JSST 2011
International Conference on Modeling and Simulation Technology
October 22-23, 2011, Tokai University Takanawa Campus,
Tokyo, JAPAN

Acceleration of computational quantum chemistry by heterogeneous computer architectures

Yuki Furukawa [1], Ryota Koga [1], and Koji Yasuda [2]

[1] X-Ability Co., Ltd., Tokyo, Japan

[2] Ecotopia Science Institute, Nagoya University, Nagoya, Japan

Abstract—Computational quantum chemistry mehods such as the Hartree-Fock (HF), the density functional theory (DFT) or the fragment molecular orbital (FMO) require heavy computational resources. In this study they are accelerated by using graphics processing units (GPUs) and the vector instruction set (AVX) of latest CPU. PRISM algorithm to evaluate the electron repulsion integrals was vectorized to utilize AVX as much as possible. We found that this new program makes the Fock matrix formation in HF 2 to 3 times faster than ever before. The Coulomb and the exchange-correlation potentials in DFT were evalualted on GPU, result in about 4 times overall speedup. The programs developed were used to accrelerate FMO. We found that our new algorithm and GPU are very suitable for the calculation of the environmental electrostatic potential. The total computational time was reduced to about 1/3.

Keywords—Fragment Molecular Orbital(FMO), GPGPU, ERI, GAMESS, SandyBridge, AVX

1 Introduction

The development of the *ab-initio* quantum chemistry and the growth of computational power make the electronic structure calculations a useful tool for the rational drag design. The fragment molecular orbital method (FMO) is one of the most famous and widely used *ab-initio* methods for proteins [1]. The quantum calculation gives us important information beyond the reach of classical molecular mechanical and coarse-grained ones. However the inherent high computational costs of quantum calculations and the huge number of drag candidates urge us to develop much faster simulation methods.

Traditionally the integration of more transistors on a chip was the mean of the performance increase of computers. As the size of a transistor approaches to the atomistic scale the performance of a computer chip will be limited by the power supply and heat problem, not by the number of transistors. Today and in future the performance per power consumption matters. A traditional CPU with longer SIMD (single instruction multiple data) unit and the streaming multiprocessors such as graphics processing units (GPUs) attract attention as the better performance per watt computing devices. In particular a GPU shows higher absolute performance and higher performance per watt than a traditional CPU by an order of magnitude today. So-called General purpose GPU (GPGPU) to use GPUs for other than the primary purpose of graphic processing is applied for various areas of computational science and engineering.

It was known for a long time that the evaluation of electron repulsion integrals (ERIs) dominates the computational cost of the Hartree-Fock (HF) and the density functional theory (DFT) calculations. Because of its intrinsic parallelism GPUs are expected to be suitable to execute this most time-consuming step. However it should be noted that a GPU has very different architecture from a traditional CPU. (i) It is a massively parallel multi-processor. Typically more than 500 processors are integrated on a chip. (ii) GPU has a high bandwidth dedicated memory, but because of the huge number of processors the bandwidth per processor is lower than that of CPU. (iii) Currently GPU is connected to CPU via PCI express bus, which is a bottleneck of data transfer. Thus tasks suitable to execute on a GPU should have the high parallelism, high computation, and low data transfer. It is almost clear that the ERI evaluation programs for CPUs could not run on a GPU as they are.

Previously one of the authors developed a special algorithm and programs to evaluate ERIs on GPUs [2,3]. It was reported that a GPU together with this special algorithm makes DFT calculation faster by an order of magnitude. Based on this work we developed the product-quality computation library, XA-CUDA-QM that is a part of XA-CHEM-SUITE [4], to accelerate widely available *ab-initio* program packages, such as GAMESS[5]. It was shown that XA-CUDA-QM and a NVIDIA's GPU successfully accelerate the evaluation of electrostatic potential in the FMO calculation [6]. In spite of this success there were two shortcomings in the previous study. (i) Only ERIs between *s* and *p* Gaussian basis functions were evaluated on GPUs. (ii) The calculation of HF exchange matrix in FMO was difficult accelerate by a GPU.

The first shortcoming stems from the limited cache size on GPUs. Typical integral evaluation algorithms calculate many intermediate integrals, which are kept on cache and memory, and then they are joined together to make the final ERIs. The number of intermediate integrals rapidly increases as the angular momentum of basis function increases. The number of single-precision words one can keep on GPU's cache was about 100 per processor, which was lower than the number of intermediate integrals to evaluate ERIs between d functions.

The second shortcoming stems from the bottleneck of data transfer between GPU and CPU. The HF exchange matrix elements calculated in FMO are rather small. Typically a fragment consists of 2 to 4 amino acid residues. The conventional self-consistent field (SCF), which calculates ERIs once and keeps them on an external storage for later use, is faster and hence preferable for this size of molecules. However because of the bottleneck explained above GPU would not be useful for conventional SCF. We also noticed that the performance of GAMESS program is much lower than the peak performance of a CPU. There should be much room for the improvements for CPU side code alone.

