X-Ability
クロスアビリティ

GPGPUによる量子化学計算の高速化
Acceleration of Quantum Chemistry by GPU

2010.10.19

株式会社クロスアビリティ
X-Ability Co.,Ltd.
古賀　良太
共同研究者：古川祐貴（同僚）、安田耕二准教授（名大）

X-Ability
クロスアビリティ

Summary of the Company

- 社名
  株式会社クロスアビリティ (X-Ability Co.,Ltd) ※役員３名のみ
- 業務内容
  計算化学関連ソフトウェアの開発、販売
  フィールドルータの開発、販売
- ビジネスモデル
  大学のユニークな研究の商用化
- 設立
  2008年1月
- 主な製品
  XA-CHEM-SUITE (XA-CUDA-QM etc.), FieldRouter

X-Ability
クロスアビリティ

Contents

1. What is Quantum Chemistry (ab initio) ?
   Approximation / Basis set / Parallelization
2. 密度汎関数法(DFT), Hartree-Fock法(HF)
3. Acceleration of DFT, HF, FMO by GPU
   静電ポテンシャルJ (Coulomb Potential)
   Hartree-Fock交換ポテンシャルK (HF exchange)
   交換相関ポテンシャル (Exchange Correlation)
4. Acceleration of Gaussian03/GAMESS-US

X-Ability
クロスアビリティ

Quantum Chemistry (ab initio)

波動方程式→電子波動関数→物質の性質を理解、予言

構成粒子（原子核と電子）の性質や力は既知

$$H(1\cdots N)\Psi(1\cdots N)=E\Psi(1\cdots N)$$

$$H=T+V_{nuc}+W,\quad T=-\frac{1}{2}\sum_i\nabla_i^2,$$ 運動エネルギー

$$V_{nuc}=-\sum_{i,c}Z_c/|r_i-R_c|$$ 原子核が電子を引っ張る

$$W=\sum_{i>j}1/|r_i-r_j|$$ 電子間の反発

3*N*変数の偏微分方程式（多体問題）

厳密解は計算できない→誤差0.001%以下で近似

X-Ability
クロスアビリティ

How to solve Quantum Chemistry

分子全体の波動関数が必要か?

大事でない所は分子力場(QM/MM):つなぎ目?

分子断片から推定(フラグメント分子軌道法):切れる?

多体問題の近似 $[-\nabla^2/2 + v_{eff}(r)]\psi_i(r) = \varepsilon_i \psi_i(r)$

平均場近似: 3次元の偏微分方程式

密度汎関数法:分散力、励起状態? ・・

摂動/クラスター展開:計算量?

基底関数展開(行列固有値問題) $\psi_i(r) = \sum_k C_k^{(i)} \chi_k(r)$

平面波、実空間grid ...

短縮Gauss関数 (GTO) :通信量小

2電子積分、高速多重極展開、行列対角化、O(N)法

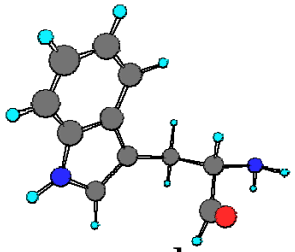
並列化、アクセラレーター

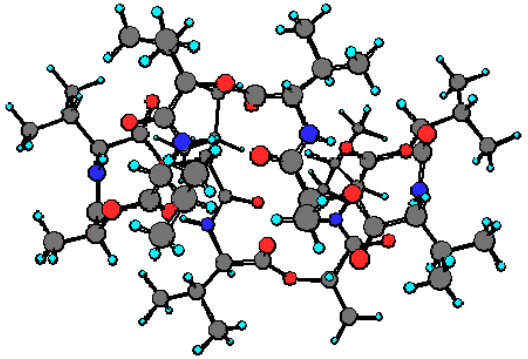
X-Ability
クロスアビリティ

Parallel Computing + Accelerator

疎粒度並列化：方程式自体を分割
　フラグメント分子軌道法、分割統治法…

細粒度並列化：アクセラレータ
　50 ～数千基底の生体分子
　密度汎関数法、平均場近似、
　　2次摂動法、クラスター展開
　速いプログラム、O(N)法を使う
　使い勝手、コスト重視
　　貧乏人でも使える


tryptophane


valinomycin

X-Ability
クロスアビリティ

What kind of basis set χ is appropriate?

電子の波動関数ψ(軌道)を線型展開 $\psi_i(r) = \sum_k C_k^{(i)} \chi_k(r)$

1. **短縮Gauss型関数 (GTO): GAMESS-US, Gaussian**

   10～30関数/原子　※推奨

2. 平面波 (PW, APW): ABINIT

   金属、半導体、周期系、内殻は擬ポテンシャル

   FFTに時間がかかる、通信ネック?

3. 実空間grid: RSDFT

   △を差分　流体同様、通信ネック?

