Quantum Chemistry on GPUs

Summary of ab initio calculation
GPU acceleration of
Density functional calculation (Gaussian),
Fragment molecular orbital calculation (GAMESS)
Density Functional Theory (DFT)

Kinetic energy

Non-local: pseudo or Hartree-Fock exchange potential

\[
[-\nabla^2/2 + V_L(r) + \hat{V}_{NL}]\psi_i(r) = \varepsilon_i\psi_i(r)
\]

Wave function

Electrostatic potential from nuclei / electrons

\[
V_L(r) = -\sum_c \frac{Z_c}{|r-R_c|} + \int \frac{\rho(r')}{|r-r'|} dr'
\]

Exchange-correlation Functional of \( \rho \)

\[
+ V_{xc}(r)
\]

Electron density

\[
\rho(r) = 2\sum_i |\psi_i(r)|^2
\]

Basis set expansion:

\[
\psi_i(r) = \sum_a C_a^{(i)} \chi_a(r)
\]

Eigenvector of Fock matrix gives expansion coef \( C \).
Choice of basis sets $\chi$

**Contracted Gaussian (GTO)**

\[ s: \chi(r) = \sum_{k=1}^{K} d_k e^{-\alpha_k (r-A)^2} \]

\[ p_x: \chi(r) = \sum_{k=1}^{K} d_k (x - A_x) e^{-\alpha_k (r-A)^2} \]

Small # of basis (10-30/atom), easy to diagonalize. Complicated program

Plane wave: communication bottleneck (FFT)

Wavelet

- Localized in real and reciprocal space

Real-space grid

- Discretize $\nabla^2$, simple program

Communication bottleneck

Large basis (200 point/Si atom, 2000/C atom)
Plane wave

# of basis function: more than $10^5$

Cost: iterative diagonalization (VASP, 97%)

\[
\hat{h} \psi = (-\nabla^2 / 2)\psi + (V_L + +\hat{V}_{NL})\psi
\]

1. Calculate $\hat{h} \psi$

\[
\psi(r) = \text{FFT}[\varphi(k)], \quad \hat{h} \varphi = \frac{1}{2} k^2 \varphi + \text{IFFT}[V_L \psi + \hat{V}_{NL} \psi]
\]

28% CPU time:

\[
0.18\%\quad 15\%\quad 5\%\quad 17\%
\]

2. Diagonalize $\hat{h}$ within $\varphi_a$:

\[
\langle \varphi_a | \hat{h} | \varphi_b \rangle C_b = \varepsilon C_a
\]

Error $g = C_a (\hat{h} \varphi_a) - \varepsilon C_a \varphi_a$

3. Orthogonalize new basis $\varphi = (h_D - \varepsilon)^{-1} g$, to $\varphi_a$. 

\[
\psi(r) = \frac{1}{\sqrt{\Omega}} \sum_k \varphi(k) \exp(ik \cdot r)
\]
FFT and BLAS parts dominate computational time. But because of data transfer just replacing them to GPU-libraries lowers the performance.

GPU task should be determined to minimize data transfer.

GPU calculates $\varphi(k)$ for a given potential $V$.

1. calculate $\hat{h}\psi$

$$\psi(r) = FFT[\varphi(k)], \quad \hat{h} \varphi = \frac{1}{2} k^2 \varphi + IFFT[V_L \psi + \hat{V}_{NL} \psi]$$

28% CPU time

2. Send matrix $\langle \varphi_a | \hat{h} | \varphi_b \rangle$ to CPU, diagonalize it, get $C_a$.

error $g = C_a (\hat{h} \varphi_a) - \varepsilon C_a \varphi_a$

3. Orthogonalize new basis $\varphi = (h_D - \varepsilon)^{-1} g$, to $\varphi_a$
Wavelet (BIGDFT)

Localized orthogonal in real and k space

$\chi_{ijk}(r) = \phi(x / \Delta - i)\phi(y / \Delta - j)\phi(z / \Delta - k)$

$\psi(r) = \sum_{ijk} \psi_{ijk} \chi_{ijk}(r)$

Variable resolution, # of basis: $10^5 \sim 6$

Operator $\rightarrow$ convolution (30 points)