In this study we report a method to overcome these shortcomings. We report the special algorithm to calculate ERIs on GPU up to d functions. We also report better implementation of ERI evaluation on a CPU. Special attention is paid to fully utilize the SIMD facility of Intel Sandy Bridge CPU, advanced vector extensions (AVX).

2 Method

In this section the theoretical background and the implementation details of our algorithms are described.

2.1 Vectorized PRISM algorithm using AVX

AVX is a SIMD instruction set extension, which is installed on Intel Sandy Bridge processors or later and will be supported on AMD processors in near future. AVX utilizes the 256-bit vector registers, which is split into four double precision floating point number (64-bit), eight single precision floating point number (32-bit) or eight single precision integer number (32-bit). Traditional CPUs before Intel Sandy Bridge only has the 128-bit SIMD operation. Hence the CPUs with AVX can process twice many data at once than the previous ones.

In LCAO (linear combination of atomic orbital) approximation the electronic orbital is expressed as a linear combination of the contracted Gaussian type orbitals (GTOs). A basis function is the sum of the primitive GTOs $\chi_a = \sum_i c_{ai}\varphi_{ai}, \quad (a = 1,2,\cdots,N)$. Here the primitive GTO is

$$\varphi_{ai} = (x-A_x)^{a_x}(y-A_y)^{a_y}(z-A_z)^{a_z}\exp(-\alpha_{ai}|\vec{r}-\vec{A}|^2)$$

where $\vec{A}$ is the center of GTO and $L_a = a_x + a_y + a_z$ is the total angular momentum of it. ERI in the *ab-initio* calculation is the six-dimensional integral

$$(ab|cd) = \iint \chi_a(\vec{r_1})\chi_b(\vec{r_1})\frac{1}{|\vec{r_1}-\vec{r_2}|}\chi_c(\vec{r_2})\chi_d(\vec{r_2})\,d\vec{r_1}d\vec{r_2}.$$

The computational cost to evaluate all ERIs is formally $O(N^4)$, where *N* is the total number of primitive GTOs. This is the most time-consuming step in the HF or DFT calculation.

The basis functions that share the center and the exponent form a shell. ERIs are calculated for the quartet of shells one by one. For instance, $(pp|pp)$ shell quartet has 3 functions (p_x, p_y, p_z) for each shell, and 3^4=81 ERIs are calculated at once. This is because these 81 ERIs share the most of the intermediate integrals result in the drastic decrease of the computational cost.

Up to now many algorithms were proposed to evaluate the huge number of ERIs. Among them PRISM method [7] is one of the fastest algorithms for various kinds of GTOs. It is widely adopted in quantum chemistry packages such as Gaussian 09 [8]. Generally the ERI evaluation and the Fock matrix formation in Hartree-Fock method are as follows. (i) Evaluate Boys function of various orders for each primitive shell quartet. (ii) Transform them to ERIs using recurrence relations. (iii) Multiply ERIs by the density matrix elements and add them to the Fock matrix elements.

Boys function in the first step is a function written as

$$[0]^{(m)} \propto F_m(T) = \int_0^1 u^{2m}\exp(-Tu^2)du.$$

With a simple transformation of the variable *u* one notices that it is mathematically equivalent to the incomplete gamma function. Boys function is often approximated by the piecewise polynomial expansion, whose coefficients are determined in advance. The domain of *T* is split into many small segments and in each segment low-order polynomial is used to approximate the function. The issue is that it is not suited for SIMD parallelization, because the coefficient depends on the value of *T*, results in many conditional branching. Hence we split the domain into only four and the function is approximated 6th to 13th polynomials. The indirect addressing is also used to eliminate the branching. The second finding is that as in Ref. [2] the variable $S = \exp(-T/8)$ gives better polynomial expansion for Boys function. The third finding is the suitable direction of the recursion relation. The largest *m* we need is the total angular momentum of the shell quartet. For numerical stability it is generally recommended that one first evaluate $F_m(T)$ of the largest *m*, and then transform it to the lower one by using the downward recursion. However we found that the upward recursion is more stable when *T* is small.

In the process (iii) the ERIs evaluated are converted to

$$G_{ij} = \sum_{k,l} P_{kl}\left[(ij|kl) - \frac{1}{2}(ik|jl)\right].$$

The first term of the right hand side expresses the electrostatic potential (J matrix), while the second term the exchange potential (K matrix). In this step the memory access cost to density and Fock matrices is greater than the floating-point operation cost, so we expect little merit to use vector operations.

Various ERI evaluation algorithms differ in the way to transform Boys functions $[0]^{(m)}$ to the contracted ERIs (*ab*|*cd*) in the second step. This step consists of successive addition and multiplication, which is suitable for the vector operations.

