GPU linear and nonlinear Poisson-Boltzmann solver module for DelPhi
José Colmenares 1, Jesús Ortiz 2 and Walter Rocchia 1 *
1 Drug Discovery and Development Department, Istituto Italiano di Tecnologia (IIT), via Morego, 30, 16163 Genova - Italy
2 Advanced Robotics Department, Istituto Italiano di Tecnologia (IIT), via Morego, 30, 16163 Genova - Italy

ABSTRACT
Summary: In this work we present a CUDA based GPU implementation of a Poisson-Boltzmann equation solver, in both the linear and nonlinear versions, using double precision. A Finite Difference scheme is adopted and made suitable for the GPU architecture. The resulting code was interfaced with the electrostatics software for biomolecules DelPhi, which is widely used in the Computational Biology community. The algorithm has been implemented using CUDA and tested over a few representative cases of biological interest. Details of the implementation and performance test results are illustrated. A speed-up of about 10 times was achieved both in the linear and nonlinear cases.


Contact: jose.colmenares@iit.it; walter.rocchia@iit.it

Supplementary information: Supplementary data are available at Bioinformatics online

1 INTRODUCTION
The Poisson-Boltzmann equation (PBE) is a widely used tool to estimate the electrostatic energy of molecular systems in ionic solution. Given the continuously increasing size of structural data for proteins and other macromolecules, and the need to deal with bigger and more complex structures, the availability of tested and reliable algorithms on the most recent and affordable computational architectures, such as GPUs, is highly desirable. In this context, we present an implementation on a GPU architecture of both a linear and a non-linear PBE solvers based on the Finite-Difference (FD) scheme. The first use of a FD approach to solve the PBE can be ascribed to Warrier and Watson (1982). Our implementation follows that adopted by the DelPhi solver and described in Nicholls and Honig (1991), which exploits the checkerboard structure of the FD discretization of the Laplace differential operator and adopts Successive Over Relaxation (SOR) to converge to the solution. To our knowledge, this is the first description of a GPU implementation of the nonlinear PBE, which follows the same approach used by DelPhi and detailed in Rocchia et al. (2001). Our implementation can be executed on any NVIDIA card with CUDA capabilities.

2 SERIAL NUMERICAL SOLUTION
The Poisson-Boltzmann equation rules the electrostatic potential of a system where free charges and dipoles react to fixed charges located on the solute. This can be thought of as an extension of the Debye-Huckel continuum electrostatic theory (see Debye and Huckel (1923)). It is successfully used in biophysics to estimate the electrostatic energy of biomolecular systems in ionic solution as shown by Grochowski and Trylska (2007). From the mathematical standpoint, the PBE is a second order, elliptic, nonlinear partial differential equation, which, in the case of a monovalent salt dissolved in the solution, takes the following form:

\[ \nabla \cdot [ε(x) \nabla \Phi(x)] = -\frac{1}{\varepsilon_0} \rho^{fixed}(x) + \kappa^2(x) \sinh(\Phi(x)) \]  

(1)

where \( \Phi \) is the electrostatic potential, \( \varepsilon \) is the local relative dielectric constant, \( \varepsilon_0 \) is the permittivity of the vacuum, and \( \kappa(x) \) is null inside the solute and it equals the reciprocal of the Debye length in the solution. This equation can be rewritten in a way that isolates the nonlinear dependence on the potential:

\[ \nabla \cdot [ε(x) \nabla \Phi(x)] - \kappa^2(x) \Phi(x) = -\frac{\rho^{fixed}(x)}{\varepsilon_0} + \kappa^2(x) (\sinh(\Phi) - \Phi) \]  

(2)

This form is particularly suitable for devising the nonlinear algorithm as an adaptation of the linear one, according to a perturbative approach. FD discretization of the PBE and application of the successive over-relaxation method leads to the following iteration stencil:

\[ \Phi_j^{(n+1)} = \omega \left( \sum_{i=1}^{n} \frac{6}{\varepsilon_i} \Phi_i^{(n)} + \frac{2}{\varepsilon_i} \frac{q_j}{\varepsilon_i} + \xi_j \right) + \left(1 - \omega\right) \Phi_j^{(n)} \]  

(3)

where \( q_j \) is the discretized fixed charge, \( h \) is the grid spacing, \( \xi \) accounts for the nonlinear correction, if present, and \( \omega \) is the over-relaxation factor. The best value for \( \omega \) can be obtained, in the linear case, from the highest eigenvalue of the iteration matrix as seen Stoer and Bulirsch (2002). This method is stable since the iteration matrix is diagonally dominant. In the nonlinear case over
relaxation can lead to divergence and an adaptive method to assign $\omega$ is used, as detailed in Rocchia et al. (2001). Dirichlet boundary conditions are usually adopted based on the analytical solution of the linearized PBE in spherical symmetry, a few considerations on possible alternatives can be found in Rocchia (2005).

