structure .1. the Formulation of a Theory for the Estimation of a Molecular Electronic-Structure Surrounded by an Anisotropic Medium. J Chem Phys. 1987; 87(2):1107–1115.(c) Zauhar RJ, Morgan RS. The Rigorous Computation of the Molecular Electric-Potential. J Comput Chem. 1988; 9(2):171–187.(d) Rashin AA. Hydration Phenomena, Classical Electrostatics, and the Boundary Element Method. J Phys Chem. 1990; 94(5):1725–1733.(e) Yoon BJ, Lenhoff AM. A Boundary Element Method for Molecular Electrostatics with Electrolyte Effects. J Comput Chem. 1990; 11(9):1080–1086.(f) Juffer AH, Botta EFF, Vankeulen BAM, Vanderploeg A, Berendsen HJC. The Electric-Potential of a Macromolecule in a Solvent - a Fundamental Approach. J Comput Phys. 1991; 97(1):144–171.(g) Zhou HX. Boundary-Element Solution of Macromolecular Electrostatics - Interaction Energy between 2 Proteins. Biophys J. 1993; 65(2):955–963. [PubMed: 8218918] (h) Bharadwaj R, Windemuth A, Sridharan S, Honig B, Nicholls A. The Fast Multipole Boundary-Element Method for Molecular Electrostatics - an Optimal Approach for Large Systems. J Comput Chem. 1995; 16(7):898–913.(i) Purisima EO, Nilar SH. A Simple yet Accurate Boundary-Element Method for Continuum Dielectric Calculations. J Comput Chem. 1995; 16(6): 681–689.(j) Liang J, Subramaniam S. Computation of molecular electrostatics with boundary element methods. Biophys J. 1997; 73(4):1830–1841. [PubMed: 9336178] (k) Vorobjev YN, Scheraga HA. A fast adaptive multigrid boundary element method for macromolecular electrostatic computations in a solvent. J Comput Chem. 1997; 18(4):569–583.(l) Totrov M, Abagyan R. Rapid boundary element solvation electrostatics calculations in folding simulations: Successful folding of a 23-residue peptide. Biopolymers. 2001; 60(2):124–133. [PubMed: 11455546] (m) Boschitsch AH, Fenley MO, Zhou HX. Fast boundary element method for the linear Poisson-Boltzmann equation. J Phys Chem B. 2002; 106(10):2741–2754.(n) Lu BZ, Cheng XL, Huang JF, McCammon JA. Order N algorithm for computation of electrostatic interactions in biomolecular systems. Proc Natl Acad Sci U S A. 2006; 103(51):19314–19319. [PubMed: 17148613] (o) Lu B, Cheng X, Huang J, McCammon JA. An Adaptive Fast Multipole Boundary Element Method for Poisson-Boltzmann Electrostatics. J Chem Theory Comput. 2009; 5(6):1692–1699. [PubMed: 19517026] (p) Bajaj C, Chen SC, Rand A. An Efficient Higher-Order Fast Multipole Boundary Element Solution For

Poisson-Boltzmann-Based Molecular Electrostatics. Siam Journal on Scientific Computing. 2011; 33(2):826–848. [PubMed: 21660123]

