STUDENT PAPER: Solving the Many-Body Polarization Problem on GPUs: Application to MOFs

Brant Tudor
University of South Florida-Tampa
3720 Spectrum Blvd, IDRB 210
Tampa, FL 33620 USA
btudor@mail.usf.edu

Brian Space
University of South Florida-Tampa
3720 Spectrum Blvd, IDRB 210A
Tampa, FL 33620 USA
bspace@mail.usf.edu

ABSTRACT
Massively Parallel Monte Carlo, an in-house computer code available at http://code.google.com/p/mpmc/, has been successfully utilized to simulate interactions between gas phase sorbates and various metal-organic materials. In this regard, calculations involving polarizability were found to be critical, and computationally expensive. Although GPGPU routines have increased the speed of these calculations immensely, in its original state, the program was only able to leverage a GPU's power on small systems. In order to study larger and evermore complex systems, the program model was modified such that limitations related to system size were relaxed while performance was either increased or maintained. In this project, parallel programming techniques learned from the Blue Waters Undergraduate Petascale Education Program were employed to increase the efficiency and expand the utility of this code.

General Terms
Algorithms, Design

Keywords
Blue Waters Undergraduate Petascale Education Program, CUDA, GPGPU, MOF, Parallel Programming, Polarization

1. INTRODUCTION
Metal-Organic Frameworks (MOFs) are highly porous, crystalline materials characterized by inorganic clusters, or nodes, connected via organic linkers. The linking molecules are roughly linear and force a relatively high level of space between the inorganic nodes. Consequently, these materials are remarkable in their high surface areas, which suggest great opportunities for applications such as gas storage via physisorptive processes. The ability to selectively control pore size, polarity and placement of functional groups on the linkers provides further opportunity for the engineering of materials suited for specific separations or catalytic activity. In order to rationally design such materials, it is desirable to understand how they work on a molecular level. For example, it would be useful to know how exactly how and why each node, linker or functional group's place within the MOF improves or retards the process of interest. Additionally, the identification of non-existent MOFs with useful properties, or the identification of useful, overlooked properties on existent MOFs, is another widely held aim.

To that end, accurate, efficient simulation of MOF materials is an area of active research. A program developed in-house, Massively Parallel Monte Carlo (MPMC), has demonstrated its effectiveness in MOF-centric and related simulations [1-3]. This program has been successfully employed to generate sorption isotherms for MOFs with high fidelity to experiment [1]. Crucial to the accuracy of such isotherms is a careful accounting of the polarization energy of the MOF, and, unfortunately, this task has proven to be a computational bottleneck [4]. Early versions of MPMC had a limited ability to utilize GPGPUs to perform these calculations. Although a significant performance boost was realized, the system size was constrained by the amount of shared memory on the card, effectively limiting the simulations to approximately 2000 atoms on the available hardware (a number only suitable for simulation of smaller MOF systems).

2. BACKGROUND
Polarization calculations in MPMC are conducted using the Thole-Applequist model [5, 6]. This model assigns each atomic site a point dipole whose interactions with all the other dipoles of the system are dictated by many-body polarization equations. Using a set of training molecules, a 3x3 polarizability tensor is calculated for each site. Then, in a static electric field, each dipole, $\vec{\mu}$, is thus represented by the product of the calculated polarizability tensor, α, and the field vector at that point, $\vec{E}^{stat}$:

$$\vec{\mu} = \alpha \vec{E}^{stat} \quad \textbf{(1)}$$

In this model, the dipole for a molecule is then treated as a collection of *N* atomic-point dipoles, which are summed to give the net dipole for the set [4]:

$$\vec{\mu}_{mol} = \sum_{i}^{N} \vec{\mu}_i = \sum_{i}^{N} \alpha_i \vec{E}_i^{stat} \quad \textbf{(2)}$$

