JCTC Journal of Chemical Theory and Computation ARTICLE

pubs.acs.org/JCTC

GPU-Based Implementations of the Noniterative Regularized-CCSD(T) Corrections: Applications to Strongly Correlated Systems

Wenjing Ma,[†] Sriram Krishnamoorthy,[*,‡] Oreste Villa,[‡] and Karol Kowalski[*,¶]

[†]Department of Computer Science and Engineering, The Ohio State University, Columbus, Ohio, United States
[‡]Computational Sciences and Mathematics Division, Pacific Northwest National Laboratory, Richland, Washington, United States
[¶]William R. Wiley Environmental Molecular Sciences Laboratory, Pacific Northwest National Laboratory, Richland, Washington, United States

ABSTRACT: The details of the graphical processing unit (GPU) implementation of the most computationally intensive (T)-part of the recently introduced regularized CCSD(T) (Reg-CCSD(T)) method [Kowalski, K.; Valiev, M. *J. Chem. Phys.* **2009**, *131*, No. 234107] for calculating electronic energies of strongly correlated systems are discussed. Parallel tests performed for several molecular systems show very good scalability of the triples part of the Reg-CCSD(T) approach. We also discuss the performance of the Reg-CCSD(T) GPU implementation as a function of the parameters defining the partitioning of the spinorbital domain (tiling structure). The accuracy of the Reg-CCSD(T) method is illustrated on three examples: the methyfluoride molecule, dissociation of dodecane, and open-shell Spiro cation (5,5′(4*H*,4*H*′)-spirobi[cyclopenta[*c*]pyrrole] 2,2′,6,6′-tetrahydro cation), which is a frequently used model to study electron transfer processes. It is demonstrated that a simple regularization of the cluster amplitudes used in the noniterative corrections accounting for the effect of triply excited configurations significantly improves the accuracies of ground-state energies in the presence of strong quasidegeneracy effects. For methylfluoride, we compare the Reg-CCSD(T) results with the CR-CC(2,3) and CCSDT energies, whereas for Spiro cation we compare Reg-CCSD(T) results with the energies obtained with completely renormalized CCSD(T) method. Performance tests for the Spiro, dodecane, and uracil molecules are also discussed.

1. INTRODUCTION

The widespread use of highly correlated methods in electronic structure calculations is contingent upon the interplay between advances in the theory and the possibility of utilizing ever-growing computer power of emerging architectures. Due to their accuracy, coupled cluster (CC) methods[1−4] have assumed a special position in high-precision calculations for molecular systems.[5−7] The well established family of iterative approximations CCSD (CC with singles and doubles),[8,9] CCSDT (CC with singles, doubles, and triples),[10,11] etc., provide an increasing level of accuracy of resulting energies. Unfortunately, due to the steep numerical complexity, CCSDT applications are very limited. The development of perturbative methods has played a large part to overcome these difficulties. In the CCSD[T][12] and CCSD(T)[13] approaches, the perturbative corrections are constructed in terms of converged CCSD cluster amplitudes. For equilibrium geometries of closed-shell systems the CCSD(T) approach is capable of providing nearly CCSDT level of accuracy. For problems that require knowledge of ground-state potential energy surfaces (PESs) and energies for stretched internuclear geometries, many approaches have been devised to alleviate the problems caused by the divergent nature of the perturbative expansion. There are two main groups of these approaches. The first class of methods is related to alternative perturbative expansions for similarity transformed Hamiltonian,[14−28] while the other class of methods is deeply rooted in the Method of Moments of Coupled Cluster equations.[29−35] Regardless of the origin, all these noniterative methods lead to very accurate results for processes where a single bond is broken. These CCSD(T)-like approaches (and approaches of even higher order) can be also used in highly accurate thermochemistry calculations.[36,37]

Given the importance of CCSD(T)-like methods for high-precision calculations, significant progress has been made toward the development of scalable CC codes[38−47] enabling calculations on large-scale molecular systems. The NWChem[48] implementation of the (T)-part of the CCSD(T) approach has been shown to scale across 250 000 cores,[49] which should be attributed to the natural parallelism of noniterative approaches. Other components of the whole CCSD(T) calculation, Hartree−Fock, 4-index transformation, and CCSD implementations, because of the much smaller task pool, scale across much smaller number of cores.

Equally important to the development of new theoretical algorithms are the implementations on emerging computer architectures. The emergence of general purpose graphic processing units (GPGPUs) has revolutionized computational science by making available an unprecedented amount of computing capability. There are several examples of successful development of GPU-based software in computational chemistry.[50−64] While GPUs have been employed in accelerating scientific calculations in the past, programming them required mapping the application to the graphic processing pipeline, a challenging task. The advent of higher-level programming support, such as through CUDA and OpenCL, has made it easier to exploit their potential. The

Received: December 16, 2010
Published: April 11, 2011

widespread availability of GPU-accelerated systems, ranging from workstations to supercomputers, stresses the need to develop algorithms to effectively utilize them, and the work's impact on researchers with access to varying computing resources. Recently, we discussed the first GPU implementation of the CCSD(T) method,[65] which due to its large flop count is ideally suited for this type of computer architectures. In this paper, we will discuss accuracies and numerical performance of the GPU implementation of the regularized CCSD(T) approach (Reg-CCSD(T)),[66] which can be used in calculations for strongly correlated systems. In analogy to the CCSD(T) method, the Reg-CCSD(T) approach is characterized by the same $n_o^3 n_u^4$ (n_o and n_u designate the number of occupied and unoccupied orbitals, respectively) numerical complexity. This high numerical overhead is associated with calculating triples correction or (T)-part in short, which is especially challenging for systems with large number of correlated electrons and large virtual space (assuming that two (T) calculations were performed with the same number N of correlated orbitals, $N = n_o + n_u = 1000$, but for two different values of n_o, $n_o = 20$, and $n_o = 400$, the latter calculations is more than 3 orders of magnitude more expensive compared to the $n_o = 20$ case). Our discussion will be based on the calculations for several challenging systems: C−F bond elongation in the methylfluoride, dissociation of dodecane, $C_{12}H_{26}$, into $C_{11}H_{23}$ and CH_3, and the mixed valence system 5,5′(4*H*,4*H*′)-spirobi[cyclopenta[*c*]pyrrole]2,2′,6,6′-tetrahydro cation (or Spiro cation for short).[67] While the methylfluoride and dodecane molecules epitomize common problems the CCSD(T) approach stumble into when both static and dynamic correlation effects play an equally important role, the Spiro molecule is frequently used in fundamental studies of charge transport processes and poses a significant challenge even for multireference perturbative methods.[67−70] The paper is organized as follows: in section 2, we give a brief overview of regularization techniques for CC theory, in section 3 details of the GPU implementation of the most expensive (T)-part are described. In section 4 we discuss the quality of the potential energy surfaces (PESs) for methylfluoride, $C_{12}H_{26}$ dissociation, and Spiro cation and illustrate the parallel performance on the example of calculations for Spiro cation and uracil.

2. THEORY

In this section, we present only the salient features of the regularized methods derived from the generating functional expansion of ref 71, where it was demonstrated that the exact energy (E) for the ground electronic state can be expanded as

$$E = E^{(A)} + \sum_{J;J\neq 0} \overline{M}_J^{(A)} \left[\frac{\partial}{\partial S_J} W(\Sigma, S) \right] \Bigg|_{S^{(A)} = T^{(A)}; S^{(R)} = 0} \quad (1)$$

Here the energy $E^{(A)}$ is an approximate energy obtained in approximate CC calculations (CC-A) defined by the approximate cluster operator $T^{(A)}$

$$T^{(A)} = \sum_{n=1}^{m_A} T_n \quad (2)$$

$$T_n = \sum_{i_1 < ... < i_n; a_1 < ... < a_n} t_{a_1 ... a_n}^{i_1 ... i_n} X_{a_1}^+ ... X_{a_n}^+ X_{i_n} ... X_{i_1} \quad (3)$$

The i_1, ..., i_n (a_1, ..., a_n) indices refer to the occupied (unoccupied) spinorbitals in the reference function $|\Phi\rangle$ and the X_p^+ (X_p) operators are creation (annihilation) operators for electrons in p-th single particle state. The cluster operator includes excitations of rank equal or lower than m_A. In practice $m_A \ll N$ (N stands for the total number of correlated electrons). In expansion (eq 1) the Σ operator corresponds to the exact cluster operator ($\Sigma = \Sigma_{n=1}^N \Sigma_n$, where the n-tuply excited many-body component Σ_n of Σ, in analogy to (eq 2) is defined by the $\Sigma_{a_1 ... a_n}^{i_1 ... i_n}$ amplitudes). The auxiliary cluster operator S ($S = \Sigma_{n=1}^N S_n$), introduced in refs 66 and 71 is chosen in such a way that the auxiliary wave function $e^S|\Phi\rangle$ is in a close vicinity of the approximate wave function $e^{T^{(A)}}|\Phi\rangle$. We also assume that the exact wave function falls into the same vicinity (these assumptions are critical from the point of view of convergence properties of the connected form of the generating functional). The $S^{(A)}$ part of the S operator in eq 1 is defined by the excitations used to define the $T^{(A)}$ operator, while the $S^{(R)}$ part of the S operator contains higher excitations. The reference function $|\Phi\rangle$ is usually represented by the Hartree−Fock (HF) determinant. In eq 1 the quantities $\overline{M}_J^{(A)}$ correspond to the matrix elements of the moments operator (see ref 71 for details) and are defined as

$$\overline{M}_J^{(A)} = \langle \Phi_{i_1 ... i_n}^{a_1 ... a_n} | \bar{H}^{(A)} | \Phi \rangle \quad (4)$$

where the string convention of ref 32 is invoked, that is, the nonzero string J corresponds to the excitation designated by the ordered set of occupied/unoccupied indices $\{i_1 < ... < i_n; a_1 < ... < a_n\}$. The similarity transformed Hamiltonian, $\bar{H}^{(A)}$, is defined as $\bar{H}^{(A)} = e^{-T^{(A)}} H e^{T^{(A)}}$, where the H operator represents the electronic Hamiltonian. The $\bar{H}^{(A)}$ operator contains connected diagrams only. The central role in eq 1 is played by the so-called generating functional $W(\Sigma,S)$, which is defined as a connected part of the overlap between the exact CC wave function and auxiliary wave function defined by the auxiliary cluster operator, that is,

