Quantum Chemistry on GPUs

名大　エコトピア　安田 耕二
クロスアビリティ　古賀良太

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

$$V_L(r) = -\sum_c \frac{Z_c}{|r - R_c|} + \int \frac{\rho(r')}{|r - r'|} dr' + V_{xc}(r)$$

Electrostatic potential from nuclei / electrons

Exchange-correlation Functional of ρ

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 χ

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

Contracted Gaussian (GTO)

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

exponent

Contraction coef

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 ∇^2, simple program

Communication bottleneck

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

Plane wave

$$\psi(r) = \frac{1}{\sqrt{\Omega}} \sum_k \varphi(k) \exp(ik \cdot r)$$

# of basis function: more than 10^5

Cost: iterative diagonalization (VASP, 97%)

$$\hat{h}\psi = \underbrace{(-\nabla^2/2)\psi}_{\text{k-space}} + \underbrace{(V_L + +\hat{V}_{NL})\psi}_{\text{Real-space}}$$

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

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

28%CPU time 0.18% 15% 5% 17%

2. Diagonalize $\hat{h}$ within φ_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. Orthgonalize new basis $\varphi = (h_D - \varepsilon)^{-1} g$, to φ_a.

Plane wave+GPU Maintz, CPC 182, 1421 (2011)

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 $\hat{h}\psi$

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

28%CPU time	0.18%	15%	5%	17%
14x by GPU		14x	14x	9x

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 φ_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:$10^{5\sim6}$

Operator→convolution (30 points)

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

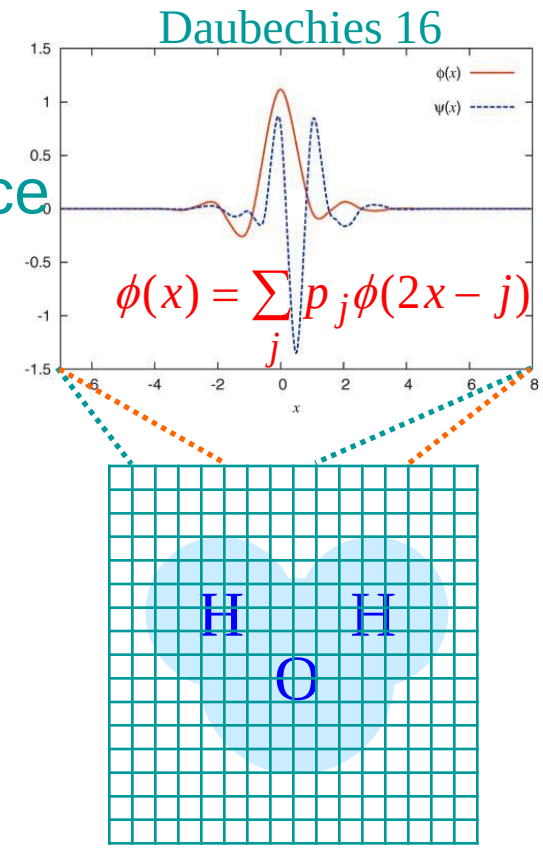
$$K_{IJK} = T_I\delta_J\delta_K + \delta_I T_J \delta_K + \delta_I\delta_J T_K$$

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

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

Wavefunction at (I,J,K)

Potential at (I,J,K)



1D convolution × 3

Simple parallel task

No global communication

Wavelet+GPU Genovese, JCP 131, 034103 (2009)

Iteratively diagonalization because of huge # of basis

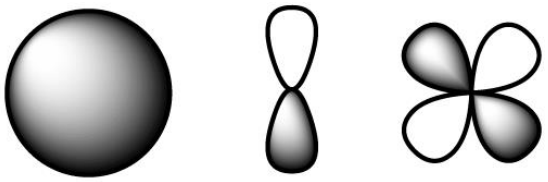
1. Electron density ρ_{ijk} 12%CPU time→ 13x by GPU
2. Potential V_{ijk} by FFT 3%
3. $\hat{h}\psi$ 15%→18x

