Two-Electron Integral Evaluation on the Graphics Processor Unit

KOJI YASUDA
Graduate School of Information Science, Nagoya University, Chikusa-ku, Nagoya 464-8601, Japan

Received 3 April 2007; Accepted 23 April 2007
DOI 10.1002/jcc.20779
Published online 5 July 2007 in Wiley InterScience (www.interscience.wiley.com).

Abstract: We propose the algorithm to evaluate the Coulomb potential in the *ab initio* density functional calculation on the graphics processor unit (GPU). The numerical accuracy required for the algorithm is investigated in detail. It is shown that GPU, which supports only the single-precision floating number natively, can take part in the major computational tasks. Because of the limited size of the working memory, the Gauss-Rys quadrature to evaluate the electron repulsion integrals (ERIs) is investigated in detail. The error analysis of the quadrature is performed. New interpolation formula of the roots and weights is presented, which is suitable for the processor of the single-instruction multiple-data type. It is proposed to calculate only small ERIs on GPU. ERIs can be classified efficiently with the upper-bound formula. The algorithm is implemented on NVIDIA GeForce 8800 GTX and the Gaussian 03 program suite. It is applied to the test molecules Taxol and Valinomycin. The total energies calculated are essentially the same as the reference ones. The preliminary results show the considerable speedup over the commodity microprocessor.

 J Comput Chem 29: 334–342, 2008

Key words: density functional; graphics processing unit; two-electron integrals

Introduction

Advances in the algorithms of the *ab initio* methods and the speedup of computers make the all-electron *ab initio* calculations of macromolecules such as proteins one of the most important applications of the quantum chemistry. Such a large system is often studied with the density functional theory (DFT)[1] and the linear-scaling algorithm, in which the computational cost scales to the system size. During the last two decades, the linear-scaling algorithm of DFT using Gaussian basis set has made great progress and today the size of a molecule we can treat is primarily restricted by the computer resources.

Until now the central processing unit (CPU) is solely responsible for processing the numerical data. However, recently the graphics processor unit (GPU) has evolved into a powerful and flexible processor. GPU is found in most modern desktop computers. It is responsible for accelerating the display of graphical elements such as polygons on a computer monitor. The most recent GPU from NVIDIA, GeForce 8800, is implemented with the general-purpose parallel processor of the single-instruction, multiple-data (SIMD) type. Because of the high computational density, the theoretical peak performance of this GPU is several times higher than the fastest commodity microprocessor. Thus, GPU would provide the alternative to accelerate the *ab initio* calculation. The reason behind such high performance is that it is specialized for compute-intensive, highly data-parallel computations. It is designed such that more transistors are devoted to data processing rather than data caching and flow control.

On the other hand this GPU has the following shortcomings. It only supports the operations on the integer and the single-precision floating-point numbers natively. Only a limited size of the working memory (registers and caches) is available on chip. To hide the pipeline stall, a lot of parallel threads should be run concurrently. These facts indicate that GPU requires completely different algorithms from the conventional CPU to work efficiently. Since the time-consuming steps in the *ab initio* calculations are often very complicated, it is not clear *a priori* whether the GPU accelerates the *ab initio* calculations or even can do some useful tasks.

Usually the most time consuming steps in DFT are the calculation of the near-field Coulomb potential, the calculation of the exchange-correlation potential, and the diagonalization of the Fock matrix. Since the evaluation of the Coulomb potential is the most complicated among them, it is the most difficult to execute on GPU. The aim of this study is to explore the algorithm to calculate the Coulomb potential efficiently on it.

The key ideas are as follows. (i) The evaluation of the electron repulsion integrals (ERIs) consumes major computational time. We

Correspondence to: K. Yasuda; e-mail: yasudak@is.nagoya-u.ac.jp

Contract/grant sponsor: Ministry of Education, Culture, Sports, Science and Technology, Japan

examined the magnitudes of ERIs and found that most of them are small so that we can safely evaluate them with the single precision. Other ERIs are evaluated on host with the double precision. (ii) We examine the algorithm to evaluate ERIs suitable for GPU. We found that the Gauss-Rys quadrature[2,3] would be the best, because it requires the least working memory. The accuracy necessary for the quadrature is examined in detail. New interpolation formula of the Rys roots and weights is presented, that is suitable for the SIMD type parallel processor.

The algorithm proposed was implemented on the GeForce 8800 GTX and the *ab initio* program package Gaussian 03.[4] It was applied to the molecules Taxol and Valinomycin. The energies calculated were essentially the same as the reference ones calculated with the double precision. The preliminary timing results show that GPU evaluates the Coulomb potential as fast as, or even faster than the recent commodity microprocessor. These results demonstrate that GPU offers a promising alternative to accelerate the density functional calculation.

Method

Algorithm Selection

The aim of this study is to explore the algorithm to calculate the Coulomb potential efficiently on the following computer. A CPU on a host computer is responsible for the double-precision calculation, while a GPU on a graphic board carries out the single-precision calculation. We assume that the GPU is much faster (about 10 times) than the host CPU. The communication speed between CPU and GPU is rather slow, which is limited by the bandwidth of the PCI express bus (several giga bytes per second).

Most of the *ab initio* calculations of molecules use the contracted Gaussian basis functions defined as,

$$\chi_A(x,y,z) = (x-A_x)^{a_x}(y-A_y)^{a_y}(z-A_z)^{a_z}\sum c_A \exp(-aR_A^2),$$
$$R_A^2 = (x-A_x)^2 + (y-A_y)^2 + (z-A_z)^2, \quad (1)$$

where the sum runs over all the primitive Gaussians, $\mathbf{A} = (A_x, A_y, A_z)$ is the position of the basis function, and c_A is the contraction coefficient. The sum $L_A = a_x + a_y + a_z$ gives the angular momentum of the basis function χ_A, where a_x, a_y, and a_z are the nonnegative integers.

Conventionally the matrix element of the Coulomb potential is calculated as,

$$J_{ab} = \sum_{cd}(ab|cd)D_{cd}, \quad (2)$$

$$(ab|cd) = \int \frac{\chi_a(r_1)\chi_b(r_1)\chi_c(r_2)\chi_d(r_2)}{|r_1 - r_2|}dr_1 dr_2, \quad (3)$$

where D_{cd} is the first-order reduced density matrix and $(ab|cd)$ is ERI over the four Cartesian Gaussian functions. The summation of eq. (2) consumes considerable computational time because of the heavy number of terms.