$$\begin{aligned} W(\Sigma, S) &= \langle \Phi | (e^{\Sigma^+} e^S)_C | \Phi \rangle = \ln(\langle \Phi | e^{\Sigma^+} e^S | \Phi \rangle) \\ &= \ln(1 + \gamma(\Sigma, S)) \end{aligned} \quad (5)$$

where subscript "C" designates connected part of a given expression and the $\gamma(\Sigma,S)$ function corresponds to the correlated part of the overlap between auxiliary and exact CC functions, that is, $\gamma(\Sigma,S) = \langle \Phi | (e^{\Sigma^+} - 1)(e^S - 1) | \Phi \rangle$, where Σ^+ is a Hermitian conjugate of the Σ operator. The features of the generating functional fully determine the basic features of the expansion in eq 1. In particular, the connectedness of $W(\Sigma,S)$ implicates the connectedness of expansion eq 1 (assuming the connectedness of cluster amplitudes). The formula (eq 1) was derived using the Taylor expansion for $\ln(1 + \gamma(\Sigma,S))$, which is only valid when the condition

$$|\gamma(\Sigma, S)| < 1 \quad (6)$$

is satisfied. This fact limits the applicability of the expansion in (eq 1) to weakly correlated systems. To maintain the form of the expansion in eq 1 in the strong interaction regime one has to artificially redefine certain parameters of the generating functional expansion in order to decrease the value of $|\gamma(\Sigma,S)|$. This type of procedure is commonly referred to as the regularization procedure. There is great flexibility as far as the choice of the regularization procedure is concerned. For the sake of simplicity, we pursue perhaps the most obvious choice corresponding to the regularization of the Σ operator in the generating functional $W(\Sigma,S)$.[71] To address this issue we adopted ideas similar to the

Tikhonov regularization, which have recently been explored by Taube and Bartlett[72] in the context of ill defined linear CC equations for quasidegenerate systems. In our procedure, described in ref 66, the regularized Σ operator (Σ_{reg}) is obtained by solving modified CC equations, which use the regularized form of the Hamiltonian (H^{reg}) defined as

$$H^{\text{reg}} = H + \omega^2 N_u \tag{7}$$

where the N_u operator is defined as $N_u = \Sigma_a X_a^+ X_a$, and represents particle number operator for particles in virtual states. The presence of this operator in the equations for regularized cluster operators Σ_{reg} is similar to the level shift techniques.

The described regularization scheme was used to define the due-to-triples corrections to energies obtained with the CCSD approach ($m_A = 2$, $T^{(A)} = T_1 + T_2$, $E^{(A)} = E^{\text{CCSD}}$). For this purpose, we used the following form of the generating functional:

$$\bar{W}(\Sigma_{\text{reg}}, S_3) \simeq \left\langle \Phi \middle| \left\{ \left(\Sigma_{\text{reg},3} + \Sigma_{\text{reg},1}\Sigma_{\text{reg},2} + \frac{1}{6}(\Sigma_{\text{reg},1})^3 \right)^\dagger S_3 \right\}_C \middle| \Phi \right\rangle \tag{8}$$

where the contributions containing only S_1 and S_2 were neglected because singly ($M_{J_1}{}^{\text{CCSD}}$) and doubly excited moments ($M_{J_2}{}^{\text{CCSD}}$) are zeroed in the process of solving CCSD equations. By substituting eq 8 into eq 1, we can derive the so-called Reg-GF(T) approximation (see ref 66)

$$E^{\text{Reg-GF(T)}} = E^{\text{CCSD}} + \left\langle \Phi \middle| \left\{ \left(\Sigma_{\text{reg},3} + \Sigma_{\text{reg},1}\Sigma_{\text{reg},2} + \frac{1}{6}(\Sigma_{\text{reg},1})^3 \right)^\dagger M_3^{\text{CCSD}} \right\}_C \middle| \Phi \right\rangle \tag{9}$$

where $\Sigma_{\text{reg},1}$, $\Sigma_{\text{reg},2}$, and $\Sigma_{\text{reg},3}$ are the regularized Σ amplitudes. While the $\Sigma_{\text{reg},1}$ and $\Sigma_{\text{reg},2}$ amplitudes are approximated by singly and doubly excited cluster amplitudes obtained by solving CCSD equations with regularized form of the electronic Hamiltonian H^{reg}, the $\Sigma_{\text{reg},3}$ amplitudes ($\tilde{\Sigma}^{i_1 i_2 i_3}_{a_1 a_2 a_3}$) are obtained in a perturbative manner

$$\tilde{\Sigma}^{i_1 i_2 i_3}_{a_1 a_2 a_3} = \frac{\bar{M}^{i_1 i_2 i_3}_{a_1 a_2 a_3}(\Sigma_{\text{reg},1}, \Sigma_{\text{reg},2})}{\varepsilon_{i_1} + \varepsilon_{i_2} + \varepsilon_{i_3} - \varepsilon_{a_1} - \varepsilon_{a_2} - \varepsilon_{a_3} - 3\omega^2} \tag{10}$$

where $\bar{M}^{i_1 i_2 i_3}_{a_1 a_2 a_3}(\Sigma_{\text{reg},1}, \Sigma_{\text{reg},2})$ are triply excited moments for regularized CCSD equations. In ref 66, the regularized version of the CCSD(T) approach (Reg-CCSD(T)) was introduced

$$E^{\text{Reg-CCSD(T)}} = E^{\text{CCSD}} + \langle \Phi | (V_N \Sigma_{\text{reg},2} + V_N \Sigma_{\text{reg},1})^+ R_0^{(3)}(\omega^2)(V_N T_2) | \Phi \rangle \tag{11}$$

where V_N is two-body part of the electronic Hamiltonian in normal product form and $R_0^{(3)}(\omega^2)$ is the ω^2-dependent resolvent defined by eq 33 of ref 66. The Reg-CCSD(T) approach can be easily implemented using existing CCSD(T) implementations. In the current form both Reg-GF(T) and Reg-CCSD(T) approaches as ω^2-dependent methods should be classified as semiempirical approaches. Despite its simplicity, the Reg-CCSD(T) method offers considerable improvements upon the CCSD(T) results especially for strongly correlated systems at the same $n_o^3 n_u^4$ numerical cost as the genuine CCSD(T) method. We believe that efficient parallel GPU implementation of the triples part of the regularized CCSD(T) method has potential to evolve into a tool that is capable of providing credible predictions for strongly correlated systems. In the forthcoming sections, we will give details of our (T) implementations and discuss their parallel performance.

3. IMPLEMENTATION DETAILS

In this section, we present the evaluation of the noniterative triples correction on GPUs. It involves a code-generation based approach to generating CUDA code from a high-level specification of tensor contractions. Several optimizations are identified in mapping the tensor contractions to the resources in a GPU. We then develop a hybrid implementation that effectively utilizes the cores and GPU accelerators available in a cluster of SMP nodes with GPUs.

To calculate triples correction two general type quantities, which appear on the right and on the left of the $R_0^{(3)}(\omega^2)$ resolvent, need to be calculated:

$$\begin{aligned}\langle \Phi^{abc}_{ijk} | V_N T_2 | \Phi \rangle = {} & v^{ij}_{ma} t^{mk}_{bc} - v^{ij}_{mb} t^{mk}_{ac} + v^{ij}_{mc} t^{mk}_{ab} \\ & - v^{ik}_{ma} t^{mj}_{bc} + v^{ik}_{mb} t^{mj}_{ac} - v^{ik}_{mc} t^{mj}_{ab} + v^{jk}_{ma} t^{mi}_{bc} - v^{jk}_{mb} t^{mi}_{ac} + v^{jk}_{mc} t^{mi}_{ab} \\ & - v^{ei}_{ab} t^{jk}_{ec} + v^{ei}_{ac} t^{jk}_{eb} - v^{ei}_{bc} t^{jk}_{ea} + v^{ej}_{ab} t^{ik}_{ec} - v^{ej}_{ac} t^{ik}_{eb} + v^{ej}_{bc} t^{ik}_{ea} \\ & - v^{ek}_{ab} t^{ij}_{ec} + v^{ek}_{ac} t^{ij}_{eb} - v^{ek}_{bc} t^{ij}_{ea}, \; (i < j < k, a < b < c)\end{aligned} \tag{12}$$

and

$$\begin{aligned}\langle \Phi^{abc}_{ijk} | V_N T_1 | \Phi \rangle = {} & v^{ij}_{ab} t^k_c - v^{ij}_{ac} t^k_b + v^{ij}_{bc} t^k_a - v^{ik}_{ab} t^j_c + v^{ik}_{ac} t^j_b \\ & - v^{ik}_{bc} t^j_a + v^{jk}_{ab} t^i_c - v^{jk}_{ac} t^i_b + v^{jk}_{bc} t^i_a, \; (i < j < k, a < b < c)\end{aligned} \tag{13}$$