   Transfer only nonzero elements.
4. Diagonalize $\hat{h}$ in ψ_a, calculate error g
5. Solve $(\frac{1}{2}\nabla^2\varphi - \varepsilon)\psi = g$ by CG 20%→10x
6. Orthoogonalize new basis ψ with ψ_a 40%→6x

   Cholesky decomposition by using BLAS

Orthogonalization $O(N^3)$: more than 80% in large system

Contracted Gaussian



s $p_{x,y,z}$ $d_{xy,xz,yz}$ $d_{x2,y2,z2}$

Linear combination of Gaussians

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

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

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

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

Matrix diagonalization : 5% CPU time

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

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

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

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

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

$$\approx\sum w_i\chi_a(r_i)V_{xc}(\rho(r_i))\chi_b(r_i)$$

$$\rho(r_i)=\sum\chi_c(r_i)D_{cd}\chi_d(r_i)$$

How to calculate ES potential?

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

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

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

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

$p(r)$ $q(r')$

②Hermite Gaussian $\sum_q (p|q) D_q$

Simpler formula of integrals

Hermite Gaussians

Expand product of two Gaussians with Hermite ones.

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

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

Herimte Gauss Λ_p
$P=(\alpha A+\beta B)/\zeta$
Exponent $\zeta=\alpha+\beta$

basis χ_a
center (A_x,A_y,A_z)
exponent α

Basis χ_b
center (B_x,B_y,B_z)
exponent β

$H_0(x)=1$
$H_1(x)=2x$
$H_2(x)=-2+4x^2$
$H_3(x)=-12+8x^3$

Expand electron density with Hermite Gaussians

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

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

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

Two-electron integral formula

$$\boxed{\int p(r_1)q(r_2)/|r_1-r_2|dr_1dr_2 = (\mathbf{p}\,|\,\mathbf{q})} = (-1)^q \boxed{[\mathbf{p}+\mathbf{q}]^{(0)}}$$

$$p(r) = H_t(x-P_x)e^{-\varsigma(x-P_x)^2} \times (y成分)\times(z成分)$$

Center **P**
exponent ζ
angular **p**

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

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

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

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

# of intermediate ints

	s	p	d
$[0]^{(m)}$	1	5	9
$[r]^{(1\sim m)}$	1	35	330
$[r]^{(0)}$	1	35	165
(p|q)	1	100	1225
FLOP	40	350	3200

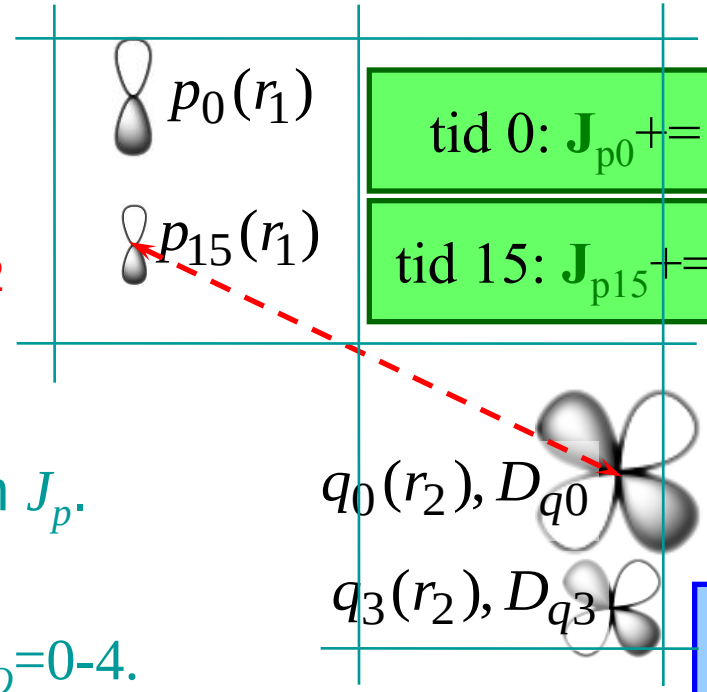
Many intermediate integrals causes register spill

Optimize to minimize it.

