Computer Physics Communications 184 (2013) 1150–1154

Contents lists available at SciVerse ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Accelerating modified Shepard interpolated potential energy calculations using graphics processing units

Hong Fu, Limin Zheng, Minghui Yang *

Key Laboratory of Magnetic Resonance in Biological Systems, State Key Laboratory of Magnetic Resonance and Atomic and Molecular Physics, Wuhan Centre for Magnetic Resonance, Wuhan Institute of Physics and Mathematics, Chinese Academy of Sciences, Wuhan, 430071, People's Republic of China

ARTICLE INFO

Article history:
Received 18 September 2012
Received in revised form
5 December 2012
Accepted 8 December 2012
Available online 14 December 2012

Keywords:
Graphics processing unit acceleration
Chemical reactive dynamics
Potential energy surface
Modified Shepard interpolation scheme

ABSTRACT

The potential energy surfaces constructed with the modified Shepard interpolation scheme have been widely used in studies of chemical reaction dynamics. However, computational costs of interpolation increase rapidly with the size of the system and the number of data points needed to achieve a given accuracy. In this work, we present a naive Graphics Processing Unit (GPU)-accelerated algorithm for modified Shepard interpolated potential energy calculations and its implementation with the PGI CUDA Fortran language. The benchmark tests on a NVIDIA Tesla C2050 using four interpolated potential energy surfaces (one for $H + H_2O \leftrightarrow H_2 + OH$, two for $H + NH_3 \leftrightarrow H_2 + NH_2$ and one for $H + CH_4 \leftrightarrow H_2 + CH_3$) demonstrated a speedup of 50-fold over the original CPU implementation on an Intel E5620 processor and the speedup increases with the system size and the number of data points. This work presents a promising GPU application in the field of chemical reaction dynamics.

1. Introduction

Graphics processing units (GPUs) are designed for acceleration in image building in a frame buffer in order to address the demands of real-time high-resolution 3D graphics computational-intensive tasks. The highly parallel multi-core structure of GPUs makes them very effective to process large blocks of data in parallel. Recently, the graphics card manufacturer NVIDIA released the "compute unified device architecture" (CUDA) development toolkit for high-end graphics cards. CUDA allows developers to program in a C-like language and to take full advantage of NVIDIA's accelerator so that the complexity for using a GPU in general-purpose scientific computing is reduced substantially. Nowadays, the GPU has emerged as a popular platform for high performance computing in many different areas [1].

One of the main applications of GPU in computational chemistry is molecular dynamics (MD) simulations which describe the motions of atoms in bio-molecules or complex molecular systems by solving Newton's second law of motion and provide important information on dynamics and thermodynamics properties [2–11]. One of the main challenges of MD simulations is that the simulation time should be long enough to sample most important conformation spaces. GPUs provide an economical means to accelerate MD simulations [2,6]. As a result, many of the mainstream MD simulation software packages (Amber [11], Gromacs [12], NAMD [13], LAMMPS [4], etc.) have incorporated GPU acceleration and some new software packages such as HOOMD-Blue [14] and ACEMD [2] have been developed especially on GPU architecture. Recently, Pande and coworkers have released the OpenMM library of GPU kernels for MD simulations [15]. So far, GPU acceleration has been demonstrated to achieve speedups from around 10 to 100 for MD simulations [13].

Another important application of GPUs is quantum chemistry [16–29]. The first application in this area is the GPU implementation of the Quantum Monte Carlo method by Anderson and coworkers [18]. Afterwards, Yasuda reported GPU applications to accelerate two-electron integral evaluation (10× speedup) and density functional calculations [19]. Martinez and coworkers have implemented a GPU algorithm for two-electron integrals (with a 130× speedup), direct self-consistent field calculations (with speedups ranging from 28× to 650×), analytical energy gradients, geometry optimization and first principles molecular dynamics on GPU [16,17]. They also released "TeraChem"—the first general-purpose quantum chemistry software package [20–22]. Others include Asadchev and coworkers who implemented the two-electron integral calculations up to *G* functions on GPUs [23], and Luigi Genovese et al. who implemented the GPU accelerating code based on BigDFT and achieved a speedup factor of 6 [24]. More recently, efforts have also been made to accelerate electronic correlation calculations [25–29].

