JCTC Journal of Chemical Theory and Computation

ARTICLE

pubs.acs.org/JCTC

Accelerating All-Atom Normal Mode Analysis with Graphics Processing Unit

Li Liu,[†,‡] Xiaofeng Liu,[†] Jiayu Gong,[‡] Hualiang Jiang,[†,§] and Honglin Li*[,†]

[†]State Key Laboratory of Bioreactor Engineering, Shanghai Key Laboratory of Chemical Biology, School of Pharmacy, East China University of Science and Technology, Shanghai 200237, China

[‡]School of Information Science and Engineering, East China University of Science and Technology, Shanghai 200237, China

[§]Drug Discovery and Design Center, Shanghai Institute of Materia Medica, Chinese Academy of Sciences, Shanghai 201203, China

ABSTRACT: All-atom normal mode analysis (NMA) is an efficient way to predict the collective motions in a given macromolecule, which is essential for the understanding of protein biological function and drug design. However, the calculations are limited in time scale mainly because the required diagonalization of the Hessian matrix by Householder-QR transformation is a computationally exhausting task. In this paper, we demonstrate the parallel computing power of the graphics processing unit (GPU) in NMA by mapping Householder-QR transformation onto GPU using Compute Unified Device Architecture (CUDA). The results revealed that the GPU-accelerated all-atom NMA could reduce the runtime of diagonalization significantly and achieved over 20× speedup over CPU-based NMA. In addition, we analyzed the influence of precision on both the performance and the accuracy of GPU. Although the performance of GPU with double precision is weaker than that with single precision in theory, more accurate results and an acceptable speedup of double precision were obtained in our approach by reducing the data transfer time to a minimum. Finally, the inherent drawbacks of GPU and the corresponding solution to deal with the limitation in computational scale are also discussed in this study.

INTRODUCTION

Predicting collective structural changes in biological macromolecules is essential in the understanding of protein function and drug design.[1] In addition to the experimental approaches such as X-ray crystallography,[2] NMR spectroscopy,[3] and single-molecule biophysical techniques,[4] some theoretical methods including molecular dynamics (MD) simulations[5] and normal mode analysis (NMA)[6] are available to explore the protein motions. Compared to MD simulation, NMA eliminates time-integration and explicit solvent degrees of freedom with a considerable computational advantage.[7] This enables NMA to be used for much larger systems[8] including viral capsids,[9] molecular motors,[10] and the ribosome.[11] Except for prediction of macromolecules' dynamic motions, another important application of NMA is to calculate the entropy change in the well-known MM/PBSA method, which is a relatively accurate but computationally expensive approach to calculate the binding free energy of the protein−protein or protein−ligand complex and is deemed more accurate than conventional scoring functions for molecular docking.[12−14]

Classical all-atom NMA assumes that protein motions, including large-scale/low-frequency and small-scale/high-frequency motions, can be described by a quasi-harmonic approximation around a local minimum on the protein energy surface,[15] and the low frequency modes can well describe large conformational changes relevant to protein function that are observed experimentally, such as the hinge opening and closing motions.[16] The approach of NMA is to diagonalize the Hessian matrix, of which $3N \times 3N$ (N is the number of atoms) elements represent the second derivative of the potential energy function along the Cartesian coordinates.[17,18] However, the application of all-atom NMA has been limited due to intensive memory consumption ($O(N^2)$) and cubic CPU time complexity ($O(N^3)$) in the case of a protein with N atoms and the resulting dramatic increase in sampling time. Even though the storage of a Hessian matrix has become less of an obstacle due to the introduction of sparse matrix techniques, the diagonalization of the all-atom matrix is still a challenge to date. Therefore, all-atom NMA is normally applied to protein systems containing at most a few hundred residues. On the other hand, the entropy change calculation is often ignored in the application of the large-scale virtual screening with MM/PBSA approach because of the repeated and extremely time-consuming NMA calculation for all of the protein−ligand complexes during the process of molecular docking,[19] which undermines the accuracy of the calculated binding free energies as well as the ranks of the hit compounds.

To solve the time-consumption problem, several coarse-grain methods, such as block normal mode (BNM)[20] and elastic network model (ENM),[21] have been put forward to reduce the size of Hessian matrix and the computational time. The BNM starts from an energy-minimized system described by an all-atom force field but considers each residue as a rigid block[20] with six translation−rotation degrees of freedom, which reduces the size of the Hessian matrix to $6N \times 6N$ in the case of a protein with N residues.[22,23] The ENM approximation regards the protein as an elastic network,[24−26] and only the Cα atoms of the individual residues are considered as nodes and connected by uniform springs.[21,22] On the basis of ENM, the size of the Hessian matrix

Received: December 20, 2010
Published: May 13, 2011

is reduced to only $3N \times 3N$ in the case of a protein with N residues. In spite of outstanding computational efficiency, the accuracy of the coarse-grained NMA result is undermined because the modes with high frequencies are eliminated.[22] Therefore, more efforts have been made to improve the computational efficiency for all-atom NMA to preserve the explicit representation of atomic degrees of freedom through reducing the time of diagonalizing the all-atom Hessian matrix.[7]

It is well-known that conventional supercomputers are typically large in size and very expensive in terms of manufacturing and maintenance. Fortunately, over the past few years, the graphics processor unit (GPU), which has tremendous raw computing power, has been rapidly evolving from a fixed-function pipeline into a programmable processor.[27] A contemporary GPU can achieve as much as 1000 GFLOPS peak performance[28] and a 1 order of magnitude speedup for single-precision arithmetic over corresponding CPU code. Due to the advantages of speed, cost, and accessibility, GPUs have been applied in many scientific computation fields, such as electrostatics,[29] molecular dynamics,[30] quantum chemistry,[31] and so on. The GPU is suitable for handling problems involving large data sets and computationally intensive calculations[32] and has the potential to revolutionize computational biology by changing the conventional batch-mode computational jobs into interactive tasks.[33]

The Householder-QR transformation algorithm is generally used to find the eigenvalues and eigenvectors of a real matrix. This algorithm has been implemented in successive parallel versions of popular linear algebra libraries such as Linear Algebra Package (LAPACK), Parallel Linear Algebra for Scalable Multicore Architectures (PLASMA), and so on. The Householder-QR transformation implemented in LAPACK leverages the idea of blocking to limit the amount of bus traffic in favor of a high reuse of the data that is present in the higher level memories, which are also the fastest ones.[34] In PLASMA, the algorithm can be represented as a sequence of small tasks that operate on square blocks of data. These tasks can be dynamically scheduled for execution based on the dependencies among them and on the availability of computational resources.[35] In this study, the Householder-QR transformation algorithm for the Hessian matrix in GROMACS was reimplemented to map onto a GPU which is amenable to parallelization on single-instruction multiple-data (SIMD) architectures using the CUDA programming toolkit.[36] CUDA adopts an extension of the C/C++ programming language, which consists of a set of keywords and a runtime library to allow the execution of parallel code and memory control on the GPU. Through evaluating the performance of GPU-accelerated NMA on some benchmark molecules with various numbers of atoms and comparing the runtimes against the single-core CPU-based NMA in GROMACS, the GPU-accelerated NMA outperforms the CPU-based one with a higher computational efficiency and achieves over 20× speedup with competitive accuracy, which highlights the GPU's potential to perform all-atom NMA on a desktop workstation in a fast and accurate way.