We used the recurrence relations found by McMarchie and Davidson [9] to transform the Boys function to the final ERIs in this study. A product of two Cartesian Gaussian functions is rewritten as a sum of the Hermite Gaussian functions, for example,

$$(x-A_x)^{a_x}\exp\left[-\alpha(x-A_x)^2\right]\cdot(x-B_x)^{b_x}\exp\left[-\beta(x-B_x)^2\right] = \sum_{t=0}^{a_x+b_x} E_t^{a_x b_x} H_t(x-P_x)\exp\left[-\zeta(x-P_x)^2\right]$$

where $\zeta = \alpha+\beta$, $P_x = (\alpha A_x + \beta B_x)/\zeta$, and H_t is a *t*-th order Hermite polynomial. Thus all we have to know is ERIs between two Hermite Gaussian functions. Hermite functions of various orders satisfy a three-term recurrence relation and McMarchie and Davidson found that this recurrence relation also results in the recurrence relation among ERIs of various angular momentum. By applying these relations to intermediate ERIs in proper order the initial Boys function of various orders are transformed to the final ERIs. This is the first example of the modern ERI evaluation algorithms based on a recurrence relation. Note that some recurrence relation can be derived by the differentiation of ERI with respect to the center *P*.

The integral transformation of PRISM consists of following five steps.

T_m: Boys function is transformed to ERIs between two Hermite Gaussians. $[0]^{(m)} \rightarrow [p|q]$

T_{bra}: a bra Hermite Gaussian is transformed to the product of two GTOs, called the bra shell-pair. $[p|q] \rightarrow [ab|q]$

T_{ket}: a ket Hermite Gaussian is transformed to the product of two GTOs. $[ab|q] \rightarrow [ab|cd]$

C_{bra}: contraction of bra shell-pair. $[ab|cd] \rightarrow (ab|cd]$

C_{ket}: contraction of ket shell-pair. $(ab|cd] \rightarrow (ab|cd)$

We can apply these operations in various orders. The important finding by Gill and co-workers is that the best order which minimizes the computational cost depends on the contraction length and the angular momentum of GTOs. The order of these operations are called 'paths', and there are 40 paths. The attractive feature of PRISM is that the best path that minimizes the computational cost is selected for a given contraction length and the angular momentum.

As an example, C++-like pseudo-code to evaluate (*ps*|*ss*) ERIs by T_m, C_{bra}, T_{ket}, C_{ket}, T_{bra} path is shown below.

```
template<class T> FunctionPSSS () {
  T (b'p' 0|ss) = 0
  T (p' 1x|ss) = 0
  T (p' 1y|ss) = 0
  T (p' 1z|ss) = 0

  for( ket contranction ) {
    T (b'p' 0|0] = 0
    T (p' 1x|0] = 0
    T (p' 1y|0] = 0
    T (p' 1z|0] = 0

    for( bra contranction ) {
      Call function to make [0](0),[0](1)
```

```
    // Tm
    T [1x|0] = Rx * [0](1)
    T [1y|0] = Ry * [0](1)
    T [1z|0] = Rz * [0](1)

    // Cbra
    T [b'p' 0|0] = Be * Pe * [0](0)
    T [p' 1x|0] = Pe * [1x|0]
    T [p' 1y|0] = Pe * [1y|0]
    T [p' 1z|0] = Pe * [1z|0]
    (b'p' 0|0] += [b'p' 0|0]
    (p' 1x|0] += [p' 1x|0]
    (p' 1y|0] += [p' 1y|0]
    (p' 1z|0] += [p' 1z|0]
   }

   // Tket
   T (b'p' 0|ss] = (b'p' 0|0)
   T (p' 1x|ss] = (p' 1x|0)
   T (p' 1y|ss] = (p' 1y|0)
   T (p' 1z|ss] = (p' 1z|0]

   // Cket
   (b'p' 0|ss) += (b'p' 0|ss]
   (p' 1x|ss) += (p' 1x|ss]
   (p' 1y|ss) += (p' 1y|ss]
   (p' 1z|ss) += (p' 1z|ss]
  }

  // Tbra
  T (pxs|ss) = (p' 1x|ss) + BAx * (b'p' 0|ss)
  T (pys|ss) = (p' 1y|ss) + BAy * (b'p' 0|ss)
  T (pzs|ss) = (p' 1z|ss) + BAz * (b'p' 0|ss)
}
template FunctionPSSS<double>();
template FunctionPSSS<double4>();
template FunctionPSSS<float8>();
```

The variables Rx, Ry, Rz, Be, Pe, BAx, BAy, and BAz are determined by the primitive shell quartets. Generally the operations above only include additions and multiplications, such as

$$(Integral\ C) = a*(Integral\ A) + b*(Integral\ B).$$

Hence exactly the same operations are applied to the shell quartets within the same shell type (i.e., angular momentum) and the same contraction lengths. This process is very suited for SIMD parallelization.