In this work, GPU acceleration is applied to the field of chemical reaction dynamics in which the nuclear motion in a molecular system is solved quantum mechanically (e.g. the time-dependent

* Corresponding author. Tel.: +86 27 87197783; fax: +86 027 87199291.
E-mail address: yangmh@wipm.ac.cn (M. Yang).

0010-4655/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.cpc.2012.12.005

wave-packet quantum dynamics method) or classical mechanically (e.g. the quasi-classical trajectory method) under the Born–Oppenheimer approximation. The concept of potential energy surface (PES) is introduced to describe the relation between energy and geometry of the system. As the number of geometries needed to construct an accurate PES increases exponentially with the size of the system, the computational costs for *ab initio* calculations and validation of the PES also increase rapidly. In fact, the number of geometries denoted as "grid points" in quantum dynamics calculations for the potential integral is often much larger than the number of expansion coefficients of wavefunctions in a basis. For example, the number of grid points exceeds 3.1×10^{10} in full-dimensional quantum dynamics studies of the $H + NH_3$ reaction, whereas the number of expansion coefficients is 3.9×10^9, resulting in a dramatic increase of computational costs.

The PES could be generated with various methods, including analytic functional fitting [30,31], spline interpolation [32], modified Shepard interpolation [33], interpolating moving least squares [34], and neural networks [35]. In this work, the GPU acceleration is applied to the modified Shepard interpolation scheme developed by Collins and coworkers [33,36–38]. This scheme calculates potential energy from *ab initio* energies and energy derivatives of known points in the PES data set and produces these points with the aid of classical trajectory simulations of reaction dynamics. Crittenden and Jordan have summarized the desirable properties of the modified Shepard interpolation scheme: it has relatively low costs in computation and exactly reproduces the original *ab initio* data, and can be applied to a large range of chemical problems [39]. However, because the potential energy of an arbitrary configuration is expressed as a weighted average of Taylor series expansions about the points in the PES data set, the computational costs scale linearly with the points in the PES data set and exponentially with the size of the system. As a result, the computational costs of the potential energy calculation with the modified Shepard interpolation scheme is usually much larger than that with an analytic potential energy function.

In this work, we present a GPU-based implementation of the modified Shepard interpolation scheme and show that this GPU acceleration could be useful for the development or evaluation of the interpolated PES and in the studies of chemical reaction dynamics.

2. The modified Shepard interpolation scheme and CPU/GPU algorithms

Because details of the modified Shepard interpolation scheme have been described in many publications [33,36–39], this scheme together with its CPU algorithm will be introduced briefly in the first subsection. In the next two subsections, we present the GPU algorithms for the exact implementation and an approximate implementation of the interpolation scheme. For simplicity, the molecular configuration whose potential energy to be calculated is denoted as the "grid point" and is distinguished from the "data point" defined in the PES data set. The term "grid point" comes from the grids defined in the quantum reaction dynamics calculations, although the GPU algorithm presented here could also be applied in quasi-classical trajectory (QCT) calculations.

2.1. The interpolation scheme and CPU algorithm

For an arbitrary grid point, its energy is expressed as a weighted average of Taylor series expansions of data points in the PES data set:

$$V(Z) = \sum_{i=1}^{ndata*ngroup} w_i(Z) T_i(Z), \tag{1}$$

where *ndata* is the number of original *ab initio* data points in the PES. By applying the molecular symmetry group elements to the original data points, the total number of data points used in the interpolation is *ndata* $*$ *ngroup*, where *ngroup* is the number of elements in the symmetry group. $Z = \{Z(X)\}$ are the interpolation coordinates and X is the $3 \times natom$ Cartesian coordinates of atoms in the molecule. Because the bond lengths are the smallest set of invariants that can give a global description of the molecular structure and their inverses were found to result in more accurate Taylor expansions for bond stretching potential functions, the PES is constructed with inverse inter-atomic distance coordinates in Collins' modified Shepard interpolation scheme,

$$Z = \{1/R_l\} \ (l = 1, \ldots, nbond), \tag{2}$$

here $nbond = natom \times (natom - 1)/2$. $T(Z, i)$ is the second-order Taylor expansion about the near data point $Z(X(i))$,

$$T_i(Z) = V|_{Z(i)} + \sum_{l=1}^{nbond} \left.\frac{\partial V}{\partial Z_l}\right|_{Z(i)} [Z - Z_l(i)] + \frac{1}{2} \sum_{k=1}^{nbond} \sum_{l=1}^{nbond} \left.\frac{\partial^2 V}{\partial Z_k \partial Z_l}\right|_{Z(i)} [Z - Z_k(i)][Z - Z_l(i)] \tag{3}$$