MATERIALS AND METHODS

Graphics Processing Unit Overview. A typical parallel program can be generalized in three parts in CUDA: (1) apply and release the memory storage; (2) execute the parallel parts as so-called kernel functions; (3) transfer data between host memory and device memory.[37] There is a scalable number of streaming multiprocessors (SMs) on a GPU, each containing eight streaming processor (SP) cores. The SMs execute a data-parallel problem which is decomposed into independent work items called threads in CUDA.

Threads are executed together in cooperative work groups called blocks, which are grouped into grids. The computation of a grid corresponds to exactly one GPU kernel invocation.[28,38] To achieve a decent performance increase compared to a CPU calculation, it is important to take three aspects into account. First, in order to fully occupy the device and provide sufficiently many threads of execution to hide various sources of latency, it requires a huge data parallel workload.[33] Second, memory access is much slower than arithmetic calculations on a GPU, and the data transfer between the host and the device has to be minimized. Third, to maximize the number of simultaneous running threads, appropriate thread usage of registers and synchronization between threads have to be determined due to the limitation of the memory size of local registers, which is only 64 KB per SM in the NVIDIA Geforce series of GPUs.[39]

Normal Mode Analysis. The harmonic approximation of the potential energy function around a minimum energy conformation is the underlying hypothesis in normal mode calculations.[40] The quadratic form of the potential energy function $V(\mathbf{r})$ can be obtained by a Taylor expansion using $\mathbf{r}^0$ (a vector consisting of the $3N$ atomic Cartesian coordinates):[24,41−43]

$$V(\mathbf{r}) \approx V(\mathbf{r}^0) + \sum_{i=1}^{3N} \left(\frac{\partial V}{\partial r_i}\right)(r_i - r_i^0) + \frac{1}{2}\sum_{i,j=1}^{3N} \left(\frac{\partial^2 V}{\partial r_i r_j}\right)(r_i - r_i^0)^{\mathrm{T}}(r_j - r_j^0) \quad (1)$$

The reference point $\mathbf{r}^0$ is chosen to correspond to a minimum of the potential energy function, where $(\partial V/\partial r_i) = 0$. In addition, the potential energy can be defined relative to this reference point, and the constant term $V(\mathbf{r}^0)$ can be taken as zero. As a result, eq 1 in matrix notation can be expressed as follows:

$$V(\mathbf{r}) \approx \frac{1}{2}\mathbf{r}^{\mathrm{T}}\mathbf{H}\mathbf{r} \quad (2)$$

where $\mathbf{H} = \sum_{i,j=1}^{3N}(\partial^2 V/\partial r_i r_j)$, $\mathbf{r}=(r_i-r_i^0)$.

At the same time, the kinetic energy K also has a quadratic form, which can be expressed as

$$K = \frac{1}{2}\dot{\mathbf{r}}^{\mathrm{T}}\mathbf{M}\dot{\mathbf{r}} \quad (3)$$

where $\mathbf{M}$ is a diagonal matrix of $3N \times 3N$ atomic masses and $\dot{\mathbf{r}}$ is the time derivative of $\mathbf{r}$. The equations of motions of a molecule in a harmonic well can be inferred from eqs 2 and 3:

$$\mathbf{M}\ddot{\mathbf{r}} = -\mathbf{H}\mathbf{r} \quad (4)$$

where $\ddot{\mathbf{r}}$ is the second time derivative of $\mathbf{r}$, and the vector $\mathbf{r}$ can be resolved by adopting the general form, which can be expressed as a function of time:

$$r(t) = r^0 + A\cos(\nu t) \quad (5)$$

ν is the fundamental vibration frequency, and A is the amplitude of the vibration. Taking the second derivative of the extension gives

$$\nu^2\mathbf{M}\mathbf{A} = \mathbf{H}\mathbf{A} \quad (6)$$

where $\mathbf{M}$ consists of a diagonal matrix whose elements are the atomic masses, and ν^2 and $\mathbf{A}$ are the eigenvalues and eigenvectors, respectively. The eigenvectors of this matrix are the normal

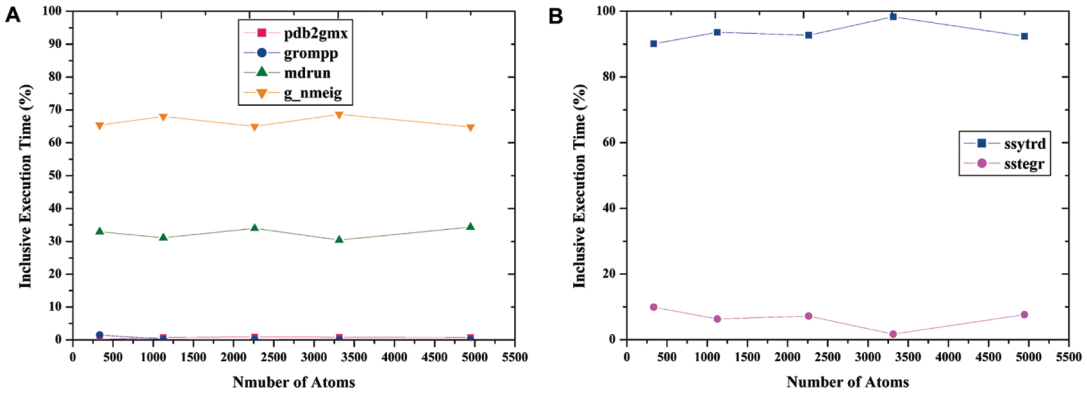

Figure 1. Runtime decomposition results of (A) the modules in the NMA calculation and (B) the subroutines of the Hessian matrix eigenvalues calculation in GROMACS 4.0. The protein test systems are 2WFU, 3CLN, 1A7R, 4AKE, and 1ATN.

modes, and the eigenvalues are the squares of the associated frequencies. Notice that when expressed in Cartesian space, a normal coordinate describes an internal collective change of the structure, except for the first six modes that correspond to global translations and rotations of the molecule with eigenvalues equal to zero.[21] Therefore, the calculation of NMA is reduced to the diagonalization of the Hessian matrix **H**.

Hessian Matrix Diagonalization. Generally speaking, an efficient algorithm for the diagonalization of a symmetric matrix usually consists of the following two stages: the original matrix is first reduced to a tridiagonal matrix **T**, which is also called the Hessenberg matrix, by a sequence of orthogonal similarity transformed as follows:

$$\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}_{A} \xrightarrow{Q_1} \begin{bmatrix} \times & \times & 0 & 0 \\ \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \end{bmatrix}_{Q_1AQ_1} \xrightarrow{Q_2} \begin{bmatrix} \times & \times & 0 & 0 \\ \times & \times & \times & 0 \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix}_{Q_2Q_1AQ_1Q_2} \tag{7}$$

Then, we need to accurately calculate its eigenvalues and eigenvectors based on QR, bisection, or divide-and-conquer algorithms etc.

The general scheme of Householder transformation method is as follows:[44,45]

Step 1 Establish symmetric matrix $\mathbf{A} = \{a_{ij} | a_{ij} = a_{ij}$ and $i,j \in [1,n]\}$.

