CATEGORY: **COMPUTATIONAL PHYSICS**

POSTER **CY14**

CONTACT NAME Lorena Barba: labarba@bu.edu

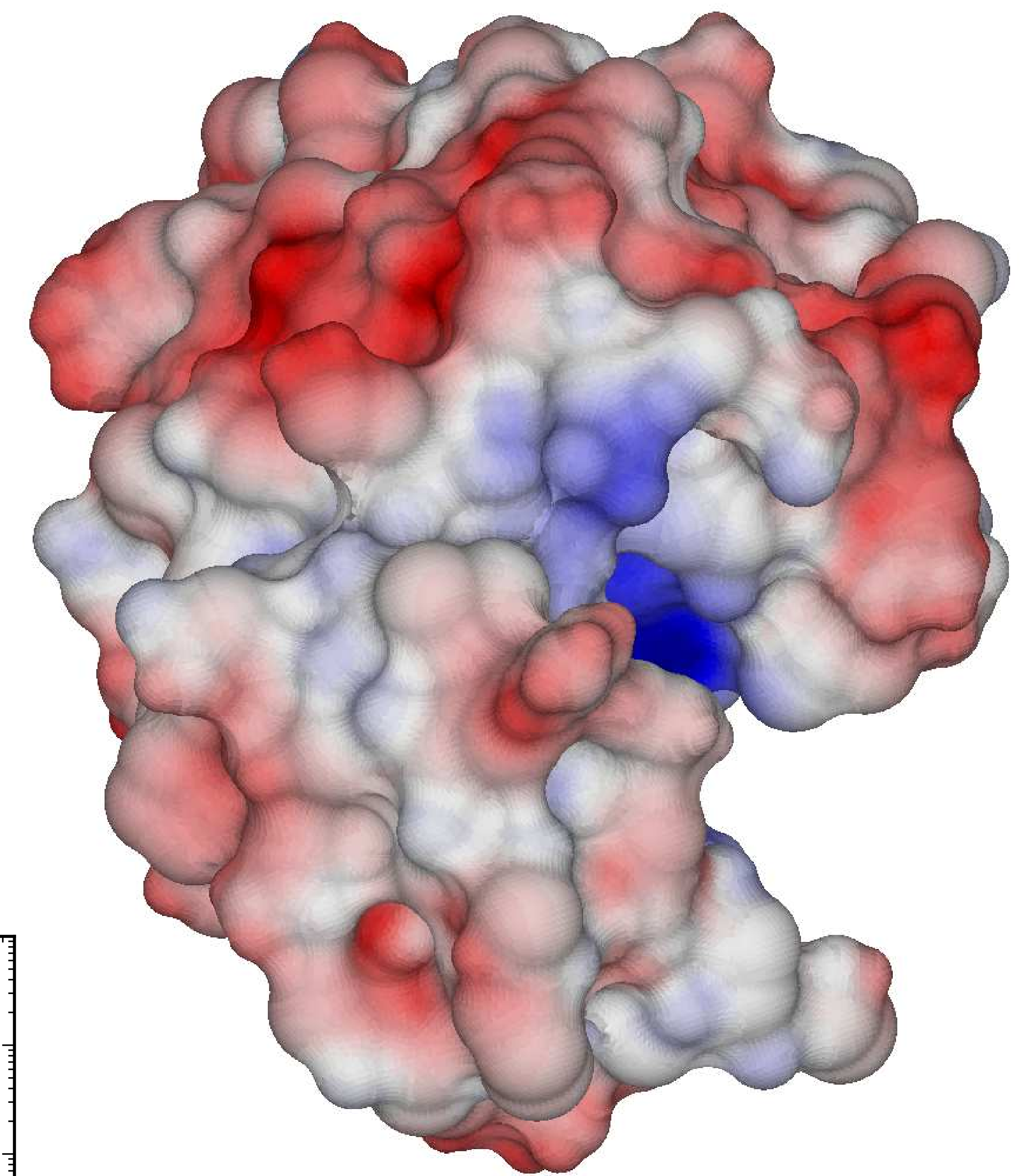
GPU TECHNOLOGY CONFERENCE

PyGBe: Python on the surface, GPUs at the heart. BEM solver for Electrostatics of Biomolecules