The most noticeable advantage of the Gaussian basis is that the product of two primitive Gaussians becomes another Gaussian (Gaussian product rule).

$$\exp\left(-aR_A^2\right)\exp\left(-bR_B^2\right) = K\exp\left(-pR_P^2\right) \quad (4)$$

$$K = \exp\{-ab(\mathbf{A}-\mathbf{B})^2/(a+b)\}$$
$$p = a+b, \quad \mathbf{P} = (a\mathbf{A}+b\mathbf{B})/(a+b) \quad (5)$$

Hence, every four-center ERI reduces to the two-center one. The product of the Gaussians can be expanded with the Hermite Gaussians,

$$(x-A_x)^{a_x}(y-A_y)^{a_y}(z-A_z)^{a_z} \times \exp\left(-aR_A^2\right)(x-B_x)^{b_x}(y-B_y)^{b_y}(z-B_z)^{b_z}\exp\left(-bR_B^2\right) = \sum_{tuv} E_t^{a_x b_x} E_u^{a_y b_y} E_v^{a_z b_z} \Lambda_t(x)\Lambda_u(y)\Lambda_v(z), \quad (6)$$

$$\Lambda_t(x) = \left(\frac{\partial}{\partial P_x}\right)^t \exp(-p(x-P_x)^2), \quad (7)$$

where H_t is the t-th order Hermite polynomial, and $E_t^{a_x b_x}$ is the McMachie-Davidson coefficient.[5] The quantities $\Lambda_u(y)$ and $\Lambda_v(z)$ are defined similarly.

By using the Gaussian product rule, we can calculate the Coulomb potential in terms of the Hermite Gaussian function. This method, called the J engine method,[6] is faster than the conventional one. Firstly we transform the density matrix in the Cartesian Gaussian basis to the Hermite Gaussian basis.

$$D_{tuv}^{Q} = \sum E_t^{c_x d_x} E_u^{c_y d_y} E_v^{c_z d_z} c_c c_d D_{cd} \quad (8)$$

The sum runs over all the primitives and the exponents (c_x, $c_y, \ldots, d_z$). The Coulomb potential is expressed with the Hermite Gaussians.

$$J_{tuv}^{P} = \sum_{Qt'u'v'} (P(tuv)|Q(t'u'v'))D_{t'u'v'}^{Q} \quad (9)$$

Here $P(tuv) = \Lambda_t\Lambda_u\Lambda_v$ is the Hermite Gaussian on center $\mathbf{P}$. These matrix elements are transformed to the Cartesian Gaussian basis.

Since the computational cost of the conventional and the J engine methods scale at least to the square of the system size, they are not appropriate for large molecules. We use the linear-scaling algorithm as follows. Coulomb potential generated by the distant electron density is calculated with the continuous fast multipole method (CFMM).[7] The near-field contribution is evaluated with the J engine method. The Schwartz inequality[8] is further used to reduce the number of ERIs to calculate. CFMM, the J engine, and the Schwartz integral screening are the established algorithms.

We measured the computational time of these three steps for the test molecules, Taxol and Valinomycin. The 3-21G and 6-31G basis sets,[9] the local density approximation (LDA) and the Perdew 91

Table 1. Computational Time of the Three Major Steps to Evaluate the Coulomb potential.

	User time (%) (s)				
Functional/Basis	Integral screening	ERI calculation	CFMM	Other	Total
Taxol ($C_{47}H_{51}NO_{14}$)					
LSDA/3-21G	9.6 (19.2)	35.1 (70.1)	3.5 (7.1)	1.8 (3.6)	50.1
LSDA/3-21G U[a]	29.0 (46.3)	26.4 (42.1)	5.0 (7.9)	2.3 (3.6)	62.7
LSDA/3-21G U/Opt[b]	17.3 (34.1)	26.4 (52.1)	3.7 (7.3)	3.3 (6.4)	50.7
PW91/6-31G	14.5 (10.7)	112.8 (83.1)	5.0 (3.7)	3.4 (2.5)	135.8
PW91/6-31G U	114.1 (52.4)	85.5 (39.3)	5.5 (2.5)	12.6 (5.8)	217.7
PW91/6-31G U/Opt	61.8 (37.2)	85.9 (51.7)	6.9 (4.1)	11.6 (7.0)	166.2
Valinomycin ($C_{54}H_{90}N_6O_{18}$)					
LSDA/3-21G	20.6 (24.6)	53.6 (64.0)	6.1 (7.3)	3.5 (4.2)	83.8
LSDA/3-21G U	59.0 (52.2)	40.5 (35.8)	7.7 (6.8)	5.8 (5.2)	113.0
LSDA/3-21G U/Opt	33.3 (38.1)	40.4 (46.3)	7.5 (8.6)	6.1 (7.0)	87.3
PW91/6-31G	34.0 (14.6)	183.5 (78.9)	9.6 (4.1)	5.6 (2.4)	232.6
PW91/6-31G U	232.6 (57.6)	135.0 (33.4)	10.4 (2.6)	25.9 (6.4)	403.9
PW91/6-31G U/Opt	119.8 (41.3)	134.1 (46.2)	10.4 (3.6)	25.8 (8.9)	290.2

Total user time of the first three SCF iteration are listed.
[a]Basis are uncontracted.
[b]Integral screening procedure reoptimized.

exchange-correlation functional (PW91),[10] and the default parameter sets of Gaussian 03 were used. Table 1 shows the total user time of the first three self-consistent field (SCF) iterations. As shown in it, CFMM occupies a small fraction of computational time (less than 8%). On the other hand, the integral screening consumes about 20% and 50% of time for the contracted and the uncontracted basis, respectively. The evaluation of ERIs occupies the rest. Thus, we should execute the integral screening and the ERI evaluation on GPU. The number of shell quartets evaluated shows the efficiency of the CFMM and the integral screening. In these examples, CFMM reduces the number of shell quartets to $1/3 \sim 1/2$. The Schwartz inequality and the density-based cutoff further reduce it: only $1/4 \sim 1/2$ of original ERIs survive. CFMM and the integral screening is essential for the fast evaluation of the Coulomb potential.

We also compared the CPU time of the conventional and the J engine methods. PRISM algorithm[8] was used to evaluate the near-field ERIs in the conventional method. We found that the J engine method is 30–40% faster than the conventional one. Interestingly, the J engine method is even fast for the uncontracted Gaussian functions. This is in sharp contrast to the conventional method, in which the use of the uncontracted Gaussians drastically increases the computational time. As seen in this table, the use of the uncontracted Gaussians increases the CPU time only about 20% for 3-21G basis, and 60–70% for 6-31G basis. The integral screening step is responsible for most of this slowdown. Optimization of this procedure by using the inline expansion improves the performance. The results denoted as "Opt" show the CPU time after the optimization. As seen in it, we can calculate the Coulomb potential with the uncontracted Gaussians as fast as with the contracted ones. Since the evaluation of ERI over the uncontracted Gaussians is much easier than the contracted ones, we carry out on GPU the integral screening and the Coulomb potential evaluation, using the J engine method and the uncontracted Gaussians.