Step 2 Iterate from eq 8 to 12 with $k = 1, 2, 3, ..., n - 1$.

$$\sigma_{k+1} = sign(a_{k+1,k})\left(\sum_{i=k+1}^{n} (a_{ik})^2\right)^{1/2} \tag{8}$$

$$\rho_{k+1} = \sigma_{k+1}(\sigma_{k+1} + a_{k+1,k}) \tag{9}$$

$$\mathbf{U}^{(k+1)} = (0, ...0, \sigma_{k+1} + a_{k+1,k}, a_{k+2,k}, ..., a_{nk}) \tag{10}$$

$$\mathbf{Q}_{k+1} = \mathbf{I} - \frac{1}{\rho_{k+1}} \mathbf{U}^{(k+1)} (\mathbf{U}^{(k+1)})^{\mathrm{T}} \tag{11}$$

$$\mathbf{Q}_{k+1} \mathbf{A} \mathbf{Q}_{k+1} = A \tag{12}$$

The first step establishes the raw Hessian matrix, which is tridiagonalized by multiplying it with the orthogonal matrix $\mathbf{Q}_i$ calculated in the second step.

Implementation Details. In order to accelerate NMA utilizing GPU, the primary step is to verify the time distribution in each part of the whole process in NMA. In GROMACS 4.0,[46] the process can be divided into four subroutines, and the corresponding runtimes against five protein systems with various numbers of atoms are shown in Figure 1A. The "pdb2gms" and "grompp" modules (preparation of coordinates and topology files) only occupy a small fraction of the overall computational time (less than 1.5%). Though the "mdrun" module (energy minimization and building a Hessian matrix from single conformation) costs about 30% of the computational time, it can be accelerated in parallel with conventional Parallel Virtual Machine (PVM) or Message Passing Interface (MPI) mechanisms across multiple CPUs.[47] The profiles of the five benchmark cases illustrate that the "g_nmeig" module (calculation of a Hessian matrix's eigenvalues) is the most computationally expensive part, which accounts for over 64% of the inclusive execution time.

The calculation of the Hessian matrix eigenvalues can be further divided into two steps, whose runtime profiling against the five benchmark systems is shown in Figure 1B. The tridiagonalization process (ssytrd module) costs the dominant fraction of the computational time (over 90%) in the Hessian matrix eigenvalues calculation process. So far, it can be concluded that the most time-consuming part in NMA is the Householder transformation, which reduces the Hessian matrix to the tridiagonal form. For this reason, the NMA calculation on GPU is reduced to the determination of the Hessian matrix eigenvalues and further to the tridiagonalization according to the characteristics of CUDA technically.

Algorithm 1. Orthogonal vector construction on CPU

set N as the order of matrix **A**
for $i = 1$ to N
  //Compute the norm of vector in row i, column i to N of matrix **A**.
  $r_i = sign(a_{i,j+1})||\mathbf{A}[i][i + 1{:}N]||$;
  //Compute the orthogonal vector **U**
  $\mathbf{U}[0{:}i - 1] = 0$;
  $\mathbf{U}[i] = \mathbf{U}[i] + sqrt(r_i)$;
  $\mathbf{U}[i + 1{:}N] = \mathbf{A}[i][i + 1{:}N]$;
  //Compute the adjusted parameter t.
  $t = 1/sqrt(r_i + sqrt(r_i)\mathbf{U}[i])$;
  Execute tridiagonalization function on GPU;
  //Replace the vector in matrix **A** needed in next iteration.
  $\mathbf{A}[i + 1][i + 1{:}N] = \mathbf{U}[i + 1{:}N]$;
end for

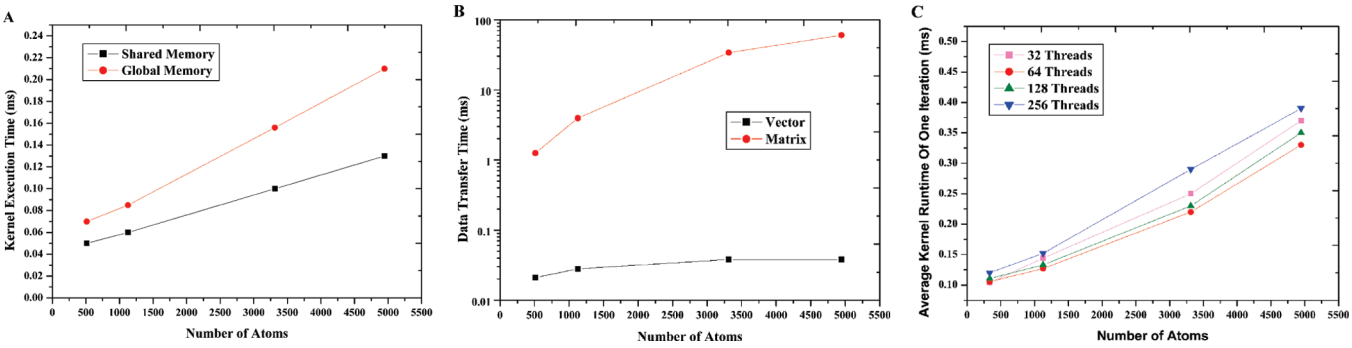

Figure 2. Optimization results of the GPU-accelerated QR Housholder decomposition: (A) kernel runtimes with different memory locations for data storage, (B) kernel runtimes with different data structure for transfer (the vertical axis is scaled in log−linear coordinates), and (C) kernel runtimes with varying numbers of threads per block.

The details of Householder transformation implemented on a hybrid GPU-based architecture are described in Algorithms 1 and 2. Algorithm 1 gives the pseudocode for the orthogonal vector construction on the CPU. The orthogonal vector is used to create the reflector, which is essential in the tridiagonalization process. Algorithm 2 gives the tridiagonalization process implemented on CUDA, which completes the reflector and converts the original matrix into the tridiagonal form. The instructions in Algorithm 2 are operated on a number of threads within all of the blocks scheduled on each stream processor in parallel.

Algorithm 2. Tridiagonalization function on GPU
set *block_size* = 64; //64 threads per block.
number_of_blocks = *N*/*block_size*;
//All blocks on SMs executed in parallel.
for each block **do**
  set *shared_memory_size* = *block_size*;
  Load vector $\mathbf{U}_i$ to shared memory of block$_i$;
  $\mathbf{U}_i' = (\mathbf{U}_i)^T$; //Compute the transpose of the orthogonal vector.
  Load vector $\mathbf{U}_i'$ to shared memory of block$_i$;
  $\mathbf{H}_i = t(\mathbf{U}_i\mathbf{U}_i')$;
  $\mathbf{H}_i = \mathbf{I}_i - \mathbf{H}_i$; //Compute the reflector.
  Load matrix $\mathbf{A}_i$ and $\mathbf{H}_i$ to shared memory of block$_i$;
  $\mathbf{A}_i = \mathbf{A}_i\mathbf{H}_i$; //Tridiagonalize the original matrix.
until all blocks are traversed;
$\mathbf{U}[i+1\colon N] = \mathbf{A}[i+1][i+1\colon N]$; //Return the vector needed in next iteration.