BOSTON UNIVERSITY

Christopher D Cooper, **Lorena A Barba**

Boston University



What's new? PyGBe

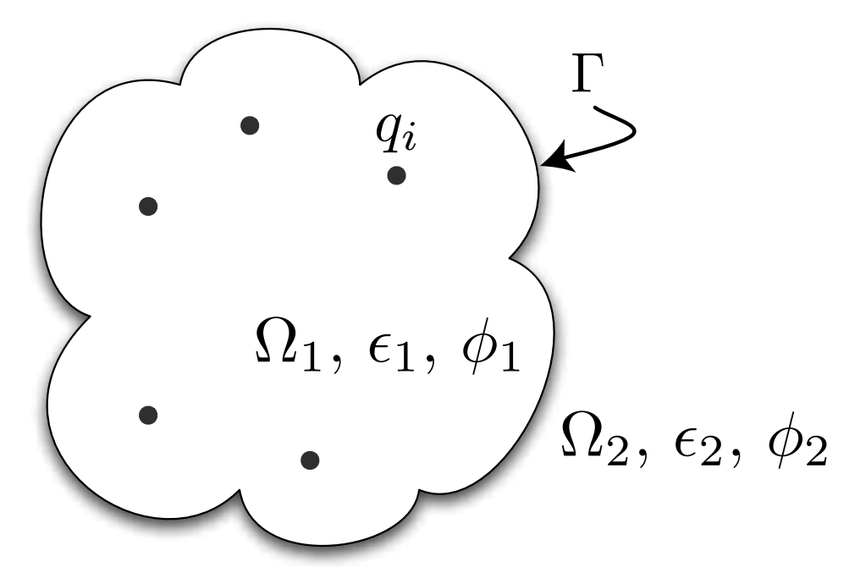
The PyGBe code (pronounced 'pig-bee') solves the linearized Poisson-Boltzmann equation using a boundary element method,BEM. The underlying dense systems are solved using a Krylov-subspace method, accelerated with a treecode to achieve O(N log N) complexity.

The code presents a Python environment for the user, while computing all kernels in CUDA (interfaced with PyCuda), for maximum ease of use, combined with high performance on GPUs.

Model: Implicit solvent

The interaction between charges in a biological system are screened by the presence of water with dissolved ions. Treating electrostatic effects via the Debye-Hücknel approximation with a Boltzmann distribution of charges, and combining with the Poisson equation for the potential field, results in the model known as "implicit solvent."

The solvent-excluded surface defines an interface between the region inside the protein and the solvent area. The Poisson equation with point charges applies inside the protein, with a low dielectric. The Poisson-Boltzmann equation (linearized) applies in the solvent area, with a high dielectric corresponding to water.


q_i
Γ
$\Omega_1,\ \epsilon_1,\ \phi_1$
$\Omega_2,\ \epsilon_2,\ \phi_2$

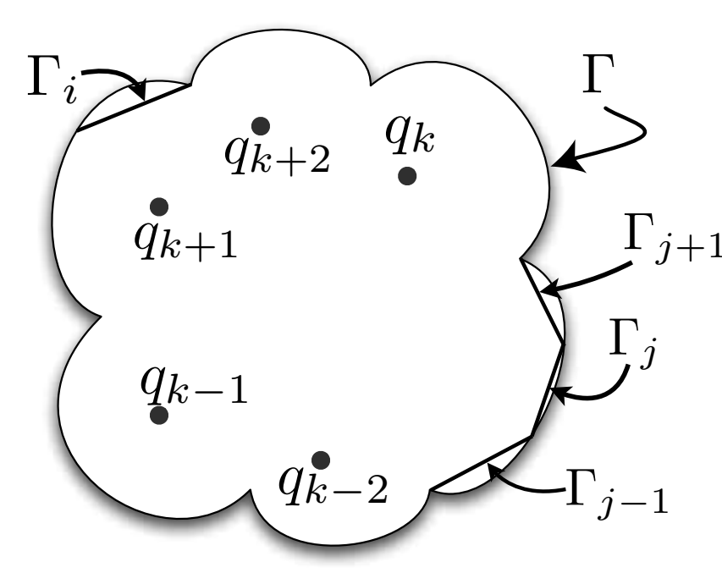
The differential system of equations is thus:

$$\nabla^2\phi_1(\mathbf{r}) = -\sum_i \frac{q_i}{\epsilon_1}\delta(\mathbf{r},\mathbf{r}_i) \quad \text{in solute } (\Omega_1)$$
$$\nabla^2\phi_2(\mathbf{r}) = \kappa^2\phi_2(\mathbf{r}) \quad \text{in solvent } (\Omega_2)$$
$$\phi_1 = \phi_2 \quad \text{on interface } \Gamma$$
$$\epsilon_1\frac{\partial\phi_1}{\partial\mathbf{n}} = \epsilon_2\frac{\partial\phi_2}{\partial\mathbf{n}}$$

Method: Boundary elements

The differential system of equations can be written in integral form, by taking convolution with the free-space Green's function of the Laplace and linearized Poisson-Boltzmann equations. We use the boundary element method, BEM, to solve the resulting system of integral equations.

We discretize the protein-solvent interface in panels, assume a distribution of the potential and its normal derivative over each panel, and then use colocation to generate a linear system.

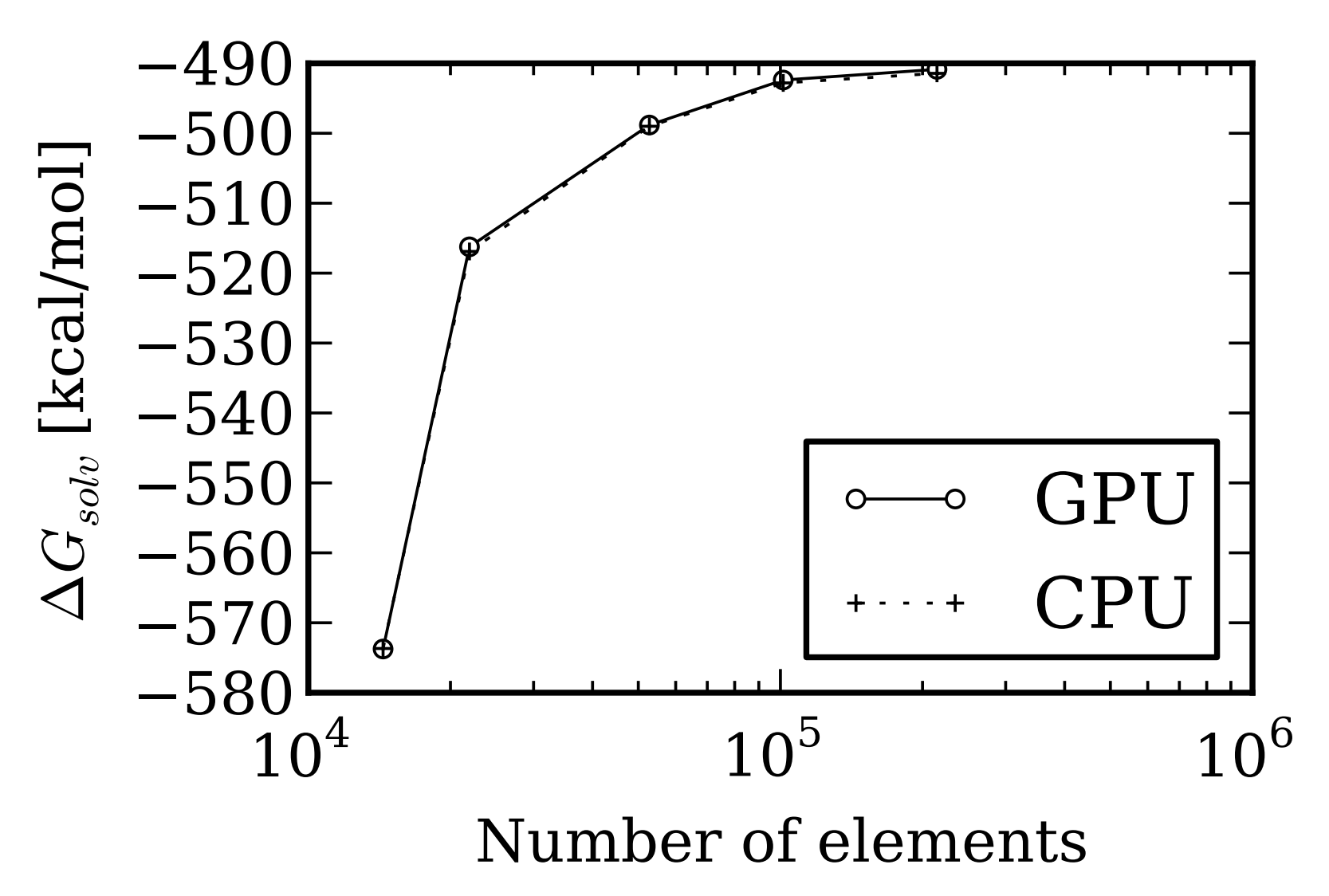


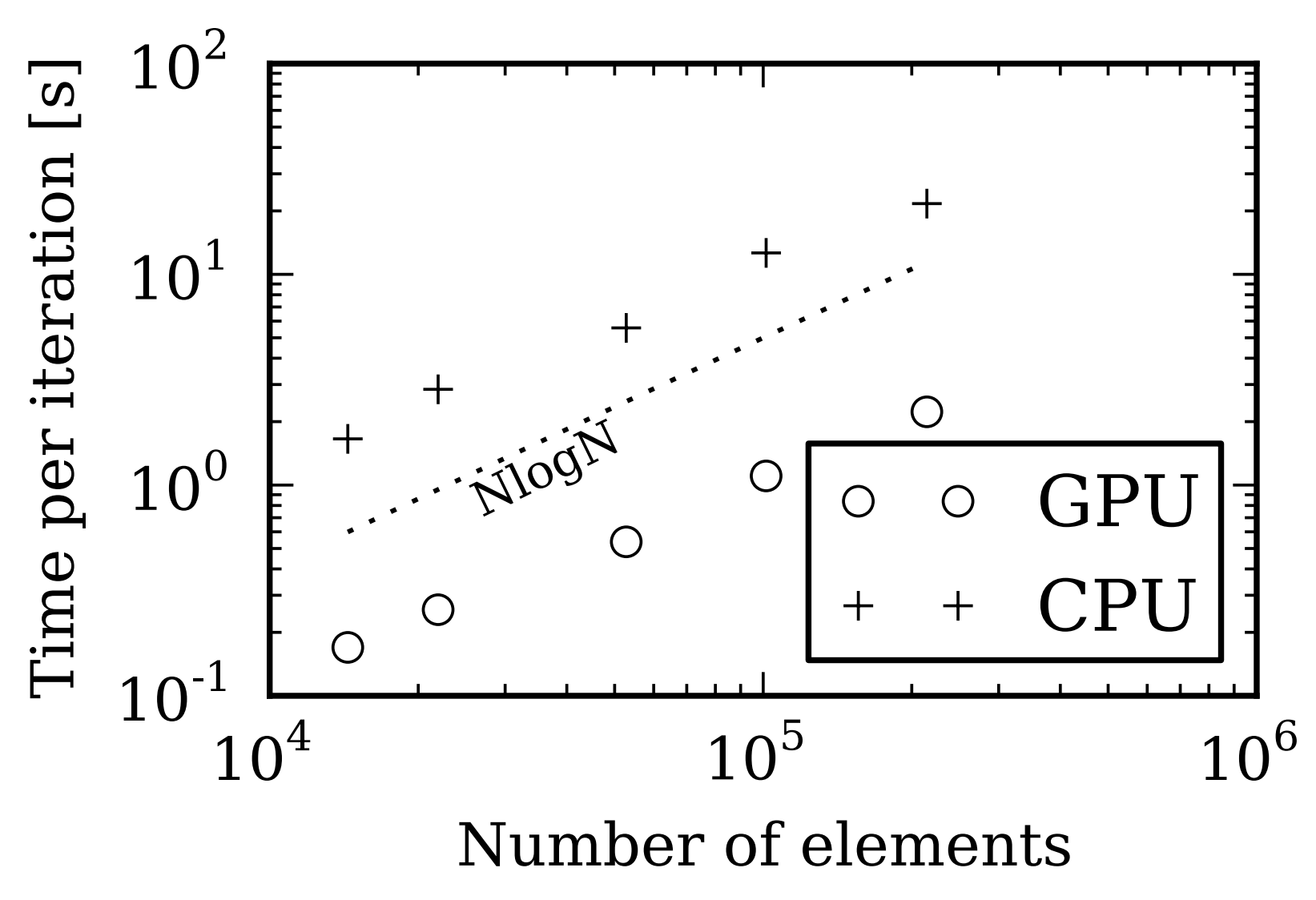
$$2\pi\phi_1(\mathbf{r}_i) - \sum_{j=1}^{N_p}\frac{\partial\phi_1}{\partial\mathbf{n}}(\mathbf{r}_j)\int_{\Gamma_j}\frac{1}{|\mathbf{r}_i-\mathbf{r}'|}\mathrm{d}\Gamma' + \sum_{j=1}^{N_p}\phi_1(\mathbf{r}_j)\int_{\Gamma_j}\frac{\partial}{\partial\mathbf{n}}\left[\frac{1}{|\mathbf{r}_i-\mathbf{r}'|}\right]\mathrm{d}\Gamma' = \sum_{k=0}^{N_c}\frac{q_k}{\epsilon_1}\frac{1}{|\mathbf{r}_i-\mathbf{r}_k|}$$
$$2\pi\phi_1(\mathbf{r}_i) + \sum_{j=1}^{N_p}\frac{\partial\phi_1}{\partial\mathbf{n}}(\mathbf{r}_j)\frac{\epsilon_1}{\epsilon_2}\int_{\Gamma_j}\frac{e^{-\kappa|\mathbf{r}_i-\mathbf{r}'|}}{|\mathbf{r}_i-\mathbf{r}'|}\mathrm{d}\Gamma' - \sum_{j=1}^{N_p}\phi_1(\mathbf{r}_j)\int_{\Gamma_j}\frac{\partial}{\partial\mathbf{n}}\left[\frac{e^{-\kappa|\mathbf{r}_i-\mathbf{r}'|}}{|\mathbf{r}_i-\mathbf{r}'|}\right]\mathrm{d}\Gamma' = 0.$$

