JCTC Journal of Chemical Theory and Computation

Accelerating Density Functional Calculations with Graphics Processing Unit

Koji Yasuda*

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

Received March 27, 2008

Abstract: An algorithm is presented for graphics processing units (GPUs), which execute single-precision arithmetic much faster than commodity microprocessors (CPUs), to calculate the exchange-correlation term in ab initio density functional calculations. The algorithm was implemented and applied to two molecules, taxol and valinomycin. The errors in the total energies were about 10^{-5} a.u., which is accurate enough for practical usage. If the exchange-correlation term is split into a simple analytic model potential and the correction to it, and only the latter is calculated with the GPU, the energy error is decreased by an order of magnitude. The resulting time to compute the exchange-correlation term is smaller than it is on the latest CPU by a factor of 10, indicating that a GPU running the proposed algorithm accelerates the density functional calculation considerably.

I. Introduction

Streaming processors such as graphics processing units (GPUs) are attracting attention as high-performance computing devices. They are potentially several times faster than commodity central processing units (CPUs) for calculating single-precision floating-point numbers because they are specialized for computation-intensive, highly data-parallel computations. They are designed such that more transistors are devoted to data processing than to data caching and flow control. Most transistors in CPUs are devoted to cache memory and control logic to reduce memory latency and pipeline stalls. The high performance of GPUs makes them an attractive way to accelerate scientific computation. GPU computation has been successfully applied to various types of applications, including gravitational dynamics,[1] molecular dynamics,[2] quantum chemistry,[3–5] and quantum Monte Carlo.[6]

However, the streaming processors have several shortcomings. They are optimized for applications in which numerical accuracy is less important than performance. Most of them do not support double-precision floating-point numbers natively. Some of them support double-precision numbers but do not exhibit high performance. The total amount of working memory (registers and caches) on a chip is small. To avoid the pipeline stalls many parallel threads must be run concurrently. Consequently the usefulness of streaming processors for certain applications is not clear a priori. Moreover, to obtain high performance, they require a fine-grade parallel algorithm that can run with few hardware resources. Such an algorithm would differ from the usual, coarse-grained parallel algorithm for conventional CPUs.

A potential area for applying GPUs is structural biology. Advances in structural biology have stimulated the investigation of the electronic structure of biological molecules. Investigations have traditionally used density functional theory (DFT) for the ground-state and time-dependent DFT for the excited states.[7] It is now becoming possible to perform all-electron ab initio calculations of macromolecules for certain molecular structures. Along with this there is a growing demand for much faster ab initio methods, as current algorithms and computers are not sufficient for elucidating the free energy surface or the relaxation of the excited state of a macromolecule from the first principle. There is thus an opportunity for high-throughput processors like GPUs to play an active role.

A density functional calculation consists of many complicated procedures, but only a few of them account for most of the computational time. Table 1 shows, for example, the computational time of the major steps for the molecules taxol and valinomycin. The Gaussian03 program,[8] the generalized

* Corresponding author e-mail: yasudak@is.nagoya-u.ac.jp.

Table 1. Computational Time for Major Steps in Density Functional Calculation Using PW91 Functional[9] and Default Parameters of Gaussian03 Program[a]

	user time in seconds (percentage)			
	taxol ($C_{47}H_{51}NO_{14}$)		valinomycin ($C_{54}H_{90}N_6O_{18}$)	
procedure	3−21G	6−31G	3−21G	6−31G
Coulomb	125 (6.9)	475 (13.5)	204 (6.1)	589 (11.3)
Fock Diag	82 (4.6)	116 (3.3)	201 (6.0)	225 (4.3)
Ex-Cor	1598 (88.4)	2922 (83.1)	2947 (87.8)	4383 (84.3)
Grid	190 (10.5)	287 (8.2)	451 (13.4)	540 (10.4)
χ	91 (5.0)	174 (4.9)	154 (4.6)	234 (4.5)
ρ	458 (25.3)	868 (24.7)	775 (23.1)	1205 (23.2)
v_{xc}	837 (46.3)	1561 (44.4)	1533 (45.7)	2363 (45.4)
total	1807 (100)	3516 (100)	3355 (100)	5200 (100)

[a] Grid: generation of points and weights; χ: calculation of AOs and gradient; ρ: calculation of density and gradient; v_{xc}: calculation of AO matrix elements of exchange-correlation potential.

gradient approximation of the PW91 functional,[9] and the 3−21G and 6−31G basis sets[10] were used. About 90−95% of the total computational time was used to solve the self-consistent field (SCF) equation. About 10% of it was used to evaluate the Coulomb potential, 85% to evaluate the exchange-correlation potential, and 5% to diagonalize the Fock matrix. The density functional calculation can be accelerated considerably if we execute these time-consuming steps on a GPU.