The CUDA implementation of the Householder transformation was executed on a single NVIDIA GeForce GTX295 (1.2 GHz), which held 480 stream processors to compute in parallel.[48] For comparison, the NMA module of GROMACS 4.0 (built with GCC 4.3.2 compilers in serial mode) was performed on a 2.50 GHz Quad-Core Intel Xeon E5420 CPU (running RedHat AS 6.0). The frequency of the CPU is more than twice higher than that of the GPU, so that it has a much better performance than a GPU on single-instruction single-data (SISD) architectures, and the random access memory (RAM) size (8 GB) is nearly 4 times larger than that of the GPU (1.7 GB). Meanwhile, we also compared the peak performance of the QR factorizations implemented in LAPACK, PLASMA, and CUDA. The block size was set to 64 in LAPACK and CUDA implementation to achieve the best performance. The number of cores in PLASMA was set to 16 with respect to the sizes of the test systems.

RESULTS AND DISCUSSION

Optimization of the Implementation. The GPU implementation was optimized from three perspectives which utilize higher level memories, minimize the amount of data transferred, and select the appropriate thread block size. In kernel functions, reading or writing global memory incurs about 400−600 clock cycles of latency, while it just needs one clock cycle of latency in shared memory.[49] In order to reduce the frequency of global memory access, we divided the huge matrix into submatrices which can be stored in the shared memory with a size of 16 KB per SM.[50] However, the maximum number of threads per block is reduced to 256 in our method, which is one-third of the maximum value in principle due to the limited size of the register memory.[51] Subsequently, only the lower triangle (or upper triangle) in eq 11 needs to be calculated, which can effectively enhance the computational speed and save the memory size, utilizing the symmetrical property of the initial matrix and the orthogonal matrix. The loop sequence should be recorded in the kernel function to get the new temporary vector generated by the tridiagonalization process. We measured the kernel runtime of different GPU implementations with different memory accessing types (as shown in Figure 2A). The performance of the shared memory implementation did not improve remarkably (by only twice compared with the global memory implementation) because the load latency was covered by intensive arithmetic calculations of abundant threads.

On the other hand, since data transfer incurs many more clock cycles than arithmetic calculations on GPU threads, it would be effective to minimize the data transfer between the host and the GPU to increase speed performance. The Household-QR algorithm implemented on GPU consists of $n-1$ loops (n is the matrix order), each of which calculates one row/column in the Hessenberg matrix. Though the Hessian matrix is totally changed after one loop, only one vector of the Hessian matrix is needed in the next loop. A temporary vector storing the results from the previous loop is set to avoid transferring the whole matrix and minimize the time spent in data transfer between the GPU and CPU. Therefore, we also measured the time needed to transfer a whole matrix and a vector between the host and GPU (Figure 2B). The cost of transferring a whole matrix is much higher than that of transferring a vector decomposed from the same matrix, which cost over 60 and 0.038 ms, respectively, resulting in transferring of the vector alone between the device memory, and the host memory can improve the performance significantly.

Furthermore, the number of local registers in each thread is very limited, and they are often used as the index of blocks and threads, which are essential for kernel functions' ability to control each thread. However, in the process of transposing the vector, we import the argument ρ_i in eq 9 to occupy one register of each thread. Then, we complete the argument and vector multiplication on the GPU for the purpose of reducing the time complexity from $O(2n)$ to $O(1)$ by duplicating the argument to each thread. It is important to synchronize all threads in the same block if there are conditional statements and loops in kernel functions on the GPU. The transposing operations on GPU are complemented by exchanging the horizontal coordinates and vertical coordinates of elements in each thread. This section also reduces the time complexity from $O(n)$ to $O(1)$ for n elements in the vector **U**.

In addition, the block size of the threads also impacts the performance of kernel implementation on the GPU. The number of threads per block should be chosen as much as possible for better time slicing and full usage of computing resources. However, the more threads per block, the fewer registers that are available per thread. In addition, the size of shared memory per stream multiprocessor (SM) is 16 KB, which is organized into 16 banks.[52] If all running threads access different banks, there are no bank conflicts, and the shared memory is the same speed as the register. If multiple threads access the same bank, the shared memory accessing time will be as long as the time required for the maximum number of threads simultaneously accessed to a single bank.[53] In order to avoid memory bank conflicts, the number of threads per block has to be set as a multiple of 32 (32, 64, 128, and 256 threads in our test), which is the minimum value of task scheduling and executing on SMs. The active thread block size cannot exceed 512 due to insufficient register memory size (64 KB per SM).[54] The blocks of 256 threads cost the most kernel runtime (see Figure 2C). The implementation with the blocks of 32 threads performed better than the one with 128 threads in the case of a smaller system, but the superiority got weaker when the test system grew larger. However, the effects on the kernel execution time are not obvious because the kernel execution time only occupies a small fraction of the total runtime. The final size of threads per block was set to 64 to achieve the best performance by avoiding the bank conflicts of shared memory and covering the load latency by assigning several active blocks on each SM.

Performance and Speedup. First, we compared the peak performance of GPU-accelerated Householder-QR factorizations with conventional (LAPACK) and parallel (PLASMA) implementation in the popular linear algebra libraries on the CPU (as shown in Figure 3). The peak performance Householder-QR factorizations implemented in CUDA can reach up to 180.6 Gflop/s on a single NVIDIA GeForce GTX295 on the average, which is much higher than that of LAPACK and PLASMA on a single CPU (the peak performances of LAPACK and PLASMA are 2.2 Gflop/s and 23.8 Gflop/s, respectively). The performance deteriorated slightly to 109.6 Gflop/s when the size of the matrix was below 3000, but it is still remarkably superior to LAPACK and PLASMA. The results indicated that the CUDA implementation of Householder-QR factorizations is much faster than conventional serial (LAPACK) and parallel (PLASMA) CPU implementations and lays a promising foundation for NMA acceleration on the GPU architecture.

To investigate the performance of the GPU-accelerated NMA calculation, five proteins were selected as the benchmark

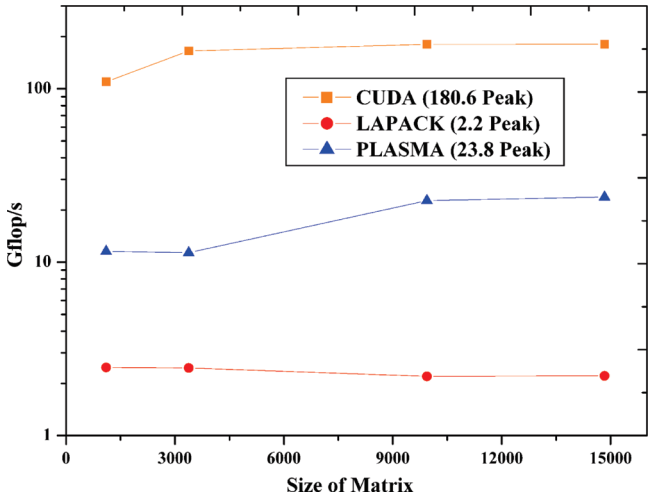

Figure 3. Comparing the peak performances of the QR factorization algorithm implemented in LAPACK, PLASMA, and CUDA. The vertical axis is scaled in log−linear coordinates.