Here, $\vec{\mu}_i$ is the dipole for an individual site, α_i is the polarizability tensor for the site, and $\vec{E}_i^{stat}$ is the electrostatic field vector at that point, for each site, *i*, in the molecule. The Thole-Applequist system is then treated as a collection of *N* dipoles and a dipole field tensor, $T_{ij}^{\alpha\beta}$. The elements of *T* are the complete set of tensors describing every induced dipole-dipole interaction in the system [4]. The product of the dipole field tensor, *T*, and a system dipole results in the many-body induced-dipole contribution to the electric field, $\vec{E}^{ind}$, at the dipole site. The dipole field tensor was designed to contain the entire induction contribution, allowing the assignment of a scalar point polarizability, α° for each site, instead of the polarizability tensor [4]:

$$\alpha_i \vec{E}_i^{stat} = \alpha_i^{\circ}\left(\vec{E}_i^{stat} + \vec{E}_i^{ind}\right) \quad \textbf{(3)}$$

$$= \alpha_i^{\circ}\left(\vec{E}_i^{stat} - T_{ij}^{\alpha\beta}\vec{\mu}_j\right) \quad \textbf{(4)}$$

If $\vec{\boldsymbol{\mu}}$ is treated as a vector, each entry of which is one of the system dipoles (each of those a vector), equation **(5)** is the result. A similar "super vector" is formed by treating vectors of the static electric field (at a point in space corresponding to each of the dipoles) in an identical fashion, the result of which is equation **(6)**.

$$\vec{\boldsymbol{\mu}} = \begin{pmatrix} \vec{\mu}_1 \\ \vec{\mu}_2 \\ \vdots \\ \vec{\mu}_N \end{pmatrix} \quad \textbf{(5)}$$

$$\vec{\boldsymbol{E}}^{stat} = \begin{pmatrix} \vec{E}_1^{stat} \\ \vec{E}_2^{stat} \\ \vdots \\ \vec{E}_N^{stat} \end{pmatrix} \quad \textbf{(6)}$$

Additionally, if matrices A and B are defined as

$$A = \left[(\alpha^{\circ})^{-1} + T_{ij}^{\alpha\beta}\right] \quad \textbf{(7)}$$

$$B = A^{-1} \quad \textbf{(8)}$$

the problem is reduced to two compact matrix equations, **(9)** and **(10)**. Matrix *A* is thus constructed such that each element is the 3x3 matrix T_{ij}. Each element of matrix *B* is also a 3x3 matrix—the site polarizability tensor characterizing each site's response to an electric field [4].

$$A\vec{\boldsymbol{\mu}} = \vec{\boldsymbol{E}}^{stat} \quad \textbf{(9)}$$

$$\vec{\boldsymbol{\mu}} = B\vec{\boldsymbol{E}}^{stat} \quad \textbf{(10)}$$

The system dipoles can therefore be found by inverting matrix *A* (giving *B*) and solving equation **(10)** directly. However, the size of matrices required to model typical MOF systems renders the computation required for matrix inversion impractical. MPMC solves these equations by guessing at the value of each point dipole and solving equation **(9)** iteratively.

3. MPMC

3.1 Limitations of the Initial Solution

MPMC typically solves for the system dipoles iteratively [7]. The initial guess for each dipole is simply the product of the scalar point polarizability and the electrostatic field vector at that point. Each dipole is considered sequentially, and is marginally corrected according to the induced contribution calculated using all the other dipoles in the system. This process is repeated for each dipole (thus concluding a single iteration), and the whole process is then repeated for the entire system until convergence to within a specified tolerance is realized. MPMC also has the ability to solve this problem through matrix inversion, but, as previously mentioned, this method is only viable for small systems.

Additionally, the original version of MPMC included support for finding the system dipoles using a General Purpose Graphics Processing Unit (GPGPU) device. This algorithm performed the iterative process previously described with only a few key differences. First, each step of the calculation updated every dipole in the system, whereas the serial algorithm incorporated the Gauss-Seidel numerical iterative technique. In this method, newly calculated dipole data replaces old dipole data as soon as it becomes available. The new values are then used in calculating all the remaining dipoles in the system. This technique can significantly decrease convergence times, but since, in the parallel algorithm, all the newly calculated dipoles become available simultaneously, the Gauss-Seidel technique was not implemented.

A test for convergence of the GPGPU polarization calculation was not implemented in the original version of MPMC. Hence, the computation would run for a preset number of iterations and results were delivered without any way of estimating their accuracy.