An algorithm for GPUs that can be used to evaluate the two-electron integrals and the Coulomb potential was previously reported.[3] Quite recently Ufimtsev and Martinez[5] proposed a similar algorithm for evaluating two-electron integrals and presented promising results. This paper describes an algorithm for GPUs to evaluate the exchange-correlation term, which is the most time-consuming step in the density functional calculation. Since this step is essentially numerical quadrature with a three-dimensional grid, it is suitable for parallel processing. Some ideas to avoid the limitations of GPUs, few hardware resources and low numerical accuracy, are also described. The algorithm was implemented and applied to two molecules, taxol and valinomycin, and the total energy was accurate enough for practical usage. The computational time was an order of magnitude lower than that of latest commodity CPUs. GPUs thus offer a considerable speedup of the density functional calculations.

II. Algorithm

The major computational task in the density functional calculation is to evaluate the exchange-correlation energy E_{xc} and the matrix elements of the exchange-correlation potential v_{xc} in terms of the basis functions (AOs), $\chi_k(r)$, for a given σ-electron density, $\rho_\sigma(r)$.[11]

$$E_{xc} = \int \varepsilon(\rho_\alpha, \rho_\beta, \gamma_{\alpha\alpha}, \gamma_{\alpha\beta}, \gamma_{\beta\beta})d^3r \tag{1}$$

$$v_{xc} = \int \left[\frac{\partial \varepsilon}{\partial \rho_\alpha}\chi_k\chi_l + \left(\frac{2\,\partial\varepsilon}{\partial \gamma_{\alpha\alpha}}\nabla\rho_\alpha + \frac{\partial\varepsilon}{\partial\gamma_{\alpha\beta}}\nabla\rho_\beta\right)\cdot\nabla(\chi_k\chi_l)\right]d^3r$$
$$\gamma_{\sigma\sigma'} = \nabla\rho_\sigma(r)\cdot\nabla\rho_{\sigma'}(r) \tag{2}$$

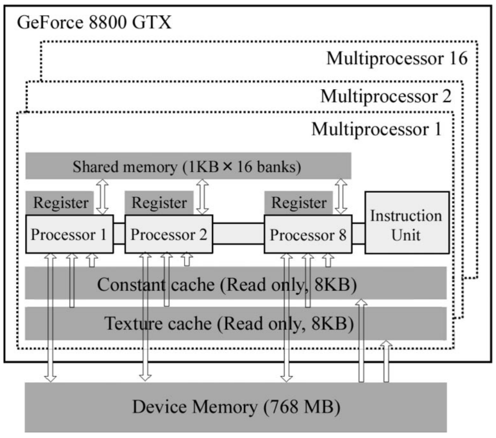

Figure 1. GeForce 8800 GTX chip has 16 multiprocessors, each containing a constant cache, a texture cache, a shared memory, and 8 processors.

The analytic integration of eqs 1 and 2 is impossible because ε is a complicated function of the electron density and gradient. We can evaluate them accurately by using the three-dimensional quadrature points (r_i) and the weights (w_i). For example, the exchange-correlation energy is given by

$$E_{xc} = \sum_i w_i\varepsilon(r_i),$$
$$\varepsilon(r_i) = \varepsilon(\rho_\alpha(r_i), \rho_\beta(r_i), \gamma_{\alpha\alpha}(r_i), \gamma_{\alpha\beta}(r_i), \gamma_{\beta\beta}(r_i)) \tag{3}$$

Any functions of position operator $\hat{r}$, like the electron density and ε, share the eigenfunctions of $\delta(r\text{-}r_0)$. Hence, the spatial grid is a natural choice and is suitable for evaluating ε and its derivatives, $\partial\varepsilon/\partial\rho_\sigma$ and $\partial\varepsilon/\partial\gamma_{\sigma\sigma'}$. The quadrature consists of four distinct steps. (i) First, the quadrature points and weights are generated. (ii) Then the electron density and the gradient on these points are calculated. (iii) These values are converted into the function ε and its derivatives, and (iv) finally the exchange-correlation energy and the matrix elements of the exchange-correlation potential are evaluated. Usually, most of the computational effort is spent on the second and fourth steps, the evaluation of the electron density, and the matrix elements of the potential. Their costs are proportional to the number of grid points and to the square of the number of AOs. To accelerate the quadrature, we must at least execute these time-consuming steps on a GPU with the architecture described below.