Accuracy

There are several issues associated with using GPU for general purpose computations. In this section, we examine the issue of the numerical accuracy, because our GPU supports natively only the integer and the single-precision floating-point numbers.

The IEEE 754 compliant, single-precision floating-point number a is represented as[11]

$$\begin{aligned} a &= \pm 2^e m, \\ -125 &\le e \le 128, \\ 1 &\le m < 2, \end{aligned} \tag{10}$$

where the exponent e is an integer, and the mantissa m is the 24-bit binary decimal. The relative error in the single-precision number is $2^{-23} \approx 1.2 \times 10^{-7}$, and hence the absolute error is about $2^{-23} \times |a|$. The result of the addition or the multiplication is rounded off to this expression.

When the precision provided by the hardware does not suffice, it is possible to carry out more accurate addition and multiplication using software. Firstly, we can calculate the exact sum of two normalized floating-point numbers a and b as follows.[12]

$$\begin{aligned} &\text{Add } 12(a, b) \\ &s = a \oplus b \\ &v = s \ominus a \\ &r = (a \ominus (s \ominus v)) \oplus (b \ominus v) \\ &\text{return}(s, r) \end{aligned} \tag{11}$$

The operations $\oplus$ and $\ominus$ are the floating point addition and subtraction with the roundoff, respectively. The value s gives the sum rounded off to the single-precision, while r catches the roundoff

error. We have the exact sum at the expense of six instructions. The procedure to evaluate the exact product was also reported, which requires 26 instructions.[13]

Next, we express a 48-bit multiprecision number r as the unevaluated sum of two single-precision numbers (r_H, r_L). The number r_H holds the most significant 24 bits of the mantissa of r. Using eq. (11) we can evaluate the sum of the multiprecision numbers as follows.[14]

$$
\begin{aligned}
&\text{Add } 22(a_H, a_L, b_H, b_L)\\
&s = a_H \oplus b_H\\
&\text{if} \quad |a_H| \geq |b_H| \quad \text{then}\\
&\quad r = (((a_H \ominus s) \oplus b_H) \oplus b_L) \oplus a_L\\
&\text{else}\\
&\quad r = (((b_H \ominus s) \oplus a_H) \oplus a_L) \oplus b_L\\
&(c_H, c_L) = \text{Add } 12(s, r)\\
&\text{return}(c_H, c_L)
\end{aligned}
\tag{12}
$$

However, since this procedure is rather expensive, we lose most of the GPU's speed advantage over the host CPU, if we replace every addition operation with it. All the operations except for the critical ones should be done with the single precision.

Generally speaking, single precision is not adequate for *ab initio* calculation. Total electronic energy often reaches 10^4 a.u., while we need at least the accuracy of 300 K $\approx 10^{-3}$ a.u. Nevertheless, we will show that GPU can do some useful tasks. A trivial one is the initial guess generation: the single-precision calculation is used only in the early iterations of SCF. Another strategy is to classify the integrals: large ERIs are evaluated on host with the double precision, while the small ones on GPU with the single precision. The absolute errors of ERIs would be small even if they are evaluated with the single precision, because ERIs themselves are small in magnitude. This division saves the computational time provided that small integrals dominate in number.

To substantiate this idea, we counted the number of terms $N(n)$ whose absolute values are in the range $[10^{-n}, 10^{-(n+1)}]$, which appeared in the Coulomb energy E_J of the near-field contribution.

$$E_J = \frac{1}{2}\sum_{pq} D_p D_q (p|q) \tag{13}$$

The 3-21G basis,[9] the local density approximation, and the converged density matrix were used. Figure 1 shows the histogram for the Valinomycin. As shown, the number of terms gradually increases as n for $n \leq 12$. Terms of order 10^{-12} are the most abundant, 5.5×10^9 in total. This histogram indicates that the terms with small n give the large contribution to the total energy. However, because of the heavy number, terms in the range $[10^{-12}, 10^{-13}]$ would also give sizable contribution to the total energy.

Each energy term $D_p D_q (p|q)/2$ expressed with the single precision has the absolute error of order $2^{-24}|D_p D_q (p|q)|$. We assume that this error has the uniform distribution. The sum of these errors becomes the error of the total Coulomb energy, which is the superposition of the Gaussian distribution. Roughly speaking, terms in the range $[10^{-n}, 10^{-(n+1)}]$ give the energy error $10^{-n}\sqrt{N(n)}$.

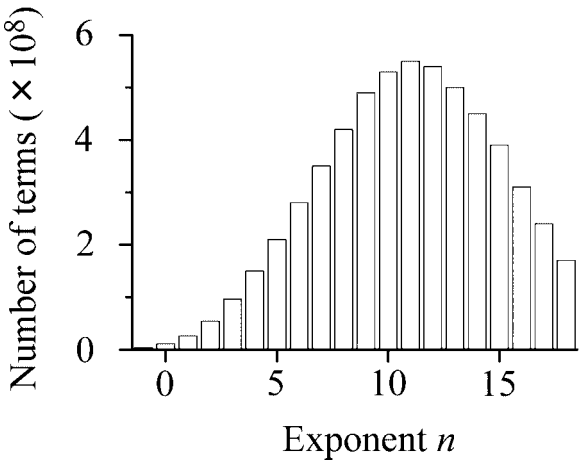

Figure 1. The number $N(n)$ of the near-field Coulomb energy terms $D_p D_q (p|q)/2$ whose absolute values are in the range $[10^{-n}, 10^{-(n+1)}]$. 3-21G basis, the local density approximation, and the converged density matrix of Valinomycin were used.

Thus, deciding from the histogram $N(n)$ most of the numerical error comes from the largest terms evaluated with the single precision. For example, if we evaluate all the terms in $[10^{-4}, 10^{-5}]$ with the single precision, the error in the total energy amounts to $2^{-23} \times 10^{-4} \times \sqrt{10^8} \approx 10^{-7}$. Because of the distribution $N(n)$ other terms smaller in magnitude give smaller errors.

The division of terms can be done efficiently using the Schwartz upper bound of ERIs: only terms whose upper-bounds are smaller than the threshold λ_{GPU} are calculated on GPU. We can control the numerical accuracy and the computational efficiency with the threshold.