6. (a) Luo R, Head MS, Moult J, Gilson MK. pK(a) shifts in small molecules and HIV protease: Electrostatics and conformation. J Am Chem Soc. 1998; 120(24):6138–6146.(b) Georgescu RE, Alexov EG, Gunner MR. Combining conformational flexibility and continuum electrostatics for calculating pK(a)s in proteins. Biophys J. 2002; 83(4):1731–1748. [PubMed: 12324397] (c) Nielsen JE, McCammon JA. On the evaluation and optimization of protein X-ray structures for pKa calculations. Protein Sci. 2003; 12(2):313–326. [PubMed: 12538895] (d) Warwicker J. Improved pK(a) calculations through flexibility based sampling of a water-dominated interaction scheme. Protein Sci. 2004; 13(10):2793–2805. [PubMed: 15388865] (e) Tang CL, Alexov E, Pyle AM, Honig B. Calculation of pK(a)s in RNA: On the structural origins and functional roles of protonated nucleotides. J Mol Biol. 2007; 366(5):1475–1496. [PubMed: 17223134]
7. (a) Shivakumar D, Deng YQ, Roux B. Computations of Absolute Solvation Free Energies of Small Molecules Using Explicit and Implicit Solvent Model. J Chem Theory Comput. 2009; 5(4):919–930. [PubMed: 26609601] (b) Nicholls A, Mobley DL, Guthrie JP, Chodera JD, Bayly CI, Cooper MD, Pande VS. Predicting small-molecule solvation free energies: An informal blind test for computational chemistry. J Med Chem. 2008; 51(4):769–779. [PubMed: 18215013]
8. (a) Swanson JMJ, Henchman RH, McCammon JA. Revisiting free energy calculations: A theoretical connection to MM/PBSA and direct calculation of the association free energy. Biophys J. 2004; 86(1):67–74. [PubMed: 14695250] (b) Bertonati C, Honig B, Alexov E. Poisson-Boltzmann calculations of nonspecific salt effects on protein-protein binding free energies. Biophysical Journal. 2007; 92(6):1891–1899. [PubMed: 17208980] (c) Brice AR, Dominy BN. Analyzing the Robustness of the MM/PBSA Free Energy Calculation Method: Application to DNA Conformational Transitions. J Comput Chem. 2011; 32(7):1431–1440. [PubMed: 21284003] (d) Luo R, Gilson HSR, Potter MJ, Gilson MK. The physical basis of nucleic acid base stacking in water. Biophys J. 2001; 80(1):140–148. [PubMed: 11159389] (e) David L, Luo R, Head MS, Gilson MK. Computational study of KNI-272, a potent inhibitor of HIV-1 protease: On the mechanism of preorganization. J Phys Chem B. 1999; 103(6):1031–1044.(f) Luo R, Gilson MK. Synthetic adenine receptors: Direct calculation of binding affinity and entropy. J Am Chem Soc. 2000; 122(12):2934–2937.(g) Luo R, Head MS, Given JA, Gilson MK. Nucleic acid base-pairing and N-methylacetamide self-association in chloroform: affinity and conformation. Biophys Chem. 1999; 78(1–2):183–193. [PubMed: 10343387]
9. (a) Marshall SA, Vizcarra CL, Mayo SL. One- and two-body decomposable Poisson-Boltzmann methods for protein design calculations. Protein Sci. 2005; 14(5):1293–1304. [PubMed: 15802649] (b) Hsieh MJ, Luo R. Physical scoring function based on AMBER force field and Poisson-Boltzmann implicit solvent for protein structure prediction. Proteins-Structure Function and Bioinformatics. 2004; 56(3):475–486.(c) Wen EZ, Luo R. Interplay of secondary structures and side-chain contacts in the denatured state of BBA1. J Chem Phys. 2004; 121(5):2412–2421. [PubMed: 15260796] (d) Wen EZ, Hsieh MJ, Kollman PA, Luo R. Enhanced ab initio protein folding simulations in Poisson-Boltzmann molecular dynamics with self-guiding forces. J Mol Graphics Modell. 2004; 22(5):415–424.(e) Lwin TZ, Luo R. Overcoming entropic barrier with coupled sampling at dual resolutions. J Chem Phys. 2005; 123(19)(f) Lwin TZ, Zhou RH, Luo R. Is Poisson-Boltzmann theory insufficient for protein folding simulations? J Chem Phys. 2006; 124(3) (g) Lwin TZ, Luo R. Force field influences in beta-hairpin folding simulations. Protein Sci. 2006; 15(11):2642–2655. [PubMed: 17075138] (h) Tan YH, Luo R. Protein stability prediction: A Poisson-Boltzmann approach. J Phys Chem B. 2008; 112(6):1875–1883. [PubMed: 18211063] (i) Tan Y, Luo R. Structural and functional implications of p53 missense cancer mutations. BMC Biophysics. 2009; 2(1):5.(j) Korman TP, Tan YH, Wong J, Luo R, Tsai SC. Inhibition kinetics and emodin cocrystal structure of a type II polyketide ketoreductase. Biochemistry. 2008; 47(7):1837–1847. [PubMed: 18205400]
10. Wang J, Cai Q, Li Z-L, Zhao H-K, Luo R. Achieving energy conservation in Poisson-Boltzmann molecular dynamics: Accuracy and precision with finite-difference algorithms. Chem Phys Lett. 2009; 468(4–6):112–118. [PubMed: 20098487]
11. (a) Meijerink JA, Vandervorst HA. ITERATIVE SOLUTION METHOD FOR LINEAR-SYSTEMS OF WHICH COEFFICIENT MATRIX IS A SYMMETRIC M-MATRIX. Mathematics of Computation. 1977; 31(137):148–162.(b) Gustafsson I. A CLASS OF FIRST ORDER