$v_i(Z)$ and $w_i(Z)$ represent the weight and relative weight of grid point $\boldsymbol{Z}$ with respect to the i-th data point, respectively,

$$w_i(Z) = \frac{v_i(Z)}{v_{tot}(Z)} \tag{4}$$

and $v_{tot}(Z)$ is the summation of the weights

$$v_{tot}(Z) = \sum_{j}^{ndata*ngroup} v_j(Z). \tag{5}$$

In practical calculations, only the data points with relative weights larger than the tolerance *wtol* are used in the Taylor expansions and the number of these data points is denoted as *nforc*,

$$V(Z) = \sum_{i=1}^{nforc} w_i(Z) T_i(Z). \tag{6}$$

However, the selection of data points by their relative weights also introduces discontinuities in the PES, as discussed in Jordan's work [40].

Usually a two-part weight function is used with parameters $p = 2$ and $q = 12$ [41],

$$v_i(Z) = \frac{1}{\left(\sum_{l}^{nbond} \left[\frac{Z_l - Z_l(i)}{d_l(i)}\right]^2\right)^p + \left(\sum_{l}^{nbond} \left[\frac{Z_l - Z_l(i)}{d_l(i)}\right]^2\right)^q} \tag{7}$$

$d^l(i)$ is the confidence length of l-th coordinates for the i-th data point and is viewed as an accuracy estimate of the Taylor expansion in each coordinate direction.

The inverse inter-atomic distance coordinate system is a redundant set of coordinates as there are only $nint = 3 \times natom - 6$ independent internal coordinates. Thus a set of *nint* internal coordinates ζ could be constructed as linear combinations of $\boldsymbol{Z}$ for each data point in the PES data set. Also, the *nint* local internal coordinates of grid points could be expressed as linear combinations of $\boldsymbol{Z}$, if it is sufficiently close to some data point $Z^0 = Z(X^0)$

$$\zeta^{X_0}(X) = U^T Z. \tag{8}$$

Then the second order Taylor expansion $T_{g\circ i}(\zeta)$ is

$$T_i(\zeta) = V[\zeta(i)] + \sum_{k=1}^{nint}[\zeta_k - \zeta_k(i)]\frac{\partial V}{\partial \zeta_k}\bigg|_{\zeta(i)} + \frac{1}{2}\sum_{k=1}^{nint}\sum_{l=1}^{nint}[\zeta_k - \zeta_k(i)][\zeta_l - \zeta_l(i)]\frac{\partial^2 V}{\partial \zeta_k \partial \zeta_l}\bigg|_{\zeta(i)} \quad (9)$$

and the potential energy at the configuration $\boldsymbol{Z}$ is

$$V(Z) = \sum_{i=1}^{nforc} w_i(Z)T_i(\zeta). \quad (10)$$

Accordingly, the CPU implementation of the interpolation scheme is divided into 3 steps:

(1) Step 1—weight calculations. At first the inverse inter-atomic distances of the grid point are calculated (Eq. (2)); then, the weights of this grid point are evaluated with respect to *ndata* × *ngroup* data points (Eq. (6)) and all *ndata* × *ngroup* weights are summed (Eq. (5)).
(2) Step 2—selection. The relative weights $w(Z)$ are calculated (Eq. (4)) and *nforc* neighboring data points are selected by comparing their relative weights $w(Z)$ with a given tolerance *wtol*. The relative weights of these *nforc* neighboring data points are evaluated again with an updated $v_{\rm tot}(Z)$ which is calculated by summing over these *nforc* data points only.
(3) Step 3—interpolations. For each of these *nforc* neighboring data points, the coordinates Z of the grid point are transformed to local internal coordinates ζ (Eq. (8)) and the Taylor expansions are calculated about data points (Eq. (9)). The potential energy of the grid point is thus obtained by a weighted average of the Taylor expansions (Eq. (10)).