The main difference between our implementation of PRISM and that in Gaussian program is the followings. In Gaussian all the computational operations to transform Boys function to the final ERIs are encoded in an integer array, called a 'driver'. The drivers needed are rebuilt at runtime based on the contraction length and the angular momentum. It also contains the memory pointers (or array indices) of all the variables (such as *A*, *B*, *C*, *a*, and *b*) in the recurrence relations. It directs the generic contraction and transformation routines the mathematical operations. The use of a driver makes the ERI evaluation program very simple: it is not necessary to generate and compile the source code for many shell types and paths.

The disadvantage of this approach is the degradation of performance because a compiler cannot know the sequence of the recurrence relations and hence it cannot optimise the program. Hence in this study source codes for all the paths are explicitly generated. The total number of functions necessary is very large, which is a shortcoming of our method. Suppose we use *s, p, sp*, and *d* type GTOs. Taking the permutation symmetry of ERI into account there are 55 patterns of angular momentum of four shells. As mentioned previously there are 40 distinct PRISM paths. Hence we have to make 2200 functions in total. In addition the source code of a function is sometimes lengthy: the path CCTTT for (*dd*|*dd*) has more than 17,000 lines. The file size of the library containing all functions is over 370 mega bytes. It does not look smart but it does not cause any serious problem thanks to a high-capacity HDD and a memory.

Another difference is the way of vectorization. Gaussian program makes batches of several dozen or hundreds of shell quartets to be applied recurrence relation at once. This technique was best suited for the ancient vector processors. It would help the automatic parallelization for recent CPUs. However we suspect that it causes the level one (L1) cash misses. In our program since eight shell quartets are handled at most cash misses would be negligible.

To deal with AVX easily, the class 'double4', which has a member variable of __m256d type, is defined as

```
class double4
{
public:
  double4& operator=(double d) {
    m_d = _mm256_set1_pd(d);
    return *this;
  }

  double4 operator+(const double4& dd) {
    return double4( _mm256_add_pd( m_d, dd.m_d ) );
  }

  // ... define other operators

  __m256d m_d;
};
```

The overload of the assignment and addition operators allows us to do without the intrinsic functions (_mm256_set1_pd, _mm256_add_pd) explicitly. The template function of C++ language shown in the pseudo-code is also useful to make the source code simple. Functions of the same instruction for double, double4 and float8 are generated once when we compile the code.

The criterion to select the appropriate path for a given type and the contraction length of shell quartet is an important issue. The widespread method is to select the path which minimizes the theoretical floating point operation (FLOP) count or the memory copy operation. However as practically better solution we measured the computational time of all paths and selected the path that minimizes it, then we recompile the code. Although these paths might be sub-optimum for a certain computer that has different architecture from ours, it is not likely to make a big difference. We can recompile the library if necessary when we install it. The actual performance difference between various CPUs is an open question.

2.2 J Matrix Engine on GPU

The calculation method of the Coulomb operator, called the J matrix, on GPU is based on Ref. [2] and we improved it in this study. The calculation of this matrix is the most time-consuming step in DFT when one uses pure functionals. ERIs are used only for J matrix in this case. In HF or DFT with hybrid functional the evaluation of the Hartree-Fock exchange potential, called K matrix is the most time-consuming step. Traditionally one first calculates all the ERIs and stores them on an external storage if necessary, and then they are transformed to the J and K matrices. However the previous study by Ufimtsev and Martinez [10] revealed that the separate calculation of both matrices is preferable to minimize the memory access. Hence we calculate J matrix separately, which enables us to use much faster J matrix engine [11] algorithm. The J matrix is the sum of the products of ERIs between Hermite Gaussian functions and the density matrix elements.

$$D_q = \sum_{cd} E_q^{cd} D_{cd}$$
$$J_p = \sum_q D_q (p|q)$$
$$J_{ab} = \sum_p E_p^{ab} J_p$$

The transformation of D_q and J_{ab} can be done before and after the summation over shell pair q, so that the cost of these steps are $O(N^2)$. Therefore the evaluation of J_p via $(p|q)$ dominates. As explained in the previous section this ERI can be evaluated by the three PRISM processes T_m, C_{bra}, and C_{ket}. Since there is no need to take T_{bra} and T_{ket} transformation, little advantages are observed to take contraction at former step even for the long contraction length. This is in sharp contrast to the K matrix evaluation. Considering the fact that GPUs have a limited shared memory and registers, we executed this summation in terms of primitive shells.

In Ref. [2] the primitive ERIs $[p|q]$ were calculated by Gauss-Rys quadrature to reduce the number of registers used. In this paper we adopted the McMarchie-Davidson recurrence relation to reduce FLOP count. The change of the preference reflects the change of the recent GPUs (Fermi core, Compute Capability 2.0 or later), in which more registers are available.