$(\nabla^2 \psi)_{IJK} = \sum_{ijk} K_{I-i,J-j,K-k} \psi_{ijk}$

$K_{IJK} = T_I \delta_J \delta_K + \delta_I T_J \delta_K + \delta_I \delta_J T_K$

$\langle \psi | V(r) | \phi \rangle = \sum_{ijk} \tilde{\psi}_{ijk} V_{ijk} \tilde{\phi}_{ijk}$

$\tilde{\psi}_{IJK} = \sum_{ijk} M_{I-i} M_{J-j} M_{K-k} \psi_{ijk}$

Wavefunction at (I,J,K)

Potential at (I,J,K)

1D convolution $\times 3$

Simple parallel task

No global communication
Iteratively diagonalization because of huge # of basis

1. Electron density $\rho_{ijk}$ 12%CPU time $\rightarrow$ 13x by GPU
2. Potential $V_{ijk}$ by FFT 3%
3. $\hat{h}\psi$ 15% $\rightarrow$ 18x
   Transfer only nonzero elements.
4. Diagonalize $\hat{h}$ in $\psi_a$, calculate error $g$
5. Solve $(\frac{1}{2} \nabla^2 \varphi - \varepsilon)\psi = g$ by CG 20% $\rightarrow$ 10x
6. Orthoogonalize new basis $\psi$ with $\psi_a$ 40% $\rightarrow$ 6x

Cholesky decomposition by using BLAS
Orthogonalization $O(N^3)$: more than 80% in large system
Contracted Gaussian

Linear combination of Gaussians

\[ s: \chi(r) = \sum_{k=1}^{K} d_k e^{-\alpha_k (r-A)^2} \]

\[ p_x: \chi(r) = \sum_{k=1}^{K} d_k (x - A_x) e^{-\alpha_k (r-A)^2} \]

Some standard cGTOs

3-21G: inner shell (K=3, 3 terms), valence (K=2)+ (K=1)
6-31G**: 6-31G+ (angular momentum of valence) +1

○: small number of basis (10~30/atom)
  easy to diagonalize, low communication
  analytical 2-electron integrals

×: convergence to complete, complicated program
Computational cost of DFT

Solve $FC = SC \varepsilon$ with basis set expansion

\[ \psi_i(r) = \sum_b C_b^{(i)} \chi_b(r) \]

\[ \langle \chi_a | -\nabla^2 / 2 + V_{\text{nuc}} + V_{\text{es}} + V_{\text{xc}} | \chi_b \rangle C_b^{(i)} = \langle \chi_a | \chi_b \rangle C_b^{(i)} \varepsilon_i \]

Matrix diagonalization: 5% CPU time

ES potential: 15 ~ 40%, $10^7 \sim 10^8$ tasks for $10^3$ basis

\[ \langle \chi_a | V_{\text{es}} | \chi_b \rangle = \sum_{cd} (ab|cd) D_{cd}, \quad D_{cd} = 2\sum C_c^{(i)} C_d^{(i)} \]

\[ (ab|cd) = \int \frac{\chi_a(r)\chi_b(r)\chi_c(r')\chi_d(r')}{|r - r'|} dr'dr \]

XC potential: 40 ~ 80%, $10^6 \sim 10^7$ tasks

\[ \langle \chi_a | V_{\text{xc}} | \chi_b \rangle = \int \chi_a(r) V_{\text{xc}}(\rho(r)) \chi_b(r) dr \]

\[ \approx \sum w_i \chi_a(r_i) V_{\text{xc}}(\rho(r_i)) \chi_b(r_i) \]

\[ \rho(r_i) = \sum \chi_c(r_i) D_{cd} \chi_d(r_i) \]
How to calculate ES potential?