where T_1 and T_2 refer either to genuine or regularized CCSD amplitudes. The $i, j, k, l, m, n, \ldots$ ($a, b, c, d, e, \ldots$) indices designate occupied (unoccupied) spinorbitals. Of the two terms described by the above equations, the first one (section 3) contributes to the $n_o^3 n_u^4$ scaling. Tensors corresponding to 2-electron integrals and doubly excited amplitudes are assumed to be antisymmetric in all pairs of lower and upper indices. In order to provide granularity for the parallel Tensor Contraction Engine[38] generated codes, the whole spinrobital domain is partitioned into smaller pieces called *tiles* which contain several spinorbitals of the same spatial- and spin-symmetry. The maximum number of elements in the tile is often referred to as the *tilesize*. This partitioning induces partitioning or block-structure of all tensors used in the CC calculations, including: amplitudes, recursive intermediates, integrals, and residual vectors. In the parallel implementation of the (T)-part each core takes care of different set of projections defined by tiles: $[i]$, $[j]$, $[k]$, $[a]$, $[b]$, $[c]$, i.e., each core generates on-the-fly the set of $\langle \Phi^{abc}_{ijk} | V_N T_2 | \Phi \rangle$ and $\langle \Phi^{abc}_{ijk} | V_N T_1 | \Phi \rangle$ projections with $i \in [i]$, $j \in [j]$, $k \in [k]$, $a \in [a]$, $b \in [b]$, $c \in [c]$. These projections are stored on the six-dimensional matrices $P3$ ($\langle \Phi^{abc}_{ijk} | V_N T_2 | \Phi \rangle$)and $R3$ ($\langle \Phi^{abc}_{ijk} | V_N T_1 | \Phi \rangle$). In our implementation, this condition is replaced by the do -loop structure for each tile corresponding to ($[i] \le [j] \le [k]$) and ($[a] \le [b] \le [c]$). This incurs a small amount of redundancy at the boundary of the conditionals (when the equality is satisfied). This has been shown in practice to be very small as compared to the total work done and is correctly incorporated through the use of appropriate constant coefficients. For example, $P3$ is defined as the following matrix

$$P3 \equiv P3(\dim[a], \dim[b], \dim[c], \dim[i], \dim[j], \dim[k]) \tag{14}$$

where $\dim[i], \ldots, \dim[c]$ are the dimensions of the corresponding tiles (they will be denoted *id, jd, kd, ad, bd, cd*). Therefore, the local

Table 1. Architectural Comparison of Tesla T10 and T20 Series

	Tesla T10	Tesla T20
num. multiprocessors (SM)	30	14
num. cores per SM	8	32
SM clock frequency	1.3 GHz	1.15 GHz
single precision peak GFLOPS	933	1030
double precision peak GFLOPS	78	515
memory frequency	800 MHz	1.5 GHz
memory bandwidth	102 GB/sec	144 GB/sec
memory interface	512 bit	384 bit
shared memory	16 KB	16 KB/48 KB
L1 cache		48 KB/16 KB
L2 cache		768 KB

memory requirement for storing $P3$ and $R3$ matrices is defined by *tilesize*^{6}. If *tilesize* equals 20 this is equivalent to 0.48 GB (in recently developed algorithm for (T)-part of TCE these tensors can be "sliced" along the first two dimensions, which lead to a less intensive use of the local memory and effectively larger *tilesize* can be used in the (T) calculations). Because the total flop count on each core associated with forming $P3$ and $R3$ tensors is proportional to *tilesize*^{6} $*n_u$, where n_u stands here for the total number of correlated virtual spinorbitals, this type of calculation is ideally suited to take advantage of GPU accelerators. The whole process is split into number of smaller tasks, where the summation goes over indices from a single tile, for example

$$P3(a,b,c,i,j,k) - = \sum_{e\,\in\,[e]} V(a,b,e,i)*T2(j,k,e,c)$$

$$(i \in [i], j \in [j], k \in [k], a \in [a], b \in [b], c \in [c]) \qquad (15)$$

where $V(a,b,e,i)$ and $T2(j,k,e,c)$ are 2-electron integrals and doubly excited amplitudes tensors. For this elementary task the flop count is equal to *tilesize*^{7}, which for *tilesize* = 20 corresponds to 1.2 GF. We expect, that the utilization of the GPU accelerators should lead to considerable speedups in the case of large numerical load, which is created by the use of larger tiles.

3.1. GPU Architecture and Execution Model. CUDA[73] is a language extension to C developed by NVIDIA to program GPGPUs. GPU devices that support the CUDA programming environment consist of several multiprocessors (SMs), each with a fast shared memory, a constant cache, a single instruction unit, and multiple processor cores.

The CUDA programming model views the GPU as an accelerator to which parts of the computation, referred to as kernels, are offloaded by the host CPU. A kernel consists of a grid of thread blocks, with each thread block consisting of multiple threads. All threads in a kernel invocation can access the global memory, which is persistent across kernel invocations. A thread-block is mapped to a multiprocessor (MP). The shared memory associated with the MP is only accessible to threads in a thread block, and is not persistent across thread blocks. Given that each MP consists of a single instruction unit, effective utilization requires that all threads execute the same instruction whenever possible. Conditional branches that diverge among the threads can greatly reduce achieved performance. The execution in a thread block in which few threads are performing work, referred to as thread block under-utilization, also inhibits performance. Each thread can identify its position in the thread block, and in the grid of thread blocks through implicit variables. These are the only distinguishing identifiers of a given thread, and are used to encode all thread-specific computation.

Table 1 summarizes the specification of the two GPU architectures that were the target of our optimizations. Tesla T10 contains the GF100 GPU processor, while Tesla T20, known as "Fermi", is the latest series of graphic card products by NVIDIA. The peak double precision performance has improved from previous generations by a factor of 6.6 (a peak of ~480 GFlops). However, the memory bandwidth and the clock speed for device memory accesses, and the data transfer rates between the host and the device, have improved by a much smaller factor. Thus, we can expect that accesses to device memory and data transfers from host memory can be even more of a bottleneck with T20 cards. Fermi has a much larger shared memory with respect to previous generations, that can also act as a Level 1 data cache. It can be configured as 48 KB shared memory and 16 KB L1 cache, or 48 KB L1 cache and 16KB shared memory. Shared memory is still allocated per thread block. The register file is also much larger. Unlike previous cards, a Level 2 cache is also available.

3.2. Multi-Dimensional Tensor Contractions. We are interested in a direct implementation of optimized tensor contractions on CUDA-programmable GPU devices. A tensor contraction can be viewed as a generalized multidimensional matrix multiplication. Often, it is transformed into a regular matrix multiplication operation through transposition operations. However, such an approach might not most effectively utilize the available hardware resources.

While matrix multiplication can be optimized assuming large dimensions lending themselves to tiling for every level of the memory hierarchy to ensure that the operation is computation-bound, the large dimensionality of the tensors could often result in each dimension being relatively small. The small size of each dimension interferes with achieving good locality. The encoding of the indices into the thread coordinates incurs high index computation overhead. The small size of the common dimension results in the equivalent of highly rectangular matrix multiplication operations, which are harder to optimize than the square versions typically employed in benchmark studies.

3.3. CUDA-Targeted Tensor Contraction Implementation. In this section, we present the various optimizations performed, with illustrative code snippets, in generating an efficient CUDA implementation of a given tensor contraction. We begin with optimizations that are generally applicable to CUDA-capable devices, in particular both T10 and T20 NVIDIA GPU cards. This discussion is followed by a presentation of the optimizations specifically targeted at the T20 cards with their latest generation Fermi GPUs. We consider a typical tensor contraction specification given by the following example:

$$\texttt{P3(b,c,a,i,k,j)} - = \sum_{l\,\in\,[l]} \texttt{V2(l,a,i,j)*T2(l,k,c,b)}$$

which is taken from the TCE generated code for the noniterative (T) correction (slightly different convention is used in eq 15. `P3`, `V2`, and `T2` are tensors, and `a`, `b`, `c`, `i`, `k`, `j`, and `l` are particle or hole indices as appropriate). Note that the GPU implementation faithfully reproduces the computation structure on the CPU, ensuring correctness of the results produced and avoiding any potential redundant computation, other than those in the CPU version.

3.3.1. Memory Management. A typical computation in the application involves numerous calls to the sequential tensor contraction. Allocating and deallocating CUDA memory, both on the host and on the device, for every kernel invocation would be expensive. We implemented a memory manager that serves allocation requests from previously allocated memory that is current not being used. The memory allocation for the arrays is as shown. The `getGPUmem` calls are the wrappers implemented for more efficient memory management

```
double  *P3_d=getGPUmem(size_of_P3);
double  *T2_d=getGPUmem(size_of_T2);
double *V2_d=getGPUmem(size_of_V2);
```

In the sample code above, `size_of_P3` denote the size of the array `P3`. It reuses the previously freed GPU memory if it is enough for P3.

3.3.2. Kernel Arguments. Accessing portions of multidimensional arrays in GPU memory requires computation of the strides along the different dimensions. This scalar arithmetic is redundantly executed by each thread and cannot be overlapped with memory or floating-point operations on these systems. We therefore compute the strides in the host CPU, as shown below, and pass them as arguments to the kernel. Note that P3 is stored as a one-dimensional array. The offsets for other arrays are calculated in a similar fashion.

```
 /* parameters calculated in host: */
 P3_offset_c = bd;
 P3_offset_a = bd*cd;
 ...
/* offset computation in GPU kernel: */
int offset_P3(b,c,a,i,k,j) {
   return b + P3_offset_c*c +
    P3_offset_a*a + ··· +
 P3_offset_j*j;
 }
 /* offset_T2(l,k,c,b)
 and offset_V2(l,a,i,j) are
   similarly defined */
```

3.3.3. Encoding Thread-Block Specific Arguments. The indices to be operated upon by a given thread are decoded from its coordinates in the thread block and thread block grid by modulo and division operations. These operations are not very efficient on GPUs as they require the use of Special Function Units. To reduce such operations, the indices of the output matrix are mapped to a two dimensional thread grid, according to the two input matrices, which means the indices from the first input matrix are mapped to the y dimension of the thread grid, and indices from the second input matrix are mapped to the *x* dimension, resulting in the configuration below.

```
dim2 dimGrid(ceil(kd/BLOCK_DIM_Y)*cd
*bd,ceil(ad/BLOCK_DIM_X)*id*jd);
```

3.3.4. Index Combining. In some tensor contractions, a sequence of indices might occur in the same order in every occurrence. These sequences of indices can be replaced by a combined index. This optimization reduces index computation overhead while improving thread block utilization. While combining of indices might not be possible for all tensor contractions, there is no negative side-effect and we employ this optimization where possible. For the illustrative tensor contraction, by using index combining, we set *x* as the multiplication of *a* and *i*, therefore it results in the following:

```
P3(b,c,x,k,j) −  = T2(l,k,c,b)*V2(l,x,j)
```

3.3.5. Dimension Flattening. Tiling of the loops in a tensor contraction enables different threads to cooperate in data movement, and enables maximal reuse of the transferred data to minimize data transfer per floating point operation. The tensor contraction could be implemented as a sequence of matrix multiplication on possibly strided data. In the example that we have been using through this section, all threads in one thread block work on elements `P3(b,c,aT:aT+16,i,kT:kT+16,j)`, assuming one block has 16 × 16 threads. This works well when the tiled dimensions are large or match the thread block configuration. The application tile sizes encountered at runtime often do not match the fixed thread block size chosen at compile time. The result is the execution of numerous thread-blocks with fewer threads than available, with ensuing poor utilization of threads blocks and poor performance. For example, a dimension size of 17 for a thread block size of 16 results in two thread blocks, with total only 17 of total 32 threads being used.