2.2. GPU programming and GPU algorithm

In the GPU programming model, the entire computational work is controlled by the code running on the CPU. Once the data are available, the CPU code copies the required data from the CPU memory to the GPU memory, then the GPU performs the computations and the results are copied from the GPU memory to the CPU memory after the computation is finished. The communication between CPU and GPU is a bottleneck in the GPU system and the data copying between the CPU memory and the GPU memory should be avoided as much as possible. The GPU can process a large number of concurrent threads and these threads are organized into thread blocks with up to 512 threads in each block. Threads inside the same block can cooperate using shared memory and barrier synchronization.

GPU acceleration is particularly suitable for the modified Shepard interpolation potential energy calculation due to two reasons: (1) because only coordinates of grid points are copied from the CPU memory to the GPU memory and potential energies are copied from the GPU memory to the CPU memory, data migration is not a rate-limiting step. For other data, they are pre-calculated and stored in the GPU memory; (2) only a few calculations demand inter-block communication. They are the summation of weights (Eq. (5)) and the weighted average of Taylor expansions (Eq. (10)). The other calculations can be executed concurrently with multiple independent thread blocks in the GPU code. Especially in the practical applications to different systems, only a few parameters about the system and the PES data set are changed and the software package will be directly applicable with no modification.

In this work, the GPU codes are programmed using the PGI CUDA Fortran language which enables one to program in Fortran with a small set of extensions. As the original interpolation program is written in Fortran, the difficulties in translating the CPU code to a GPU one are substantially reduced. The GPU algorithm is also divided into 3 steps.

(1) Step 1—weight calculations. The weight calculations of a grid point with respect to *ndata* × *ngroup* data points are mapped to GPU threads and these *ndata* × *ngroup* threads are organized as a one-dimensional grid of thread blocks (the number of thread blocks is denoted as *nblock*) and each block contains 256 threads. The weights are saved in a global array *vtmn* and the sum reduction performs an intra-block summation leading to a local sum of weights, *vtblock*. Another sum reduction performs a summation over *vtblock* leading to the sum of all weights $v^{\rm tot}$ (Eq. (5)).
(2) Step 2—selection. Again, all data points are mapped to *ndata* × *ngroup* GPU threads and these threads are organized as a one-dimensional grid of thread blocks containing 256 threads. Each thread calculates the relative weight for a data point with *vtmn* and $v_{\rm tot}$ obtained in Step 1 (Eq. (4)). Inside a thread block, a stream compaction algorithm [42] is employed to collect the data points which have relative weights larger than the tolerance *wtol*. All *nforc* neighboring data points with larger relative weights are collected by another stream compaction calculation and the weights for these data points are summed using the same sum reduction algorithm in Step 1.
(3) Step 3—interpolations. Comparing to *ndata* × *ngroup*, the value of *nforc* is relatively small (usually several hundreds at most) and the GPU algorithm should be carefully designed and tested. In the calculation of the local internal coordinates (Eq. (8)), *nint* × *nforc* coordinates are mapped to GPU threads and these threads are organized to *nint* thread blocks containing 256 threads. While in the calculation of the weighted average of the Taylor expansions (Eq. (10)), *nforc* threads are organized to one thread block containing 256 threads. Each thread carries out the calculation of the Taylor expansion about a data point and sum reduction performs the weighted average leading to the potential energy of the grid point. Our testing showed that the algorithm with one thread block has higher computational efficiency than that with multiple thread blocks. The detailed analysis indicates that the higher efficiency is attributable to the small value of *nforc* and to the avoidance of a second sum reduction in the algorithm with multi-thread blocks.

2.3. GPU algorithm for the approximate method

The algorithm described in the last subsection is denoted as the "exact method" since it is the exact implementation of the original interpolation scheme. The exact method could be applied to quantum dynamics and quasi-classical trajectory calculations. Practically, an "approximate" method has also been widely used in many studies [43] of quantum dynamics in order to reduce computational costs. The method is described as follows:

While in the exact method, the *nforc* neighboring data points used in the Taylor expansion should be selected for each grid point individually, the approximate method uses a common set of *nforc* neighboring data points for a group of grid points and the weight calculation is only performed for one grid point. As the weight calculation is the most computational intensive part in the interpolation scheme, the overall computational costs are substantially reduced without significant loss of accuracy. In quantum dynamics studies, all potential energies are pre-calculated and saved in memory. Generally these potential energies are calculated point by point. However, they could also be organized into a number of groups and calculated group by group with each group containing 3^{ndof} grid points and there are 3 neighboring grids for each degree of freedom (DOF). Here *ndof* is the number of DOF in the quantum dynamics calculations. The central grid point is then used to produce the common set of *nforc*

neighboring data points for the Taylor expansions, with the same algorithm in the exact method.