Discretize $\nabla^2 J(r) = \rho(r)$ in real space

$$(J_{i+1,j} + J_{i-1,j} + J_{i,j+1} + J_{i,j-1} - 4J_{i,j})/\hbar^2 = \rho_{i,j}$$

Conjugate gradient: sparse matrix, communication

Solve it in reciprocal space

$k^2 J(k) = \rho(k)$

Only for smooth density, FFT requires communication

Coulomb’s law $J(r) = \int \frac{\rho(r')}{|r-r'|} dr' \quad \rho(r') = \sum_{cd} D_{cd} \chi_c(r') \chi_d(r')$

① 2e integral $\sum_{cd} (ab|cd) D_{cd} \quad (ab|cd) = \int \chi_a(r) \chi_b(r') \chi_c(r') \chi_d(r') \frac{dr' dr}{|r-r'|}$

② Hermite Gaussian $\sum_q (p|q) D_q$

Simpler formula of integrals
Hermite Gaussians

Expand product of two Gaussians with Hermite ones.

\[(x - A_x)e^{-\alpha(x-A_x)^2}(x - B_x)e^{-\beta(x-B_x)^2} = \sum_{t=0}^{2} E_t^{11} H_t(x - P_x)e^{-\zeta(x-P_x)^2}\]

\[|ab) = \sum E_p^{ab} |p]\]

basis \(\chi_a\)
center \((A_x, A_y, A_z)\)
exponent \(\alpha\)

Hermite Gauss \(\Lambda_p\)
P = \((\alpha A + \beta B)/\zeta\)
Exponent \(\zeta = \alpha + \beta\)

Basis \(\chi_b\)
center \((B_x, B_y, B_z)\)
exponent \(\beta\)

Expand electron density with Hermite Gaussians

\[\rho(r) = \sum_{ab} D_{ab} \chi_a(r) \chi_b(r) = \sum_p D_p \Lambda_p(r)\]

\[(ab | cd) = \int \frac{\chi_a(r) \chi_b(r) \chi_c(r') \chi_d(r')}{|r - r'|} dr'dr\]

\[J_{ab} = \sum_{cd} (ab | cd) D_{cd} = \sum_p E_p^{ab} \sum_q [p | q] D_q\]

H_0(x) = 1
H_1(x) = 2x
H_2(x) = -2 + 4x^2
H_3(x) = -12 + 8x^3
Two-electron integral formula

\[ \int p(r_1)q(r_2)|_{r_1-r_2}dr_1dr_2 = (p|q) = (-1)^q[p+q]^{(0)} \]

\[ p(r) = H_t(x-P_x)e^{-\zeta(x-P_x)^2} \times (y \text{成分}) \times (z \text{成分}) \]

Simultaneously calculate \((p_x|p_x)\ldots(p_y|p_z)\)

\[ [0]^{(m)} = (\text{定数}) \int_0^1 u^{2m} e^{-Tu^2} \ du, \]

\[ [r]^{(m)} = R_i[r-1_i]^{(m+1)} - (r_i-1)[r-2_i]^{(m+1)} \]

\[ [(002)]^{(1)} = R_z[(001)]^{(2)} - (2-1)[(000)]^{(2)} \]

Many intermediate integrals causes register spill
Optimize to minimize it.

### # of intermediate ints

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>p</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]^{(m)}</td>
<td>1</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>[r]^{(1\sim m)}</td>
<td>1</td>
<td>35</td>
<td>330</td>
</tr>
<tr>
<td>[r]^{(0)}</td>
<td>1</td>
<td>35</td>
<td>165</td>
</tr>
<tr>
<td>(p</td>
<td>q)</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>FLOP</td>
<td>40</td>
<td>350</td>
<td>3200</td>
</tr>
</tbody>
</table>
ES potential: implementation

\[ J_p = \sum_{q} (p \mid q) D_q \]

\[ (p \mid q) = \int \frac{p(r_1)q(r_2)}{|r_1 - r_2|} \, dr_1 \, dr_2 \]

1 thread calculates \((p \mid q)\), multiply \(D_q\), accumulate in \(J_p\).

16\(p \times 4q\) SIMD parallel.

Different kernels for \(L_p, L_q=0-4\).

\((p \mid q)\) are recalculated (10～40FLOPs / integral).

Host gets only \(J\) (FLOP/byte ratio～200).

Integral symmetry \((p \mid q)=(q \mid p)\) is not used.