A. Architecture of GeForce 8800. The GPU used was NVIDIA's GeForce 8800 GTX.[12] As shown in Figure 1, it consists of 16 sets of SIMD multiprocessors with on-chip shared memory. Each multiprocessor is composed of eight processors. At any given clock cycle, each processor in each multiprocessor executes the same instruction but for different data. Each multiprocessor has four kinds of on-chip memory: (i) 1024 local registers per processor, (ii) memory shared by all the processors (the amount available to each multiprocessor is 16 KB divided into 16 banks), (iii) read-only constant cache that is shared by all the processors to speed up reads from the constant memory space (the amount available is 64 KB with a cache of 8 KB per multiprocessor), and (iv) read-only texture cache that is shared by all the processors to speed up reads from the texture memory space

(8 KB per multiprocessor). There is also an off-chip 768 MB memory device that is not cached. Its latency is about 400−600 clock cycles. The latency of the registers and the shared and global memories is hidden if enough independent threads are launched. It is recommended that more than 196 threads be run concurrently on each multiprocessor. The threads can be synchronized on a multiprocessor but not on different multiprocessors.

This GPU follows most of the IEEE-754 standard for single-precision binary floating-point arithmetic. The IEEE 754 compliant, single-precision floating-point number a is represented by[13]

$$\begin{aligned} &a = \pm 2^e m, \\ &-125 \le e \le 128, \\ &1 \le m < 2 \end{aligned} \tag{4}$$

where exponent e is an integer, and mantissa m is a 24-bit binary decimal. The relative error in the single-precision number is $2^{-23} \approx 1.2 \times 10^{-7}$, so the absolute error is about $2^{-23} \times |a|$. The results of the addition and the multiplication are rounded off. The throughput of the instructions is four cycles for floating-point addition and multiplication and for fused addmul operation.

B. Quadrature Grid. The quadrature grid used is the standard pruned one of 75 radial shells and 302 angular points centered at each atom. The number of grid points is about 7×10^3 per atom. The nearby points in the same cube, called a "grid box", are grouped. The cube is 4 a.u. in each dimension. The weights of the quadrature are generated with the fuzzy Voronoi cell method of Becke.[14] Grid generation requires little computational time compared to generation of the weights (5−13% of time, Table 1). Note that the Gaussian03 program recalculates them at every SCF iteration. The computational effort is reduced by calculating them only once on the host computer and then storing them in the external storage.

C. Basis Functions on Points. We need the values of the AOs, $\chi_k(r_i)$, and their gradient, $\nabla\chi_k(r_i)$, at grid points r_i. The contracted Cartesian Gaussian basis function centered at A is given by[15]

$$\begin{aligned} &\chi_k(x, y, z) = (x-A_x)^{a_x}(y-A_y)^{a_y}(z-A_z)^{a_z}\sum c_A \exp(-aR_A^2), \\ &R_A^2 = (x-A_x)^2 + (y-A_y)^2 + (z-A_z)^2 \end{aligned} \tag{5}$$

where c_A is the contraction coefficient, a is the primitive Gaussian exponent, and a_x, a_y, and a_z are non-negative integers. Their evaluation requires about 5% of the computational time (Table 1). The calculation of sufficiently small terms is skipped in the Gaussian03 program as well as in the present algorithm. The absolute minimum of eq 5 and the gradient is calculated for each grid box, and only those AOs with a minimum greater than threshold λ_{AO} are kept. They are called "significant AOs". Every batch of points has a different set of significant AOs (30−700 in the examples studied). Thus, except for small molecules, it is not realistic to keep all the calculated $\chi_k(r_i)$ and $\nabla\chi_k(r_i)$ in the external storage for later use. They have to be recalculated at every SCF iteration. The wide range in the number of significant AOs means that constructing an efficient algorithm requires special care.

From the significant AOs the GPU calculates the density and the density gradient at the grid points as well as the AO matrix elements of the exchange-correlation potential. Because of the huge number of terms and the slow communication speed between the host memory and GPU, it is best to calculate $\chi_k(r_i)$ and $\nabla\chi_k(r_i)$ on the GPU as well. However, the internal memory (4096 words per multiprocessor) is not large enough to store the calculated AOs, even for a small batch of points, as there are potentially 700 × 4 words for each grid point. It is in principle possible to store some of them on a large, external graphic memory at the expense of long latency. However it was found to be better to deal with every 32 AOs in turn in order to keep as many as possible in the shared memory. Trial and error indicated that the best implementation is that 128 threads on a multiprocessor calculate 4 primitive Gaussians of the 32 significant AOs on 32 points in parallel.