Gauss-Rys Quadrature

In this section, we perform the error analysis of the ERI evaluation procedure to elucidate the most dangerous operation in it. There are several methods to calculate ERIs.[2,3,5,8,15–18] Since GPU has very small on-chip memory, we examine the Rys polynomial method,[2,3] that needs the smallest memory. Gauss-Rys quadrature gives the following formula of ERI between the contracted Gaussian basis functions.

$$
\begin{aligned}
(ab|cd) = 2\pi^{5/2}\sqrt{\alpha} \sum_{\substack{tuv\\ t'u'v'}} (-1)^{t'+u'+v'} C_{AB} C_{CD} R^{-l} \sum_{k=1}^{n} w_{nk} s_{nk}^{l}\\
\times H_{t+t'}(n_x s_{nk}) H_{u+u'}(n_y s_{nk}) H_{v+v'}(n_z s_{nk})\\
l = t + u + v + t' + u' + v'\\
s_{nk} = \sqrt{\alpha} R r_{nk}
\end{aligned}
\tag{14}
$$

Here r_{nk} and w_{nk} are the k-th root and the quadrature weight of the Rys polynomial of order $n = [l/2]+1$, respectively. They depend on the value $\beta = \sqrt{\alpha}R$, because Rys polynomial forms the orthonormal set under the inner product $\int_0^1 \cdots \exp(-\beta^2 s^2) ds$. Indices t, u, v, t', u', and v' run over all the nonnegative integers such that the sum of

them equals to L, where $L = L_A + L_B + L_C + L_D$. Other quantities are defined as follows.

$$C_{AB} = c_A c_B E_t^{a_x b_x} E_u^{a_y b_y} E_v^{a_z b_z} p^{-3/2}$$
$$C_{CD} = c_C c_D E_{t'}^{c_x d_x} E_{u'}^{c_y d_y} E_{v'}^{c_z d_z} q^{-3/2}$$
$$q = c + d, \quad \mathbf{Q} = (c\mathbf{C} + d\mathbf{D})/(c + d)$$
$$\alpha = pq/(p + q), \quad \mathbf{R} = \mathbf{P} - \mathbf{Q}$$
$$(n_x, n_y, n_z) = \mathbf{R}/R$$

ERIs over the primitive Hermite Gaussians are given with the simpler formula,

$$(p|q) = (-1)^{t'+u'+v'} \frac{2\pi^{5/2}}{pq\sqrt{p+q}} R^{-l} \times \sum_{k=1}^{n} w_{nk} s_{nk}^{l} H_{t+t'}(n_x s_{nk}) H_{u+u'}(n_y s_{nk}) H_{v+v'}(n_z s_{nk}). \quad (15)$$

Firstly, let us examine the condition for the weight w_{nk} so that the absolute error of ERI is smaller than ϵ. Since ERI is the linear form of the weight, $(ab|cd) = \sum_{nk} w_{nk} U_{nk}$, the accuracy necessary for w_{nk} is reciprocal to U_{nk}. For example, when the threshold ϵ is 10^{-6} and the maximum value of the coefficient U_{nk} is 10^2, the weight w_{nk} should be given within the absolute errors of 10^{-8}. The accuracy necessary for the root s_{nk} is determined with the same kind of linearlized formula.

To know the accuracy required for the Rys roots and weights, we consider the two-center ERIs of arbitrary separation R. This is because the three- and the four-center ERIs are smaller in magnitude than the two-center ones due to the exponential prefactor [eq. (5)], and hence less sensitive to the errors of the roots and weights.

The complicated task to find the maximum value of U_{nk} can be simplified as follows. First, define the functions,

$$C_{AB}(l) = c_A c_B p^{-3/2} \max_{\substack{tuv \\ a_x \cdots b_z}} \left| E_t^{a_x b_x} E_u^{a_y b_y} E_v^{a_z b_z} \right|, \quad (16)$$

$$G(l, s) = \max_{\substack{tuv \\ n_x n_y n_z}} |s^l H_t(n_x s) H_u(n_y s) H_v(n_z s)|, \quad (17)$$

under the condition that

$$t + u + v = l, \quad n_x^2 + n_y^2 + n_z^2 = 1,$$
$$a_x + a_y + a_z = L_A, \quad b_x + b_y + b_z = L_B.$$

The function $C_{CD}(l)$ is defined similarly. Then, the upper bound of ERI becomes

$$|(ab|cd)| \le 2\pi^{5/2} \sum_{l=0}^{L} \alpha^{(l+1)/2} \sum_{l'=0}^{l} C_{AB}(l') C_{CD}(l - l') \times \sum_{k=1}^{n} w_{nk} G(l, s_{nk}) / (\sqrt{\alpha} R)^l, \quad (18)$$

where $n = [l/2] + 1$. The maximum value of the coefficient of the weight w_{nk} determines the necessary accuracy. Note that the Rys root and weight are the functions of $\beta = \sqrt{\alpha} R$, and the accuracy required also depends on it.

Next, we search for the maximum $D(l)$ over all the combination of the shells (A, B, C, D).

$$D(l) = \max_{ABCD} \alpha^{(l+1)/2} \sum_{l'=0}^{l} C_{AB}(l') C_{CD}(l - l') \quad (19)$$

The quantities C_{AB}, C_{CD}, and $D(l)$ are the only terms that explicitly depend on the basis set used. The maximum value of the coefficient of the weight $w_{nk}(\beta)$ is

$$U_{nk}(\beta) = 2\pi^{5/2} \max_{l \in [2n-2, 2n-1]} D(l) G(l, s_{nk}(\beta)) \beta^{-l}. \quad (20)$$

The accuracy necessary for the weight $w_{nk}(\beta)$ is reciprocal to it, $\epsilon U_{nk}(\beta)^{-1}$.

The two-electron integral is a function of Rys root: $(ab|cd) = I(r_{nk})$. The error in the Rys root δr induces the error in the two-electron integral $I(r_{nk} + \delta r) - (ab|cd) \approx V_{nk} \delta r$, where $V_{nk} = \partial (ab|cd)/\partial r_{nk}$. Thus, the accuracy necessary for the Rys root r_{nk} is given with the maximum value of V_{nk}. Using the following auxiliary function $H(l, s)$,

$$H(l, s) = \max_{\substack{t,u,v \\ n_x n_y n_z}} \left| \frac{\partial}{\partial s} s^l H_t(n_x s) H_u(n_y s) H_v(n_z s) \right|, \quad (21)$$

the upper bound of V_{nk} is expressed as

$$V_{nk}(\beta) = 2\pi^{5/2} \max_{l \in [2n-2, 2n-1]} D(l) H(l, s_{nk}(\beta)) \beta^{-l}. \quad (22)$$

Note that the upper bounds [eqs. (20) and (22)] are solely determined with the basis set information and does not depend on the geometries.

To investigate the accuracy necessary for the Rys roots and weights, we calculate the coefficients U_{lk} and V_{lk} of eqs. (20) and (22). We used the STO-3G, STO-6G, 3-21G, and 6-31G basis sets[9] of the first and the second-row elements (H, He, ..., Ne). The coefficients U_{lk} and V_{lk} are then converted to the absolute errors allowed for the root and weight to ensure the integral accuracy of 10^{-7}. Table 2 presents the results as well as the number of the binary digits to represent the roots and the weights. The Rys roots are in increase order $r_{n1} < r_{n2} < \cdots$.