Within the approximate method, the computational costs for weight calculation are greatly reduced. For a molecule containing *natom* atoms and studied with full-dimensional quantum dynamics, the number of grid points in a group is 3^{nint} with $nint = 3 \times natom - 6$ (for example, $3^{nint} = 729$ for 4-atom systems and $3^{nint} = 19,683$ for 5-atom systems) and the number of data points to perform weight calculations is $(3^{nint}-1) \times nforc + ndata \times ngroup$ and $3^{nint} \times ndata \times ngroup$ in the approximate and the exact methods, respectively. As the value of $ndata \times ngroup$ is usually much larger than the value of *nforc*, the superiority of the approximate method is obvious for quantum dynamics calculations. However, for quasi-classical trajectory calculations, the potential energy should be calculated point by point and the approximate method cannot be applied directly.

The GPU algorithm for the approximate method is a combination of the GPU and CPU algorithms introduced above. At first the coordinates of all grid points in a group are copied from the CPU memory to the GPU memory. For the calculation of weights and the selection of *nforc* neighboring data points in Steps (1) and (2), the method employed is the same as the GPU algorithm in the exact method and the calculation is only performed for the central grid point in the group. In the calculation of the weighted average of the Taylor expansions, the 3^{ndof} grid points in the group are mapped to the GPU threads and therefore 3^{ndof} threads are organized as a one-dimensional grid of thread blocks which contain 256 threads. Each thread performs all calculations and produces the potential energy for the mapped grid point. As a result, the potential energies for all 3^{ndof} grid points are copied from the GPU memory to the CPU memory.

3. Results and discussions

Benchmark testing has been performed with four interpolated PES (see Table 1) to assess the performance of the GPU algorithm in both the exact and approximate methods. As shown in Table 1, the four PESs are different in system size and number of data points. As a result, the benchmark testing is expected to provide unbiased assessment about the efficiency of the GPU algorithm in interpolated potential energy calculations. The four PESs employed here are:

(1) The YZCL2 PES for the $H + H_2O \leftrightarrow H_2 + OH$ reaction. The PES is developed with the CCSD(T)/avqz method for energies and the CCSD(T)/avtz method for energy gradients and contains 1972 data points [44]. This PES has been employed in many chemical dynamics studies, both classical and quantum mechanically [45,46].
(2) A PES for the $H + NH_3 \leftrightarrow H_2 + NH_2$ reaction. This PES is constructed with the CCSD(T)/avdz method and contains 2000 data points [47].
(3) A PES for the $H + CH_4 \leftrightarrow H_2 + CH_3$. This PES is a subset of the recently published PES by Zhou and coworkers [43] and contains 2344 data points.
(4) A PES for the $H+NH_3 \leftrightarrow H_2+NH_2$ reaction recently developed in our group. Comparing to PES (2), the *ab initio* calculations were carried out at the CCSD(T)/avtz level and there are 3397 data points in this PES [48].

The computer used in this testing is installed with an Intel E5620 host CPU running with 4 cores at 2.4 GHz and a NVIDIA Tesla C2050 GPU with 3 GB memory and 448 cores running at 1.15 GHz. The PGI CUDA Fortran compiler is used for all the codes and all calculations are carried out with double precision accuracy. When the comparisons are made between CPU and GPU, the CPU code runs in the serial mode.

Table 1
The information of PES.