Finally, simulations utilizing the GPGPU device were limited to 2048 atoms due to the manner in which MPMC employed the GPU's shared memory system. This constraint renders the GPGPU algorithm useful only in simulations of relatively small system size. A MOF simulator should ideally be able to handle system sizes of 10,000+ atoms in order to be useful for several MOFs of current and future interest to investigators.

3.2 Updated Program Model

Several changes to improve and expand the functionality of MPMC were realized.

3.2.1 Maximum System Size Expansion

The 2048 atom cap imposed on simulations was the first limitation addressed during the course of this project. In the updated program model, each GPU thread was assigned a single system dipole. Each thread calculates its dipole's interaction with every other dipole in the system, and sums these interactions to arrive at the dipole vector to be used in the next iteration. Since

every thread needs access to the vector data of every other dipole, it only makes sense to load the dipole information into shared memory so that each thread in the block can access it. This precludes the need for each of these threads to access the data individually from global memory (a relatively time consuming process to be avoided when possible) [8]. However, since shared memory is fairly limited, it is impossible to fit all the dipole data in this memory system simultaneously (for moderate to large-sized systems). This situation is amenable to a tiled model of data handling such that the complete set of dipole information resides in global memory and is moved in and out of shared memory as needed, one block at a time **(FIGURE 1)**. Organizing the data in this manner shifts the limitation of system size from shared memory to one of global memory and/or maximum grid size. Obviously, the global memory of the GPGPU device must be large enough to hold dipole data for the entire system. However, since each thread is responsible for a dipole and each thread block executes a limited number of threads, the maximum grid size (which dictates the total number of blocks) is ultimately responsible for determining the maximum number of threads [8], and therefore the maximum number of dipoles (i.e. atoms). Fortunately, on current hardware, the system sizes imposed by these limitations number in the millions of atoms, thus transforming MPMC's prohibiting considerations from those of system size to one of computational duration.

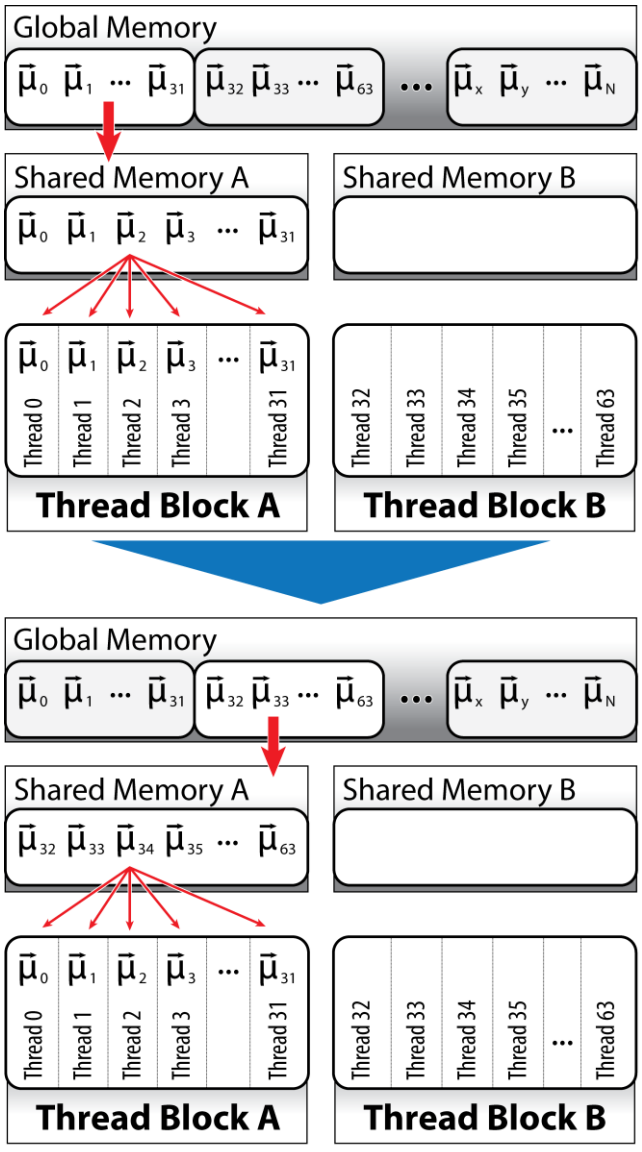

FIGURE 1: In this example, Thread Block A (TBA) is executing, while B sits idle. In the top panel, TBA pulls global vector data for the first block of dipoles. Once all threads have processed this information, it pulls data for the second block, as illustrated in the bottom panel. This continues until TBA has processed all dipole-dipole interactions for which it is responsible. Other thread blocks will do likewise (either concurrently or sequentially) until all dipole-dipole interactions in the system have been calculated.