ES potential: implementation

$$J_p = \sum_q (p\,|\,q) D_q$$

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



tid 0: $\mathbf{J}_{p0} += (\mathbf{p}_0|\mathbf{q}_3)\mathrm{D}_{q3}$

tid 15: $\mathbf{J}_{p15} += (\mathbf{p}_{15}|\mathbf{q}_3)\mathrm{D}_{q3}$

Shared mem: p_k, q_l centers, exponents, D_q

1 thread calculates $(p|q)$, multiply D_q, accumulate in J_p.

$16p \times 4q$ SIMD parallel.

Different kernels for L_P, L_Q=0-4.

$(p|q)$ are recalculated (10～40FLOPs / integral).

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

Integral symmetry $(p|q)=(q|p)$ is not used.

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

Accuracy requirement

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

s	e_0	…	e_7	f_1	…	f_{23}

We examined numbers and magnitudes of terms in

$$J_p = \sum_q (p \mid q) D_q$$

We assume roundoff errrors are random

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

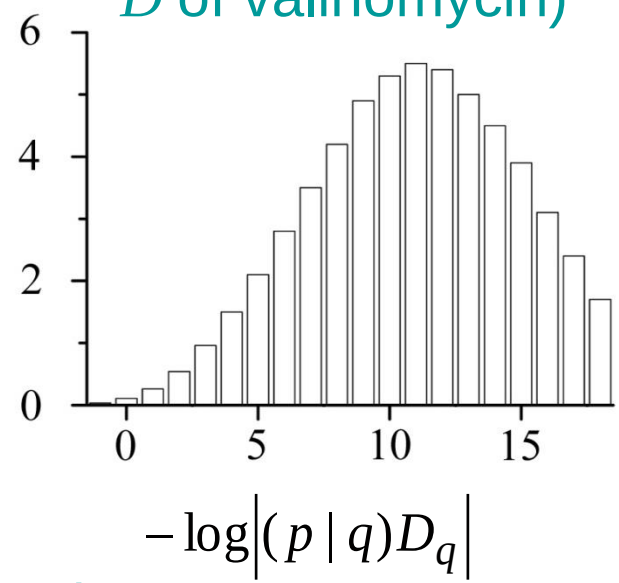
Relative error: $1/\sqrt{\#\text{ of terms}}$

Small terms mainly contributes to J:

single precision (GPU) is OK.

Roundoff error of J comes from large integrals: double precision (CPU) required.

# of terms in J_p ($\times 10^8$, converged D of valinomycin)



procedure for ES potential

① Reorder basis in a FMM box

$|p)$: decreasing order of $(p|p)^{1/2}$

$|q)$: 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）

	GPU (NVIDIA GTX580 1581 GFLOPS)		CPU (INTEL i7 3930K 6 cores 154/77 GFLOPS)	
momentum(LP,LQ)	Non-opt	Opt	SSE (4 float vector)	G09 double
0,0	624 (40)	625 (40)	69 (45)	29 (37)
1,1	669 (42)	666 (42)	54 (35)	20 (26)
2,2	903 (57)	893 (57)	72 (47)	17 (22)
3,3	1050 (66)	1100 (70)	94 (61)	11 (14)
4,4	590 (20)	675 (43)	49 (32)	9.5 (12)
5,4	203 (13)	645 (41)	37 (24)	9.2 (12)

Optimized kernel works better for high angular momentum L.

Preformance of Gaussian09 dropped for large L.

Errors in Energy, ES potential

		taxol $C_{47}H_{51}NO_{14}$		valinomycin $C_{54}H_{90}N_6O_{18}$	
	Threshold for GPU	LDA 321G	PW91 631G	LDA 321G	PW91 631G
E (au)	All	-1.7[-3]	-3.2[-3]	-1.1[-3]	-1.1[-3]
	0.1	-8.9[-5]	-8.6[-5]	-2.2[-5]	-1.4[-4]
	10^{-3}	-2.0[-8]	-4.7[-7]	-2.2[-7]	-6.7[-7]
J	All	1.2[-5]	1.4[-5]	1.5[-5]	1.5[-5]
	0.1	1.0[-6]	3.8[-6]	1.1[-6]	4.2[-6]
	10^{-3}	9.6[-9]	2.0[-8]	9.2[-9]	2.0[-8]