	System	Reaction	*natom*	*nbond*	*nint*	*ngroup*	*ndata*
1[a]	OH_3	$H + H_2O \leftrightarrow H_2 + OH$	4	6	6	6	1972
2[b]	NH_4	$H + NH_3 \leftrightarrow H_2 + NH_2$	5	10	9	24	2000
3[c]	CH_5	$H + CH_4 \leftrightarrow H_2 + CH_3$	6	15	12	120	2344
4[d]	NH_4	$H + NH_3 \leftrightarrow H_2 + NH_2$	5	10	9	24	3397

[a] Ref. [41].
[b] Ref. [45].
[c] Ref. [46].
[d] Ref. [47].

Table 2
The running time and speedups with the exact method.

	CPU		GPU		Speedup
	Time (s)	Percentage	Time (s)	Percentage	
PES 1, OH_3	Ngrid = 1 476 225				
Step 1. Weight	642	78.58	35	32.41	18.3
Step 2. Selection	162	19.83	51	47.22	3.2
Step 3. Interpolation	13	1.59	22	20.37	0.6
Total	817		108		7.6
PES 2, NH_4	Ngrid = 78 732				
Step 1. Weight	408	91.69	6	46.15	68.0
Step 2. Selection	35	7.87	5	38.46	7.0
Step 3. Interpolation	2	0.45	2	15.35	1.0
Total	445		13		34.2
PES 3, CH_5	Ngrid = 26 244				
Step 1. Weight	1130	93.93	14	63.64	80.7
Step 2. Selection	72	5.99	7	31.82	10.3
Step 3. Interpolation	1	0.08	1	4.55	1.0
Total	1203		22		54.7

Table 2 lists the running times and speedups of the exact implementation with PES (1)–(3) in which the numbers of data points are all around 2000. The running time and speedup for each step are also provided for analyzing the performance of the GPU algorithm applied in different computational tasks.

From Table 2, one can see that Step 1 consumed 78%–94% of the running time in the CPU version and this percentage increases with the system size. The increasing of the percentage is attributed to that *nbond* (used in Step 1) scales quadratically while *nint* (used in Step 3) scales linearly with the system size. Consequently the percentage of the other two steps all decreases with the system size. Similar variations could also been found in the GPU version with Step 1 consuming 32%–64% of the running time. The GPU acceleration apparently has a higher efficiency in this step than in other steps.

The total speedups obtained range from 7 to 54 times, with speedups increasing with the system size. This is important because it demonstrates the promise of GPU acceleration in large systems. Table 2 also lists the speedup of each step in calculating the interpolated potential energies. The speedups obtained in Step 1 range from 18 to 81 and the speedup achieved is 3–10 times in Step 2. For Step 3, the performance of the GPU is the same as or worse than that of the CPU due to the small value of *nforc*.

The running times and speedups with the approximate method are listed in Table 3. As the calculations for the first two steps are carried out only one time for all $ndata \times ngroup$ data points its running time can be neglected in the calculation. Only the total running times and speedups are presented for the approximate implementation. One may notice that the overall speedups with the approximate method range from 2.3 to 3.6 times, which are much smaller than the corresponding values with the exact method. This is because the computational costs of the most intensive weight calculations are greatly reduced by the approximation method employed here. Another interesting observation is that the computational costs increase rapidly with the system size.

Table 3
The running time and speedups with the approximate method.

PES	*ngrid*	CPU (s)	GPU (s)	Speedup
PES 1, OH_3	45,562,500[a]	122	53	2.3
PES 2, NH_4	2,519,424[b]	97	28	3.5
PES 3, CH_5	419,904[c]	71	20	3.6

[a] 62 500 groups with each group containing $3^6 = 720$ grid points.
[b] 6561 groups with each group containing $3^9 = 19\,683$ grid points.
[c] 64 groups with each group containing $3^8 = 6561$ grid points.

Table 4
Comparison of the running time with two NH_4 PES.

	ndata	CPU	GPU	Speedup
PES 2, NH_4	2000	445	13	34.2
PES 4, NH_4	3937	877	22	39.9

We also performed benchmark testing with PESs 2 and 4 to assess the performance of the GPU acceleration with different numbers of data points (see Table 4). The number of data points in PES 4 is about two times larger than that in PES 2. The running time of the CPU implementation with PES 4 is also almost two times that with PES 2. While in the GPU version, the ratio of the running times between PES 4 and PES 2 is about 1.7 and consequently the speedup increases from 34.2 to 39.9. It is noticeable that the GPU acceleration remains for PESs with a larger number of data points.