FACTORIZATION METHODS. BIT Numerical Mathematics. 1978; 18(1):142–156.(c) Eisenstat SC. EFFICIENT IMPLEMENTATION OF A CLASS OF PRECONDITIONED CONJUGATE-GRADIENT METHODS. Siam Journal on Scientific and Statistical Computing. 1981; 2(1):1–4. (d) Meijerink JA, Vandervorst HA. GUIDELINES FOR THE USAGE OF INCOMPLETE DECOMPOSITIONS IN SOLVING SETS OF LINEAR-EQUATIONS AS THEY OCCUR IN PRACTICAL PROBLEMS. J Comput Phys. 1981; 44(1):134–155.

12. (a) Götz AW, Williamson MJ, Xu D, Poole D, Le Grand S, Walker RC. Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 1. Generalized Born. J Chem Theory Comput. 2012; 8(5):1542–1555. [PubMed: 22582031] (b) Páll S, Hess B. A flexible algorithm for calculating pair interactions on SIMD architectures. Comput Phys Commun. 2013; 184(12):2641–2650.(c) Salomon-Ferrer R, Götz AW, Poole D, Le Grand S, Walker RC. Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 2. Explicit Solvent Particle Mesh Ewald. J Chem Theory Comput. 2013; 9(9):3878–3888. [PubMed: 26592383]
13. (a) Ufimtsev IS, Martínez TJ. Quantum Chemistry on Graphical Processing Units. 1. Strategies for Two-Electron Integral Evaluation. J Chem Theory Comput. 2008; 4(2):222–231. [PubMed: 26620654] (b) Asadchev A, Gordon MS. New Multithreaded Hybrid CPU/GPU Approach to Hartree–Fock. J Chem Theory Comput. 2012; 8(11):4166–4176. [PubMed: 26605582] (c) Titov AV, Ufimtsev IS, Luehr N, Martinez TJ. Generating Efficient Quantum Chemistry Codes for Novel Architectures. J Chem Theory Comput. 2013; 9(1):213–221. [PubMed: 26589024]
14. (a) Colmenares J, Ortiz J, Rocchia W. GPU linear and non-linear Poisson–Boltzmann solver module for DelPhi. Bioinformatics. 2014; 30(4):569–570. [PubMed: 24292939] (b) Colmenares J, Galizia A, Ortiz J, Clematis A, Rocchia W. A Combined MPI-CUDA Parallel Solution of Linear and Nonlinear Poisson-Boltzmann Equation. BioMed Research International. 2014; 2014:560987. [PubMed: 25013789]
15. Press, WH., Teukolsky, SA., Vetterling, WT., Flannery, BP. Numerical recipes : the art of scientific computing. Cambridge University Press; Cambridge [Cambridgeshire]; New York: 1986.
16. Alcouffe RE, Brandt A, Dendy JE, Painter JW. THE MULTI-GRID METHOD FOR THE DIFFUSION EQUATION WITH STRONGLY DISCONTINUOUS COEFFICIENTS. Siam Journal on Scientific and Statistical Computing. 1981; 2(4):430–454.
17. Ruge, JWKS. Algebraic multigrid. In: McCormick, SF., editor. Multigrid Methods. Vol. 3. SIAM; Philadelphia: 1987. p. 73-130.
18. Stuben K. ALGEBRAIC MULTIGRID (AMG) - EXPERIENCES AND COMPARISONS. Applied Mathematics and Computation. 1983; 13(3–4):419–451.
19. (a) Stuben K. A review of algebraic multigrid. Journal of Computational and Applied Mathematics. 2001; 128(1–2):281–309.(b) Gandham R, Esler K, Zhang Y. A GPU accelerated aggregation algebraic multigrid method. Computers & Mathematics with Applications. 2014; 68(10):1151–1160.
20. Vaněk P, Mandel J, Brezina M. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. Computing. 1996; 56(3):179–196.
21. [accessed October 1st, 2016] Nvidia NVIDIA CUDA Basic Linear Algebra Subroutines library. https://developer.nvidia.com/cublas
22. [accessed October 1st, 2016] Nvidia NVIDIA CUDA Sparse Matrix library. https://developer.nvidia.com/cusparse
23. [accessed October 1st, 2016] Nvidia NVIDIA CUSP library. https://developer.nvidia.com/cusp
24. Case, DA., Betz, RM., Botello-Smith, W., Cerutti, DS., Cheatham, TEI., Darden, TA., Duke, RE., Giese, TJ., Gohlke, H., Goetz, AW., Homeyer, N., Izadi, S., Janowski, P., Kaus, J., Kovalenko, A., Lee, TS., LeGrand, S., Li, P., Lin, C., Luchko, T., Luo, R., Madej, B., Mermelstein, D., Merz, KM., Monard, G., Nguyen, H., Nguyen, HT., Omelyan, I., Onufriev, A., Roe, DR., Roitberg, A., Sagui, C., Simmerling, CL., Swails, J., Walker, RC., Wang, J., Wolf, RM., Wu, X., Xiao, L., York, DM., Kollman, PA. Amber 2016. University of California; San Francisco: 2016.
25. Cornell WD, Cieplak P, Bayly CI, Gould IR, Merz KM, Ferguson DM, Spellmeyer DC, Fox T, Caldwell JW, Kollman PA. A 2ND GENERATION FORCE-FIELD FOR THE SIMULATION OF PROTEINS, NUCLEIC-ACIDS, AND ORGANIC-MOLECULES. J Am Chem Soc. 1995; 117(19):5179–5197.