systems: the D14A variant of the *Drosophila* insulin-like peptide 5 (2WFU, 335 atoms), calmodulin (3CLN, 1126 atoms), adenylate kinases (4AKE, 3313 atoms), actin (1ATN, 4946 atoms), and alcohol dehydrogenase (1ADB, 6783 atoms). Table 1 summarizes the execution time of GROMACS 4.0 on a CPU in serial mode and our GPU implementation at both single and double data precisions. The benchmark results revealed that the GPU-accelerated Householder transformation did not show a performance advantage over the CPU for the small systems (2WFU) with 335 atoms, which can be attributed to the fact that for such a small protein system, the computational resources of GPU are not fully utilized, and the extra cost of multithread scheduling and synchronization occupies a relatively large fraction of the overall runtime. However, the GPU code achieved a more than 20× speedup over the CPU for the Householder transformation of larger systems in the cases of 4AKE, 1ATN, and 1ADB. Unfortunately, the NVIDIA GeForce GTX295 used in our study can only access a maximum of 1.7 GB VRAM, which limits the upper size of the simulation system to 7000 atoms with single precision and 5000 atoms with double precision, correspondingly. The tridiagonalization process of the largest protein 1ADB in the test with single precision costs more than 1.6 h on the CPU in serial mode, but it only costs about 219 s on the GPU in the multithread parallel mode. At the same time, the second largest protein 1ATN with double precision costs more than 1.1 h on the CPU, while it costs only 165 s on the GPU in our study. Although the 20× speedup of our GPU NMA implementation is not as high as those observed for GPU-accelerated MD simulations and may not be enough to make an abiding difference in NMA calculation for single protein systems, the speed advantage of GPU-accelerated NMA is promising for repeated and intensive NMA calculations involved in the MM/PBSA approach for large-scale virtual screening.

One of the determinants of the GPU-accelerated performance was the size of the intensive data. The speedup of GPU calculation increased with increasing size of the matrix for single and double precisions because a huge workload can fully utilize the computational resources of the GPU and hide various sources of latency including memory access and thread synchronization. Although the benchmark system size is limited by the global memory size in our study, the scale of the performance can

Table 1. Performance Comparison of the Runtime of Householder Transformation with Single and Double Precision on the CPU and GPU

			single precision			double precision		
PDB ID	number of residues	number of atoms	CPU runtime (s)	GPU runtime (s)	speedup	CPU runtime (s)	GPU runtime (s)	speedup
2WFU	46	335	1.031	2.491	<1	2.167	2.699	<1
3CLN	143	1126	44.584	10.127	4.402	63.542	15.103	4.207
4AKE	428	3313	1774.563	82.119	21.609	2162.016	103.157	20.959
1ATN	630	4946	2881.651	114.558	25.155	4029.183	164.954	24.426
1ADB	748	6783	5943.227	219.386	27.09			

The speedup is the runtime of Householder transformation on the GPU versus the runtime of the Householder transformation routines in GROMACS on the CPU. The GPU runtime was measured from the CUDA implementation of the Householder transformation (the double-type version was compiled by the nvcc compiler using option -arch compute_13, -code sm_13). The CPU runtime was measured by Householder transformation subroutines "ssytrd" with single precision and "dsytrd" with double precision in the lapack library of GROMACS 4.0.

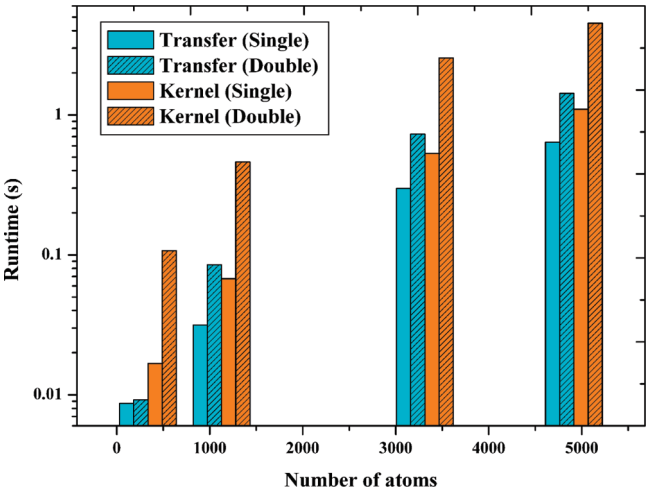

Figure 4. The distribution of data transfer and kernel runtime of the single (float) and double precisions Householder transformation implementation on GPU. Note that the vertical axis is scaled in log−linear coordinates.

be maintained for larger protein systems on more powerful GPU cards with larger global memory as the latest Fermi architecture provided by NVIDIA (as many as 6 GB). Actually, the speedup of the GPU implementation over the CPU one could be even greater for larger systems in principle because the performance cannot be simply scaled by the number of atoms between the GPU and the CPU calculations. For the CPU case, the scaling is very close to $O(N^3)$, indicating that Householder transformation of the Hessian matrix is the dominant cost. However, the speedup of the GPU over the CPU increased faster than linearly with the number of atoms, which can be attributed to the time complexity of $O(1)$ as mentioned above if the latencies incurred from data transfer are ignored because the intensive data were loaded to the GPU and operated across all available cores simultaneously regardless of the data size involved.

The double precision code is expected to be more time-consuming because the number of single precision cores (32 bits) on the GPU is about 8 times more than that of the double precision cores (64 bits).[55] As shown in Figure 4, the transfer time of the double precision implementation increases almost twice, and the kernel runtime costs about 5−8 times more than the single precision one. The differences became more apparent for larger test systems. The differences in speed for the single and double codes are reasonable given that the double type is twice longer than the single one in storage and transfer, and there are 8 times more single precision cores in the GPU. However, as shown in Table 1, the speedup for double precision only descends slightly compared with the single precision, which can be attributed to the fact that the kernel execution time on GPU only occupies a very small fraction in the overall program runtime, especially with larger protein systems.

Accuracy. Another important consideration in evaluating all-atom NMA implementation is the accuracy of the modes it produces. Previous work has shown that single precision float point is sufficient to produce high quality results in molecular dynamics,[56] but in practice, double precision is always used in the calculations in a way that avoids an unnecessary loss of accuracy. Since double precision code reduced the performance by about 8 times compared to the single precision one, as expected, it is worth investigating whether any potential error may have resulted from the precision truncation to make a compromise between the efficiency and accuracy.

To test the accuracy of our CUDA implementation, we performed the Householder transformation calculation of the Hessian matrix from the NMA of adenylate kinase (4AKE) with single and double precisions and generated the structural snapshots of corresponding modes with GROMACS from the decomposition results. The root-mean-square deviation (RMSD) between the snapshots of corresponding modes from the GPU code and GROMACS calculated on CPU were compared to evaluate the accuracy of the GPU implementation. Thirty snapshots from the 10 lowest-frequency vibration modes (modes 7−16) calculated by GROMACS were chosen as the reference of the NMA results. The average RMSD values of the 30 snapshots in each mode between single/double precision code and the reference structures were calculated. The average RMSDs between the modes generated by double precision code and GROMACS are in the range of 0.0021−0.042 Å (Figure 5A), and most of the heavy atoms' position deviations between the most different snapshot conformations of node 8 from the GPU and CPU are below 0.1 Å (as shown in Figure 5B). Considering that the resolution of the crystal structure (4AKE) is as high as 2.20 Å, the tiny structural deviations between the GPU and CPU NMA calculation results can be almost ignored. However, the average RMSDs calculated from single precision modes are higher than the double precision one and up to about 0.25 Å.