4. Conclusions

Generally, potential energy calculations in the quantum reaction dynamics studies and the development/validation of PESs consume a large amount of CPU time. In this work, we have presented a GPU-accelerated algorithm for modified Shepard interpolated potential energy calculations and implemented this algorithm with the PGI CUDA Fortran language. Benchmark testing has been performed with four interpolated PESs for both the exact interpolation scheme and an approximate scheme. The results indicated that the GPU-accelerated algorithm could reduce the computational costs dramatically in the potential energy calculations with a modified Shepard interpolation scheme.

For the exact method, the GPU implementation achieved up to a 50-fold speedup over the original CPU implementation and speedup increases with the system size. For the approximate method, the speedups range from 2.3 to 3.6 as the most computational intensive step has been removed. However, although the approximate method is faster than the exact one, it can only be applied in quantum dynamics calculations in which all potential energies are calculated and saved on disk or in memory before the dynamics calculations start. We also tested the performance of the GPU acceleration for the same system with different numbers of points in the PES data set. The results show that the speedup also increases with the increase of the number of data points in the PES.

In conclusion, the speedups achieved in the GPU acceleration increase with both the system size and the number of data points. This work thus demonstrates the promise of GPU acceleration in the studies of chemical reaction dynamics.

Acknowledgments

We gratefully acknowledge support from the National Science Foundation of China (Project Nos. 20921004, 21073229 and 20833007). The authors also acknowledge Prof. Hua Guo (University of New Mexico) and Dr. Jun Zeng (Computist Bio-Nanotech Pty Ltd (Melbourne) and Monash Institute of Pharmaceutical Sciences), for many discussions and revisions of the manuscript.