Sort \(P, Q\) shells, loop over only whose Schwartz upper bound is large enough. \(\sqrt{(p \mid p)} \times \sqrt{(q \mid q)} D_q\)
Accuracy requirement

Float number: \( r = \pm f \times 2^{e+127} \)

We examined numbers and magnitudes of terms in

\[
J_p = \sum_q (p \mid q)D_q
\]

We assume roundoff errors are random

\[\delta J \approx |\text{average of terms}| \times \sqrt{\# \text{ of terms}}\]

Relative error: \(1/\sqrt{\# \text{ of terms}}\)

Small terms mainly contributes to \(J\):

- single precision (GPU) is OK.
- Roundoff error of \(J\) comes from large integrals: double precision (CPU) required.

<table>
<thead>
<tr>
<th># of terms in (J_p)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(( \times 10^8, \text{converged}))</td>
</tr>
</tbody>
</table>

\(D\) of valinomycin

\[-\log |(p \mid q)D_q|\]
procedure for ES potential

① Reorder basis in a FMM box  
   \(|p\rangle\): decreasing order of \((p|p)^{1/2}\)  
   \(|q\rangle\): decreasing order of \((q|q)^{1/2} D_q\)

② Send \(p, q, D_q\) to GPU

③ Calculate \(J_p = \sum_q (p|q)D_q\) on GPU  
   1 thread calculates \((p|q)\).
   Integral symmetry are not used (1/2 efficiency).

④ Send \(J_p\) to the host, transform it to \(J_{ab}\).

Communication time < 10% of computation time  
Host-GPU bandwidth is enough to get \(J\), but not \((p|q)\).
Performance of kernels

Performance of kernels (in GFLOPS), peak performance ratio (%) (4096 P shells × 512 Q shells, random data)

<table>
<thead>
<tr>
<th>momentum (LP,LQ)</th>
<th>GPU (NVIDIA GTX580 1581 GFLOPS)</th>
<th>CPU (INTEL i7 3930K 6 cores 154/77 GFLOPS)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Non-opt</td>
<td>Opt</td>
</tr>
<tr>
<td>0,0</td>
<td>624 (40)</td>
<td>625 (40)</td>
</tr>
<tr>
<td>1,1</td>
<td>669 (42)</td>
<td>666 (42)</td>
</tr>
<tr>
<td>2,2</td>
<td>903 (57)</td>
<td>893 (57)</td>
</tr>
<tr>
<td>3,3</td>
<td>1050 (66)</td>
<td>1100 (70)</td>
</tr>
<tr>
<td>4,4</td>
<td>590 (20)</td>
<td>675 (43)</td>
</tr>
<tr>
<td>5,4</td>
<td>203 (13)</td>
<td>645 (41)</td>
</tr>
</tbody>
</table>

Optimized kernel works better for high angular momentum L. Performance of Gaussian09 dropped for large L.
Errors in Energy, ES potential

<table>
<thead>
<tr>
<th></th>
<th>taxol $C_{47}H_{51}NO_{14}$</th>
<th>valinomycin $C_{54}H_{90}N_{6}O_{18}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threshold for GPU</td>
<td>LDA 321G</td>
<td>PW91 631G</td>
</tr>
<tr>
<td>$E$ (au)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>All</td>
<td>-1.7[-3]</td>
<td>-3.2[-3]</td>
</tr>
<tr>
<td>$J$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>All</td>
<td>1.2[-5]</td>
<td>1.4[-5]</td>
</tr>
<tr>
<td>0.1</td>
<td>1.0[-6]</td>
<td>3.8[-6]</td>
</tr>
<tr>
<td>$10^{-3}$</td>
<td>9.6[-9]</td>
<td>2.0[-8]</td>
</tr>
</tbody>
</table>

We can control errors by changing the GPU threshold. Threshold=$10^{-3}$ is enough, 90% integrals are calculated in single precision.
Exchange-correlation term

\[ \langle \chi_a | V_{xc} | \chi_b \rangle \approx \sum_i w_i \chi_a(r_i) f(\rho(r_i)) \chi_b(r_i) \]

\( V_{xc} \): numerical quadrature (7000 points/atom)

① Electron density \( \rho(r_i) = \sum \chi_a(r_i) D_{ab} \chi_b(r_i) \) (15% time)