To solve this problem, we optimize the code by *dimension flattening*. This is illustrated in Figure 1. In the dimension-flattened version, we group the indices of the output array into two dimensions, according to the origin of the indices. In the example we have been using, the two groups are `(k,c,b)` and `(a,i,j)`. Tiling is done on the grouped indices, instead of a single index as in the original algorithm. Thus, different threads in one thread block could compute output elements that have more than two different indices. The order in which the indices are flattened has an impact on the data access costs for the arrays. We considered flattening orders that favors inputs and outputs and found the order favoring the inputs to be most effective. This is the order used in the experiments.

3.3.6. Pipelined Execution. In a parallel execution of coupled cluster calculations, the data is transferred from remote to local memory for processing by the host CPU. This data needs to be transferred to the GPU memory prior to kernel invocation and the result transferred back to host memory. CUDA enables overlap of this data transfer with kernel invocation through the use of streams. To ensure efficient data transfer, we use the outermost dimension of the output tensor as the streaming index. The number of kernel invocations is the same as the value of the streaming index. dimension.

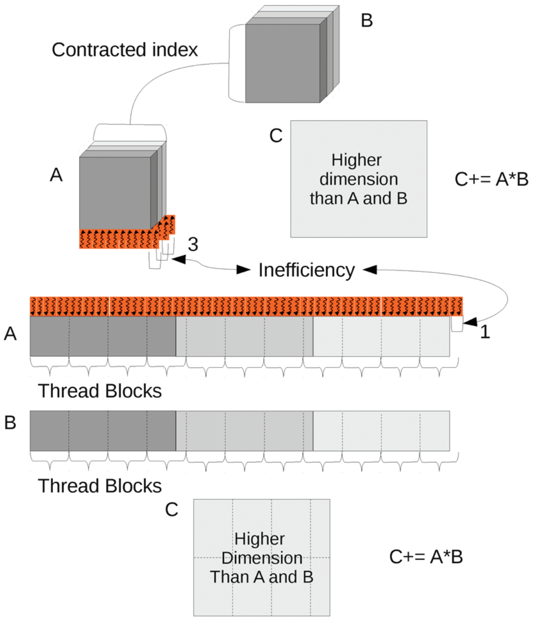

Figure 1. Illustration of dimension flattening. The solid lines correspond to different two-dimensional regions. The dotted lines correspond to the mapping of the data blocks to the thread blocks. A and B matrices are implicitly flattened into two-dimensional arrays. This implicit flattening is used to map the work to be performed to the thread blocks. As shown in the figure, this results in most thread-blocks being fully utilized.

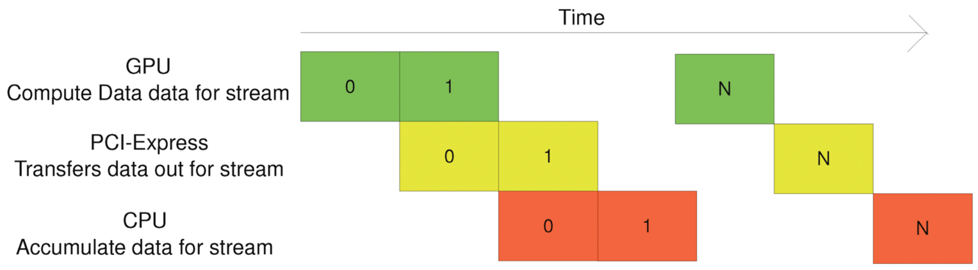
Time
GPU
Compute Data data for stream
PCI-Express
Transfers data out for stream
CPU
Accumulate data for stream
0
1
N

Figure 2. Three-stage pipelined execution with CPU and GPU concurrently participating in the contraction.

Building upon the streaming strategy we also implement a pipelined approach with the host participating in the contraction by performing the accumulation and thus avoiding the initial copy of the large tensors in the GPU memory. With this approach we traded off data transfer with computation, allowing a better utilization of the available resources and reducing the amount of traffic on the PCI-Express bus. This three-stage pipelined design ensures that the different components of the architecture: the CPU, the GPU, and the PCI-express bus are all utilized to minimize the execution time. Figure 2 shows the concept above-discussed, assuming the total amount of streams to transfer equal to *N*.

3.3.7. Hybrid Execution. The above optimization, with the GPU and CPU collaborating in performing a single contraction, is a clear instance of hybrid execution. However, general high-performance SMP nodes (as the one used in our experiments) have more CPU computing cores than GPUs. As each GPU in an SMP node requires a separate core that drives the kernel execution and participates in its pipelined processing, the "spare" cores can be used to compute serially other contractions. This can be seen as a second level of hybrid execution. We implemented this approach introducing a high level load balancer to distribute the contractions. This design maximally utilizes the computational resources available to reduce the time to solution.

3.3.8. Optimizations Specific to the Fermi (T20) GPU Architecture. The CUDA code generator had to be modified to adapt to the different architectural trade-offs presented by the Fermi GPU architecture. Here we identify the specific architectural differences and the adaptations we undertook to account for them.

3.3.8.1. Elimination of PCI Express Data Transfer. As discussed above, the data transfer across the PCI express can be effectively overlapped with the kernel execution on the GPU, through different pipelining approaches. The greater factor of improvement in the computation rate on the Fermi GPU architecture as compared to the improvement in memory bandwidth results in the execution being bound by data transfer, even with the pipelining approaches. To achieve the best execution time, we modified the application to keep the intermediate in memory, across tensor contractions. The inputs need to be transferred into the GPU. The intermediate is computed and translated into a scalar contribution to the energy value, which is transferred back to the host memory. The compute structure is shown below:

```
double *P3_s = getGPUmem(size_inter-
mediate);
/*intermediate tile */
double *P3_d = getGPUmem(sizeof(
double));
/*energy scalar*/
ccsd_t_single(P3_s);
/*Moves inputs to GPU memory;*/
/*Result updated in GPU memory*/
ccsd_t_doubles_1((P3_s); /*as above*/
ccsd_t_doubles_2(P3_s); /* -do- */
compute_energy(P3_s, P3_d); /* -do- */
transfer_to_cpu(P3_d); /*one element */
```

3.3.8.2. Register Tiling. Fermi includes a wider register file and larger shared memory enabling register tiling, which improves data reuse and reduces the data transfer with the GPU memory hierarchy. We modified our implementation such that each thread contributes to 16 output elements, rather than 1 as in the original algorithm. All contributions are stored in 16 double precision registers and finally written back to GPU memory.

3.3.8.3. Scalar Optimizations. While the double precision performance is improved by a factor of 6.6 in Fermi, the scalar instructions do not see a corresponding improvement in performance. However, register tiling results in each thread performing more computation enabling scalar optimizations across calculations for the 16 output elements. Each write to the GPU memory is enclosed in a boundary check to ensure that thread has a valid contribution to make, in cases when the tile being executed is smalled the thread block size. We coalesce the condition checks to minimize them in the common case. The original boundary checking order for four output elements is as shown below:

```
if(thread_y<total_y)
  p3[offset_1]+=tlocal1
if(thread_y+16<total_y)
  p3[offset_2]+=tlocal2;
if(thread_y+16*2<total_y)
  p3[offset_3]+=tlocal3;
if(thread_y+16*3<total_y)
  p3[offset_4]+=tlocal4;
```

The modified order we employ is shown below:

```
if(thread_y+16*3<total_y)
  {p3[offset_1]+=tlocal1;
  p3[offset_2]+=tlocal2;
  p3[offset_3]+=tlocal3;
  p3[offset_4]+=tlocal4;
  }
 else if(thread_y+16*2<total_y)
  {p3[offset_1]+=tlocal1;
  p3[offset_2]+=tlocal2;
  p3[offset_3]+=tlocal3;
  }
 else if(thread_y+16<total_y)
  {p3[offset_1]+=tlocal1;
  p3[offset_2]+=tlocal2;
  p3[offset_3]+=tlocal3;
  }
 else if(thread_y<total_y)
  {p3[offset_1]+=tlocal1;
  }
```

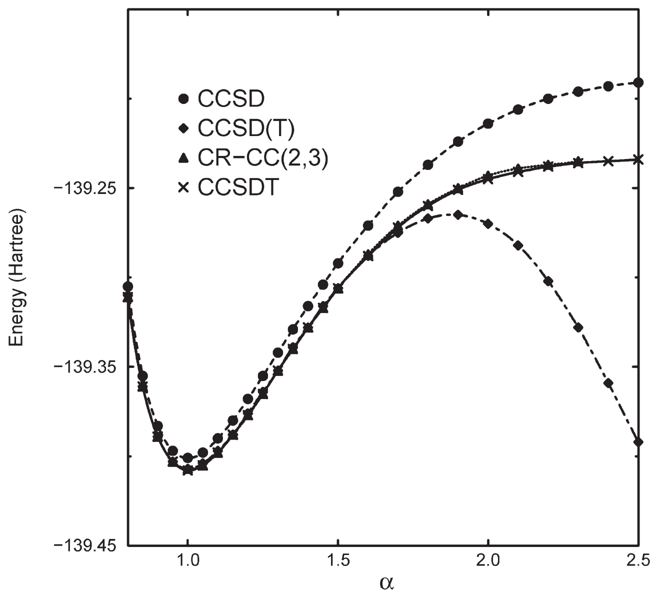

Figure 3. CC energies for methylfluoride system as a function of C−F elongation obtained with the cc-pVDZ basis set (see text for details).