   Si:200点/原子、C:2000点/原子

4. Wavelet: BigDFT

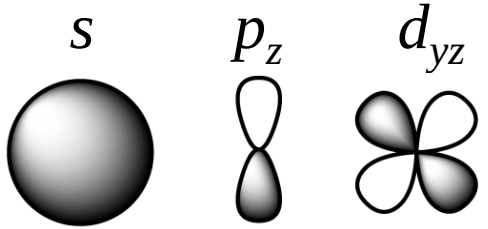
   基底関数極限が分かる(必要?)

X-Ability
クロスアビリティ

Contracted Gaussian basis set χ

原子核中心のGauss関数の線型結合

形 p型: x,y,z, d型: x^2,y^2,z^2,xy,xz,yz



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

指数

短縮係数

種類いっぱい。元素毎に表が整備されている

3-21G: 内殻3G×1個、価電子2G×1個+ 1G×1個

6-31G**: 角運動量+1の1Gも加える

長所: 基底数が少なくて済む、対角化容易、通信小

解析的に2電子積分

短所: プログラム複雑

X-Ability
クロスアビリティ

Procedure of Hartree-Fock method

$$F_{\mu\nu} = H_{\mu\nu}^{core} + \sum_{a}^{2/N}\sum_{\lambda\sigma} C_{\lambda a} C_{\sigma a}^{*}\left[2(\mu\nu|\sigma\lambda) - (\mu\lambda|\sigma\nu)\right]$$

$$= H_{\mu\nu}^{core} + \sum_{\lambda\sigma} P_{\lambda\sigma}\left[(\mu\nu|\sigma\lambda) - \frac{1}{2}(\mu\lambda|\sigma\nu)\right] \quad \text{SCF}$$

$$= H_{\mu\nu}^{core} + G_{\mu\nu}$$

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

Density Matrix : Initial Guessで初期値を作った後、非線形方程式を解いている間（SCFサイクル）アップデートされ続ける。 $F(C)C = SC\varepsilon$

Electron Replusion Integral J-matrix : Coulomb Potential J。理屈では一度計算してメインメモリにおけばいいのだが、$O(N^4)$ のため少し基底Nが大きくなると置けなくなる。ディスクI/Oは時間がかかり毎回演算するのがリーズナブルとなるが大変。

Electron Replusion Integral K-matrix : $O(N^4)$、HF exchange K。J-matrixと同様の問題を抱えるが、使用レジスタの量が多く、メモリ少のSIMD型への実装が難しい。

X-Ability
クロスアビリティ

Problem of Contracted Gaussian basis set

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

$[ab|$と$|cd]$対から、$[p|$と$|q]$を決め

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3～5倍速いが、中間積分増（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)

X-Ability
クロスアビリティ

How to solve Coulomb Potential J

1. 2電子積分 $(ab|cd)D_{cd}$ $(ab|cd)=\int\frac{\chi_a(r)\chi_b(r)\chi_c(r')\chi_d(r')}{|r-r'|}dr'dr$

   Hartree-Fock交換項Kと同時計算可能（CPUでは）
2. Density fit $(ab|c)D_c$

   $O(N)$でない？　余り速くない？
3. Hermite Gauss基底 $(p|q)D_q$
4. 高速多重極子展開(CFMM)も組み合わせる

   $O(N)$, 核電荷も同時処理可能
5. 実空間grid+FFT

   内殻電子密度は別に扱う

   広がった基底には有効
6. 有限要素法

X-Ability
クロスアビリティ

Procedure using Hermite Gaussian basis set

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

t次Hermite多項式

同じ中心のGauss関数はHermite Gaussで展開可能

①電子密度をHermite Gaussで展開 $\rho(r) = \sum_q D_q |q]$

②HG間の2電子積分と静電ポテンシャルを計算 $\sum_q [p|q]D_q$
　pについて並列化

③それをGauss関数に逆変換

普通のGauss関数より2電子積分が簡単

方向だけ違う積分$[p_x|p_x]$…$[p_y|p_z]$ …を同時計算

通信が遅く、2電子積分$[p|q]$は回収できない

X-Ability
クロスアビリティ

Calculation of J-matrix

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

(中心**P**, 角運動量**p**, 指数ζ)のHG $\Lambda_{\mathbf{p}}$と、
(**Q**, **q**, η)のHG $\Lambda_{\mathbf{q}}$の間の2電子積分

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

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

$$[0]^{(m)} = (\text{定数})\int_0^1 u^{2m}e^{-Tu^2}du, \quad T = \frac{\zeta\eta}{\zeta+\eta}R^2$$

HG対の距離Rと指数（広がり）ζ, ηから計算

中間積分の数

	s	p	d
[p|q]	1	100	1225
$[r]^{(0)}$	1	35	165
$[r]^{(1\text{-}m)}$	1	35	330
$[0]^{(m)}$	1	5	9
FLOP	40	800	15000