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 $\rho(r_i) = \sum_{ab} \chi_a(r_i) D_{ab} \chi_b(r_i)$ (15% time)

sparse vector × matrix × vector product

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

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

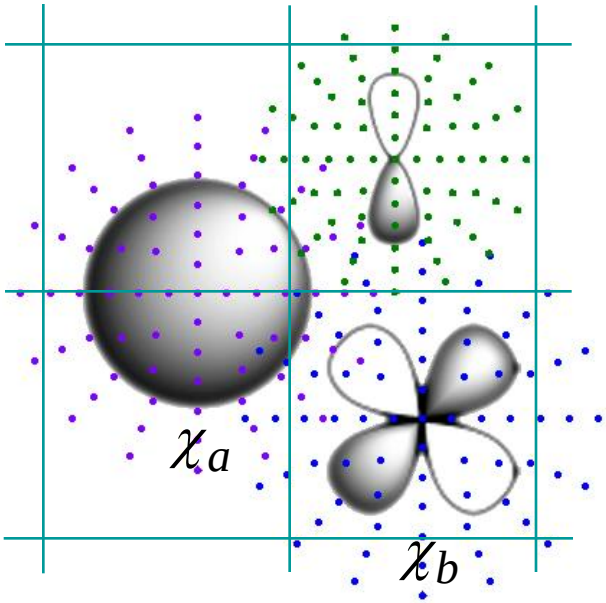
② XC potential f_i at r_i

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

matrix product (N~100)

send f_i , receive V_{ab}

FLOP / byte～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=20～400]: difficult to off-load calculation to GPU.

× calculate $\chi_k(r_i)$, $\nabla\chi_k(r_i)$ with host and send them to GPU

each element only used *n* times.

× keep $\chi_k(r_i)$, $\nabla\chi_k(r_i)$ on GPU's device mem.

1000 atoms =7 × 10^6 points

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

○ 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 χ_a and 32 quadrature points r_i.

128 threads repeat ①～③ in SIMD way.

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

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

C_n, α_n, R_n are broadcasted from tex cache.

② read 32×32 matrix D_{ab} from device mem, calculate matrix product

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

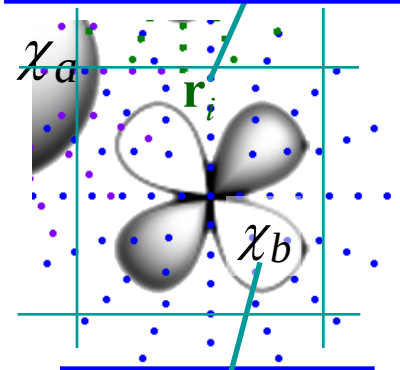
③ calculate $\chi_{32}(r_i)$～$\chi_{63}(r_i)$ for r_0～r_{31},

inner product $\rho(r_i) += \sum_a \chi_a(r_i) A_a(r_i)$

tid 0: $\mathbf{r}_0$
①$\chi_{0\text{-}31}(\mathbf{r}_0)$
②$\mathbf{A}_{0\text{-}31}(\mathbf{r}_0)$
③$\chi_{32\text{-}63}(\mathbf{r}_0)$, ρ_0

tid 31: $\mathbf{r}_{31}$
①$\chi_{0\text{-}31}(\mathbf{r}_{31})$
②$\mathbf{A}_{0\text{-}31}(\mathbf{r}_{31})$
③$\chi_{32\text{-}63}(\mathbf{r}_{31})$, ρ

Shared mem:
$\chi_k(r_i)$, $A_k(r_i)$, D_{kl}



Tex cache:
Basis info
C_n, α_n, R_n

Exchange-correlation potential

$\mathbf{r}_0, \mathbf{f}_0, \mathbf{g}_0$ tid 0 $\mathbf{F}_{0\text{-}31}(\mathbf{r}_0)$ $\chi_{0\text{-}31}(\mathbf{r}_0) / \chi_{32\text{-}63}(\mathbf{r}_0)$	$\mathbf{r}_{31}, \mathbf{f}_{31}, \mathbf{g}_{31}$ tid 31 $\mathbf{F}_{0\text{-}31}(\mathbf{r}_{31})$ $\chi_{0\text{-}31}(\mathbf{r}_{31}) / \chi_{32\text{-}63}(\mathbf{r}_0)$

Shared mem: $F_k(r_i)$, $\chi_k(r_i)$
Tex cache:basis info
Device mem: V_{kl}

Pick 32 basis χ_a and 32 quadrature points r_i.