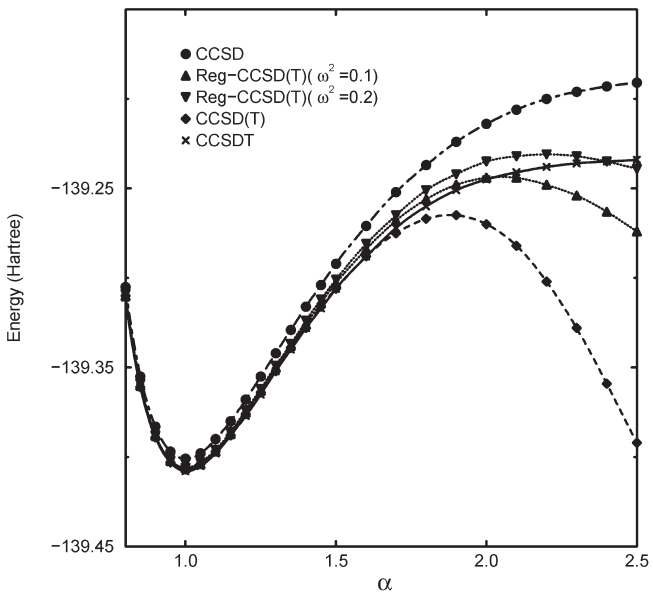

Figure 4. Regularized CC energies for methylfluoride system as a function of C−F elongation obtained with the cc-pVDZ basis set (see text for details).

Note that the above optimizations do not have an impact on the older GPU architecture, while being crucial to maximize performance on Fermi.

4. RESULTS AND DISCUSSION

In this section, we discuss the quality of the Reg-CCSD(T) method when applied to several challenging molecular systems and provide several illustrative performance examples of our GPU implementation of the noniterative triples Reg-CCSD(T) correction.

For the H_3CF system we used the cc-pVDZ basis set,[74] for the dodecane we employed 6-311G basis set,[75] whereas for Spiro cation we used Sadlej's basis set (POL1),[76] which is composed of 486 atomic basis set functions. The geometry of the methylfuoride molecule was optimized with the B3LYP method[77] using cc-pVTZ basis set.[74] The geometry of Spiro cation is as discussed in ref 78. The geometry of dodecane was optimized with the B3LYP approach using cc-pVTZ basis set. In all calculations reported here core electrons were not correlated.

It is well-known that various renormalized CC approaches can provide correct description of processes involving single bond breaking. The first two examples are to illustrate the performance of the Reg-CCSD(T) methods for these processes. We start our discussion from the ground-state singlet potential energy surface of the methylfluoride molecule H_3CF as a function of the C−F bond elongation. The B3LYP/cc-pVTZ equilibrium values of the $H-C$ ($R_{H-C}(eq)$) and $C-F$ ($R_{C-F}(eq)$) bond lengths are equal to 1.09021 and 1.38655 Å, respectively, whereas the equilibrium H−C−H angle is equal to 109.878 deg. The geometry of the system is defined by a single parameter α (see Figure 3 and Figure 4), which defines the C−F distance (R_{C-F} as $R_{C-F} = \alpha R_{C-F}(eq)$). In the present studies we consider the set of geometries with corresponding α's falling into the 0.8,2.5 interval.

It has been already shown[79] that for larger internuclear distances the restricted Hartree−Fock (RHF) determinant is a rather poor choice of the reference, which results in divergent behavior of perturbative triples estimates of the CCSD(T) method. These problems are eliminated by the iterative inclusion of connected triply excited clusters which is sufficinet to obtain nearly MRCI(Q)[80,81] level of accuracy for geometries considered here (see ref 79). Since the approximate CCSDT method (the CCSDT-1b approach[12]) has been considered in ref 79, we should expect that the full CCSDT results provide further improvement of the CCSDT-1b energies, which can in turn be used to calibrate the accuracy of the regularized CCSD(T) method.

In Figure 3, we show the CCSD, CCSD(T), CR-CC(2,3),[33−35] and CCSDT energies. The CR-CC(2,3) calculations were perfromed using the GAMESS implementation.[82,83] While the CCSD(T) method provides a good approximation of the CCSDT results at the equilibrium region (the CCSD(T) energy error with respect to the CCSDT energy is as small as 0.5 milli-Hartree) it fails at larger internuclear distances. For example the CCSD(T) error for α = 2.5 is equal to −158 milliHartree. This is a direct consequence of the poor choice of the RHF reference. The absolute values of the largest T_1 and T_2 amplitudes: 0.60 and 0.96 for α = 2.5, provide a good illustration of these problems. The very efficient CR-CC(2,3) approach is capable of removing all deficiences characterizing the CCSD(T) in this case. For example, the errors of the CR-CC(2,3) method for α = 1.0, 1.5, 2.0, and 2.3 are equal to 0.02, 0.5, 1.8, and 0.6 milliHartree, respectively (the CR-CC(2,3) results are reported up to α = 2.3 only; for larger distances NWChem and GAMESS were converging to different RHF solutions).

It is interesting to analyze the extent to which the regularization procedure can offset the problems plaguing the CCSD(T) approach. In Figure 4 we compare the Reg-CCSD(T) results with the CCSDT ones for two choices of the ω^2 parameter: 0.1 and 0.2. The impact of the choice of ω^2 on the Reg-CCSD(T) accuracies was a subject of discussion in ref 66. Using the example of the HF molecule we concluded that for strong quasidegeneracy the choice of larger ω^2 may be beneficial. Indeed, for the HF system the best agreement between CCSDT and Reg-CCSD(T) for stretched geometries was obtained for larger values of ω^2 (see ref 66 for details). This can also be observed using the H_3CF

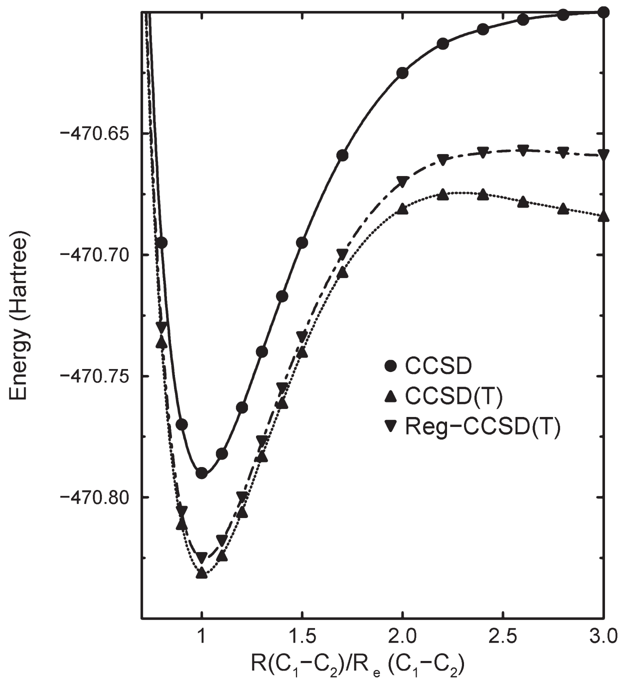

Figure 5. CCSD, CCSD(T), Reg-CCSD(T)(ω^2 = 0.1) energies as functions of the C_1-C_2 bond stretch in dodecane.

example. Despite of the fact that the Reg-CCSD(T)(ω^2 = 0.1) approach is capable of reducing the −158 milliHartree CCSD-(T) error at α = 2.5 almost 4-fold, still large error of −40 milliHartree remains. The increase of the ω^2 value results in further reduction of the Reg-CCSD(T)(ω^2 = 0.1) error down to −5 milliHartrre obtained with the Reg-CCSD(T)(ω^2 = 0.2) approach. At the same time the overall error of the Reg-CCSD(T) (ω^2 = 0.2) approach do not exceed 10 milliHartree for all geometries considered here (the Reg-CCSD(T)(ω^2 = 0.2) errors are 1.8, 5.1, 9.96, and −4.89 milliHartree for α = 1.0, 1.5, 2.0, and 2.5, respectively). This clearly demonstrate the advantages of using stronger regularization for quasidegenerate systems and also demonstrates that more flexible regularization methods should be developed in order to minimize the errors for the equilibrium and stretched geometries.

Recently, the state-of-the-art CR-CCSD(2,3) method[33−35] was used to describe ground-state PES corresponding to the $C_{12}H_{26}$ dissociation into $C_{11}H_{23}$ and CH_3.[84] It was shown that the CR-CC(2,3) curve along bond breaking coordinates (corresponding to varied C_1-C_2 separation, see Figure.3 of ref 84) smoothly approaches dissociation limit without any unphysical barriers, which are often observed in the methods employing many-body perturbation theory. With Reg-CCSD(T) method we performed similar studies using 6-311G basis set. The results of our calculations are shown in Figure 5. One can notice that the CCSD(T) curve discloses (in analogy to the methyfluoride) typical symptoms of perturbative breakdown. As in the previous example (H_3CF), for large internuclear distance, the doubly excited amplitudes assume the largest values (the largest amplitude assumes −0.8 value). In contrast to methylfluoride no big T_1 amplitudes have been observed in the CCSD calculations for dodecane. The unphysical hump of the CCSD(T) method is located around 2.2 $R_e(C_1-C_2)$. The CCSD(T) energy for 3$R_e(C_1-C_2)$ is located around 9 milliHartree below the CCSD(T) energy calculated for the "hump" geometry. The Reg-CCSD(T) approach to a large extent eliminates this pathological behavior: the analogous difference is reduced to 1.8 milliHartree. At the same time the Reg-CCSD(T) method yields energy of the CCSD(T) quality at the equilibrium geometry.