3.2.2 Gauss-Seidel in Parallel

The original GPGPU algorithm did not attempt to implement the Gauss-Seidel iterative method of using newly calculated dipole information in the calculations for later dipoles. From outside the GPU kernel, all the dipoles appear to be updated simultaneously, so a treatment of this nature simply is not possible. However, from inside the kernel, once a thread block has completed, it is possible for each thread to overwrite its value in global memory with its newly calculated value **(FIGURE 2)**. This treatment will allow any subsequent calculations to use the latest available information for their own computations. This technique updates a block of dipoles at a time, and as such effects a coarse-grained version of the Gauss-Seidel method. Typically, several thread blocks will be executing concurrently and these blocks will not be able to take advantage each other's updates, thus it is expected that this modification will only become significant on larger system sizes where only a small portion of the total number of the required thread blocks can run concurrently.

3.2.3 Convergence Verification

Prior to this work, MPMC set a fixed number of iterations for the GPGPU algorithm and the level of convergence obtained after this number of iterations was what any dependent calculations were forced to use. After extensive testing, it became apparent that, in many cases, the set number of iterations was sufficient for a high level of convergence. However, in some cases it was not. Worse, the program was unable to tell if a set of dipoles converged, so the user received no warning that their calculation may be suspect.

From inside the kernel, before each thread updates its data in global memory (for Gauss-Seidel), modifications were made such that each thread now copies its original dipole data into a local register. The difference between the old dipole data and the newly calculated dipole is squared and stored in an output array which can then be examined by the function that launched the kernel. Outside the kernel, in the calling function, the transfer of the squared-difference data from the GPGPU device to the host machine can take a significant amount of time compared to a single iteration. In some cases, the transfer duration can take *longer* than a single iteration, more than doubling the length of the total calculation. To mitigate this effect, the squared dipole differences are only downloaded and examined after every tenth iteration.

3.2.4 Energy Calculations in Parallel

The Monte Carlo portion of MPMC aims to identify low-energy system configurations. As such, the purpose of calculating the

system dipoles is to quantify an energy contribution from polarization effects. The time required to calculate this energy tends to vary widely. Kinetic and coulombic energies are also considered and the combined time required for these calculations, depending on the duration of the polarization energy, can be mildly to highly significant by comparison. Finally, MPMC can calculate an energy contribution due to van der Waals effects. This computation relies on matrix diagonalization and, when utilized, invariably takes the longest of any of the calculations. Using the Open Multi-Processing API (OpenMP), MPMC is now able to split into three concurrent threads of execution, one of which is responsible for both the kinetic and coulombic energy calculation, another of which is responsible for the polarization energy calculation, and the last of which is responsible for calculating the van der Waals energy contribution.