128 threads repeat ①～④ in SIMD way

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

$$F_k(r_i) = \sum_i f_i \chi_k(r_i) + \mathbf{g}_i \cdot \nabla \chi_k(r_i)$$

② transpose $F_k(r_i)$

③ calculate $\chi_{32}(r_i)$～$\chi_{63}(r_i)$ for r_0～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

Errors in Energy and XC potential

		Taxol		Valinomycin	
	Cutoff*	PW91 321G	PW91 631G	PW91 321G	PW91 631G
E (au)	1.[-15]	6.5[-5]	3.0[-5]	8.0[-5]	5.4[-5]
	1.[-6]	6.5[-5]	3.1[-5]	7.9[-5]	5.7[-5]
	1.[-4]	6.3[-5]	3.1[-5]	7.8[-5]	5.6[-5]
v_{xc}	1.[-15]	4.8[-5]	4.8[-5]	6.3[-5]	5.9[-5]
	1.[-6]	4.8[-5]	4.8[-5]	6.3[-5]	5.9[-5]
	1.[-4]	4.8[-5]	5.0[-5]	6.4[-5]	6.1[-5]

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

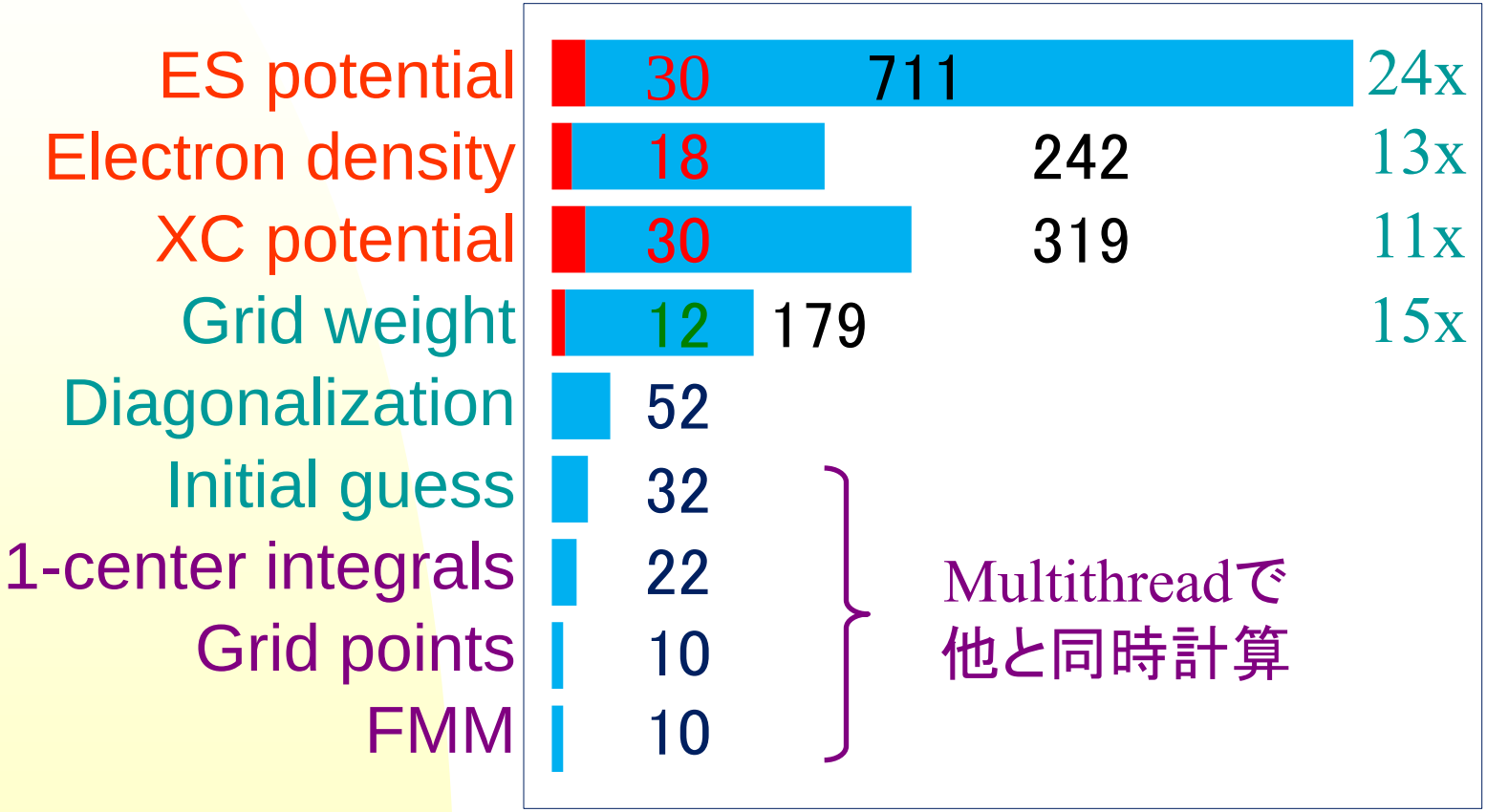
* Basis functions of $\chi_k(r_i)$ < cutoff are discarded.