The main goal of studying the Spiro cation is to characterize the lowest doublet state of A_2 symmetry along a defined reaction pathway, which corresponds to the electron transfer from one π to other π moiety (see refs 67−70 and 78 for details). The Spiro cation has two equivalent C_{2v} minima, with a D_{2d} intermediate geometry (the neutral ground state is D_{2d}). In our studies of electron transfer we used the geometric change parameter ζ from the work of ref.,[70] which defines a simple linear mixing of the two mirror image C_{2v} minima (Q_A and Q_B)

$$Q(\zeta) = \left(\frac{1}{2} - \zeta\right) Q_A + \left(\frac{1}{2} + \zeta\right) Q_B \qquad (16)$$

The barrier region corresponds to ζ = 0.0 value.

Recently, the Spiro cation was the subject of intensive studies (see refs 68−70 and 78) using high-level theory including multireference perturbative (MRPT) approaches such as CASPT2,[85] various orders of the NEVPT method,[86] and second order of multiconfigurational quasi-degenerate perturbation theory (MCQDPT2)[87]). Using the approximate pathway (eq 16) it was demonstrated that the CASPT2, NEVPT2, and MCQDPT2 formalisms experience a serious problem (an unphysical minimum) in the description of correlation effects in the vicinity of the avoided crossing between the 1^2A_2 and 2^2A_2 states. This unphysical minimum on the ground-state PES in the vicinity of the avoided crossing region can be removed either by invoking higher orders of theory (NEVPT3) or by averaging the orbital energies of two charge-localized one particle states.[70] In ref 78, we showed that the single reference CC methods are capable of providing a satisfactory description of the ground-state PES as a function of the ζ parameter, avoiding to a large extent the problems plaguing multireference methods. In this paper we compare the Reg-CCSD(T) results with those obtained with the CCSD(T) and CR-CCSD(T) (version CR-CCSD(T),IA of ref 88) approaches. The CCSD(T), Reg-CCSD(T) (ω^2 = 0.1), and CR-EOMCCSD(T) energies for the 1^2A_2 state are shown in Table 2 and Figure 6. One should notice that the CCSD(T) corrections to the CCSD energies are in excess of 100 milliHartree for all geometries discussed here. Moreover, the presence of the overlap denominator in the CR-CCSD(T) correction makes the CR-CCSD(T) corrections two times smaller than the CCSD(T) ones. It is interesting to notice that the Reg-CCSD(T) results are invariably between the CCSD(T) and CR-CCSD(T) energies. The same observation is valid for the barrier heights which are equal to 0.078, 0.065, 0.069, and 0.077 eV for the CCSD, CCSD(T), Reg-CCSD(T), and CR-CCSD(T) methods, respectively. Unlike higher orders of MRPT theory, the single reference formulations suffer from cuspy behavior at the transition state geometry (ζ = 0.0). We believe that this feature can be eliminated by employing MCSCF orbitals. Another unresolved issue is related to a "kink" at all levels of CC theory in the vicinity of ζ = ± 0.25. We expect that this may be associated with the reference change or ROHF instability at that region. Unfortunately, using even very small ζ increments and continuing the ROHF solution from the equilibrium geometry we could not find an alternative solution.

We now focus on evaluating the performance of our implementation. The experiments were performed on two clusters,

Table 2. Comparison of the CC Energies As Functions of ζ-Parameter for the Spiro Moleclue[a]

geom.	CCSD	CCSD(T)	Reg-CCSD(T)	CR-CCSD(T)
ζ= 1.50	−611.36232	−611.46506	−611.45046	−611.40588
ζ= 1.40	−611.36396	−611.46675	−611.45214	−611.40755
ζ= 1.30	−611.36540	−611.46826	−611.45363	−611.40902
ζ= 1.20	−611.36666	−611.46959	−611.45494	−611.41030
ζ= 1.10	−611.36773	−611.47072	−611.45606	−611.41139
ζ= 1.00	−611.36861	−611.47167	−611.45699	−611.41228
ζ= 0.90	−611.36930	−611.47243	−611.45773	−611.41298
ζ= 0.80	−611.36980	−611.47300	−611.45829	−611.41351
ζ= 0.70	−611.37013	−611.47340	−611.45867	−611.41384
ζ= 0.60	−611.37026	−611.47361	−611.45885	−611.41398
ζ= 0.50	−611.37021	−611.47362	−611.45885	−611.41394
ζ= 0.40	−611.36997	−611.47346	−611.45866	−611.41371
ζ= 0.30	−611.36954	−611.47312	−611.45829	−611.41330
ζ= 0.20	−611.36921	−611.47289	−611.45803	−611.41298
ζ= 0.10	−611.36840	−611.47217	−611.45728	−611.41216
ζ= 0.05	−611.36793	−611.47175	−611.45684	−611.41168
ζ= 0.00	−611.36740	−611.47125	−611.45634	−611.41117

[a] In all calculations Sadlej's TZ basis set was used, and core electrons were not correlated.

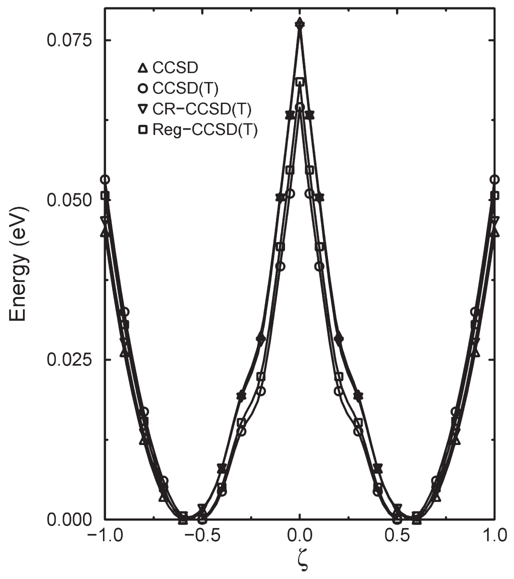

Figure 6. CC 1^2A_2 PESs obtained for the Spiro molecule described using POL1 basis set.[76] The lowest point of a given theory has been shifted to zero.

one with Tesla T10 GPUs, and the other with the Fermi cards. The T10 cluster consists of 64 nodes while the T20 cluster contains 16 nodes. Each node on the cluster with Tesla T10 GPUs has two Quad-Core Intel Xeon X5560 CPUs, with a frequency of 2.80 GHz, and 8 MB L2 cache. Two nodes share one Tesla S1070 box, implying that every node has two Tesla T10 GPUs. Each node on the Fermi cluster is equipped with two Quad-Core Intel Xeon E5520 CPUs, with the frequency of 2.27 GHz. Each node has a single GPU. PCI Express 2.0 is used for I/O between the host and the device on both systems. GNU 4.1.2 and NVCC 2.3 compilers are used for compilation, and CUBLAS 2.3 was used for the cublas-dgemm based algorithm. We begin with an evaluation of the individual block contributions, which constitute a parallel tensor contraction, on single GPUs, followed by the full parallel execution on the two clusters of GPUs.

Table 3. Performance of Different Algorithms (In Milliseconds) on Microbenchmark

problem size	cublasDgemm	baseline	combining	flattening	pipelining
(16,16,16,16,16,16,16)	464	109	101	105	50
(17,17,17,16,16,16,16)	336	233	216	131	59
(10,10,10,19,19,18,19)	114	65	60	44	21

In Table 3, we show the effects of our optimizations. These experiments are done on a micro benchmark which only includes one block contribution, on 3 problem sizes, using a single process. The comparison is among 5 versions, explained as below.

cublasDgemm. This is based on the original algorithm, in which each tensor contraction is implemented with index permutation and dgemm operations. We replaced the dgemm function with the Fortran wrapper and CUBLAS call, a library function provided for dgemm in CUDA.

Baseline. The basic CUDA algorithm described in the previous section.

Combining. Baseline. version with index combination.

Flattening. Combining version with flattened index.

Pipelining. Flattening version with pipelining to overlap data movement and kernel execution.

Among the 3 problem sizes, the first one has all dimensions set as 16, which fits the thread block configuration perfectly; the second problem size has the first dimensions as 17, resulting in big difference between thread block configuration and matrix index size; the third one is a random problem size picked from the NWChem trace. From the table, we can see that the baseline version is already doing better than the cublas version, because of reduced permutation operations. Pipelining is playing an important role in improving performance. Index combining improves performs further. Index flattening provides significant improvement when the problem size does not fit in the thread block configuration.

In the following paragraphs, we present our experiments on the full Reg-CCSD(T) execution. We demonstrate the scalability and performance improvements due to GPU execution using three systems: the Spiro, uracil, and dodecane molecules.

We did a comparison between GPU and CPU versions by running the two versions of code for the Spiro molecule. Since each node has only two GPUs, we ran 2 processes on each node, each driving a GPU. With 32 nodes, the time for CPU version is 42776 s, and the time of GPU version is 6950 s. So the GPU version yields a speedup of more than 6. When using 7 processes on each node, the GPU version has 2 processes using GPU and 5 processes using CPU only. This mixed version also has a speedup of about 3 over the version of using 7 CPU processes on each node. This demonstrates the speed-ups achieved by the GPU-based approach as compared to the CPU-only implementation. To provide a perspective on these times with respect to the overall execution time, we measured the times involved in the different modules executed. For the mixed version utilizing 2 GPUs and 5 CPUs on 32 nodes, for a total of 224 processes, the Hartee-Fock routine consumed 170 s, the four-index tranformation procedure consumed 2061 s, and the iterative CCSD

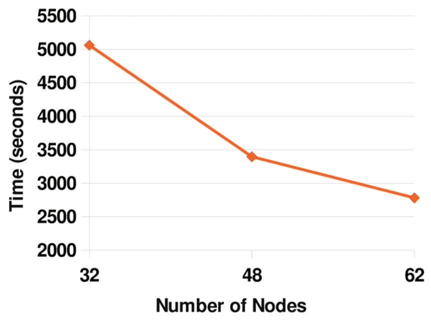

Figure 7. Running time for the (T) correction for Spiro molecule.