Anderson et al. reported an optimized quantum Monte Carlo (QMC) algorithm[6] for a different GPU (NVIDIA 7800 GTX), in which the basis function evaluation on spatial points is the one of the rate-limiting steps. The computational effort of QMC differs from that of DFT considerably. About 73% is focused on basis function evaluation and 11% on dense matrix multiplication. On the other hand in DFT only 5% is used for the basis function evaluation (Table 1). Most of the effort is used to generate the electron density and the exchange-correlation potential, which are essentially the matrix multiplication processes. Another noticeable difference is that the proposed algorithm is a linear-scaling one, because it screens the grid points and the basis functions. In the QMC algorithm, localization procedures which lead to sparser matrices are ignored. Since the matrices with the proposed algorithm are much smaller than the QMC ones, they must be kept on faster shared memory to avoid memory latency. Moreover the lack of enough shared memory means that some of the basis function values must be recalculated. However, the low latency of the shared memory more than compensates for this extra recalculation.

D. Electron Density and Gradient. The electron density, $\rho_\sigma(r_i)$, and gradient, $\nabla\rho_\sigma(r_i)$, at the grid point are calculated for a given first-order reduced density matrix (1-RDM) of the σ-electron P^σ.

$$\rho_\sigma(r_i) = \sum_{k,l} P^\sigma_{kl}\chi_k(r_i)\chi_l(r_i) \tag{6}$$

$$\nabla\rho_\sigma(r_i) = 2\sum_{k,l} P^\sigma_{kl}\chi_k(r_i)\nabla\chi_l(r_i) \tag{7}$$

The summation runs over the significant AOs. As shown in Table 1, this step consumes 20−25% of the computational time, mostly for the calculation of the matrix-vector products, $A_l(r_i) = \Sigma_k P^\sigma_{kl}\chi_k(r_i)$. The $\rho_\sigma(r_i)$ and $\nabla\rho_\sigma(r_i)$ are calculated on a GPU as follows. First, the host computer sends the grid points, significant AOs, and 1-RDM elements to the external graphic memory. The GPU calculates the values of a set of 32 significant AOs, χ_k, on 32 points, r_i, concurrently. It then reads a 32 × 32 submatrix of P_{kl} from the external graphic memory and multiplies it by $\chi_k(r_i)$ to form product $A_l(r_i)$.

Next it calculates $\chi_l(r_i)$ and $\nabla\chi_l(r_i)$, which are multiplied by $A_l(r_i)$, to form the density and the gradient. Once all AOs have been processed, the results are sent back to the host computer. The communication time to send the 1-RDM elements from the host computer to the GPU dominates the total communication time, which is about 10% of the GPU computation time in the example below.

E. Functional on Points. The electron density and gradient at the grid points are then converted into the exchange-correlation potential, ε, and its derivatives, $\partial\varepsilon/\partial\rho_\sigma$ and $\partial\varepsilon/\partial\gamma_{\sigma\sigma'}$. They are converted into the total exchange-correlation energy [eq 3] and the quantities f and **g**.

$$f(r_i) = \frac{w_i}{2}\frac{\partial\varepsilon(r_i)}{\partial\rho_\alpha} \tag{8}$$

$$g(r_i) = w_i\left(\frac{2\,\partial\varepsilon(r_i)}{\partial\gamma_{\alpha\alpha}}\nabla\rho_\alpha + \frac{\partial\varepsilon(r_i)}{\partial\gamma_{\alpha\beta}}\nabla\rho_\beta\right) \tag{9}$$

The host computer executes this step with double-precision accuracy, because the computational cost scales linearly with the number of points, so it requires little computational effort.

F. Matrix Elements of Exchange-Correlation Potential. Finally, the GPU calculates

$$A_{kl} = \sum_i B_{ki}\chi_l(r_i) \tag{10}$$

$$B_{ki} = f(r_i)\chi_k(r_i) + \mathbf{g}(r_i)\cdot\nabla\chi_k(r_i) \tag{11}$$

The sum runs over the batch of grid points, while the indices k and l run over the significant AOs. The AO matrix elements of the exchange-correlation potential, eq 2, are given by $v_{xc} = A_{kl}+A_{lk}$. This last step, which is essentially the evaluation of the matrix-matrix product, eq 10, consumes 40−45% of the computational time (Table 1). For a given batch of points, the GPU first evaluates a set of 64 AOs and the gradient for 32 points concurrently. They are converted into the matrix B_{ki} and stored in the internal shared memory. The matrix product between B_{ki} and $\chi_l(r_i)$ is calculated, and A_{kl} is updated. Because of the memory limitation, A_{kl} is kept in the external graphic memory. After all points are processed, the calculated matrix elements are sent back to the host computer. This communication time dominates the total communication time; it takes about 25% of the GPU computational time.

III. Accuracy Requirement

At first glance, the most serious issue with the present GPU is the numerical accuracy: it supports only single-precision floating-point numbers natively, and ab initio calculation obviously requires double-precision arithmetic. For example, the total electronic energy of a molecule should be calculated within an accuracy of 10^{-4} a.u., which is 1/10th the thermal energy of room temperature. Since the total electronic energy is about 40 au per carbon atom, the roundoff error of the total energy of a large molecule surpasses it. It is in principle possible to carry out multiprecision addition and multiplication by software.[16] However, since this is expensive, most operations should be done with single-precision accuracy.