The first result (2007)

Core2 Quad 2.66GHz (**1core**) + G80

Valinomycin ($C_{54}H_{90}N_6O_{18}$) PW91/6-31G 882 basis

Intel Fortran +MKL + CUDA (β release)



G80 (333GFLOPs) is 60 times faster than a Core2 Quad.

Hartree-Fock exchange

$$\int \chi_a(r_1)\frac{D(r_1,r_2)}{|r_1-r_2|}\chi_b(r_2)dr_1dr_2 = \sum_{cd}(ac|bd)D_{cd}$$

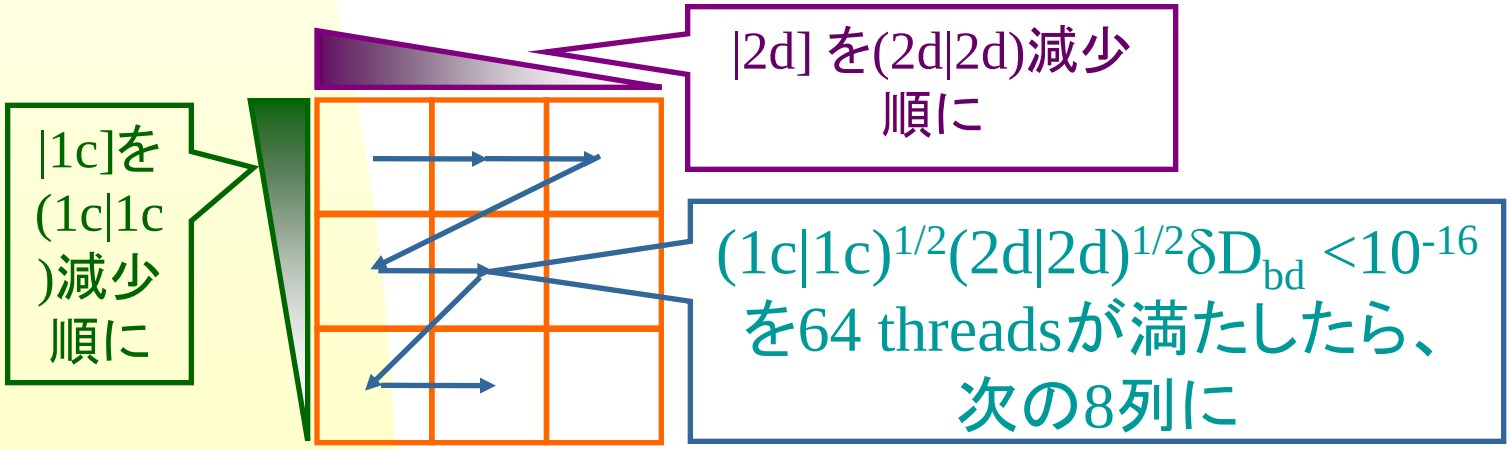
Un-contract Gaussian （3～7x FLOP）

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



Contraction problem

基底χ_a:K個のGauss関数の線型結合 $\chi_a(r) = \sum_{k=1}^{K} \boxed{d_{ak}} e^{-\alpha_k (r-A)^2}$

線型結合をばらしてK^4個の積分を求め

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

② K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）。

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

① 誤差関数→ $[0]^{(m)}$ → $[r]^{(0)}$

② 4重のK求和

③ $(r)^{(0)}$ → (p|q) → (p|cd) → (ab|cd)

Prism algorithm

[ab|と|cd]対に対して、[p|と|q]を決め

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

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

① 誤差関数→ $[0]^{(m)} \rightarrow [r]^{(0)}$