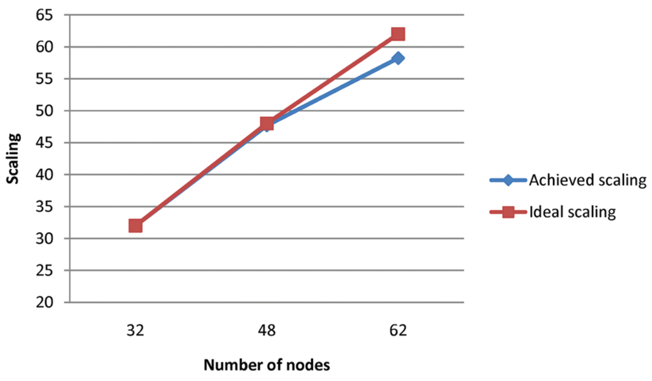

Figure 8. Scaling of the (*T*) correction for Spiro molecule.

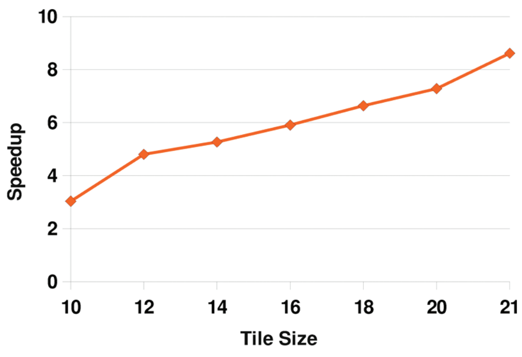

Figure 9. Speedup of GPU for the uracil molecule.

routines 167 s. In comparison, 5079 s were consumed by the noniterative triples correction.

The scalability of the GPU implementation is evaluated on 32, 48, and 62 nodes of the T10-based cluster. The execution times are shown in Figure 7 and scalability in Figure 8. The scalability is plotted with a baseline scaling of 32 on 32 nodes. We observe that the execution time scales almost linearly, achieving good scalability in addition to its speed-up over the CPU-only implementation.

The Spiro molecule represents systems characterized by relatively high symmetry (all calculations were performed using C_{2v} symmetry). This fact results in tiles which may be too small to provide optimum flop count for efficient use of GPU cores (tiles correspond to the partitioning of the spinorbital domain, which in turns implicates the block structure of all multidimensional tensors used in CC calculations: cluster amplitudes, recursive

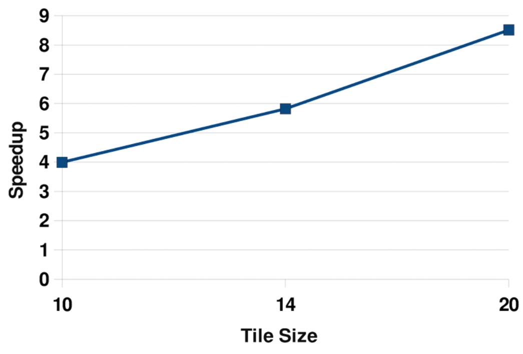

Figure 10. Speedup of GPU for the dodecane molecule.

intermediates, and residual vectors; the length of tiles will be commonly referred to as the tilesize).

In order to test the impact of tilesize on the GPU performance we tested GPU speedup on the uracil molecule in 6-31G* basis set where the spatial symmetry was not invoked. This situation commonly occurs in calculations for large systems without symmetry where tilesize can be sufficiently large. Of special importance is to understand the impact of tilesize on the GPU speedup.

The experiments are run on 30 nodes, with 2 processes on each node. The speedup of GPU over CPU version is shown in Figure 9. It can be seen that with larger tile size, which implies more FLOPS per process (proportional to $(tilesize)^7$), the speedup of GPU is more obvious. While for small tilesizes (tilesize = 10) the GPU speedup is rather modest (around 3), for larger tiles (tilesize = 21) the speedup is much better (around 8.75). From data shown in Figure 9 we should expect that the further increase in the tilesize should result in a further improvement in the GPU speedup.

To verify the generality of these observation, we evaluated the impact of tile sizes on the dodecane molecule. Shown in Figure 10, we observe speedups improving with tile sizes, reaching more than a factor of 8 with a tile size of 20.

5. CONCLUSIONS

We demonstrated that the Reg-CCSD(T) approach can improve the accuracies of the CCSD(T)method in studies of strongly correlated closed- and open-shell systems. We showed that for the methylfluoride, the regularization procedure can to a large extent eliminate the problems characteristic for the standard CCSD(T) approach. Our studies on the dodecane dissociation clearly indicate that the Reg-CCSD(T) approach provides significant improvements of the CCSD(T) results for the stretched geometries. We hope that this observation is generally valid for processes involving single bond breaking. For dodecane we showed that $\omega^2 = 0.1$ regularization is sufficient to obtain reliable shape of potential energy surface. The Reg-CCSD(T) results obtained for the methylfluoride suggest that the regularization procedures for singly and doubly excited clusters may require various ω^2 values, especially for states with large T_1. We also demonstrated that the Reg-CCSD(T) energies for the Spiro cation are free of the problems plaguing multireference approaches. We believe that for strong quasidegeneracy effects the use of larger values of ω^2 leads to more reliable results. It should be also stressed that the cost of triples part of Reg-CCSD(T) approach is exactly the same as its CCSD(T) analog. This problem will be pursued in a separate paper. The new GPU implementation of the Reg-CCSD(T) codes showed a great promise as far as the parallel performance of the most computationally N^7 part is concerned. We hope that the further advances in GPU-based technology will enable reliable CC calculations for medium (large) molecular systems on desktop (massively parallel) GPU computers.

■ AUTHOR INFORMATION

Corresponding Authors

*E-mail: sriram@pnl.gov (S.K.); karol.kowalski@pnl.gov (K.K.).

■ ACKNOWLEDGMENT

The work related to the development of the GPU Reg-CCSD(T) implementation was supported by the Extreme Scale Computing Initiative (SK,OV,KK), a Laboratory Directed Research and Development Program at Pacific Northwest National Laboratory. The calculations were performed using the Barracuda cluster in EMSL, a national scientific user facility sponsored by the Department of Energy's Office of Biological and Environmental Research and located at Pacific Northwest National Laboratory. The Pacific Northwest National Laboratory is operated for the U.S. Department of Energy by the Battelle Memorial Institute under Contract DE-AC06-76RLO-1830. The Barracuda cluster was purchased with funds received under the American Recovery and Reinvestment Act.