The important technique is to sort the shell pairs. It is widely accepted that we can skip the evaluation of very small integrals (less than 10^{-10} a.u. cut-off). So one first evaluates the upper bound of ERI by using the Schwartz inequality,

$$D_q[p|q] \le D_q \sqrt{[p|p][q|q]} \le cutoff$$

and calculates ERI which passed it. This reasonable screening causes the serious parallel efficiency problem on GPU. Since more than 32 threads run concurrently in SIMD fashion on GPU to evaluate their respective ERIs, all the threads should run the same program even when only one ERI passes the cut-off test. The rest threads calculate ERIs which are essentially zero. To avoid such waste we pack the significant shell quartets densely. We sorted shell pairs p as decreasing order of $\sqrt{[p|p]}$, and sorted q as decreasing order of $D_q\sqrt{[q|q]}$. It also enables for the threads in the same block to exit the loops over p and q as soon as possible all together. However, this method failed to utilize the integral symmetry $[p|q]=[q|p]$. The computational cost is thus twice of the ideal one.

In Ref. [2] ERIs whose upper bound were greater than the cut-off above and also were smaller than the second cut-off were evaluated on GPU with single precision. Other ERIs which were greater than the second cut-off were evaluated on CPU with double precision. Although the number of ERIs evaluated on CPU is only about 1/10 of that on GPU, we found that the computational time on CPU is sometimes longer than that on GPU, because the performance of GPU is much superior. Since the double precision operations are supported on recent NVIDIA's GPUs (Compute Capability 1.3 or later), we developed the double precision version of the GPU kernels and used it for larger ERIs.

2.3 ESP Calculation in FMO

FMO is an efficient method to calculate the electronic structure of large proteins with practical computational cost. This section describes the acceleration of FMO by GPU. A molecule to solve is split into many fragments in FMO, and the wave function of a fragment is determined by the standard *ab-initio* methods, such as HF. The wave function of a fragment is determined under the environmental electrostatic potential (ESP) of the neighbour fragments. This is a self-consistent problem, called the self-consistent charge (SCC), and the calculations are repeated until wave functions of all the fragments converge. At each SCC iteration we need ESP generated by all the neighbour fragments. The computational cost for ESP is higher than that of SCF calculation of each fragment: it typically occupies 70-90% of the total time. Therefore, the ESP is the first candidate to accelerate FMO. J matrix engine described above is suitable for it because ESP is just a Coulomb potential from the environment.

A particularity of ESP is that the integral symmetry of ERI $[p|q]=[q|p]$ cannot be used practically even when ERIs are evaluated on CPUs. This is because a bra shell-pair belongs to the fragment to be calculated and a ket belongs to one of the neighbour fragments. If one wants to use integral symmetry, the ERIs on a CPU should be transferred to other CPUs via rather slow networks. Because of it the disadvantage of the symmetry in GPU kernels does not matter.

2.4 Evaluation of Exchange Correlation Matrix in DFT

The calculation method for the exchange correlation matrix on GPU is mainly based on Ref. [3]. Assuming closed shell for simplicity the exchange-correlation energy functional $\varepsilon = \varepsilon(\rho(\vec{r}), \gamma(\vec{r}))$ is expressed as a function of the electron density $\rho(\vec{r})$ and the gradient of it, $\gamma(\vec{r}) = \nabla\rho(\vec{r}) \cdot \nabla\rho(\vec{r})$. The total exchange correlation energy is

$$E_{xc} = \int \varepsilon(\rho,\gamma) dr \cong \sum_i w_i \varepsilon(\rho(\vec{r}_i), \gamma(\vec{r}_i)),$$

where $\vec{r}_i$ and w_i are quadrature grid points and weights, respectively. The exchange correlation matrix are expressed in terms of ε and the first derivative of ε by ρ and γ as follows.

$$\begin{aligned}\langle \chi_k | v_{xc} | \chi_l \rangle &= \left\langle \chi_k \left| \frac{\delta E_{xc}}{\delta \rho} \right| \chi_l \right\rangle \\ &= \int \left(\chi_k \frac{\partial \varepsilon}{\partial \rho} \chi_l + \frac{\partial \varepsilon}{\partial \gamma} \nabla\rho \cdot \nabla(\chi_k \chi_l) \right) dr \\ &\cong \sum_i \chi_k(\vec{r}_i) \left[\frac{\partial \varepsilon(\vec{r}_i)}{\partial \rho} \chi_l(\vec{r}_i) + 2 \frac{\partial \varepsilon(\vec{r}_i)}{\partial \gamma} \nabla\rho(\vec{r}_i) \cdot \nabla\chi_l(\vec{r}_i) \right]\end{aligned}$$

The evaluation steps of exchange correlation matrix are (i) to determine the value of ρ and γ on the grid points, (ii) to determine the value of $\partial\varepsilon/\partial\rho$ and $\partial\varepsilon/\partial\gamma$ from the value of ρ and γ on the grid points, and (iii) to evaluate the elements of exchange correlation matrix from $\partial\varepsilon/\partial\rho$ and $\partial\varepsilon/\partial\gamma$. The calculation cost for (ii) is an order of the number of grid points (N_G), and that for (i) and (iii) is about $N_G \times N_B$, where N_B is the number of basis functions. So we accelerated steps (i) and (iii) by GPU. The step (ii) was executed on CPU and the data is transferred from the GPU's device memory to the host memory.