② Primitiveを求和

③ $(r)^{(0)} \rightarrow (p|q) \rightarrow (p|cd) \rightarrow (ab|cd)$

Primitiveの求和は好きな位置でできる。

Hatree-Fock results by Martinez

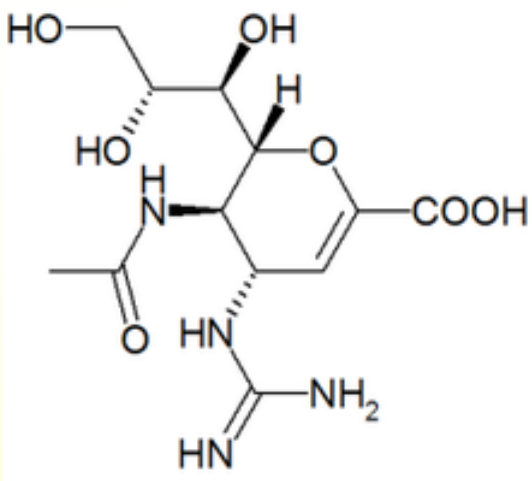
Time of the first SCF iteration (sec, Huckel guess, 10^{-10} cutoff)

		GAMESS			G03	GPU	
CPU/GPU		PenD	Core2Quad 1 core	Corei7 1 core	Core2Quad 1 core	GTX280	
GFLOPS		12	5.6	13.2	2*2.8=5.6	933	167x
taxol	3-21G	**282**	**157**	**87**	**110**	**3.0**	37x
	6-31G	**477**	**285**	**150**	**153**	**7.5**	20x
valino mycin	3-21G	**730**	**378**	**209**	**253**	**5.5**	46x
	6-31G	**1226**	**789**	**361**	**342**	**14**	24x

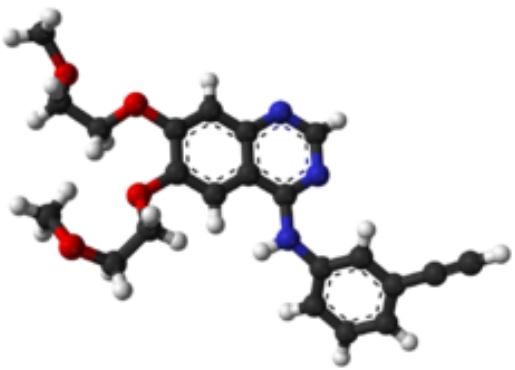
GAMESS is slower than Gaussian

GTX280 (160x faster) shows 20-40x acceleration, because integrals are recalculated 6 times, uncontracted Gaussian needs 2.5～7x FLOP cost.

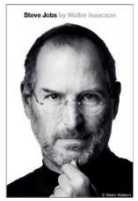
Drugs generated by Computational Chemistry



©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)

Ab initio method for large insulators (protein etc.)