誤差関数→$[0]^{(m)}$→$[\mathbf{r}]^{(0)}$→**[p|q]**の順に計算

中間積分は多数。Kernelは多種類。

天体、MDよりずっと複雑

途中過程でレジスタを大量に消費する。Occupancy悪いが仕方がない。

X-Ability
クロスアビリティ

Integral Screening for J-matrix

① 基底関数の積$|ab)$で、χ_aとχ_bが遠ければ捨てる

$|ab)$, $|p]$の数～O(N)

40～70%が残る

② CFMMで扱わない、近くの $(ab|$と$|cd)$対だけ計算

$(ab|cd)$, $[p|q]$の数～O(N)

20～40%が残る

③ 三角不等式 $(ab|cd)^2 \le (ab|ab)(cd|cd)$ を使い、10^{-10}以上の積分だけ計算

50～80%が残る

1～3で積分数は1/7～1/13になる

不規則、小サイズの密データ（50程度）

X-Ability
クロスアビリティ

Procedure of J-matrix

① Shell対 $|ab)$, $|p]$, D_qを作る

primitive HGにばらす

② 近接したFMM箱内の$|p]$と$|q]$を並び替える

$|p]$: $[p|p]^{1/2}$の減少順

$|q]$: $[q|q]^{1/2}$ D_qの減少順

③ GPUにp, q, D_qを送る

④ GPUで　$J_p = \sum_q [p|q] D_q$　を並列計算

各threadが1組の$[p|q]$を計算

積分対称性は使わない（効率1/2)