Results: Lysozyme molecule

One of the most important quantities obtained using this model is the solvation free energy, which is widely used to study proteins, e.g., to obtain binding affinities.

In this example, we show the results of running PyGBe with a lysozyme molecule, using increasingly refined surface meshes. The numbers of panels vary from 14,000 to 215,000 (approximately).





Details: Validation technical report

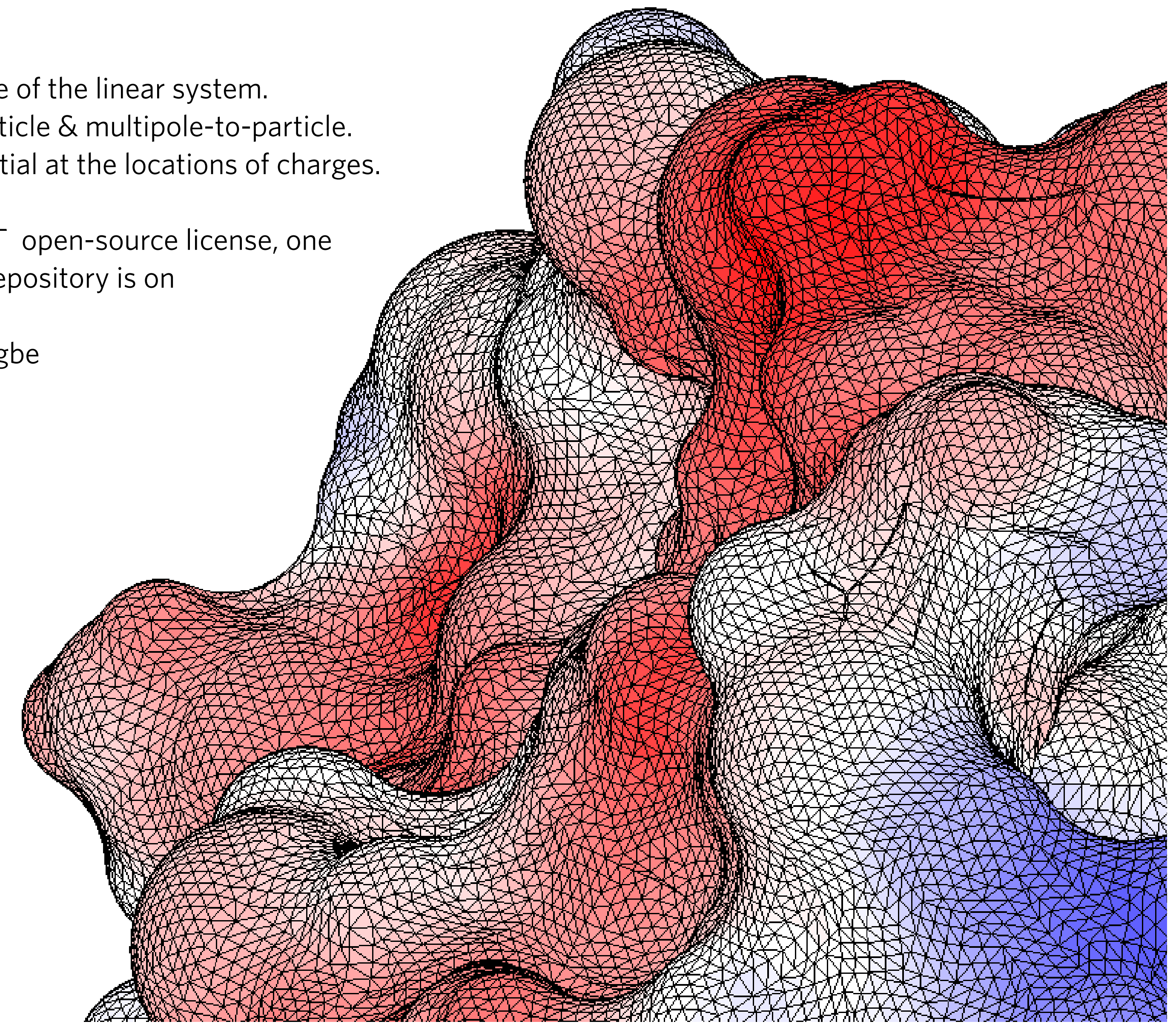
Validation of the PyGBe code for Poisson-Boltzmann equation with boundary element methods. Christopher Cooper, Lorena A. Barba. fig**share**. http://dx.doi.org/10.6084/m9.figshare.154331

Code: GPU-capable, Open-source under MIT license

All of the user-visible code in PyGBe is Python, and we interface to Cuda via PyCuda for the most computationally intensive parts of the algorithm. The parts that run on GPU are the following:

▶ Generation of the right-hand-side of the linear system.
▶ Treecode kernels: particle-to-particle & multipole-to-particle.
▶ Calculation of the reaction potential at the locations of charges.

The code is available under the MIT open-source license, one of the most permissive. The code repository is on Bitbucket:
https://bitbucket.org/cdcooper/pygbe