Since the roundoff error should be unbiased and nearly random, the sum of the N single-precision numbers with an absolute value is about s and contains the relative error of $2^{-23}sN^{-1/2}$. It decreases as the number of terms increases. Single-precision arithmetic can be used when this error estimate is smaller than the tolerance.

Table 2. Exponent and Coefficient Values of Model Exchange-Correlation Potential [Eq *23*].

element	C		N		O		H
ζ	3.0	1.0	3.0	1.0	3.0	1.0	2.0
d	−0.198	−2.035	−0.847	−1.783	−1.616	−1.402	−1.658

The two strategies previously proposed[3] for calculating the Coulomb potential on a GPU are to use the GPU only in the early SCF iterations and to calculate the small electron repulsion integrals (ERIs) by GPU. The absolute errors of these ERIs are small. The host computer calculates the rest of the large ERIs with double-precision accuracy. These two strategies were found to work well. The accuracy requirement for the exchange-correlation term should be less severe than for the Coulomb term because the former is generally an order of magnitude smaller than the latter.

The exchange-correlation matrix elements should allow lesser accuracy than the density and the gradient because of the variational principle and the conservation of the number of electrons. The small error in the matrix elements induces a small change in the SCF density. However, since the SCF density satisfies the energy variational principle, $\delta E/\delta\rho(r) = \mu$, the first-order energy change always vanishes.

A deviation in the density or the gradient induces energy error of the first-order through eq 1. This error is reduced as follows. Two model potentials, $v_{local}(r)$ and $v_{gc}(r)$, that respectively mimic the local and gradient-dependent parts of the exchange-correlation potential at the ground-state density are used.

$$v_{local}(r) \approx \frac{\partial\varepsilon}{\partial\rho_\alpha} \tag{12}$$

$$v_{gc}(r) = \nabla\cdot\mathbf{V} \tag{13}$$

$$\mathbf{V}(r_i) \approx -\left(\frac{2\,\partial\varepsilon}{\partial\gamma_{\alpha\alpha}}\nabla\rho_\alpha + \frac{\partial\varepsilon}{\partial\gamma_{\alpha\beta}}\nabla\rho_\beta\right) \tag{14}$$

The total exchange-correlation energy is given as the sum of the reference energy, E_1, and the correction to it, E_2.

$$E_{xc} = E_1 + E_2 \tag{15}$$

$$E_1 = \int\{v_{local}(r) + v_{gc}(r)\}\rho(r)d^3r = Tr\{(v_{local} + v_{gc})P\} \tag{16}$$

$$E_2 = \sum_i w_i\{\varepsilon(r_i) - v_{local}(r_i)\rho(r_i) + \mathbf{V}(r_i)\cdot\nabla\rho(r_i)\} \tag{17}$$

The total 1-RDM and the density are denoted as $P = P^\alpha + P^\beta$ and $\rho = \rho_\alpha + \rho_\beta$, respectively. The equivalence of eqs 1 and 15 can be seen from Green's theorem.

$$\int(v_{gc}\rho + \mathbf{V}\cdot\nabla\rho)d^3r = 0 \tag{18}$$

The model potentials, $v_{local}(r)$ and $v_{gc}(r)$, are chosen so that the AO matrix elements can be calculated analytically. Thus, we can evaluate E_1 analytically with double-precision accuracy, so correction E_2 simply needs to be approximated by numerical quadrature with single-precision accuracy. Assuming that the model potential approximates well the true exchange-correlation potential and that the numerical quadrature is accurate, the first-order error of E_{xc} with respect to $\rho(r_i)$ and $\nabla\rho(r_i)$ vanishes because

$$\delta\varepsilon(r_i) \approx v_{local}(r_i)\delta\rho(r_i) - \mathbf{V}(r_i)\cdot\delta(\nabla\rho(r_i)) \qquad (19)$$

Note that the calculation of $v_{local}(r_i)$ and $v_{gc}(r_i)$ requires little computational effort. Vector field $\mathbf{V}$ represents the electronic field of the imaginary charge distribution, $v_{gc}(r)$.