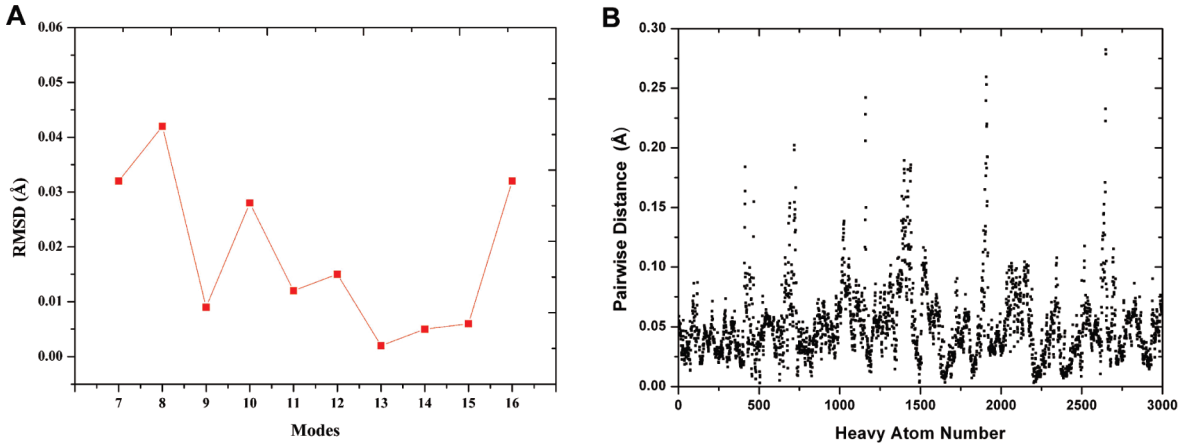

Figure 5. Accuracy comparison of *t* NMA calculation on GPU and CPU. (A) Average RMSD values of the top 10 modes of 4AKE (modes 7−16) between the results of GPU (double precision) and CPU implementation. (B) Distribution of pairwise distances for the heavy atoms between the most different snapshot conformations in mode 8 of 4AKE from GPU and CPU.

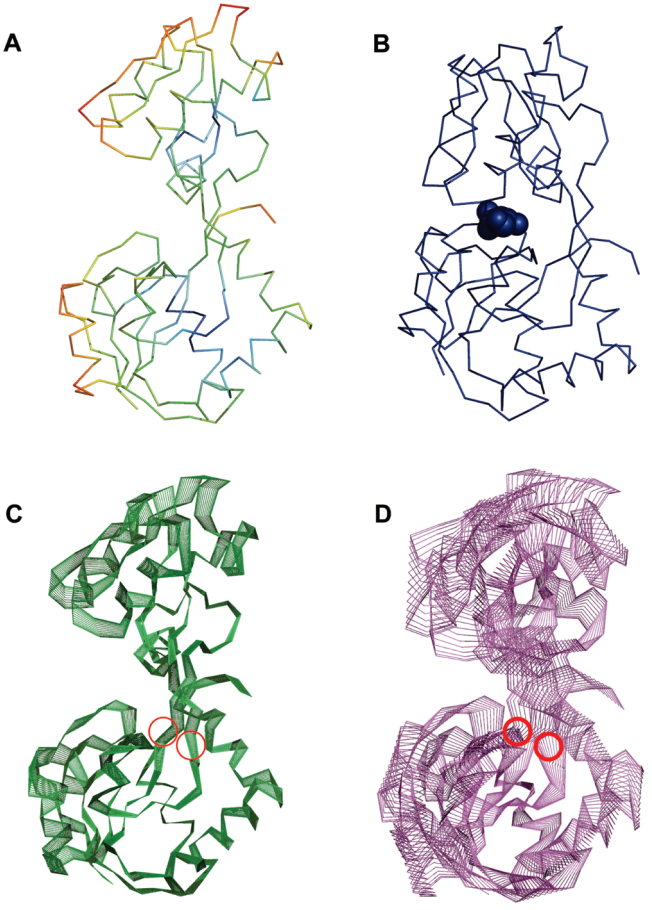

Figure 6. Collective motion prediction results for the allosteric conformational changes observed between (A) free (1GGG) and (B) substrate-bound (1WDN) forms of the glutamine-binding protein with (C) GPU-accelerated all-atom NMA and (D) ENM. The structure of the free glutamine-binding protein is colored according to the atom position deviations aligned onto the substrate-bound form: red color for deviation as high as 14.4 Å and blue color for deviation as low as 0.7 Å. The glutamine is represented in the sphere model. The superposition of intermediate structures for modes 7 from the GPU and ENM calculation are displayed. The two hinge regions are shown circled in red.

Although the accuracy of the faster single precision code is about 100 times lower than the double precision one, the errors of the results are still acceptable considering the resolutions of most of the crystal structures used for NMA calculation are higher than 1.0 Å, indicating that the results obtained from GPU with both single and double precisions are as accurate as the equivalent calculated from common CPU implementation in GROMACS.

Application. NMA is widely applied to predict the functional collective motions of the large macromolecules. Although the previous study of Tirion evidenced that a hypothetical force field with uniform (single-parameter) harmonic potentials (like ENM) yields global modes, which are almost indistinguishable from those obtained from a detailed force field with specific nonlinear terms,[48] all-atom NMA is still considered more accurate than the coarse-grained one because the magnitude of the excursions along the normal modes is almost unknown, and more attention must be paid when interpreting the resulted motions.[8] We analyzed the allosteric conformational changes observed between the free (1GGG) and substrate-bound (1WDN) forms of the glutamine-binding protein with GPU-accelerated all-atom NMA.[57,58] This ellipsoidal protein has two globular domains connected by two hinges and exhibits a large-scale movement of the two hinges connecting the two globular domains upon glutamine binding (as shown in Figure 6A and B). From the superposition of intermediate structures for mode 7 of the ligand-free structure (as shown in Figure 6C), the collective motion trend of the two domains around the hinges can be apparently observed to close the deep binding cleft for glutamine binding, which corresponds well with the most deviated regions between the ligand-free and ligand-bound regions, as shown in Figure 6A. Moreover, the hinge regions with less structural excursion variances are also apparent and coincide well with the crystal structures, which is more accurate and distinct than that of the ENM results (as shown in Figure 6C and D).

CONCLUSION AND FUTURE WORK

We have implemented Householder-QR transformation for Hessian matrix on GPU, which is the most time-consuming step in all-atom NMA. As expected, GPU implementation achieved an acceptable speedup compared with CPU. The results revealed

that the performance of GPU improved with the increasing computational workload, which can fully occupy the idle computational units and hide various sources of latency. At the same time, the influence of the data precision was also verified from the perspectives of both speed and accuracy. Data transfer and the amount of processing cores significantly impacted the performance of GPU when the precision of data type increases. The results of our calculations illustrate that all-atom NMA may be suitable for GPU acceleration for large systems without any coarse-grained approximation. The GPU code has been integrated into the GROMACS package as a standalone module to provide seamless acceleration of all-atom NMA for the molecular modeling community, which can be provided on request.