\( \sum \chi_a(r_i) D_{ab} \chi_b(r_i) \)

sparse vector × matrix × vector product

→ dense matrices (\( N \sim 100 \)), vector \([\chi_a(r_i)]\) are recalculated on GPU

send \( D \) matrix, receive \( \rho(r_i) \)

② XC potential \( f_i \) at \( r_i \)

③ \( V_{xc} \) matrix \( \sum_i f_i \chi_a(r_i) \chi_b(r_i) \) (20%)

matrix product (\( N \sim 100 \))

send \( f_i \), receive \( V_{ab} \)

FLOP / byte \( \sim 10^2 \)
Reducing data transfer time

\[ \rho(r_i) = \sum_{kl}^{n} D_{kl} \chi_k(r_i) \chi_l(r_i) \]

matrix size [nonzero \( \chi_k(r) \) at \( r_i \)] is small \([n=20 \sim 400]\): difficult to off-load calculation to GPU.

\( \times \) calculate \( \chi_k(r_i), \nabla \chi_k(r_i) \) with host and send them to GPU each element only used \( n \) times.

\( \times \) keep \( \chi_k(r_i), \nabla \chi_k(r_i) \) on GPU’s device mem.

1000 atoms \( =7 \times 10^6 \) points

\( \chi_k(r_i), \nabla \chi_k(r_i) \) amount to \( 2 \sim 45GB \)

\( \circ \) calculate \( \chi_k(r_i), \nabla \chi_k(r_i) \) with GPU, keep some on GPU’s shared mem, others recalculate.

matrix blocking with \( n_1 = 32 \)
Electron density: implementation

Pick 32 basis $\chi_a$ and 32 quadrature points $r_i$.

128 threads repeat ①～③ in SIMD way.

① calculate $\chi_0(r_i) \sim \chi_{31}(r_i)$ for $r_0 \sim r_{31}$

$$\chi_a(r_i) = \sum_n C_n e^{-\alpha_n (r_i-R_n)^2}$$

$C_n, \alpha_n, R_n$ are broadcasted from tex cache.

② read $32 \times 32$ matrix $D_{ab}$ from device mem, calculate matrix product

$$A_a(r_i) = \sum_b D_{ab} \chi_b(r_i)$$

③ calculate $\chi_{32}(r_i) \sim \chi_{63}(r_i)$ for $r_0 \sim r_{31}$,

inner product $\rho(r_i)^+ = \sum_a \chi_a(r_i) A_a(r_i)$
# Exchange-correlation potential

<table>
<thead>
<tr>
<th>$r_0, f_0, g_0$</th>
<th>tid 0</th>
<th>$r_{31}, f_{31}, g_{31}$</th>
<th>tid 31</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_{0-31}(r_0)$</td>
<td></td>
<td>$F_{0-31}(r_{31})$</td>
<td></td>
</tr>
<tr>
<td>$\chi_{0-31}(r_0) / \chi_{32-63}(r_0)$</td>
<td></td>
<td>$\chi_{0-31}(r_{31}) / \chi_{32-63}(r_0)$</td>
<td></td>
</tr>
</tbody>
</table>

Pick 32 basis $\chi_a$ and 32 quadrature points $r_i$.

128 threads repeat ①～④ in SIMD way

① calculate $\chi_0(r_i) \sim \chi_{31}(r_i)$, $F_0(r_i) \sim F_{31}(r_i)$ for $r_0 \sim r_{31}$

$$F_k(r_i) = \sum f_i \chi_k(r_i) + g_i \cdot \nabla \chi_k(r_i)$$

② transpose $F_k(r_i)^i$

③ calculate $\chi_{32}(r_i) \sim \chi_{63}(r_i)$ for $r_0 \sim r_{31}$

④ 32 × 32 matrix product $V_{kl} + = \sum_i F_k(r_i) \chi_l(r_i)$

All steps are executed on GPU to reduce communication

**Shared mem:** $F_k(r_i), \chi_k(r_i)$

**Tex cache:** basis info

**Device mem:** $V_{kl}$
## Errors in Energy and XC potential

<table>
<thead>
<tr>
<th>Cutoff*</th>
<th>Taxol</th>
<th>Valinomycin</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PW91 321G</td>
<td>PW91 631G</td>
</tr>
<tr>
<td>$E$ (au)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\nu_{xc}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

XC potential calculated in single precision induced errors of $10^{-5}$ au, which is within chemical accuracy.