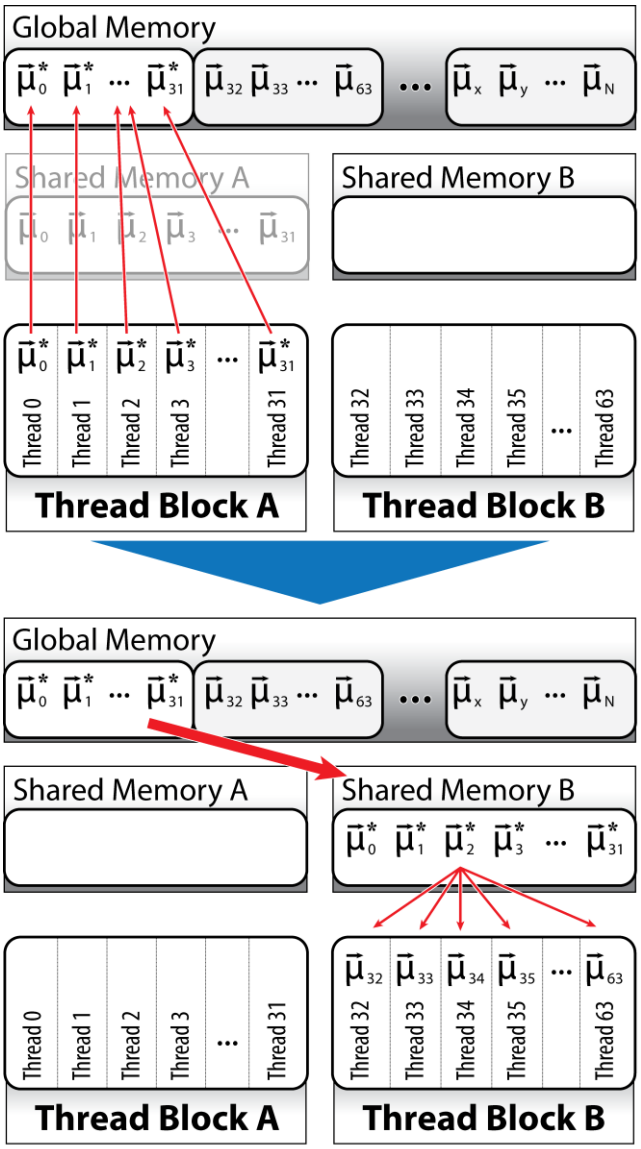

FIGURE 2: A block-wise, parallel Gauss-Seidel iterative method. In the top panel, Thread Block A (TBA) has finished executing and immediately updates the global data with the newly calculated values for its dipoles (updated values are denoted with an asterisk). In the lower panel, when Thread Block B (or any subsequent thread block) executes, it is able to leverage the latest calculated values for TBA's dipole data in its own calculations, speeding convergence.

3.2.5 Van der Waals Calculations Using MAGMA

The final modification made to MPMC was to utilize the Matrix Algebra on GPU and Multi-core Architectures library (MAGMA) in order to compute the van der Waals energy contribution. The original routine calls for a matrix diagonalization via the LAPACK routine `dsyev_()`. It was a simple matter to construct an alternate routine, to be used in the event that a GPGPU device was detected. The two routines were practically identical in all respects except that the new one makes a call to MAGMA's `magma_dsyevd()` instead of to the equivalent LAPACK function.

4. RESULTS

The updated version of GPGPU portion of MPMC is able to reproduce the results of the original with perfect fidelity for system sizes less than or equal to 2048 atoms, in approximately the same amount of time. For larger systems, no direct comparison can be made since the older version is unable to produce a result, although the computation is performed six to eight times faster on the GPU than the CPU. Comparing GPU results against data obtained through matrix inversion, presumed exact, reveals that calculations on typical systems are within five percent error.

Performance increases due to the multi-threaded, OpenMP handling of the energy calculations, though present, is difficult to quantify. The combined calculation time for the kinetic and coulombic contributions represents roughly 10 to 50 percent of the total calculation time, and this figure varies widely from iteration to iteration. Effectively, the total calculation time is now reduced to the duration of whichever calculation takes the longest (coulombic/kinetic, polarization, or van der Waals), plus a small penalty for the overhead required to establish the threads. On test systems, the net speedup of the multithreaded treatment was typically around 20 percent.

The use of the MAGMA routine in the calculation of the van der Waals energies is able to exactly reproduce the LAPACK result. However, the calculations are completed in approximately half the time.

5. FUTURE WORK

The accuracy of the GPGPU polarization calculation is lower than ideal, on the order of three percent error. Different techniques are being tried in order to increase the accuracy of these results, as well as to decrease convergence times. Additionally, the version of MPMC under discussion was designed to simulate a crystalline material and a single species of sorbate. Currently, efforts are underway to modify the program such that it can simulate multiple sorbate species simultaneously introduced into the material.