3 Results and Discussion

The algorithms explained in the previous section were implemented to make the libraries, XA-CUDA-QM and XA-AVX-QM. The accuracy and the performance of the product were measured by applying them to several molecules. The *ab-initio* program package of GAMESS 2010R3 was modified to join the libraries developed in this study. They are expected to replace the most time-consuming part of HF and DFT calculations, without changing the results. We compare the energy and some properties calculated with those of the original GAMESS.

The specifications of a computer used are: Intel Core i5 2500 running at 3.30GHz (4 cores), a GTX580 GPU, and 8 GB DDR3 memory. NVIDIA CUDA 3.0, Intel Composer XE (12.0) compiler and MKL 10.3 library are used.

3.1 Acceleration of Hartree-Fock calculation using the AVX library

The molecules calculated were valinomycin (molecular mass: 1111) and paclitaxel (molecular mass: 854) with 3-21G and 6-

31G basis set. Table 1 shows the timing information and the calculated HF energy.

Table 1. Computational times and the Hartree-Fock total energies of molecules, paclitaxel and valinomycin.

	Time [sec]	Total energy [a.u.]
paclitaxel, 3-21G		
GAMESS	184.048	-2895.7814570171
this work	78.271	-2895.7814570169
speedup/error	x 2.35	0.0000000002
paclitaxel, 6-31G		
GAMESS	324.386	-2910.6633340322
this work	153.632	-2910.6633340179
speedup/error	x 2.11	0.0000000157
valinomycin, 3-21G		
GAMESS	476.829	-3750.9205018138
this work	155.481	-3750.9205017267
speedup/error	x 3.07	0.0000000871
valinomycin, 6-31G		
GAMESS	752.839	-3770.0595984968
this work	323.098	-3770.0595984236
speedup/error	x 2.33	0.0000000732

As shown in Table 1 the HF calculations were accelerated by a factor of 2.1-3.1 times compared to the original ones. More than 95% of total computational time was used to evaluate ERIs and J and K matrices in the original GAMESS program. The energies calculated were the same in the accuracy of 10^{-7} a.u. We found that the acceleration ratio for 3-21G basis is higher than that of 6-31G basis. The difference of the ERI algorithm would partly explain it. The Pople-Hehre algorithm [12] used in GAMESS is suitable for highly contracted basis because the contraction is performed at an early stage of ERI transformation. Another plausible reason is the inefficiency of the current AVX library to evaluate the Boys function. The fraction of this cost increases as the contraction length increases, because they are calculated for all primitive shell quartets. The current library evaluates the Boys function unnecessarily more accurately than required. The optimization of this part is in underway.

3.2 Acceleration of DFT calculation using the GPU library

The molecules calculated were the same as the previous section: valinomycin and paclitaxel with 3-21G and 6-31G basis set. The so-called PW91 functional proposed by Pedrew and Wang [13] was used. Table 2 summarizes the timing information and the total energies of DFT calculation.

DFT calculation by GAMESS starts to solve HF. After the convergence of HF to some loose threshold it switches to the real DFT calculation. Hence the results of the DFT calculations include an effect of the AVX library. As the DFT calculation converges GAMESS uses finer numerical grid to evaluate the exchange-correlation potential. As shown in Table 2, the DFT calculations were accelerated by a factor of 3.6-4.0 compared to the original GAMESS. Less influence of basis sets was observed than in Table 1. The energy differences between the present and the original GAMESS are about $5\text{-}6\times10^{-3}$ a.u., which are much larger than in Table 1. We suspect the origin of this discrepancy as the implementation issue. Some parameters in the exchage-corelation functionals in GAMESS seem diferent from ours and those in Gaussian 09. In fact, the total energy of paclitaxel (3-21G) calculated by Gaussian 09 program was -2912.45580647 a.u. It matches only in the accuracy of 4×10^{-3} a.u. to the original GAMESS. On the other hand there is no reason for the discrepancy of the HF energy except for the numerical round-off error, Schwartz cut-off of ERIs, and the polynomial approximation of the Boys function. Taking these facts into account we can say that the right value of energy was calculated.