AUTHOR INFORMATION

Corresponding Author

*Phone/Fax: +86-21-64250213. E-mail: hlli@ecust.edu.cn.

ACKNOWLEDGMENT

This work was supported by the Special Fund for Major State Basic Research Project (Grants 2009CB918501 and 2011CB910200), the National Natural Science Foundation of China (Grants 20803022), the Shanghai Committee of Science and Technology (Grants 09dZ1975700 and 10431902600), the Major National Scientific and Technological Project of China (Grants 2011ZX09307-002-03, 2009ZX09501-001, and 2009ZX09301-001), and the Fundamental Research Funds for the Central Universities. H.L. is also sponsored by the Shanghai Rising-Star Program (Grant 10QA1401800) and Program for New Century Excellent Talents in University. We also thank Prof. Rolf Hilgenfeld from University of Lübeck for the careful reading and polishing of this manuscript when he held a Chinese Academy of Sciences (CAS) Professorship.

ABBREVIATIONS

BNM, Block Normal Mode; CPU, Central Processing Unit; CUDA, Compute Unified Device Architecture; ENM, Elastic Network Model; GPU, Graphics Processing Unit; LAPACK, Linear Algebra Package; MD, Molecular Dynamics; MPI, Message Passing Interface; NMA, Normal Mode Analysis; PLASMA, Parallel Linear Algebra for Scalable Multicore Architectures; PVM, Parallel Virtual Machine; RAM, Random Access Memory; RMSD, Root Mean Square Deviation; SIMD, Single-Instruction Multiple-Data; SISD, Single-Instruction Single-Data; SM, Stream Multiprocessor; SP, Streaming Processor

REFERENCES

(1) Lindahl, E.; Azuara, C.; Koehl, P.; Delarue, M. NOMAD-Ref: visualization, deformation and refinement of macromolecular structures based on all-atom normal mode analysis. *Nucleic Acids Res.* **2006**, *34*, 52–56.

(2) Smyth, M. S.; Martin, J. H. J. X-Ray crystallography. *J. Clin. Pathol.* **2000**, *53*, 8–14.

(3) Wishart, D. S.; Sykes, B. D.; Richards, F. M. The chemical shift index: a fast and simple method for the assignment of protein secondary structure through NMR spectroscopy. *Biochemistry* **1992**, *31*, 1647–1651.

(4) Ritchie, K.; Iino, R.; Fujiwara, T.; Murase, K.; Kusumi, A. The fence and picket structure of the plasma membrane of live cells as revealed by single molecule techniques (Review). *Mol. Membr. Biol.* **2003**, *20*, 13–18.

(5) Karplus, M.; Petsko, G. A. Molecular dynamics simulations in biology. *Nature* **1990**, *347*, 631–639.

(6) Tama, F.; Sanejouand, Y. H. Conformational change of proteins arising from normal mode calculations. *Protein Eng.* **2001**, *14*, 1–6.

(7) Sedeh, R. S.; Bathe, M.; Bathe, K. J. The subspace iteration method in protein normal mode analysis. *J. Comput. Chem.* **2010**, *31*, 66–74.

(8) Bahar, I.; Lezon, T. R.; Bakan, A.; Shrivastava, I. H. Normal mode analysis of biomolecular structures: functional mechanisms of membrane proteins. *Chem. Rev.* **2010**, *110*, 1463–1497.

(9) Tama, F.; Valle, M.; Frank, J.; Brooks, C. L., III. Dynamic reorganization of the functionally active ribosome explored by normal mode analysis and cryo-electron microscopy. *Proc. Natl. Acad. Sci. U. S. A.* **2003**, *100*, 9319–9323.

(10) Ma, J. Usefulness and limitations of normal mode analysis in modeling dynamics of biomolecular complexes. *Structure* **2005**, *13*, 373–380.

(11) Tama, F.; Brooks, C. L., III. Symmetry, form, and shape: guiding principles for robustness in macromolecular machines. *Annu. Rev. Biophys. Biomol. Struct.* **2006**, *35*, 115–133.

(12) Kuhn, B.; Kollman, P. A. Binding of a diverse set of ligands to avidin and streptavidin: an accurate quantitative prediction of their relative affinities by a combination of molecular mechanics and continuum solvent models. *J. Med. Chem.* **2000**, *43*, 3786–3791.

(13) Swanson, J. M.; Henchman, R. H.; McCammon, J. A. Revisiting free energy calculations: a theoretical connection to MM/PBSA and direct calculation of the association free energy. *Biophys. J.* **2004**, *86*, 67–74.

(14) Obiol-Pardo, C.; Rubio-Martinez, J. Comparative evaluation of MMPBSA and XSCORE to compute binding free energy in XIAP-peptide complexes. *J. Chem. Inf. Model.* **2007**, *47*, 134–142.

(15) Brooks, B.; Karplus, M. Harmonic dynamics of proteins: normal modes and fluctuations in bovine pancreatic trypsin inhibitor. *Proc. Natl. Acad. Sci. U. S. A.* **1983**, *80*, 6571–6575.

(16) Delarue, M.; Sanejouand, Y. H. Simplified normal mode analysis of conformational transitions in DNA-dependent polymerases: the elastic network model. *J. Mol. Biol.* **2002**, *320*, 1011–1024.

(17) Marechal, J. D.; Perahia, D. Use of normal modes for structural modeling of proteins: the case study of rat heme oxygenase 1. *Eur. Biophys. J.* **2008**, *37*, 1157–1165.

(18) Suhre, K.; Sanejouand, Y. H. ElNemo: a normal mode web server for protein movement analysis and the generation of templates for molecular replacement. *Nucleic Acids Res.* **2004**, *32*, 610–614.

(19) Brown, S. P.; Muchmore, S. W. Rapid estimation of relative protein-ligand binding affinities using a high-throughput version of MM-PBSA. *J. Chem. Inf. Model.* **2007**, *47*, 1493–1503.

(20) Tama, F.; Gadea, F. X.; Marques, O.; Sanejouand, Y. H. Building-block approach for determining low-frequency normal modes of macromolecules. *Proteins* **2000**, *41*, 1–7.

(21) Hollup, S. M.; Salensminde, G.; Reuter, N. WEBnm@: a web application for normal mode analyses of proteins. *BMC Bioinf.* **2005**, *6*, 52–60.

(22) Bahar, I.; Rader, A. J. Coarse-grained normal mode analysis in structural biology. *Curr. Opin. Struct. Biol.* **2005**, *15*, 586–592.

(23) Tama, F.; Brooks, C. L., III. Diversity and identity of mechanical properties of icosahedral viral capsids studied with elastic network normal mode analysis. *J. Mol. Biol.* **2005**, *345*, 299–314.

(24) Hinsen, K. Analysis of domain motions by approximate normal mode calculations. *Proteins: Struct., Funct., Bioinf.* **1998**, *33*, 417–429.

(25) Yang, L.; Song, G.; Jernigan, R. How well can we understand large-scale protein motions using normal modes of elastic network models? *Biophys. J.* **2007**, *93*, 920–929.