■ REFERENCES

(1) Coester, F. *Nucl. Phys.* **1958**, *7*, 421–424.
(2) Coester, F.; Kümmel, H. *Nucl. Phys.* **1960**, *17*, 477–485.
(3) Čížek, J. *J. Chem. Phys.* **1966**, *45*, 4256–4266.
(4) Paldus, J.; Shavitt, I.; Čížek, J. *Phys. Rev. A* **1972**, *5*, 50–67.
(5) Crawford, T. D.; Schaefer, H. F. *Rev. Comput. Chem.* **2000**, *14*, 33–136.
(6) Piecuch, P.; Kowalski, K.; Pimienta, I. S. O.; McGuire, M. J. *Int. Rev. Phys. Chem.* **2002**, *21*, 527–655.
(7) Bartlett, R. J.; Musiał, M. *Rev. Mod. Phys.* **2007**, *79*, 291–352.
(8) Purvis, G. D.; Bartlett, R. J. *J. Chem. Phys.* **1982**, *76*, 1910–1918.
(9) Cullen, J. M.; Zerner, M. C. *J. Chem. Phys.* **1982**, *77*, 4088–4109.
(10) Noga, J.; Bartlett, R. J. *J. Chem. Phys.* **1987**, *86*, 7041–7050.
(11) Noga, J. *J. Chem. Phys.* **1988**, *89*, 3401–3401.
(12) Urban, M.; Noga, J.; Cole, S. J.; Bartlett, R. J. *J. Chem. Phys.* **1985**, *83*, 4041–4046.
(13) Raghavachari, K.; Trucks, G. W.; Pople, J. A.; Head-Gordon, M. *Chem. Phys. Lett.* **1989**, *157*, 479–483.
(14) Stanton, J. F. *Chem. Phys. Lett.* **1997**, *281*, 130–134.
(15) Stanton, J. F.; Gauss, J. *J. Chem. Phys.* **1995**, *103*, 1064–1076.
(16) Stanton, J. F.; Gauss, J. *Theor. Chim. Acta* **1996**, *93*, 303–313.
(17) Kucharski, S. A.; Bartlett, R. J. *J. Chem. Phys.* **1998**, *108*, 5243–5254.
(18) Kucharski, S. A.; Bartlett, R. J. *J. Chem. Phys.* **1998**, *108*, 5255–5264.
(19) Crawford, T. D.; Stanton, J. F. *Int. J. Quantum Chem.* **1998**, *70*, 601–611.
(20) Gwaltney, S. R.; Head-Gordon, M. *Chem. Phys. Lett.* **2000**, *323*, 21–28.
(21) Gwaltney, S. R.; Head-Gordon, M. *J. Chem. Phys.* **2001**, *115*, 2014–2021.
(22) Gwaltney, S. R.; Byrd, E. F. C.; Van Voorhis, T.; Head-Gordon, M. *Chem. Phys. Lett.* **2002**, *353*, 359–367.
(23) Hirata, S.; Nooijen, M.; Grabowski, I.; Bartlett, R. J. *J. Chem. Phys.* **2001**, *114*, 3919–3928.
(24) Hirata, S.; Nooijen, M.; Grabowski, I.; Bartlett, R. J. *J. Chem. Phys.* **2001**, *115*, 3967–3968.
(25) Bomble, Y. J.; Stanton, J. F.; Kállay, M.; Gauss, J. *J. Chem. Phys.* **2005**, *123*, 8.
(26) Kállay, M.; Gauss, J. *J. Chem. Phys.* **2005**, *123*, 13.
(27) Taube, A. G.; Bartlett, R. J. *J. Chem. Phys.* **2008**, *128*, 13.
(28) Taube, A. G.; Bartlett, R. J. *J. Chem. Phys.* **2008**, *128*, 9.
(29) Kowalski, K.; Piecuch, P. *J. Chem. Phys.* **2000**, *113*, 18–35.
(30) Kowalski, K.; Piecuch, P. *J. Chem. Phys.* **2000**, *113*, 5644–5652.
(31) McGuire, M. J.; Piecuch, P.; Kowalski, K.; Kucharski, S. A.; Musiał, M. *J. Phys. Chem. A* **2004**, *108*, 8878–8893.
(32) Kowalski, K.; Piecuch, P. *J. Chem. Phys.* **2005**, *122*, 12.
(33) Piecuch, P.; Włoch, M. *J. Chem. Phys.* **2005**, *123*, 10.
(34) Piecuch, P.; Włoch, M.; Gour, J. R.; Kinal, A. *Chem. Phys. Lett.* **2006**, *418*, 467–474.
(35) Włoch, M.; Gour, J. R.; Piecuch, P. *J. Phys. Chem. A* **2007**, *111*, 11359–11382.
(36) Tajti, A.; Szalay, P. G.; Császár, A. G.; Kállay, M.; Gauss, J.; Valeev, E. F.; Flowers, B. A.; Vázquez, J.; Stanton, J. F. *J. Chem. Phys.* **2004**, *121*, 11599–11613.
(37) Bomble, Y. J.; Vázquez, J.; Kállay, M.; Michauk, C.; Szalay, P. G.; Császár, A. G.; Gauss, J.; Stanton, J. F. *J. Chem. Phys.* **2006**, *125*, 8.
(38) Hirata, S. *J. Phys. Chem. A* **2003**, *107*, 9887–9897.
(39) Lotrich, V.; Flocke, N.; Ponton, M.; Yau, A. D.; Perera, A.; Deumens, E.; Bartlett, R. J. *J. Chem. Phys.* **2008**, *128*, 15.
(40) Kuś, T.; Lotrich, V. F.; Bartlett, R. J. *J. Chem. Phys.* **2009**, *130*, 7.
(41) Janowski, T.; Ford, A. R.; Pulay, P. *J. Chem. Theory Comput.* **2007**, *3*, 1368–1377.
(42) Janowski, T.; Pulay, P. *Chem. Phys. Lett.* **2007**, *447*, 27–32.
(43) Janowski, T.; Ford, A. R.; Pulay, P. *Mol. Phys.* **2010**, *108*, 249–257.
(44) Bentz, J. L.; Olson, R. M.; Gordon, M. S.; Schmidt, M. W.; Kendall, R. A. *Comput. Phys. Commun.* **2007**, *176*, 589–600.
(45) de Jong, W. A.; Bylaska, E.; Govind, N.; Janssen, C. L.; Kowalski, K.; Muller, T.; Nielsen, I. M. B.; van Dam, H. J. J.; Veryazov, V.; Lindh, R. *Phys. Chem. Chem. Phys.* **2010**, *12*, 6896–6920.
(46) Kowalski, K.; Krishnamoorthy, S.; Villa, O.; Hammond, J. R.; Govind, N. *J. Chem. Phys.* **2010**, *132*, 154103.
(47) Yoo, S.; Apra, E.; Zeng, X. C.; Xantheas, S. S. *J. Phys. Chem. Lett.* **2010**, *1*, 3122–3127.
(48) Valiev, M.; Bylaska, E. J.; Govind, N.; Kowalski, K.; Straatsma, T. P.; Van Dam, H. J. J.; Wang, D.; Nieplocha, J.; Apra, E.; Windus, T. L.; de Jong, W. A. *Comput. Phys. Commun.* **2010**, *181*, 1477–1489.
(49) Apra, E.; Harrison, R.; de Jong. W. A.,; Rendell, A.; Tipparaju, V.; Xantheas, S.; Olsen, R. *Proc. of the ACM/IEEE Supercomp. 2009 Conf.* 2009; pp 66:1−66:7.
(50) Anderson, A. G.; Goddard, W. A.; Schröder, P. *Comput. Phys. Commun.* **2007**, *177*, 298–306.
(51) Owens, J. D.; Luebke, D.; Govindaraju, N.; Harris, M.; Kruger, J.; Lefohn, A. E.; Purcell, T. J. *Comput. Graphics Forum* **2007**, *26*, 80–113.
(52) Stone, J. E.; Phillips, J. C.; Freddolino, P. L.; Hardy, D. J.; Trabuco, L. G.; Schulten, K. *J. Comput. Chem.* **2007**, *28*, 2618–2640.
(53) Hardy, D. J.; Stone, J. E.; Schulten, K. *Parallel Comput.* **2009**, *35*, 164–177.
(54) Stone, J. E.; Hardy, D. J.; Ufimtsev, I. S.; Schulten, K. *J. Mol. Graph. Model.* **2010**, *29*, 116–125.
(55) Yasuda, K. *J. Comput. Chem.* **2008**, *29*, 334–342.
(56) Yasuda, K. *J. Chem. Theory Comput.* **2008**, *4*, 1230–1236.
(57) Ufimtsev, I. S.; Martinez, T. J. *J. Chem. Theory Comput.* **2008**, *4*, 222–231.
(58) Ufimtsev, I. S.; Martinez, T. J. *J. Chem. Theory Comput.* **2009**, *5*, 1004–1015.
(59) Ufimtsev, I. S.; Martinez, T. J. *J. Chem. Theory Comput.* **2009**, *5*, 2619–2628.
(60) Anderson, J. A.; Lorenz, C. D.; Travesset, A. *J. Chem. Phys.* **2008**, *227*, 5342–5359.
(61) Vogt, L.; Olivares-Amaya, R.; Kermes, S.; Shao, Y.; Amador-Bedolla, C.; Aspuru-Guzik, A. *J. Phys. Chem. A* **2008**, *112*, 2049–2057.

(62) Friedrichs, M. S.; Eastman, P.; Vaidyanathan, V.; Houston, M.; Legrand, S.; Beberg, A. L.; Ensign, D. L.; Bruns, C. M.; Pande, V. S. *J. Comput. Chem.* **2009**, *30*, 864–872.
(63) van Meel, J. A.; Arnold, A.; Frenkel, D.; Zwart, S. F. P.; Belleman, R. G. *Mol. Simul.* **2008**, *34*, 259–266.
(64) Eastman, P.; Pande, V. S. *J. Comput. Chem.* **2010**, *31*, 1268–1272.
(65) Ma, W.; Krishnamoorthy, S.; Villa, O.; Kowalski, K. *IEEE Intl. Conf. Cluster Comp.* **2010**, 207–216.
(66) Kowalski, K.; Valiev, M. *J. Chem. Phys.* **2009**, *131*, 12.
(67) Farazdel, A.; Dupuis, M.; Clementi, E.; Aviram, A. *J. Am. Chem. Soc.* **1990**, *112*, 4206–4214.
(68) Pastore, M.; Helal, W.; Isti, S. E.; Leininger, T.; Malrieu, J. P.; Maynau, D.; Angeli, C.; Cimiraglia, R. *J. Chem. Phys.* **2008**, *128*, 9.
(69) Helal, W.; Evangelisti, S.; Leininger, T.; Maynau, D. *J. Comput. Chem.* **2009**, *30*, 83–92.
(70) Pastore, M.; Helal, W.; Angeli, C.; Evangelisti, S.; Leininger, T.; Cimiraglia, R. *THEOCHEM* **2009**, *896*, 12–17.
(71) Kowalski, K.; Fan, P. D. *J. Chem. Phys.* **2009**, *130*, 11.
(72) Taube, A. G.; Bartlett, R. J. *J. Chem. Phys.* **2009**, *130*, 14.
(73) NVIDIA, NVIDIA CUDA C Programming Guide; http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/CUDA_C_Programming_Guide.pdf (accessed 11/9/2010).
(74) Dunning, T. H. *J. Chem. Phys.* **1989**, *90*, 1007–1023.
(75) Krishnan, R.; Binkley, J. S.; Seeger, R.; Pople, J. A. *J. Chem. Phys.* **1980**, *72*, 650–654.
(76) Sadlej, A. J. *Collect. Czech. Chem. C.* **1988**, *53*, 1995–2016.
(77) Stephens, P. J.; Devlin, F. J.; Chabalowski, C. F.; Frisch, M. J. *J. Phys. Chem.* **1994**, *98*, 11623–11627.
(78) Glaesemann, K. R.; Govind, N.; Krishnamoorthy, S.; Kowalski, K. *J. Phys. Chem.* **2010**, *114*, 8764–8771.
(79) Schütz, M. *J. Chem. Phys.* **2002**, *116*, 8772–8785.
(80) Knowles, P. J.; Werner, H. *J. Chem. Phys. Lett.* **1988**, *145*, 514–522.
(81) Werner, H. J.; Knowles, P. J. *J. Chem. Phys.* **1988**, *89*, 5803–5814.
(82) Schmidt, M. W.; Baldridge, K. K.; Boatz, J. A.; Elbert, S. T.; Gordon, M. S.; Jensen, J. H.; Koseki, S.; Matsunaga, N.; Nguyen, K. A.; Su, S. J.; Windus, T. L.; Dupuis, M.; Montgomery, J. A. *J. Comput. Chem.* **1993**, *14*, 1347–1363.
(83) Piecuch, P.; Kucharski, S. A.; Kowalski, K.; Musiał, M. *Comput. Phys. Commun.* **2002**, *149*, 71–96.
(84) Li, W.; Piecuch, P.; Gour, J. R.; Li, S. H. *J. Chem. Phys.* **2009**, *131*, 30.
(85) Andersson, K.; Malmqvist, P. A.; Roos, B. O.; Sadlej, A. J.; Wolinski, K. *J. Phys. Chem.* **1990**, *94*, 5483–5488.
(86) Angeli, C.; Cimiraglia, R.; Evangelisti, S.; Leininger, T.; Malrieu, J. P. *J. Chem. Phys.* **2001**, *114*, 10252–10264.
(87) Nakano, H. *J. Chem. Phys.* **1993**, *99*, 7983–7992.
(88) Kowalski, K.; Piecuch, P. *J. Chem. Phys.* **2004**, *120*, 1715–1738.