* Basis functions of $\chi_k(r_i) < $ cutoff are discarded.
The first result (2007)

Core2 Quad 2.66GHz (1 core) + G80
Valinomycin (C$_{54}$H$_{90}$N$_{6}$O$_{18}$) PW91/6-31G 882 basis
Intel Fortran +MKL + CUDA (β release)

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES potential</td>
<td>30</td>
<td>24x</td>
</tr>
<tr>
<td>Electron density</td>
<td>18</td>
<td>13x</td>
</tr>
<tr>
<td>XC potential</td>
<td>30</td>
<td>11x</td>
</tr>
<tr>
<td>Grid weight</td>
<td>12</td>
<td>15x</td>
</tr>
<tr>
<td>Diagonalization</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>Initial guess</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>1-center integrals</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>Grid points</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>FMM</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

G80 (333GFLOPs) is 60 times faster than a Core2 Quad.
Hartree-Fock exchange

\[ \int \chi_a(r_1) \frac{D(r_1, r_2)}{|r_1 - r_2|} \chi_b(r_2) dr_1 dr_2 = \sum_{cd} (ac|bd)D_{cd} \]

Un-contract Gaussian (3~7x FLOP)

Use only (ac|bd)=(bd|ac) symmetry (4x FLOP)

8 × 8 threads calculate 1 \( K_{ab} \) element

Reorder |ac), |bd) to minimize memory access

\( K_{12} \)の計算

|2d] を(2d|2d)減少順に

(1c|1c)\(^{1/2}\) (2d|2d)\(^{1/2}\) \( \delta D_{bd} < 10^{-16} \) を64 threadsが満たしたら、次の8列に
Contraction problem

基底 $\chi_a : K$ 個の Gauss 関数の線型結合

線型結合をばらして $K^4$ 個の積分を求める

① 誤差関数 → $[0]^{(m)}$ → $[r]^{(0)}$ → $[p \mid q]$ → $[p \mid cd]$ → $[ab \mid cd]$

② $K$ 求和

$$ (ab \mid cd) = \sum_{i=1}^{K} \sum_{j=1}^{K} \sum_{k=1}^{K} \sum_{l=1}^{K} d_{ai}d_{bj}d_{ck}d_{dl} [a_i b_j \mid c_k d_l] $$

先に $K$ 和を計算すると 3 ～ 7 倍速い（Prism 法）

中間積分の種類が増える（GPU でレジスタ不足）。

$$ a' b' p' (r)^{(m)}_{c' d' q'} = \sum_{i=1}^{K} \sum_{j=1}^{K} \sum_{k=1}^{K} \sum_{l=1}^{K} d_{ai}d_{bj}d_{ck}d_{dl} \frac{\alpha^{a'} \beta^{b'} \gamma^{c'} \delta^{d'} \zeta^{p'} \eta^{q'}}{\xi^{p'} \eta^{q'}} [r]^{(m)} $$

① 誤差関数 → $[0]^{(m)}$ → $[r]^{(0)}$

② 4 重の $K$ 求和

③ $(r)^{(0)}$ → $(p \mid q)$ → $(p \mid cd)$ → $(ab \mid cd)$
Prism algorithm

[ab|と|cd]対に対して、[p|と|q]を決め

① 誤差関数→ [0]^(m) → [r]^(0) → [p|q] → [p|cd] → [ab|cd]の順に計算し、

② Primitiveを求和

(ab|cd) = \sum_{i=1}^{K} \sum_{j=1}^{K} \sum_{k=1}^{K} \sum_{l=1}^{K} d_{ai}d_{bj}d_{ck}d_{dl}[a_{i}b_{j} | c_{k}d_{l}]

先に②の和を計算できる。計算量が減るが、中間積分の種類が増える。

③ Primitiveの求和は好きな位置でできる。
Hatree-Fock results by Martinez

Time of the first SCF iteration (sec, Huckel guess, \(10^{-10}\) cutoff)