We can also separate exchange-correlation potential v_{xc} into reference term v_1 and the correction to it, v_2.

$$v_{xc} = v_1 + v_2 \qquad (20)$$

$$v_1 = (k)v_{local} + v_{gc}|l| \qquad (21)$$

$$v_2 = \sum_i w_i\left[\left(\frac{\partial\varepsilon(r_i)}{\partial\rho_\alpha} - v_{local}(r_i)\right)\chi_k(r_i)\chi_l(r_i) + \left(\frac{2\,\partial\,\varepsilon(r_i)}{\partial\gamma_{\alpha\alpha}}\nabla\rho_\alpha(r_i) + \frac{\partial\varepsilon(r_i)}{\partial\gamma_{\alpha\beta}}\nabla\rho_\beta(r_i) + \mathbf{V}(r_i)\right)\cdot\nabla(\chi_k(r_i)\chi_l(r_i))\right] \qquad (22)$$

The reference term can be calculated analytically, so only the correction to it needs to be calculated with the GPU. The error in v_{xc} is lower because the former term is calculated exactly.

Here, the model potential v_{local} is expanded with the spherical Gaussian functions centered on each atom A, while v_{gc} is ignored.

$$v_{local} = \sum_A\sum_i d_i\exp(-\zeta_i(r-A)^2) \qquad (23)$$

The exponent and coefficient of the model potential used in this paper are summarized in Table 2. They were determined as follows. The self-consistent ground-state density of a molecule, $CH_3CH(CHO)NH_2$, was calculated using the PW91 functional[9] and 3−21G basis set.[10] The AO matrix elements of the local part of the exchange-correlation potential were calculated from this density. The linear coefficients of the model potential, d_i, were then determined by minimizing the error

$$L = \mathrm{Tr}\{P(v - v_{local})P(v - v_{local})\} \qquad (24)$$

where P is the ground-state 1-RDM. The same potential was used for the same elements in a molecule. The potentials for elements C, N, and O were expanded with the two spherical Gaussians, while that of H was expanded with only one Gaussian. The exponents ζ_i were determined from the dimensional argument and were not optimized.

IV. Results and Discussion

The algorithm described above was implemented using the beta release of the CUDA toolkit[12] to develop the GPU code, which was then linked with a modified Gaussian03 program.[8] In this section the accuracy of the total SCF energy and the exchange-correlation potential calculated with the GPU are first examined. The density functional calculations for taxol ($C_{47}H_{51}NO_{14}$) and valinomycin ($C_{54}H_{90}N_6O_{18}$) were performed using the 3−21G and 6−31G basis sets[10] and the PW91 functional[9] The electron density, the density gradient, and the matrix elements of the exchange-correlation potential were calculated with the GPU, while the rest were calculated on the host computer with double-precision accuracy. The host computer had a Pentium4 (2.8 GHz) CPU, 2 GB of memory, and the GeForce 8800 GTX graphic card; it ran Scientific Linux version 4.3.[17] An INTEL Fortran compiler (version 9) was used to compile the program.

Table 3 summarizes the errors in the total energy for various AO cutoffs and model potentials. The algorithm for GPU was also executed on the host computer with double-precision accuracy to calculate the reference values. The fast multipole method (FMM), the special Coulomb engine (FofCou), and the tight SCF convergence criterion were used. As shown in the table the errors were about 6 × 10^{-5} a.u. on average when the model exchange-correlation potential was not used. They were of the same order of magnitude as the default SCF convergence criterion for single-point calculation. The error remained almost constant as the AO cutoff was increased up to $\lambda_{AO} = 10^{-6}$, indicating that this threshold is high enough for the molecules studied. When the model exchange-correlation potential was used, the energy error decreased by an order of magnitude, to about 6

Table 3. Error in Total SCF Energy[a]

	energy error in a.u.			
	taxol ($C_{47}H_{51}NO_{14}$)		valinomycin ($C_{54}H_{90}N_6O_{18}$)	
λ_{AO}	3−21G	6−31G	3−21G	6−31G
	Without Model Potential			
10^{-3}	5.58[−5]	3.75[−5]	8.00[−5]	5.39[−5]
10^{-6}	5.81[−5]	5.30[−5]	6.92[−5]	5.82[−5]
10^{-15}	5.81[−5]	5.03[−5]	6.91[−5]	6.01[−5]
	With Model Potential			
10^{-3}	−8.80[−7]	−6.94[−6]	−1.71[−6]	−5.17[−6]
10^{-6}	5.98[−6]	5.17[−6]	8.20[−6]	6.08[−6]
10^{-15}	6.61[−6]	5.06[−6]	8.09[−6]	6.39[−6]

[a] λ_{AO} is the threshold for significant AOs. Reference values were calculated with double-precision accuracy, and $\lambda_{AO} = 10^{-15}$. Numbers in square brackets indicate powers of ten.