26. Naumov M. Incomplete-LU and Cholesky preconditioned iterative methods using CUSPARSE and CUBLAS. Nvidia white paper. 2011

27. Bell, N., Garland, M. Nvidia Technical Report NVR-2008-004. Nvidia Corporation; 2008. Efficient sparse matrix-vector multiplication on CUDA.

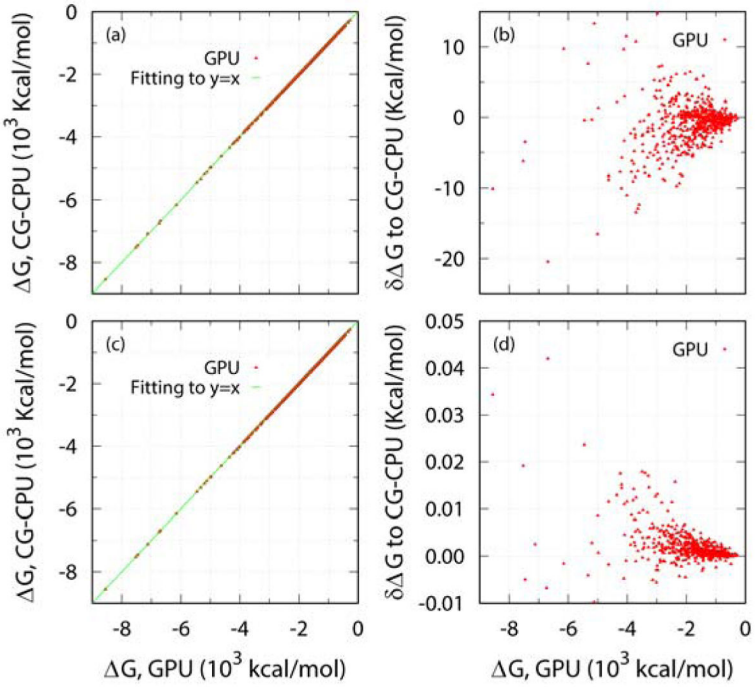


Figure 1.

Correlations (a), (c) and differences (b), (d) of electrostatic solvation energies on GPU (Jacobi with CUSP library and DIA matrix format) and on CPU (CG) for the protein test set. Free space boundary condition was used. The convergence criterion was set to 10^{-3} (a), (b) and 10^{-6} (c), (d). The linear regression slopes are 0.999931 and 0.999996 for 10^{-3} and 10^{-6} criterion respectively, and the correlation coefficients are 1.0 for both.