<table>
<thead>
<tr>
<th>CPU/GPU</th>
<th>GAMESS</th>
<th></th>
<th></th>
<th>G03</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PenD</td>
<td>Core2Quad</td>
<td>Corei7</td>
<td>Core2Quad</td>
<td>GTX280</td>
</tr>
<tr>
<td>GFLOPS</td>
<td>12</td>
<td>5.6</td>
<td>13.2</td>
<td>2*2.8=5.6</td>
<td>933</td>
</tr>
<tr>
<td>taxol</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-21G</td>
<td>282</td>
<td>157</td>
<td>87</td>
<td>110</td>
<td>3.0</td>
</tr>
<tr>
<td>6-31G</td>
<td>477</td>
<td>285</td>
<td>150</td>
<td>153</td>
<td>7.5</td>
</tr>
<tr>
<td>valino</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-21G</td>
<td>730</td>
<td>378</td>
<td>209</td>
<td>253</td>
<td>5.5</td>
</tr>
<tr>
<td>mycin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6-31G</td>
<td>1226</td>
<td>789</td>
<td>361</td>
<td>342</td>
<td>14</td>
</tr>
</tbody>
</table>

GAMESS is slower than Gaussian
GTX280 (160x faster) shows 20-40x acceleration, because integrals are recalculated 6 times, uncontracted Gaussian needs 2.5~7x FLOP cost.
Drugs generated by Computational Chemistry

©wikipedia
Zanamivir
trade name Relenza
Influenza Inhibitor

©wikipedia
Erlotinib hydrochloride
(trade name Tarceva)
Lung cancer, pancreas cancer

Steve Jobs might have used this to extend his life…
Procedure of FMO (Fragment MO)

1. Divide a molecule into N fragments.
2. Generate initial density matrices for all fragments (monomers).
3. Prepare **environmental electrostatic potentials (ESP)** from previous density matrices.
4. Solve Fock equations for all monomers $F^I_C = S^I_C e^I$, for $I=1$ to $N$.
5. Are all monomer densities converged?
   - **NO**
     1. Prepare environmental electrostatic potentials for all fragment pairs (dimers) from monomer densities.
     2. Solve Fock equations for all monomers $F^I_C = S^I_C e^I$, for $I=2..N$, $J=1..I-1$.
   - **YES**
     1. Calculate total energy and properties of the molecule.

Ab initio method for large insulators (protein etc.)

Bottleneck
Acceleration of Environmental electrostatic potential (ESP)

- Utilize GPGPU ERI J-matrix
cd: basis on neighbor fragments

- Decompose protein into each amino acid
  - Conventional SCF is effective in normal amino acid because ERIs can be kept on disk (# of basis < 180).

ERI calculation is not a bottleneck in FMO.
ESP is dominant.

\[
J_{ab} = \sum_{cd} (ab|cd)D_{cd}
\]

Fragment A
Fragment B
Test Model

Insulin (PDBID:2HIU)

Small Protein
44 amino acids
About 400 atom
System configuration

(1) Intel Core i7-3930K @3.2GHz (6 core),
    **GTX580 x 2**, 32GB, CUDA 4.0
(2) Intel Core i7-3930K @3.2GHz (6 core),
    **GTX580 x 4**, 32GB, CUDA 4.0
(3) Intel Core i7-3930K @3.2GHz (6 core),
    **GTX680 x 2**, 32GB, CUDA 4.0
(4) Intel Xeon E5-2650 @2.00GHz (32 core), a
    **TESLA K20m x 2**, 64GB, CUDA 5.0
+ Intel Composer XE 12.0 + MKL 10.3
## Total Energy Calculation of 2HIU

<table>
<thead>
<tr>
<th></th>
<th>Time [sec]</th>
<th>Total Energy [a.u.]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original GAMESS</td>
<td>3279.944</td>
<td>-21635.4488652520</td>
</tr>
<tr>
<td>Our GPGPU work</td>
<td>790.527</td>
<td>-21635.4488649211</td>
</tr>
</tbody>
</table>