Table 4. Computational Time for Exchange-Correlation Terms[a]

	user time in seconds			
	taxol ($C_{47}H_{51}NO_{14}$)		valinomycin ($C_{54}H_{90}N_6O_{18}$)	
λ_{AO}	3−21G	6−31G	3−21G	6−31G
	GPU			
10^{-3}	36.4	45.5	66.1	75.0
10^{-6}	46.1	61.4	83.9	100.4
10^{-15}	73.7	110.1	140.8	193.8
	Host			
10^{-3}	667.9	955.2	932.1	1371
10^{-6}	1513	2385	2704	4106
10^{-15}	4449	6822	10577	15739
	Original Gaussian03			
	1598	2922	2947	4383

[a] GPU: electron density, gradients, and AO matrix elements of v_{xc} were calculated on GPU. Host: the same program was executed on host computer with double-precision accuracy.

Chart 1

```
GPU kernel for density
  arrays on shared mem: chi[64][32], rdmchi[32][32]
read a grid point from gmem
initialize $\rho$ and $\nabla\rho$
loop over AO batch
  chi = 0
  loop over primitives in batch
    read primitive information from gmem
    calculate basis functions on a grid point
    increase chi
  loop end
  loop over AO batch
    read RDM elements $P$ from gmem
    calculate matrix-vector product, rdmchi = $P\times$ chi
    loop over primitives in batch
      read primitive information from gmem
      calculate primitive functions and their derivatives
      multiply them by rdmchi
      increase $\rho$ and $\nabla\rho$
    loop end
  loop end
loop end

GPU kernel for potential
  arrays on shared mem: chi[32][32], fchi[64][32]
loop over grid points
  read a grid point, f, and g of Eqs. (8) and (9) from gmem
  loop over primitives in a batch
    read primitive information from gmem
    calculate primitive functions and their derivatives
    multiply them by f or g
    increase fchi
  loop end
  32 threads concurrently transpose matrix fchi
  loop over AO batch
    chi = 0
    loop over primitives in batch
      read primitive information from gmem
      calculate primitive functions on a grid point
      increase chi
    loop end
    read $A_{kl}$ of Eq. (10) from gmem
    calculate matrix product $A_{kl} = A_{kl}$ + chi $\times$ fchi
    write $A_{kl}$ to gmem
  loop end
loop end
```

$\times\ 10^{-6}$ a.u. on average. With the proposed algorithm, the GPU calculated total energies accurate enough for practical use. Note that the reference energy itself changed when the model potential was used. This difference, which is about 10^{-5} a.u., is attributed to the grid discretization error.

Finally, the computational time to calculate the exchange-correlation terms is compared. The results are summarized in Table 4. "Host" denotes the execution time of the same GPU algorithm on the host computer with double-precision accuracy. The execution time of the original Gaussian03 program is also shown. As shown in the table the GPU executed the same program about 40 times faster than the host computer (Pentium4, 2.8 GHz). The benchmark result for the same molecule[18] indicates that a standard commodity processor (INTEL Core2 Duo, 3.0 GHz) is about 4 times faster than the host CPU used here. Thus, the GPU used here is about five to ten times faster than the latest commodity CPU. Note that the GPU results include the communication and host code execution time. The host code, which calculates the grid weights and exchange-correlation potential on points, consumed about half the GPU time for $\lambda_{AO} = 10^{-6}$. The execution time of the original Gaussian03 program was close to the time for $\lambda_{AO} = 10^{-6}$ on the host computer. This is in accordance with the observation that $\lambda_{AO} = 10^{-6}$ is the best cutoff for the proposed algorithm without degradation of the energy. Note that the Gaussian03 program uses a more complex two-step procedure to select the significant AOs. In short, a GPU running the proposed algorithm accelerates the density functional calculation considerably.

However, the application of the GPU to quantum chemistry has a limitation. It is useful only for simple tasks. To minimize the communication time between the host computer and GPU, a series of tasks should be done on the GPU. However, the GPU's internal memory will soon run short storing the intermediate data. The evaluation of two-electron repulsion integrals between four contracted Gaussians is a fundamental process of ab initio calculation. However, it is difficult for the current GPU to execute efficient algorithms of the McMurchie-Davidson or Obara-Saika[15] type, because of the shortage of fast memory. Thus, we need a new algorithm to accelerate the hybrid density functional calculation. This is why Vogt et al. evaluated the electron repulsion integrals on a host computer and used a GPU for the postcalculation.[4] Other ab initio methodologies require more or less new algorithms. In particular, it would be hopeless to automatically convert the current ab initio program of the host computer to run on the GPU.

On the other hand, the issue of accuracy is much easier to manage in practice. We often have to solve an equation iteratively. A GPU would be suitable for such problems, since lower accuracy is allowed in the early stages of the iteration. An example is the diagonalization of the Fock matrix with the linear scaling algorithm.[19] This method minimizes the energy in terms of the purified 1-RDM iteratively. The major computational task is to calculate the gradient from the products of the Fock matrix and 1-RDM. A GPU can be used without problems in the early stages of this iteration. A study on accelerating the Fock matrix diagonalization is in progress, and the results will be published soon.