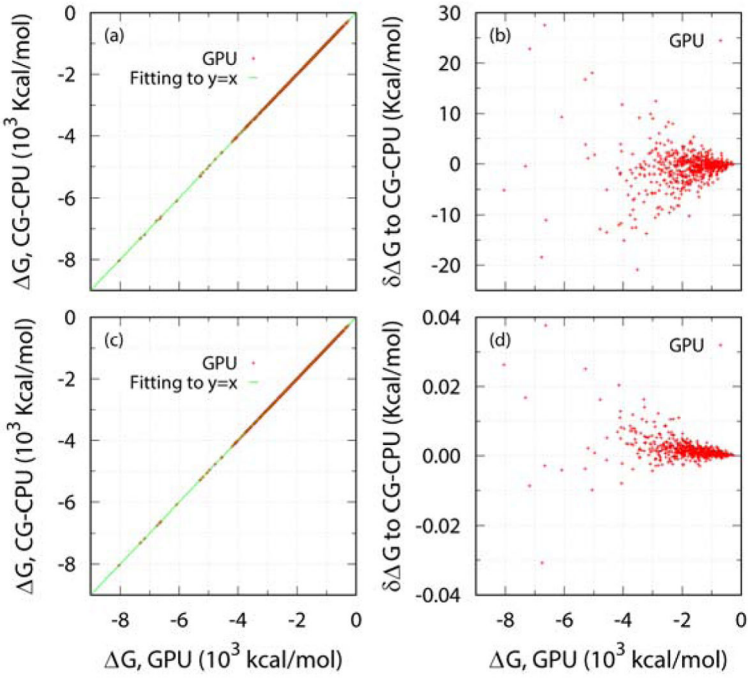


Figure 2.

Correlations (a), (c) and differences (b), (d) of electrostatic solvation energies on GPU (Jacobi with CUSP library and DIA matrix format) and on CPU (CG) for the protein test set. The periodic boundary condition (PBC) was used. The convergence criterion was set to 10^{-3} (a), (b) and 10^{-6} (c), (d). The linear regression slopes are 0.999639 and 1.0 for 10^{-3} and 10^{-6} criterion respectively, and the correlation coefficients are 1.0 for both.

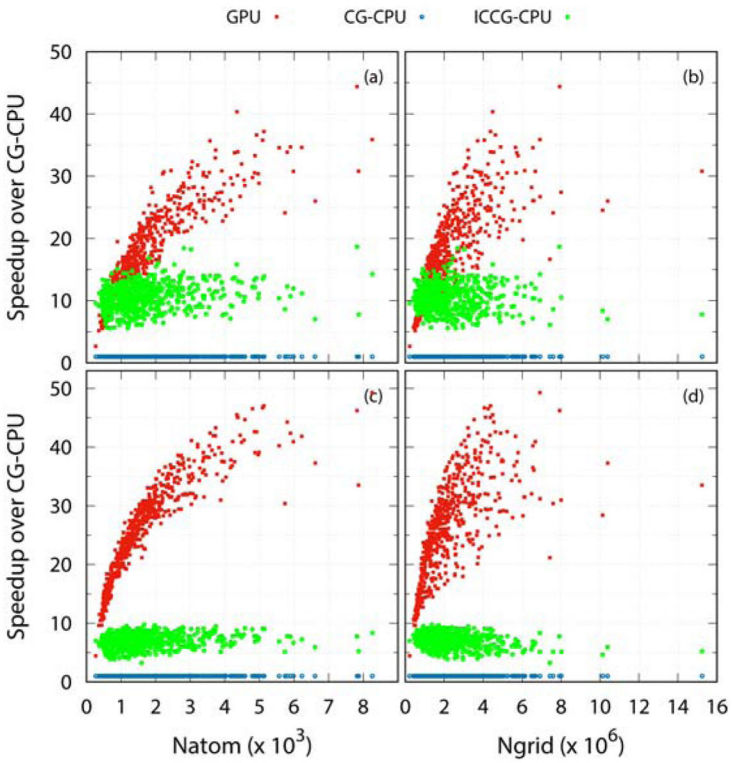


Figure 3.