As shown in it, the accuracy necessary are largely different for each root or weight: the larger the root is the smaller the error should be. This is because that the integrand of eq. (14) or (15) is a rapidly increasing polynomial of order $2l$. We can clearly see it in the third weights. The weight w_{33} requires the accuracy of $10^{-8} \sim 10^{-9}$, while w_{31} needs only 10^{-3} to 10^{-5}. The accuracy necessary decreases as the argument β increases.

Table 2 also indicates that some roots and weights require the multiprecision accuracy to reduce the absolute error of ERI

Table 2. The Accuracy Necessary for the Rys Roots and Weights to Achieve the Integral Accuracy of 10^{-7} as a Function of $\beta = \sqrt{\alpha}R$.

	Absolute error (Number of bits)			
β	[0,6]	[6,12]	[12,24]	[24,∞]
r_{11}	4.4[−8] (24)	6.5[−8] (23)	9.1[−8] (22)	1.3[−7] (21)
r_{21}	1.7[−7] (22)	5.4[−7] (20)	1.0[−6] (18)	2.1[−6] (17)
r_{22}	9.1[−8] (24)	1.6[−7] (23)	3.3[−7] (21)	6.6[−7] (20)
r_{31}	5.4[−6] (16)	2.7[−5] (14)	8.4[−5] (12)	3.3[−4] (9)
r_{32}	3.9[−8] (25)	1.2[−7] (23)	4.6[−7] (21)	2.6[−6] (18)
r_{33}	1.3[−8] (27)	5.0[−8] (25)	1.1[−7] (23)	6.3[−7] (21)
w_{11}	2.4[−8] (26)	2.4[−8] (25)	2.4[−8] (24)	2.4[−8] (24)
w_{21}	6.3[−7] (21)	1.7[−6] (19)	3.2[−6] (17)	6.4[−6] (16)
w_{22}	4.2[−8] (23)	4.3[−8] (21)	7.3[−8] (19)	1.5[−7] (18)
w_{31}	4.2[−5] (14)	1.9[−4] (12)	5.6[−4] (10)	2.2[−3] (7)
w_{32}	7.9[−8] (23)	1.8[−7] (20)	4.8[−7] (18)	2.6[−6] (15)
w_{33}	3.5[−9] (24)	3.2[−9] (21)	3.9[−9] (19)	2.0[−8] (17)
β	7.3[−8] (24)	8.1[−7] (24)	2.3[−6] (23)	6.4[−6] (23)

Numbers in the square brackets indicate powers of 10, while those in parentheses are the number of the binary digits of the mantissa necessary to represent the roots and the weights.

below 10^{-7}. For example, w_{11} needs at most 26 bits of mantissa. If we express it with the single-precision number, it would contain the roundoff error eight times larger than acceptable tolerance. Multiprecision arithmetic is required for them.

To circumvent this problem, in this study, we calculate only the ERIs small in magnitude with the single precision. The accuracy requirement of the root and the weight we presented so far is that the absolute error of the two-center ERI should be smaller than 10^{-7}. There are a lot of ERIs which are small in magnitude because of the exponential prefactor of eq. (5). This prefactor also reduces the absolute error of ERI. By selecting the integral screening threshold λ_{GPU}, we safely evaluate them on GPU.

The accuracy required for the roots and weights in turn determine that for β. Since a root r_{nk} is a function of β, the error $\delta\beta$ induces the error of a root,

$$\delta r_{nk} \approx \frac{dr_{nk}}{d\beta}\delta\beta. \tag{23}$$

Similar equation holds for the weight. In Table 2 we present the maximum allowed error of β as well as the number of the binary digits of the mantissa to represent β. Since the roots and weights are slowly varying function of β, lower accuracy is allowed for β than for the roots and weights.

The evaluation of the Hermite polynomial $H_t(x)$ in eqs. (14) and (15) with the three-term recursion may be a potentially dangerous step. Because of the cancellation of the significant bit, the relative error of $H_t(x)$ becomes large when x is close to the zero point of H_t. However, since the zero points of the n-th and the $(n+1)$-th orthonormal polynomials are always different, such error does not propagate. Hence, the error accumulation to ERI is rather small.

Rys Interpolation Table

In this section, we discuss the efficient procedure to evaluate the roots and weights of the Rys polynomial $R_n(s)$. There are several methods to calculate them.[3] We can determine the roots on the fly by solving the algebraic equation $R_n(s) = 0$ numerically. The weights of the first-, second-, and the third-orders can be expressed with the roots and the incomplete gamma function.[3] However, to avoid the complicated calculus we approximate both roots and weights with the piecewise polynomial fit.

In ref. 3 King et al. proposed the following fit: the Chebyshev expansion for the small and the intermediate β, and the following asymptotic form for the large β.

$$\begin{aligned} f(\beta) &= f_0\beta^{-\frac{1}{2}} + e^{-x}g(\beta) \\ g(\beta) &= \sum_j g_j(\beta - \beta_0)^j \end{aligned} \tag{24}$$

The sum runs over some negative and positive integers, while f_0, β_0, and g_j are constants. The order and the coefficients of the polynomial are determined to satisfy the accuracy of one part in 10^{13}. Typically, the 5-th to 15-th order polynomial is used for 10 segmented regions of β.

On the other hand, we need the following interpolation tables. (i) The same formula is desirable for all the regions of β except for the coefficients because each thread on the SIMD processor executes the same operation on different data. (ii) We need not keep the same precision for all the roots and weights and for all the range of β, because the necessary accuracy is different. The interpolation formula we developed is as follows. We split the entire region of β into four. In the region of the smallest β, the roots and the weights themselves are approximated, while in other regions the asymptotic forms are subtracted and the rests are approximated. Since the function to approximate decays exponentially for large β, we expand it in terms of the new variable y,

$$y = -\exp\{(\beta - \beta_i)/8\}.$$

This new variable y provides faster convergence than the simple Chebyshev expansion in terms of β. The calculation of exponential requires no extra work on SIMD processor. It is always necessary even if we use the formula of King et al., because the asymptotic form of eq. (24) has the exponential factor.

By carefully selecting the accuracy for the roots and weights, we can drastically reduce the number of the coefficients of the polynomials. For example, to ensure the tolerance 10^{-5} of the weight w_{31}, 4-th order polynomial is sufficient.