Table 2. Computational times and the DFT total energies of molecules, paclitaxel and valinomycin.

	time [sec]	Total energy [a.u.]
paclitaxel, 3-21G		
GAMESS	1014.439	-2912.4520720717
this work	279.097	-2912.4570178922
speedup/error	x 3.63	0.0049458205
paclitaxel, 6-31G		
GAMESS	1307.251	-2927.6946871371
this work	344.638	-2927.6996862771
speedup/error	x 3.79	0.0049991400
valinomycin, 3-21G		
GAMESS	2303.663	-3772.7680853481
this work	635.573	-3772.7744935064
speedup/error	x 3.62	0.0064081483
valinomycin, 6-31G		
GAMESS	3190.326	-3792.3719433246
this work	795.796	-3792.3784019260
speedup/error	x 4.01	0.0064586014

We also examine the performance in detail of the most time consuming steps. Table 3 shows the results of valinomycin with 3-21G basis.

Table 3. The details of computational time for the DFT calculation of valinomycin (3-21G basis). [unit:sec]

	GAMESS	this work	speedup
HF Fock Matrix	251.950	81.424	x 3.09
DFT J matrix	620.071	2.091	x 296.54
DFT exchange-correlation matrix	1059.445	173.544	x 6.10
Total	2303.663	635.573	x 3.62

Fock matrix (J and K matrices) formation in HF method was accelerated 3.1 times by using our PRISM method parallelized with AVX. Since this step is executed on the same CPU in either method speedup comes from purely the software improvement. J matrix formation in DFT was accelerated 297 times by using the GPU package. This is a surprisingly large number, because the theoretical peak performance of GPU we used is only about 10 times compared to CPU. Note that GAMESS does not use J matrix engine and ERIs are explicitly evaluated. This difference would partly explain the results, because generally J engine is twice faster than the usual ERI algorithms. However the most plausible reason is the inefficiency or the bad implementation of the original GAMESS. Our result implies that there is still much room for improvement in GAMESS.

The evaluation of exchange-correlation matrix was accelerated 6.1 times by using the GPU. We found that the generation of the quadrature grid and the weights was another bottleneck of the computation. It is a next candidate to accelerate by GPU.

3.3 Acceleration of FMO by using GPU and AVX libraries

The libraries developed (XA-CUDA-QM and XA-AVX-QM) were used to accelerate FMO. We choose as the test molecule the complex of the protease domain and the N-terminal module of the coagulation factor Xa and the inhibitor [14]. The molecule was split into 284 fragments. The basis set used was 6-31G basis

set. Table 4 shows the timing information and the calculated energy of FMO method.

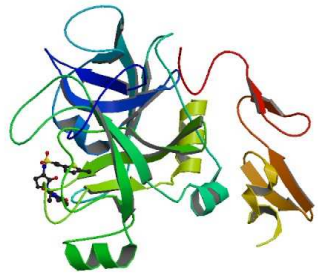

Fig. 1 Structure of the molecular complex calculated in FMO [15].

Table 4. Computational time and the FMO total energy of the molecular complex calculated

	Time [sec]	Total energy [a.u.]
GAMESS	134036.8	-118004.4295238
this work	40264.9	-118004.4295004
speedup/error	x 3.33	0.0000234

The error of the total energy was 2.4×10^{-5} a.u. The original calculation time of 134036.8 seconds by GAMESS was reduced to 40264.9 seconds. By using AVX and GPU we achieved more than three times acceleration. Table 5 compares the dipole moments of the first ten fragments.

Table 5. The x component of the dipole moment in atomic unit of each residue calculated with FMO2 approximation.

Residue	GAMESS	this work	error
1(ILE)	-58.81871	-58.81871	0.00000
2(VAL)	4.97865	4.97865	0.00000
3(GLY)	-2.94988	-2.94988	0.00000
4(GLY)	0.70733	0.70733	0.00000
5(GLN)	10.65749	10.65749	0.00000
6(GLU)	46.78936	46.78936	0.00000
7(CYS)	-4.34418	-4.34418	0.00000
8(LYS)	-27.77172	-27.77172	0.00000
9(ASP)	2.95062	2.95062	0.00000

As shown in it the dipole moment of each residue was essentially the same: the errors were within 10^{-5} a.u. In short we succeeded in the acceleration of whole FMO calculation without any degradation of results.

We examine the computational time of FMO in detail. Table 6 compares the time for the generation of ESP and SCF calculation of all the fragments. As shown the time-consuming step of the original program is the evaluation of ESP, which covers more than 70% of total time. We successfully accelerated it by using J matrix algorithm on GPU previously described: we achieved 13 times speedup. This acceleration ratio is much smaller than that of J matrix calculation for larger molecules described in previous section. One reason is the small size of fragments. The fraction of the overhead of GPU calculation, such as host-device data transfer or transformation of the data structures, increases as the size of molecule decreases. Another explanation is the bad implementation of the original GAMESS. We suspect the cache misses, which explains the rapid decay of the performance for large molecules.