6. REFLECTIONS

The summer portion of the Blue Waters Undergraduate Petascale Education Program (BW-UPEP) provided training and instruction at the Urbana-Champagne campus of the University of Illinois. During this program, various technologies and techniques for scientific coding on parallel and supercomputer architectures were

discussed and elucidated. Of particular interest to this project was the training on GPGPU programming through NVIDIA Compute Unified Device Architecture (CUDA) as well as the Open Multi-processing API (OpenMP) maintained by the OpenMP Architecture Review Board. The workshop introduced students to various algorithmic models, concepts and issues that were particularly useful to the current project, such as deconstruction of large repetitious problems into loosely coupled blocks appropriate for efficient handling by GPGPU devices, concurrent processing of dissimilar tasks through multi-threading, and, perhaps most importantly, how to leverage both techniques within a single program. Resources for learning any one of the technologies abound, but an area where the program excelled was instruction on how to effectively harness all these technologies to work together within a single project.

Through the work started during the BW-UPEP program, I was able to foster a deep understanding of the architecture sitting underneath the hood of various high performance computing systems. Whereas before, I had only superficial experience with supercomputers, I currently develop scientific software and perform research computation on my own university's local research computing cluster, as well as on many of the computing systems made available through the NSF's Extreme Science and Engineering Discovery Environment (XSEDE) project. Speaking from personal experience, I believe undergraduates who have an interest in scientific computing stand to gain a considerable amount of confidence, experience and expertise by attending such a program as the BW-UPEP. The abundant knowledge and support available during the development of various pedagogical codes, as well as the guidance received regarding submission of these jobs to actual work environments (research computing clusters of universities with ties to the program), made it much easier to "leave the nest" and create and submit my own computational jobs to world-class research computing facilities throughout the academic world.

I am currently in the early stages of my Doctoral program in theoretical and computational chemistry at the University of South Florida, and the skills and knowledge acquired through the BW-UPEP program have definitely helped to jumpstart my career therein. The time saved by not having to start from scratch in learning the basics of HPC coding (*or* the ins-and-outs of interaction with research computing environments) may have shaved a semester or more off my time in graduate school. In classes oriented around high performance computation and scientific coding, I find that while my peers spend much of their time trying to frame the posed problems in a manner suitable for parallel computation, the practical experience gained through the Blue Waters program often allows me to skip this step and immediately begin to identify opportunities to make the code more efficient in terms of the low-level hardware, e.g. efficient use of cache, shared memory systems, coalesced memory accesses, etc. My association with the BW-UPEP has proven to be an invaluable advantage in this regard and my ardent gratitude toward the program remains steadfast.

7. AKNOWLEDGMENTS

Funding for this work was provided by the National Science Foundation's Office of CyberInfrastructure through the Blue Waters Undergraduate Petascale Education Program.

The authors would like to acknowledge the use of the services provided by Research Computing at the University of South Florida. Simulations and development were performed on the University of South Florida's Research Computing Center, CIRCE, with additional support received from the Space Foundation (Basic and Applied Research).

8. REFERENCES

[1] J. L. Belof, *et al.*, "A Predictive Model of Hydrogen Sorption for Metal−Organic Materials," *The Journal of Physical Chemistry C*, vol. 113, pp. 9316-9320, 2009.

[2] *An Accurate and Transferable Intermolecular Diatomic Hydrogen Potential for Condensed Phase Simulation*, 4, 2008.

[3] J. L. Belof, *et al.*, "On the Mechanism of Hydrogen Storage in a Metal−Organic Framework Material," *Journal of the American Chemical Society*, vol. 129, pp. 15202-15210, 2007.

[4] J. L. Belof, *Theory and simulation of metal-organic materials and biomolecules*. Tampa: Theses and Dissertations. Paper 1851. http://scholarcommons.usf.edu/etd/1851, 2009.

[5] J. Applequist, *et al.*, "An Atom Dipole Interaction Model for Molecular Polarizability. Application to Polyatomic Molecules and Determination of Atom Polarizabilities," *Journal of the American Chemical Society*, vol. 94, pp. 2952-2960, 1972.

[6] B. T. Thole, "Molecular polarizabilities calculated with a modified dipole interaction," *Chemical Physics*, vol. 59, pp. 341-350, 1981.

[7] J. L. Belof, "Massively Parallel Monte Carlo (MPMC)," http://code.google.com/p/mpmc/, 2007.

[8] (2012). *NVIDIA CUDA C Programming Guide Version 4.2*.