(26) Zadra, A.; Brunet, G.; Derome, J. An empirical normal mode diagnostic algorithm applied to NCEP reanalyses. *J. Atmos. Sci.* **2002**, *59*, 2811–2829.

(27) Haque, I. S.; Pande, V. S. PAPER--accelerating parallel evaluations of ROCS. *J. Comput. Chem.* **2010**, *31*, 117–132.

(28) Owens, J.; Houston, M.; Luebke, D.; Green, S.; Stone, J.; Phillips, J. GPU computing. *Proc. IEEE* **2008**, *96*, 879–899.

(29) Hardy, D. J.; Stone, J. E.; Schulten, K. Multilevel Summation of Electrostatic Potentials Using Graphics Processing Units. *Parallel Comput.* **2009**, *35*, 164–177.
(30) Phillips, J. C.; Braun, R.; Wang, W.; Gumbart, J.; Tajkhorshid, E.; Villa, E.; Chipot, C.; Skeel, R. D.; Kale, L.; Schulten, K. Scalable molecular dynamics with NAMD. *J. Comput. Chem.* **2005**, *26*, 1781–1802.
(31) Anderson, A. G.; Goddard, W. A.; Schroder, P. Quantum Monte Carlo on graphical processing units. *Comput. Phys. Commun.* **2007**, *177*, 298–306.
(32) Che, S.; Boyer, M.; Meng, J. Y.; Tarjan, D.; Sheaffer, J. W.; Skadron, K. A performance study of general-purpose applications on graphics processors using CUDA. *J. Parallel. Distr. Comput.* **2008**, *68*, 1370–1380.
(33) Stone, J. E.; Hardy, D. J.; Ufimtsev, I. S.; Schulten, K. GPU-accelerated molecular modeling coming of age. *J. Mol. Graphics Model.* **2010**, *29*, 116–125.
(34) Buttari, A.; Langou, J.; Kurzak, J.; Dongarra, J. A class of parallel tiled linear algebra algorithms for multicore architectures. *Parallel Comput.* **2009**, *35*, 38–53.
(35) Buttari, A.; Langou, J.; Kurzak, J.; Dongarra, J. Parallel tiled QR factorization for multicore architectures. *Concur. Comput.—Pract. E* **2008**, *20*, 1573–1590.
(36) Belleman, R.; Bédorf, J.; Portegies Zwart, S. High performance direct gravitational N-body simulations on graphics processing units II: An implementation in CUDA. *New. Astron.* **2008**, *13*, 103–112.
(37) Hong, S.; Kim, H. An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness. *SIGARCH Comput. Archit. News* **2009**, *37*, 152–163.
(38) Lindholm, E.; Nickolls, J.; Oberman, S.; Montrym, J. NVIDIA Tesla: A Unified Graphics and Computing Architecture. *IEEE Micro* **2008**, *28*, 39–55.
(39) Nickolls, J.; Buck, I.; Garland, M.; Skadron, K. Scalable parallel programming with CUDA. *Queue* **2008**, *6*, 40–53.
(40) Ghysels, A.; Van Speybroeck, V.; Pauwels, E.; Catak, S.; Brooks, B. R.; Van Neck, D.; Waroquier, M. Comparative study of various normal mode analysis techniques based on partial Hessians. *J. Comput. Chem.* **2010**, *31*, 994–1007.
(41) Case, D. Normal mode analysis of protein dynamics. *Curr. Opin. Struct. Biol.* **1994**, *4*, 285–290.
(42) Tirion, M. Large amplitude elastic motions in proteins from a single-parameter, atomic analysis. *Phys. Rev. Lett.* **1996**, *77*, 1905–1908.
(43) Ma, J. New advances in normal mode analysis of supermolecular complexes and applications to structural refinement. *Curr. Protein Pept. Sci.* **2004**, *5*, 119–123.
(44) Martin, R.; Wilkinson, J. Similarity reduction of a general matrix to Hessenberg form. *Numer. Math.* **1968**, *12*, 349–368.
(45) Dongarra, J. J.; van de Geijn, R. A. Reduction to condensed form for the Eigenvalue problem on distributed memory architectures. *Parallel Comput.* **1992**, *18*, 973–982.
(46) Lange, O. F.; Schafer, L. V.; Grubmuller, H. Flooding in GROMACS: accelerated barrier crossings in molecular dynamics. *J. Comput. Chem.* **2006**, *27*, 1693–1702.
(47) Berendsen, H.; van der Spoel, D.; Van Drunen, R. GROMACS: a message-passing parallel molecular dynamics implementation. *Comput. Phys. Commun.* **1995**, *91*, 43–56.
(48) Gao, W.; Huyen, N.; Loi, H.; Kemao, Q. Real-time 2D parallel windowed Fourier transform for fringe pattern analysis using Graphics Processing Unit. *Opt. Express.* **2009**, *17*, 23147–23152.
(49) Levy, T.; Cohen, G.; Rabani, E. Simulating Lattice Spin Models on Graphics Processing Units. *J. Chem. Theory Comput.* **2010**, *6*, 3293–3301.
(50) Tunbridge, I.; Best, R. B.; Gain, J.; Kuttel, M. M. Simulation of Coarse-Grained Protein-Protein Interactions with Graphics Processing Units. *J. Chem. Theory Comput.* **2010**, *6*, 3588–3600.
(51) Garland, M.; Le Grand, S.; Nickolls, J.; Anderson, J.; Hardwick, J.; Morton, S.; Phillips, E.; Zhang, Y.; Volkov, V. Parallel computing experiences with CUDA. *IEEE Micro.* **2008**, *28*, 13–27.
(52) Yasuda, K. Accelerating density functional calculations with graphics processing unit. *J. Chem. Theory Comput.* **2008**, *4*, 1230–1236.
(53) Zhang, Y.; Cohen, J.; Owens, J. D. Fast Tridiagonal Solvers on the GPU. *Acm. Sigplan. Notices* **2010**, *45*, 127–136.
(54) Harvey, M. J.; De Fabritiis, G. An implementation of the smooth particle mesh Ewald method on GPU hardware. *J. Chem. Theory Comput.* **2009**, *5*, 2371–2377.
(55) Manavski, S.; Valle, G. CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. *BMC Bioinf.* **2008**, *9*, 10–19.
(56) Friedrichs, M. S.; Eastman, P.; Vaidyanathan, V.; Houston, M.; Legrand, S.; Beberg, A. L.; Ensign, D. L.; Bruns, C. M.; Pande, V. S. Accelerating molecular dynamic simulation on graphics processing units. *J. Comput. Chem.* **2009**, *30*, 864–872.
(57) Hsiao, C. D.; Sun, Y. J.; Rose, J.; Wang, B. C. The crystal structure of glutamine-binding protein from *Escherichia coli*. *J. Mol. Biol.* **1996**, *262*, 225–242.
(58) Sun, Y. J.; Rose, J.; Wang, B. C.; Hsiao, C. D. The structure of glutamine-binding protein complexed with glutamine at 1.94 Å resolution: comparisons with other amino acid binding proteins1. *J. Mol. Biol.* **1998**, *278*, 219–229.