Implementation

In this section, we briefly present the algorithm and the implementation to calculate the Coulomb potential on GPU. The block structure of GPU, NVIDIA GeForce 8800 GTX, is shown in Figure 2. It consists of 16 sets of the SIMD multiprocessors with the on-chip shared memory. Each multiprocessor is composed of eight processors. At any given clock cycle each processor of the multiprocessor executes the same instruction, but operates on different data. Each multiprocessor has on-chip memory of the four kinds. (i) One set of local registers per processor. (ii) A memory that is shared by all the processors. The amount of shared memory available

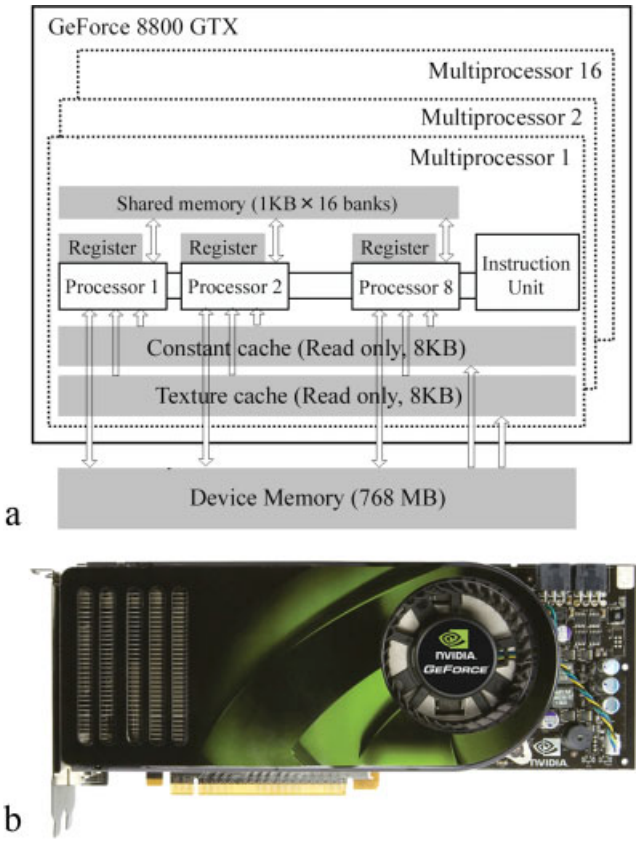

Figure 2. (a) Structure of the GeForce 8800 GTX chip. A chip has 16 multiprocessors (box of thin dashed line), which contain a constant cache, a texture cache, a shared memory and eight processors. (b) NVIDIA GeForce 8800 GTX Graphics Card. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

per multiprocessor is 16 kilo bytes (KB) divided into 16 banks. (iii) A read-only constant cache that is shared by all the processors to speed up reads from the constant memory space. The amount of constant memory available is 64 KB with a cache of 8 KB per multiprocessor. (iv) A read-only texture cache that is shared by all the processors to speed up reads from the texture memory space. The size of cache is 8 KB per multiprocessor. We also have an off-chip device memory which is not cached and whose total size is 768 mega bytes. This GPU follows most of the IEEE-754 standard for single-precision binary floating-point arithmetic.

As a rule of thumb, we execute the intensive and highly data-parallel computations on GPU, while other complicated tasks that do not take time on the host computer. In the present study GPU is responsible for the screening and the evaluation of ERIs and the reduction of them to the Coulomb matrix. All the other calculations are executed on the host computer. Generally speaking, GPU suffers from a bottleneck to transfer data from or to CPU, because the bandwidth of the interface (several giga bytes per second) is much slower than GPU's processing speed (500 GFLOP per second). This communication speed is not sufficient for GPU to send all ERIs calculated to the host. Thus, the algorithm we adopted is as follows.

Since we have large amount of device memory we send to it all the shell-pair data and the density matrix elements before the calculation. As the calculation starts, 64 threads are launched on a multiprocessor. They first store the coordinates and the Gaussian exponents of the 16 bra shells (denoted as P) on the shared memory. The coordinates and exponents of a ket shell (denoted as Q) are broadcasted one by one to all the threads via the texture cache. Each Q shell is examined whether it gives, ERIs greater than the threshold λ_{GPU} between one of 16 P shells. If a Q shell is accepted, ERIs are calculated in parallel; otherwise the next one is examined. Thus, 64 threads calculate ERIs between 16 P shells and 4 Q shells concurrently. Interpolation tables of Rys roots and weights are broadcasted via the constant cache. The density matrix elements are broadcasted via the texture cache and the ERIs calculated are contracted with them to make the Coulomb matrix elements. The accumulation of the Coulomb matrix was done with the 48-bit multiprecision addition. These procedures are repeated until all the P and Q shell pairs in the near-field CFMM box are treated. Finally, only the Coulomb matrix elements are sent back to the host. Currently, symmetry of ERI was not used to reduce the computational cost. The kernel program of GPU and the interface were developed with the β release of the NVIDIA CUDA development toolkit.[19]

Results

The algorithm we presented so far was applied to the test molecules, Taxol and Valinomycin to verify the numerical accuracy and the computational efficiency. The 3-21G and 6-31G basis sets[9] and the local density approximation (LDA) and the Perdew 91 exchange-correlation functional (PW91)[10] were used. The specification of the host computer is as follows: Intel Pentium 4 processor running at 2.8 GHz with 2 giga byte of DDR 400 memory. Operating system is the Scientific Linux distribution 4.3.[20] Gaussian source program was compiled using the Intel Fortran compiler version 9.1.

Table 3 compares the maximum error of the matrix elements of the Coulomb potential in the Hermite Gaussian basis as a function of the threshold λ_{GPU}. The Coulomb matrix elements whose upper bounds are smaller than λ_{GPU} were calculated on GPU with the single precision, while others were calculated on host with the double precision. The reference value was calculated with the double precision. The default initial guess of Gaussian 03 was used. All the Coulomb terms are calculated on GPU in the "GPU only" results. As shown in it the errors of the matrix element decrease monotonically as the threshold does. They are about $10^{-5}\lambda_{GPU}$ in these examples. In particular, the errors of the "GPU only" results are about 10^{-5}. We also measured the mean-square errors of the Coulomb matrix elements. They are somewhat larger than but the same order of magnitude as the maximum errors. On the other hand, the average of the errors were very small, about 10^{-13}, due to the error cancellation.

Next we solved the Kohn-Sham equation iteratively to determine the total electronic energy for various values of λ_{GPU}. The default initial guess of Gaussian 03, CFMM, and the DIIS extrapolation were used. The convergence criterion needs some consideration. As shown in Table 3, our Coulomb matrix contains the numerical error δJ, which acts as the random perturbation. Hence, the tightest convergence of the density matrix we can achieve is about $\delta J/\delta\epsilon$, where $\delta\epsilon$ is the energy gap between the highest-occupied molecular orbital

Table 3. The Maximum Error of the Matrix Elements of the Coulomb Potential and the Error of the Converged Total Energies as a Function of the Threshold λ_{GPU}.