References

[1] http://www.nvidia.com/object/gpu-applications.html.
[2] M.J. Harvey, G. Giupponi, G. De Fabritiis, J. Chem. Theory Comput. 5 (2009) 1632.
[3] N. Schmid, C.D. Christ, M. Christen, A.P. Eichenberger, W.F. Van Gunsteren, Comput. Phys. Comm. 183 (2012) 890.
[4] W.M. Brown, P. Wang, S.J. Plimpton, A.N. Tharrington, Comput. Phys. Comm. 182 (2011) 898.
[5] I.V. Morozov, A.M. Kazennov, R.G. Bystryi, G.E. Norman, V.V. Pisarev, V.V. Stegailov, Comput. Phys. Comm. 182 (2011) 1974.
[6] B.G. Levine, J.E. Stone, A. Kohlmeyer, J. Comput. Phys. 230 (2011) 3556.
[7] N. Ganesan, B.A. Bauer, T.R. Lucas, S. Patel, M. Taufer, J. Comput. Chem. 32 (2011) 2958.
[8] T.D. Nguyen, C.L. Phillips, J.A. Anderson, S.C. Glotzer, Comput. Phys. Comm. 182 (2011) 2307.
[9] P.H. Colberg, F. Hoefing, Comput. Phys. Comm. 182 (2011) 1120.
[10] D.C. Rapaport, Comput. Phys. Comm. 182 (2011) 926.
[11] A.W. Gotz, M.J. Williamson, D. Xu, D. Poole, S. Le Grand, R.C. Walker, J. Chem. Theory Comput. 8 (2012) 1542.
[12] D. van der Spoel, B. Hess, Wiley Interdiscip. Rev.: Comput. Mol. Sci. 1 (2011) 710.
[13] J.E. Stone, J.C. Phillips, P.L. Freddolino, D.J. Hardy, L.G. Trabuco, K. Schulten, J. Comput. Chem. 28 (2007) 2618.
[14] J.A. Anderson, C.D. Lorenz, A. Travesset, J. Comput. Phys. 227 (2008) 5342.
[15] https://simtk.org/home/openmm.
[16] I.S. Ufimtsev, N. Luehr, T.J. Martínez, J. Phys. Chem. Lett. 2 (2011) 1789.
[17] I.S. Ufimtsev, T.J. Martínez, Comput. Sci. Eng. 10 (2008) 26.
[18] A.G. Anderson, W.A. Godard III, P. Schroeder, Comput. Phys. Comm. 177 (2007) 298.
[19] K. Yasuda, J. Chem. Theory Comput. 4 (2008) 1230.
[20] I.S. Ufimtsev, T.J. Martínez, J. Chem. Theory Comput. 4 (2008) 222.
[21] I.S. Ufimtsev, T.J. Martínez, J. Chem. Theory Comput. 5 (2009) 1004.
[22] I.S. Ufimtsev, T.J. Martínez, J. Chem. Theory Comput. 5 (2009) 2619.
[23] A. Asadchev, V. Allada, J. Felder, B.M. Bode, M.S. Gordon, T.L. Windus, J. Chem. Theory Comput. 6 (2010) 696.
[24] L. Genovese, M. Ospici, T. Deutsch, J.F. Méhaut, A. Neeloy, S. Goedecker, J. Chem. Phys. 131 (2009) 034103.
[25] L. Vogt, R. Olivares-Amaya, S. Kermes, Y.H. Shao, C. Amador-Bedolla, A. Aspuru-Guzik, J. Phys. Chem. A 112 (2008) 2049.
[26] M.A. Watson, R. Olivares-Amaya, R.G. Edgar, T. Arias, A. Aspuru-Guzik, Comput. Sci. Eng. 12 (2010) 40.
[27] R. Olivares-Amaya, M.A. Watson, R.G. Edgar, L. Vogt, Y.H. Shao, A. Aspuru-Guzik, J. Chem. Theory Comput. 6 (2010) 135.
[28] W.J. Ma, S. Krishnamoorthy, O. Villa, K. Kowalski, J. Chem. Theory Comput. 7 (2011) 1316.
[29] A.E. DePrince, J.R. Hammond, J. Chem. Theory Comput. 7 (2011) 1287.
[30] J.N. Murrell, S. Carter, S.C. Farantos, P. Huxley, A.J.C. Varandas, Molecular Potential Energy Functions, Wiley, New York, 1984.
[31] X. Zhang, B.J. Braams, J.M. Bowman, J. Chem. Phys. 124 (2006) 021104.
[32] R. Siebert, P. Fleurat-Lessard, R. Schinke, M. Bittererová, S.C. Farantos, J. Chem. Phys. 116 (2002) 9749.
[33] M.J.T. Jordan, K.C. Thompson, M.A. Collins, J. Chem. Phys. 102 (1995) 5647.
[34] Y. Guo, A. Kawano, D.L. Thompson, A.F. Wagner, M. Minkoff, J. Chem. Phys. 121 (2004) 5091.
[35] D.F.R. Brown, M.N. Gibb, D.C. Clary, J. Chem. Phys. 105 (1996) 7597.
[36] K.C. Thompson, M.J.T. Jordan, M.A. Collins, J. Chem. Phys. 108 (1998) 8302.
[37] R.P.A. Bettens, M.A. Collins, J. Chem. Phys. 111 (1999) 816.
[38] R.P.A. Bettens, M.A. Collins, M.J.T. Jordan, D.H. Zhang, J. Chem. Phys. 112 (2000) 10162.
[39] D.L. Crittenden, M.J.T. Jordan, J. Chem. Phys. 122 (2005) 044102.
[40] M.J.T. Jordan, K.C. Thompson, M.A. Collins, J. Chem. Phys. 102 (1995) 5647.
[41] M.A. Collins, Theor. Chem. Acc. 108 (2002) 313.
[42] M. Harris, S. Sengupta, J. Owens, Parallel prefix sum (Scan) in CUDA in GPU Gems 3 (Chapter 29).
[43] Y. Zhou, B.N. Fu, C.R. Wang, M.A. Collins, D.H. Zhang, J. Chem. Phys. 134 (2011) 064323.
[44] M.H. Yang, D.H. Zhang, M.A. Collins, S.Y. Lee, J. Chem. Phys. 115 (2001) 174.
[45] B.N. Fu, E. Kamarchik, J.M. Bowman, J. Chem. Phys. 133 (2010) 164306.
[46] B.N. Fu, D.H. Zhang, J. Chem. Phys. 136 (2012) 194301.
[47] G.E. Moyano, M.A. Collins, Theor. Chem. Acc. 113 (2005) 225.
[48] Y. Yang, M.H. Yang, An interpolated PES for $H_2 + NH_2 \rightarrow H + NH_3$ reaction to be published elsewhere.