Comparison between PB solvers on GPU (Jacobi with CUSP library and DIA matrix format) and on CPU with free space boundary condition for the protein test set, as functions of number of atoms and grids respectively. The convergence criterion was set to 10^{-3} (a), (b) and 10^{-6} (c), (d).

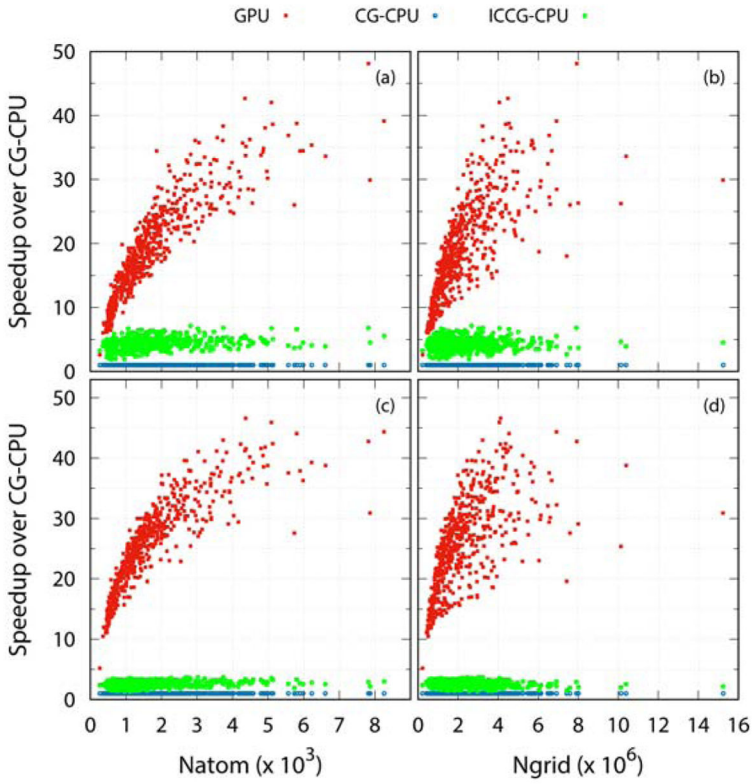


Figure 4.

Comparison between PB solvers on GPU (Jacobi with CUSP library and DIA matrix format) and on CPU (ICCG) with PBC for the protein test set, as functions of number of atoms and grids respectively. The convergence criterion was set to 10^{-3} (a), (b) and 10^{-6} (c), (d).

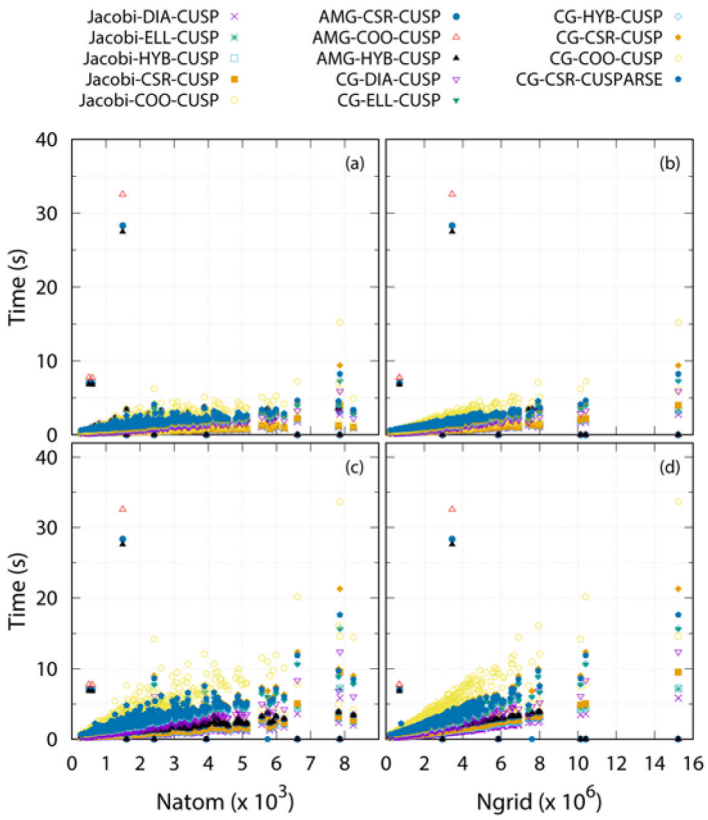


Figure 5.