The rest 30% of computational time is attributed to the regular SCF calculations of small but many fragments. The direct SCF method is unfavourable for them because most of the fragments are so small that we can store all ERIs in a host memory. As shown in Table 3 the SCF part is only slightly accelerated. Our current AVX library has the limitation that the fine-tuned AVX library is used only for the direct SCF algorithm. On the other hand in the FMO calculation most of the fragments were so small that all the ERIs are stored in the memory and the Fock matrix is calculated by the conventional SCF algorithm, which explains the marginal improvement of this part. Note that it is rather straightforward to use AVX library for conventional SCF. We expect that it will accelerate regular SCF.

Table 6. Computational time to generate the environmental electrostatic potential (ESP) and to solve SCF for every fragment in FMO calculation [unit:sec].

	ESP	SCF	Total
GAMESS	99554.2	14490.7	134036.8
this work	7721.2	12218.5	40264.9
speedup	x 12.89	x 1.186	x 3.33

4 Conclusion

In conclusion, the software libraries (XA-CUDA-QM, XA-AVX-QM) developed in this study replace the most time-consuming steps in the Hartree-Fock and DFT calculations. They successfully made the quantum chemical calculation by GAMESS 2.1-3.1 times faster for Hartree-Fock, 3.6-4.0 times for DFT, and 3.3 times for FMO. AVX were found to be useful to accelerate the ERI evaluation on CPUs for Hartree-Fock method. The total energy and the dipole moments in FMO calculation were found to be essentially the same. GPU made the calculation of environmental electrostatic potential 13 times faster than multi-core CPUs.

References

[1] K. Kitaura et al., *Chem. Phys. Letters.* 313,701(1999); 312,319(1999)

[2] K. Yasuda, *J. Comput. Chem.* **2008**, *29*, 334-342.

[3] K. Yasuda, *J. Chem. Theory. Comput.* **2008**, *4*, 1230-1236.

[4] http://x-ability.jp/ScienceCalc.html

[5] M.W.Schmidt, K.K.Baldridge, J.A.Boatz, S.T.Elbert, M.S.Gordon, J.H.Jensen, S.Koseki, N.Matsunaga, K.A.Nguyen, S.J.Su, T.L.Windus, M.Dupuis, J.A.Montgomery, *J. Comput. Chem.*, **14**, 1347-1363(1993)

[6] 241st American Chemical Society national meeting, 2011 March, Aneheim, CA, COMP division 177.

[7] M.W. Gill and J.A. Pople, *Int. J. Quant. Chem*, **40**, 753-772(1991)

[8] Gaussian 09, Revision A.1, M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, B. Mennucci, G. A. Petersson, H. Nakatsuji, M. Caricato, X. Li, H. P. Hratchian, A. F. Izmaylov, J. Bloino, G. Zheng, J. L. Sonnenberg, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. Bearpark, J. J. Heyd, E. Brothers, K. N. Kudin, V. N. Staroverov, R. Kobayashi, J. Normand, K. Raghavachari, A. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, N. Rega, J. M. Millam, M. Klene, J. E. Knox, J. B. Cross, V. Bakken, C. Adamo, J. Jaramillo, R. Gomperts, R. E. Stratmann, O. Yazyev, A. J. Austin, R. Cammi, C. Pomelli, J. W. Ochterski, R. L. Martin, K. Morokuma, V. G. Zakrzewski, G. A. Voth, P. Salvador, J. J. Dannenberg, S. Dapprich, A. D. Daniels, O. Farkas, J. B. Foresman, J. V. Ortiz, J. Cioslowski, and D. J. Fox, Gaussian, Inc., Wallingford CT, 2009.

[9] L.E. McMurchie and E.R. Davidson, *J. Comp. Phys.*, **26**, 218-231 (1978)

[10] I.S. Ufimtsev and T.J. Martinez, *J. Chem. Theory. Comput*, **4**, 222 (2008); ibid. **5**, 1004 (2009).

[11] Y. Shao and M. Head-Gordon, *Chem. Phys. Lett.*, **323**, 425-433 (2000)

[12] J.A. Pople and W.J. Hehre, *J. Comp. Phys.*, **27**, 161-168 (1978)
[13] J.P. Perdew, et al., *Phys. Rev. Lett*, **46**, 6671-6687 (1992)
[14] PDB ID: 2UWL Young, R. J., Brown, D., Burns-Kurtis, C. L., Chan, C., Convery, M. A., Hubbard, J. A., Kelly, H. A., Pateman, A. J., Patikis, A., Senger, S., Shah, G. P., Toomey, J. R., Watson, N. S., Zhou, P, Bioorg. Med. Chem. Lett. 17, 2927 (2007).
[15] Image from the RCSB PDB (www.pdb.org) of PDB ID 2UWL.