3 GPU IMPLEMENTATION

The GPU implementation borrows from the algorithm originally given by Nicholls and Honig (1991). All the calculations related to the stencil are done on the GPU card (or equivalent device). Interestingly, the stencil in Eq. (3) shows a checkerboard (“odd and even”) structure, implying that updating a point requires only the knowledge about its nearest neighbors, which are of opposite parity. Therefore, the execution can be divided in two segments, alternating the update of odd and even points. Within each segment, the calculations are independent, so any further parallelization is trivial. The physical grid is therefore partitioned in two logical sub-grids, odd and even. Our GPU implementation further exploits this structure and loads alternately odd and even points to the “texture” memory of the device to optimize the memory access. Dedicated data structures separately address charged grid points, and those that are at the boundary between regions with different dielectric.

A thread starts from every grid point of the bottom $xy$ slice of each sub-grid and then proceeds along the $z$ direction. A single step along $z$ in a sub-grid corresponds to an increment of 2 in the physical grid. Nearby threads in a $xy$ slice of a sub-grid are gathered in blocks, and are given access to a common shared memory. Basically, a thread does a loading step, followed by an updating step and finally it moves to the upper slice in the same sub-grid. For example, before updating an odd point a thread loads to the shared memory the even point having its same index. Since all the threads of a block act in parallel, and thanks to a purposely designed indexing scheme, an odd thread block in one step loads simultaneously all the even grid points needed for the updating task with the exception of the neighboring points that pertain to the adjacent blocks and of those that are in the $z$ direction. We cope with the first problem by adding a suitable overlap between blocks borders. The “$z-1$” and the “$z+1$” points of the physical grid are not present in the shared memory. The number of these accesses is halved by saving each “$z+1$” point in a temporary variable, which plays the role of the “$z-1$” point once the thread has moved to the upper slice. In the Supplementary information a graphical description of the algorithm is provided, while a more detailed explanation in the case of the linear PBE is given in Colmenares et al. (2013). Similarly to the DelPhi approach, the nonlinearity is treated as an additive charge-like term, which affects only the grid points that are located in solution and which is gradually inserted during the solution. Whether or not to update the nonlinear term is decided at the CPU level, based on an heuristic approach as in the DelPhi code (Rocchia et al. (2001)).

4 RESULTS

The results below show the speed up between the serial code executed on a CPU with an AMD Opteron (1.4GHz) chip and a Tesla Kepler K20m. The solver was run on the Fatty Acid Amide Hydrolase (FAAH) protein with 8325 atoms. A monovalent salt concentration of 0.15M was used. The relative dielectric constant of the molecule was taken as 2.0 and that of the solution as 80.0. The Debye length is roughly 8Å. The speedup of the nonlinear algorithm is comparable to that of the linear one. In fact, the former benefits from a larger number of floating point operations but it suffers from a larger number of data transfers.

<p>| Table 1. FAAH protein - 8325 atoms - 297x297x297 grid points |
|---------------------------------|------------------|------------------|</p>
<table>
<thead>
<tr>
<th></th>
<th>Linear solver</th>
<th>Non-linear solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration</td>
<td>0.015s (0.18s, x10.60)</td>
<td>0.035s (0.44s, x12.57)</td>
</tr>
<tr>
<td>Total</td>
<td>10.250s (1m38s, x 9.61)</td>
<td>10.14s (1m37s, x 9.55)</td>
</tr>
</tbody>
</table>

5 FUNDING

This work was supported by the National Institutes of Health [1R01GM093937-01].

6 ACKNOWLEDGEMENTS

The Authors gratefully acknowledge the IIT platform Computation and Dr. Decherchi for help in linking the module to the DelPhi code.

REFERENCES