Average time used by different CPU and GPU solvers for selected test proteins. The detail of each solver combination is elaborated in Table 1. The GPU/ICCG solver is not listed due to their extremely long execution times. The convergence criterion was set to 10^{-3} (top) and 10^{-6} (bottom).

Table 1

Details of tested solver combinations.

Solver Combination	Description
Jacobi-DIA-CUSP	Jacobi-preconditioned CG using CUSP library with DIA matrix format
Jacobi-ELL-CUSP	Jacobi-preconditioned CG using CUSP library with ELL matrix format
Jacobi-HYB-CUSP	Jacobi-preconditioned CG using CUSP library with HYB matrix format
Jacobi-CSR-CUSP	Jacobi-preconditioned CG using CUSP library with CSR matrix format
Jacobi-COO-CUSP	Jacobi-preconditioned CG using CUSP library with COO matrix format
AMG-CSR-CUSP	Smoothed-aggregation-based AMG using CUSP library with CSR matrix format
AMG-COO-CUSP	Smoothed-aggregation-based AMG using CUSP library with COO matrix format
AMG-HYB-CUSP	Smoothed-aggregation-based AMG using CUSP library with HYB matrix format
CG-ELL-CUSP	CG using CUSP library with ELL matrix format
CG-HYB-CUSP	CG using CUSP library with HYB matrix format
CG-CSR-CUSP	CG using CUSP library with CSR matrix format
CG-COO-CUSP	CG using CUSP library with COO matrix format
CG-DIA-CUSP	CG using CUSP library with DIA matrix format
CG-CSR-cuSPARSE	CG using cuSPARSE library with CSR matrix format
ICCG-CSR-cuSPARSE	ICCG using cuSPARSE library with CSR matrix format
CG-CPU	CG on CPU using standard system library with matrix free coding
ICCG-CPU	ICCG on CPU using standard system library with matrix free coding

Table 2

Average time (in second) used by CPU and GPU solvers for eight selected test proteins and representative solvers. The CG and Jacobi-preconditioned CG on GPU were carried out with CUSP library and DIA matrix format, and the SA-AMG-preconditioned CG on GPU was carried out with CUSP library and COO matrix format, while the ICCG solver on GPU was implemented with cuSPARSE library and CSR matrix format. The timing scheme for each solver include all execution time of the core routine code, i.e. time elapsed on device (GPU) and on host (CPU) and on transferring data between the device and the host. Both 10^{-3} and 10^{-6} criteria were used for comparison.

	Protein	Ngrid	CPU			GPU		
			CG	ICCG	CG	ICCG	Jacobi-CG	SA-AMG-CG
Convergence 10^{-3}	1pmc	739431	1.81	0.21	0.32	24.65	0.26	0.57
	1e01	1010510	3.36	0.33	0.44	37.31	0.26	0.68
	1ghc	1244220	5.02	0.46	0.54	51.33	0.31	0.79
	1f53	1466600	4.73	0.32	0.53	42.48	0.30	0.89
	1e0a	1651190	4.47	0.47	0.52	60.57	0.32	0.97
	1ev0	1912380	5.48	0.79	0.61	91.38	0.41	1.06
	1dz7	2160050	7.62	0.86	0.72	106.19	0.43	1.15
	1ap0	2603130	6.75	1.22	0.67	144.31	0.51	1.34
Convergence 10^{-6}	1pmc	739431	4.39	0.67	0.52	61.99	0.32	0.60
	1e01	1010510	7.28	0.96	0.69	84.98	0.36	0.72
	1ghc	1244220	9.87	1.32	0.86	118.73	0.44	0.81
	1f53	1466600	12.68	1.47	1.05	129.50	0.47	0.92
	1e0a	1651190	9.75	1.8	0.85	162.12	0.50	1.01
	1ev0	1912380	13.18	2.42	1.05	219.29	0.61	1.11
	1dz7	2160050	15.72	2.79	1.20	249.95	0.65	1.21
	1ap0	2603130	18.78	3.59	1.43	325.58	0.80	1.39