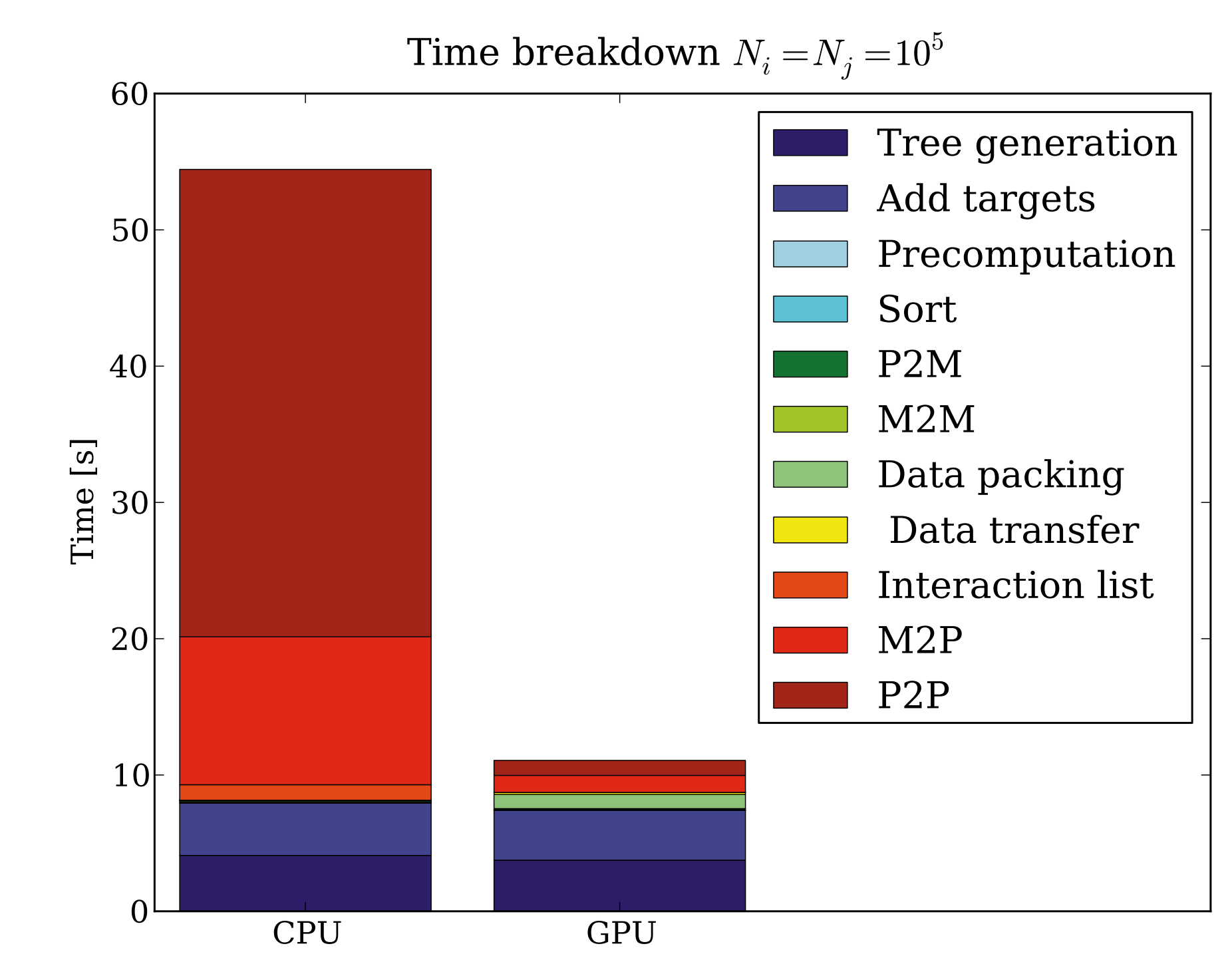


Performance: Treecode on GPU

The performance of PyGBe is dominated by the treecode evaluations for the matrix-vector multiplications in the linear solver. The timing breakdown below shows the two most computationally intensive parts of the treecode, as they were moved to the GPU.

The key to obtaining this performance on the GPU was the design of the data layout to obtain maximum parallelism and bandwidth usage. To improve runtime efficiency, some parts of PyGBe were written in C++ and wrapped in Python. Also, in the CPU version of the code, the multipole-to-particle and particle-to-particle interactions of the treecode were written in C++ and wrapped in Python.

All tests were run on a single Intel Xeon 2.67 GHz CPU core, or an NVIDIA Tesla C2075 GPU.



Want more? Papers and software are online

All the codes developed in our group are free (like free beer) and open source. To download them, follow the links from our group website:
http://barbagroup.bu.edu

Acknowledgement:
We are grateful to Jaydeep Bardhan for his help with this work.

GPU Technology Conference, March 2013