Divide a molecule into N fragments.

↓

Generate initial density matrices for all fragments (monomers).

↓

Prepare **environmental electrostatic potentials (ESP)** from previous density matrices.

Bottleneck

↓

NO

Solve Fock equations for all monomers $F^I C^I = S^I C^I e^I$, for I=1 to N.

↓

Are all monomer densities converged?

↓ YES

Prepare environmental electrostatic potentials for all fragment pairs (dimers) from monomer densities.

↓

Solve Fock equations for all monomers $F^I C^I = S^I C^I e^I$, for I=2..N, J=1..I-1.

↓

Calculate total energy and properties of the molecule.

Acceleration of Environmental electrostatic potential (ESP)

Fragment A Fragment B

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

- 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).

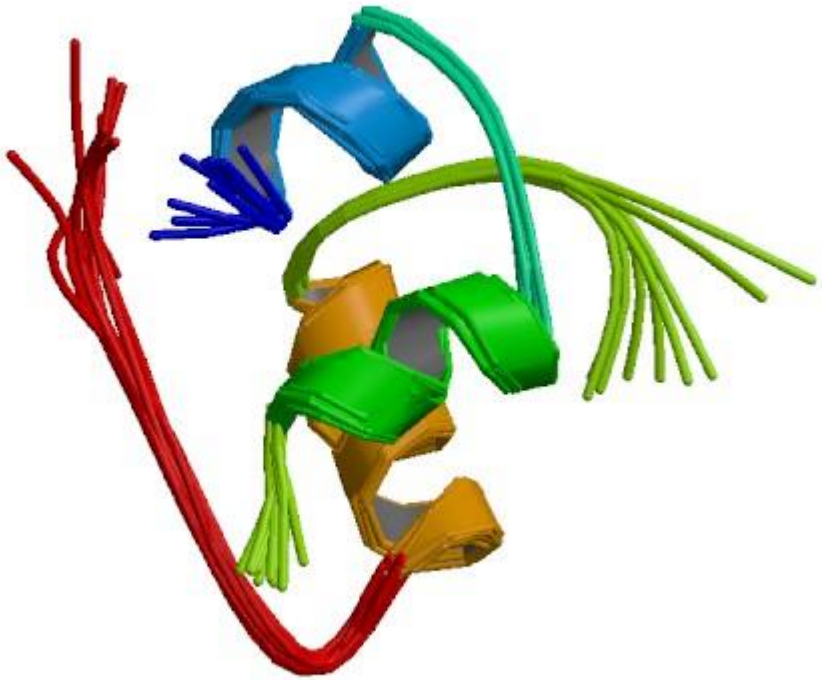
ERI calculation is not a bottleneck in FMO.

ESP is dominant.

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

	Time [sec]	Total Energy [a.u.]
Original GAMESS	3279.944	-21635.4488652520
Our GPGPU work	790.527	-21635.4488649211

(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

	GAMESS	This work	speedup
ESP part(GPU)	2571.490	170.897	x 15.0
HF-SCF part(host)	708.454	619.630	x 1.14
Total	3279.944	790.527	x 4.15

(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)

	Time [sec]	Total Energy [a.u.]
Original GAMESS	7026.264	-21635.4488652592
Our work [GTX580 x 2](1)	1670.931	-21635.4488649623
Our work [GTX680 x 2](3)	1708.522	-21635.4488649862

(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)

	Time [sec]	Total Energy [a.u.]
Original GAMESS	3467.644	-21635.4488651915
Our work [TESLA K20m x 2](4)	1629.643	-21635.4488651367

(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)

Basis set / method	(1) 3.2GHz 6 core + GTX680 (512 core) x 2		(2) 2.0GHz 32 core + TESLA K20 (2496 core) x 2	
	ESP	Total	ESP	Total
6-31G / FMO2, Single Point	13.8	4.1	3.3	2.1
6-31G* / FMO2, Single Point	16.6	4.4	4.7	2.5
6-31G / FMO3, Single Point	19.5	3.7	4.8	2.3
6-31G / FMO2, Grad	10.6	2.6	2.9	1.4

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.