Acknowledgment. I am grateful to Prof. H. Nakatsuji at the Quantum Chemistry Research Institute, Prof. J. Makino at the National Astronomical Observatory of Japan, Prof. T. Ebisuzaki at RIKEN, and Dr. H. Matsubara at RIKEN for their encouragement. This research was supported in part by a grant from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

Appendix

The pseudocodes of the GPU kernels for calculating the density and the exchange-correlation potential are presented here (see Chart 1) . Each kernel describes the task of a thread; 128 threads on each multiprocessor execute the same kernel for different data, i.e., different grid points or basis functions. The host computer is responsible for organizing the input data, including the grid points, basis function parameters, and RDM elements. The abbreviation "gmem" stands for the external graphic memory on the GPU.

References

(1) Hamada, T.; Iitaka, T. http://arxiv.org/abs/astro-ph/0703100 (accessed April 1, 2007). See also http://www.gpgpu.org/ (accessed May 13, 2008).

(2) Stone, J. E.; Phillips, J. C.; Freddolino, P. L.; Hardy, D. J.; Trabuco, L. G.; Schulten, K. *J. Comput. Chem.* **2007**, *28*, 2618–2640.

(3) Yasuda, K. *J. Comput. Chem.* **2008**, *29*, 334–342.

(4) Vogt, L.; Olivares-Amaya, R.; Kermes, S.; Shao, Y.; Amador-Bedolla, C.; Aspuru-Guzik, A. *J. Phys. Chem. A* **2008**, *112*, 2049–2057.

(5) Ufimtsev, I. S.; Martinez, T. J. *J. Chem. Theory Comput.* **2008**, *4*, 222–231.

(6) Anderson, A. G.; Goddard, W. A., III; Schroeder, P. *Comput. Phys. Commun.* **2007**, *177*, 298–306.

(7) (a) Kohn, W.; Sham, L. J. *Phys. Rev.* **1965**, *140*, A1133−A1138. (b) Parr, R. G.; Yang, W. In *Density-Functional Theory of Atoms and Molecules*; Oxford University Press: New York, 1989. (c) Gross, E. K. U.; Dreizler, R. M. In *Density Functional Theory*; Plenum: New York, 1995.

(8) *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.

(9) (a) Perdew, J. P. In *Electronic Structure of Solids '91*; Akademie Verlag: Berlin, 1991; p 11. (b) Perdew, J. P.; Burke, K.; Wang, Y. *Phys. Rev. B* **1996**, *54*, 16533–16539.

(10) (a) Hehre, W. J.; Stewart, R. F.; Pople, J. A. *J. Chem. Phys.* **1969**, *51*, 2657–2664. (b) Binkley, J. S.; Pople J. A.; Hehre W. J, *J. Am. Chem. Soc.* **1980**, *102*, 939–947. (c) Hehre; Ditchfield, R.; Pople, J. A. *J. Chem. Phys.* **1972**, *56*, 2257–2261. (d) Dill, J. D.; Pople, J. A. *J. Chem. Phys.* **1975**, *62*, 2921–2923.

(11) Johnson, G. G.; Gill, P. M. W.; Pople, J. A. *J. Chem. Phys.* **1993**, *98*, 5612–5626.

(12) http://www.nvidia.com/object/cuda_home.html (accessed April 1, 2007).

(13) *IEEE standard for binary foating-point arithmetic*; ANSI/IEEE Standard, Std 754-1985, New York, 1985.

(14) Becke, A. D. *J. Chem. Phys.* **1988**, *88*, 2547–2553.

(15) Helgaker, T.; Jorgensen, P.; Olsen, J. In *Molecular Electronic-Structure Theory*; John Wiley: 2000; pp 336−426.

(16) (a) Knuth, D. In *The Art of Computer Programming, volume 2, Seminumerical Algorithms*; Addison Wesley: MA, 1998. (b) Dekker, T. J. *Numerische Mathematik* **1971**, *18*, 224–242. (c) Hida, Y.; Li, X.; Bailey, D. H. In *15th IEEE Symposium on Computer Arithmetic*; Vail, CO, 2001; p 155.

(17) https://www.scientificlinux.org/ (accessed April 1, 2007).

(18) http://www.hpc.co.jp/appli/solution/gaussian/gaussian_bench_test397.html (accessed April 1, 2007).

(19) (a) Li, X. P.; Nunes, R. W.; Vanderbilt, D. *Phys. Rev. B* **1993**, *47*, 10891–10894. (b) Daw, M. S. *Phys. Rev. B* **1993**, *47*, 10895–10898. (c) Goedecker, S. *Rev. Mod. Phys.* **1999**, *71*, 1085–1123.

CT8001046