	Taxol		Valinomycin	
Threshold (λ_{GPU})	LSDA 3-21G	PW91 6-31G	LSDA 3-21G	PW91 6-31G
Error of Coulomb matrix				
GPU only	1.2[−5]	1.4[−5]	1.5[−5]	1.5[−5]
1	1.0[−5]	1.4[−5]	1.2[−5]	1.5[−5]
0.1	1.0[−6]	3.8[−6]	1.1[−6]	4.2[−6]
0.01	9.4[−8]	1.9[−7]	9.0[−8]	2.2[−7]
1×10^{-3}	9.6[−9]	2.0[−8]	9.2[−9]	2.0[−8]
1×10^{-4}	1.0[−9]	1.4[−9]	8.7[−10]	1.8[−9]
Error of total energy (a.u.)				
GPU only	−1.7[−3]	−3.2[−3]	−1.1[−3]	−1.1[−3]
1	−1.4[−3]	−3.0[−3]	−5.1[−4]	−9.4[−4]
0.1	−8.9[−5]	−8.6[−5]	−2.2[−5]	−1.4[−4]
0.01	−6.5[−7]	−6.3[−6]	−1.5[−6]	−9.6[−6]
1×10^{-3}	−2.0[−8]	−4.7[−7]	−2.2[−7]	−6.7[−7]
1×10^{-4}	−1.0[−8]	−2.0[−8]	−2.1[−7]	−3.0[−8]

The Coulomb matrix elements whose Schwartz upper bound is smaller than λ_{GPU} are calculated by GPU with the single precision. Numbers in the square brackets indicate powers of 10.

(HOMO) and the lowest-unoccupied molecular orbital (LUMO). We imposed the convergence criterion $10^{-4}\min(1, \lambda_{GPU})$ for the density matrix. Incremental Fock build was not used, because the error accumulation sometimes destabilizes the SCF convergence.

Table 3 summarizes the errors of the total energies as a function of λ_{GPU}. It clearly demonstrates that the energy error decreases as the threshold does. Energy errors become 10^{-5} a.u. (3 K) for $\lambda_{GPU} = 10^{-2}$, which is accurate enough for practical purposes. On the other hand, the energy errors of the "GPU only" results are much larger, about 10^{-3} a.u. (300 K). They are not appropriate for the production run. However, the density matrix converged provides the accurate initial guess. Only two or three SCF iterations are required to converge for other values of λ_{GPU}. We also found that the convergence speed of the SCF iterations was almost independent of λ_{GPU}. The energies of HOMO and LUMO were reproduced well. The threshold $\lambda_{GPU} = 1 \times 10^{-3}$ reproduces them within the error of 10^{-6} a.u.

Table 4 summarizes the computational time of the Coulomb potential evaluation during the whole SCF iteration. In this table, the computational time elapsed on host was further divided into four: the integral screening, ERI calculation, CFMM, and others. The GPU time in this table is the sum of the data transfer, the integral screening, and the ERI calculation. The default SCF convergence threshold of Gaussain 03 was used. Incremental Fock formation was used for the "Host only" calculation.

As shown in this table, GPU clearly accelerates the evaluation of the Coulomb potential. Total computational time of the threshold $\lambda_{GPU} = 10^{-3}$ is about 1/3 of the reference ("Host only") one. The ERI calculation time in this table indicates that about 80–90% of ERIs were evaluated on GPU. If all ERIs are evaluated on GPU, we have 8–15 times of acceleration. Although this "GPU only" results do not give the energy of production-quality, the density matrix serves as the accurate initial guess. Hence the best way is to evaluate all ERIs on GPU in the early SCF iteration, and after the modest convergence the threshold $\lambda_{GPU} = 10^{-3}$ is used. We also examined the data transfer time between the host and GPU. It was only about 2% of the GPU time.

It is interesting to compare the actual computation time with the theoretical peak performance. The peak performance of the host and of GPU are 5.6 GFLOPS and 350 GFLOPS, respectively. Although GPU is 60 times faster than the host at most, we got only 8–15 times of the speedup. There is much room for improvement, for example, the use of the two-electron integral symmetry and the pipelining the instruction. We feel that the latter is more important. Since the evaluation of ERIs is the complicated procedure it uses registers more than 40 at most. Thus, we cannot launch enough threads to hide the register read after write stall. Better software pipelining overcomes this bottleneck, and hence better compiler is anticipated.

Finally, we shortly discuss issues not covered in this article. The evaluation of the exchange-correlation potential remains the major computational bottleneck. It consumes CPU time longer than the Coulomb potential evaluation. We can calculate it efficiently on GPU, because it requires a little memory due to the grid formulation. Fock matrix diagonalization can be reduced to the calculation of the matrix product in the linear-scaling method. GPU evaluates the matrix product efficiently. The acceleration of them as well as the full DFT calculation on GPU is now in progress and will be presented in future publication.

Table 4. Computational Time in Seconds of the Coulomb Potential Evaluation During the SCF Iteration.

	Host					
(λ_{GPU}) Threshold	Integral screening	ERI calculation	CFMM	Other	GPU	Total
Taxol/LSDA/3-21G						
GPU only	0.0	0.0	14.7	6.8	16.8	21.6
1×10^{-3}	19.7	20.5	12.0	7.7	12.3	60.0
Host only	27.7	118.1	10.1	9.6	0.0	165.5
Taxol/PW91/6-31G						
GPU only	0.0	0.0	17.9	14.1	31.9	32.0
1×10^{-3}	52.1	39.4	17.8	18.5	29.2	127.8
Host only	48.1	395.2	16.7	18.2	0.0	478.1
Valinomycin/LSDA/3-21G						
GPU only	0.0	0.0	24.0	12.1	23.8	36.1
1×10^{-3}	35.7	26.5	20.4	13.2	18.4	95.8
Host only	46.1	157.2	19.9	14.9	0.0	238.0
Valinomycin/PW91/6-31G						
GPU only	0.0	0.0	32.6	31.8	57.4	64.5
1×10^{-3}	93.6	54.1	27.8	33.0	45.4	208.6
Host only	74.2	470.2	22.8	24.1	0.0	591.3

The entries "GPU only" and "Host only" indicate that all the near-field Coulomb terms are calculated by the GPU and by the host, respectively. The time spent by the host is further divided into the integral screening, ERI calculation, CFMM, and others.

Conclusions

In this article we demonstrated that we can evaluate the Coulomb potential in the *ab initio* density functional calculation on the graphics processor unit (GPU). The numerical accuracy required for the algorithm was investigated in detail. We showed that GPU, which supports only the single-precision floating-point number natively, can evaluate most of the two-electron repulsion integrals (ERIs) without depredating the accuracy of the total energy. Because of the limited size of the working memory the Gauss-Rys quadrature to evaluate ERIs is investigated in detail. We presented the error analysis and the new interpolation formula of the roots and weights, which is suitable for the processor of the single-instruction multiple-data type. We propose to evaluate ERIs on GPU whose upper bounds are smaller than the threshold, while the others on the host. The algorithm was implemented on NVIDIA GeForce 8800 GTX and the Gaussian 03 program suite. It was applied to the test molecules Taxol and Valinomycin. The accuracy of the Coulomb potential and the total energies indicate that the algorithm proposed works fine and that we can control the numerical error by the threshold. The preliminary results show the considerable speedup over the commodity microprocessor.