(1) Intel Core i7-3930K @3.2GHz (6 core), GTX580 x 2, 32GB, CUDA 4.0
(2) Intel Core i7-3930K @3.2GHz (6 core), GTX580 x 4, 32GB, CUDA 4.0
(1) + (2)

Error of total FMO energy is very small.
No errors were observed in the energies of amino acids (monomer fragments).
Computational Time of FMO2-ESP and HF-SCF

<table>
<thead>
<tr>
<th></th>
<th>GAMESS</th>
<th>This work</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP part (GPU)</td>
<td>2571.490</td>
<td>170.897</td>
<td>x 15.0</td>
</tr>
<tr>
<td>HF-SCF part (host)</td>
<td>708.454</td>
<td>619.630</td>
<td>x 1.14</td>
</tr>
<tr>
<td>Total</td>
<td>3279.944</td>
<td>790.527</td>
<td>x 4.15</td>
</tr>
</tbody>
</table>

(1) Intel Core i7-3930K @3.2GHz (6 core), GTX580 x 2, 32GB, CUDA 4.0
(2) Intel Core i7-3930K @3.2GHz (6 core), GTX580 x 4, 32GB, CUDA 4.0
(1) + (2)

Acceleration ratio of ESP is higher than total one. Integrals needed in monomer HF-SCF were kept on host-side memory. It is faster than direct SCF by GPGPU.
# We accelerate HF-SCF with AVXed PRISM algorithm.
# GeForce Benchmark (FMO2 HF-SCF Single Point)

<table>
<thead>
<tr>
<th></th>
<th>Time [sec]</th>
<th>Total Energy [a.u.]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original GAMESS</td>
<td>7026.264</td>
<td>-21635.448652592</td>
</tr>
<tr>
<td>Our work <a href="1">GTX580 x 2</a></td>
<td>1670.931</td>
<td>-21635.4488649623</td>
</tr>
<tr>
<td>Our work <a href="3">GTX680 x 2</a></td>
<td>1708.522</td>
<td>-21635.4488649862</td>
</tr>
</tbody>
</table>

(1) Intel Core i7-3930K @3.2GHz (6 core), GTX580 x 2, 32GB, CUDA 4.0
(3) Intel Core i7-3930K @3.2GHz (6 core), GTX680 x 2, 32GB, CUDA 4.0
# 12 threads
### TESLA Benchmark (FMO2 HF-SCF Single Point)

<table>
<thead>
<tr>
<th></th>
<th>Time [sec]</th>
<th>Total Energy [a.u.]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original GAMESS</td>
<td>3467.644</td>
<td>-21635.448651915</td>
</tr>
<tr>
<td>Our work <a href="4">TESLA K20m x 2</a></td>
<td>1629.643</td>
<td>-21635.448651367</td>
</tr>
</tbody>
</table>

(4) Intel Xeon **E5-2650@2.0GHz** x 2 (16 core), **TESLA K20m x 2**, 64GB, CUDA 5.0
# Total 64 threads in 2 nodes
Comparison between acceleration rates of ESP and total time (Insuline)

<table>
<thead>
<tr>
<th>Basis set / method</th>
<th>(1) 3.2GHz 6 core + GTX680 (512 core) x 2</th>
<th>(2) 2.0GHz 32 core + TESLA K20 (2496 core) x 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ESP</td>
<td>Total</td>
</tr>
<tr>
<td>6-31G / FMO2, Single Point</td>
<td>13.8</td>
<td>4.1</td>
</tr>
<tr>
<td>6-31G* / FMO2, Single Point</td>
<td>16.6</td>
<td>4.4</td>
</tr>
<tr>
<td>6-31G / FMO3, Single Point</td>
<td>19.5</td>
<td>3.7</td>
</tr>
<tr>
<td>6-31G / FMO2, Grad</td>
<td>10.6</td>
<td>2.6</td>
</tr>
</tbody>
</table>

Performance ratio of (2)/(1) is 1.1 by perk performance.
Conclusion about GPGPU FMO

- Mixed precision method worked very fine.
- Conventional algorithm with CPU was suitable for HF-SCF part in FMO2/3. Faster CPUs are also desirable.
- Acceleration of ESP benefited both energy and grad calculations.
Thank you for your attention.