Acknowledgements

This paper is dedicated to Prof. H. Nakatsuji on the occasion of his retirement from Kyoto University. The author also acknowledges Prof. J. Makino at National Astronomical Observatory of Japan, Prof. T. Ebisuzaki at RIKEN, and Dr. Matsubara at RIKEN for the valuable discussion and encouragements. Some parts of this work stem from our joint work of the GRAPE-DR project.

Appendix

In this appendix we discuss the small R limit of the functions $G(l, s)$ and $H(l, s)$. First of all their large R limit is not interesting because the scaled Rys root s_{nk} approaches some finite value as $R \to \infty$. Because of the denominator R^{-l} the large R limit play no role to the upper bound of ERI, eq. (18).

As R approaches zero s_{nk} are proportional to $\sqrt{\alpha}R$, and the lowest-order terms in the Hermite polynomial dominate in eqs. (17) and (21). The lowest-order term of the i-th Hermite polynomial $H_i(x)$ is $(-2)^{i/2}(i-1)!!$ for even i and $-(-2)^{(i+1)/2}i!!x$ for odd i. Thus, the maximum value of $H_t(n_x s)H_u(n_y s)H_v(n_z s)$ in eq. (17) is $(-2)^{l/2}(l-1)!!$ for even $l = t+u+v$ and $-(-2)^{(l+1)/2}l!!s$ for odd l.

The limit of the function $H(l, s)$ can be derived similarly. It approaches $(-2)^{l/2}(i-1)!!ls^{l-1}$ for even l and $-(-2)^{(l+1)/2}l!!ls^{l-1}$ for odd l. Thus, the maximum value of $H_t(n_x s)H_u(n_y s)H_v(n_z s)$ in eq. (21) is $(-2)^{l/2}(l-1)!!$ for even $l = t+u+v$ and $-(-2)^{(l+1)/2}l!!s$ for odd l.

References

1. (a) Kohn, W.; Sham, L. J Phys Rev 1965, 140, A1133; (b) Parr, R. G.; Yang, W. Density-Functional Theory of Atoms and Molecules; Oxford University Press: New York, 1989; (c) Gross, E. K. U.; Dreizler, R. M. Density Functional Theory; Plenum: New York, 1995.
2. Dupuis, M.; Rys, J.; King, H. F. J Chem Phys 1976, 65, 111.
3. King, H. F.; Dupuis, M. J Comput Phys 1976, 21, 144.
4. Gaussian 03, Revision B.01, Frisch, M. J.; Trucks, G. W.; Schlegel, H. B.; Scuseria, G. E.; Robb, M. A.; Cheeseman, J. R.; Montgomery, J. A., Jr.; Vreven, T.; Kudin, K. N.; Burant, J. C.; Millam, J. M.; Iyengar, S. S.; Tomasi, J.; Barone, V.; Mennucci, B.; Cossi, M.; Scalmani, G.; Rega, N.; Petersson, G. A.; Nakatsuji, H.; Hada, M.; Ehara, M.; Toyota, K.; Fukuda, R.; Hasegawa, J.; Ishida, M.; Nakajima, T.; Honda, Y.; Kitao, O.; Nakai, H.; Klene, M.; Li, X.; Knox, J. E.; Hratchian, H. P.; Cross, J. B.; Adamo, C.; Jaramillo, J.; Gomperts, R.; Stratmann, R. E.; Yazyev, O.; Austin, A. J.; Cammi, R.; Pomelli, C.; Ochterski, J. W.; Ayala, P. Y.; Morokuma, K.; Voth, G. A.; Salvador, P.; Dannenberg, J. J.; Zakrzewski, V. G.; Dapprich, S.; Daniels, A. D.; Strain, M. C.; Farkas, O.; Malick, D. K.; Rabuck, A. D.; Raghavachari, K.; Foresman, J. B.; Ortiz, J. V.; Cui, Q.; Baboul, A. G.; Clifford, S.; Cioslowski, J.; Stefanov, B. B.; Liu, G.; Liashenko, A.; Piskorz, P.; Komaromi, I.; Martin, R. L.; Fox, D. J.; Keith, T.; Al-Laham, M. A.; Peng, C. Y.; Nanayakkara, A.; Challacombe, M.; Gill, P. M. W.; Johnson, B.; Chen, W.; Wong, M. W.; Gonzalez, C.; Pople, J. A. Gaussian, Inc., Pittsburgh PA, 2003.
5. McMurchie, L. E.; Davidson, E. R. J Comput Phys 1978, 26, 218.
6. (a) White, C. A.; Head-Gordon, M. J Chem Phys 1996, 104, 2620; (b) Challacombe, M.; Schwegler, E. J Chem Phys 1997, 106, 5526.
7. White, C. A.; Johnson, B.; Gill, P.; Head-Gordon, M. Chem Phys Lett 1994, 230, 8.
8. Gill, P. M. W. Adv Quant Chem 1994, 25, 141.
9. (a) Hehre, W. J.; Stewart, R. F.; Pople, J. A. J Chem Phys 1969, 51, 2657; (b) Binkley, J. S.; Pople J. A.; Hehre W. J. J Am Chem Soc 1980, 102, 939; (c) Hehre, W. J.; Ditchfield, R.; Pople, J. A. J Chem Phys 1972, 56, 2257; (d) Dill, J. D.; Pople, J. A. J Chem Phys 1975, 62, 2921.
10. (a) Perdew, J. P. Electronic Structure of Solids '91; Akademie Verlag: Berlin, 1991, 11; (b) Perdew, J. P.; Burke, K.; Wang, Y. Phys Rev B 1996, 54, 16533.
11. IEEE standard for binary foating-point arithmetic; ANSI/IEEE Standard, Std 754-1985, New York, 1985.
12. Knuth, D. The Art of Computer Programming, vol. 2: Seminumerical Algorithms; Addison Wesley: MA, 1998.
13. Dekker, T. J. Numerische Mathematik 1971, 18, 224.
14. Hida, Y.; Li, X.; Bailey, D. H. In the 15th IEEE Symposium on Computer Arithmetic; Vail, Co, 2001, 155.
15. Helgaker, T.; Jorgensen, P.; Olsen, J. Molecular Electronic-Structure Theory; John Wiley: New York 2000.
16. Shavitt, I. Methods in Computational Physics; Academic Press: New York, 1963, Vol. 2, p. 1.
17. Obara, S.; Saika, A. J Chem Phys 1986, 84, 3963.
18. Ishida, K. Int J Quant Chem 1996, 59, 209.
19. http://www.nvidia.com/.
20. https://www.scientificlinux.org/.