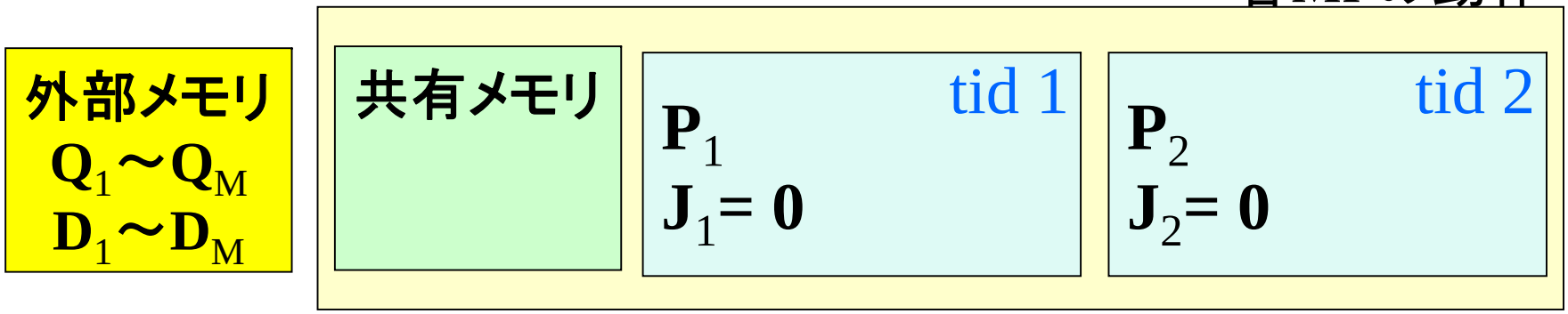
⑤ 積算後の J_pをhostに返し、J_{ab}に変換

通信量：計算量の10%以下

X-Ability
クロスアビリティ

Coulomb Potential J : Parallelization

各MPの動作



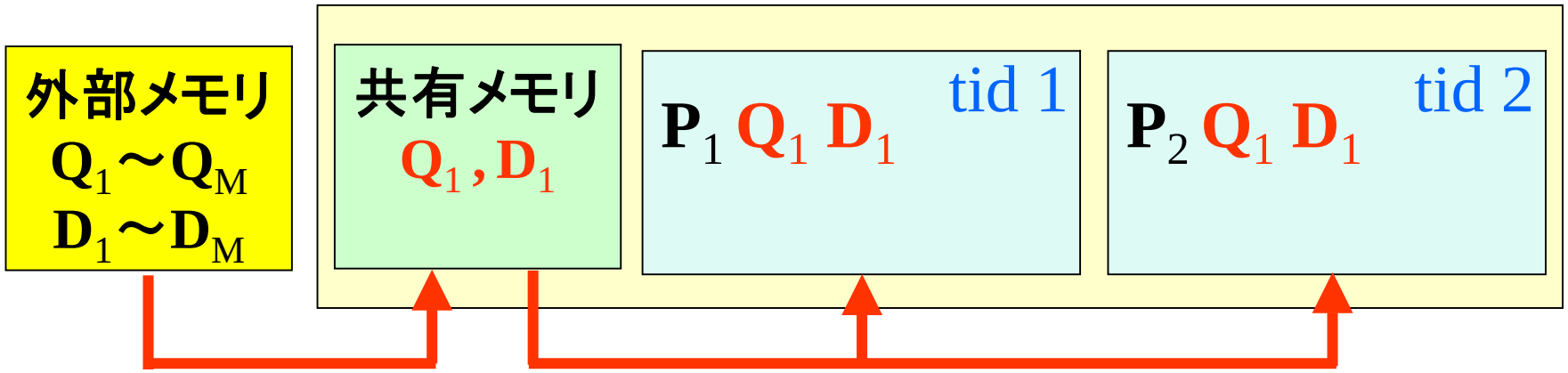
$$J_i = \sum_{j=1}^{N} [P_i|Q_j]D_j$$

近接したshell pairをGPUに送る

64スレッドが16 P shell × 4 Q shellの積分を計算

X-Ability
クロスアビリティ

Coulomb Potential J : Parallelization

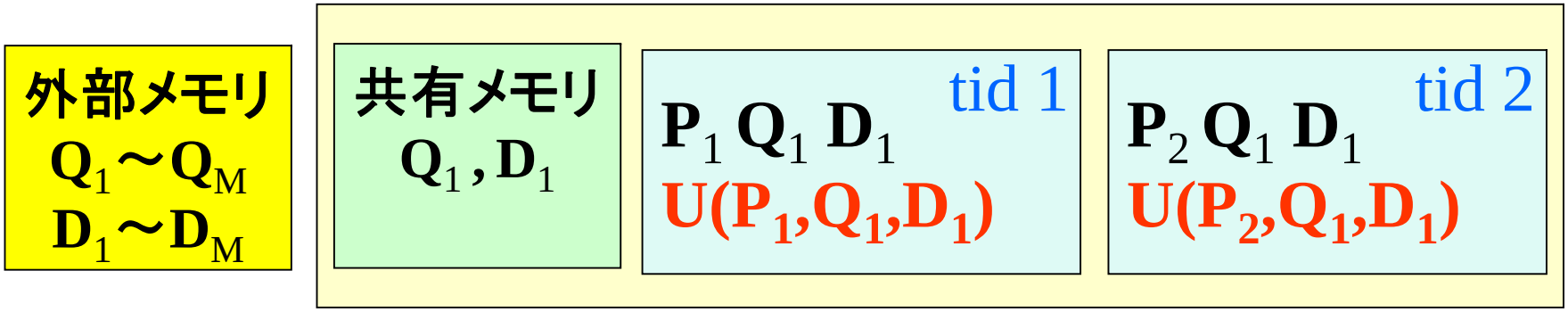


最初のQ shellを読む

外部メモリ読み込み中は, 他のスレッドが計算する

X-Ability
クロスアビリティ

Coulomb Potential J : Parallelization



CFMMで計算済か?

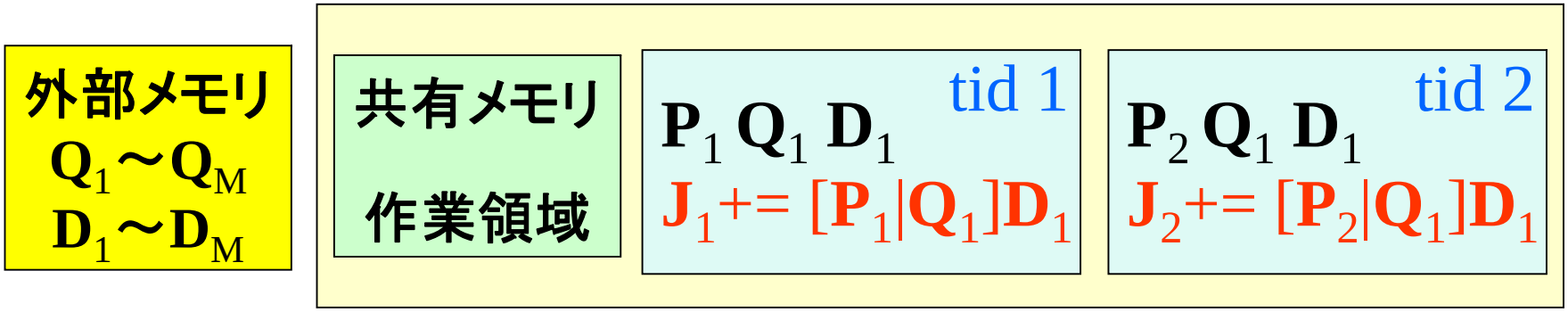
Schwarz上限値Uを求め, 計算必要?

積分cutoff < U < 単精度cutoff

計算不要ならkernel終了

X-Ability
クロスアビリティ

Coulomb Potential J : Parallelization



32スレッドは同時に積分計算し, *J* に加算する
*J*は倍精度が良い

X-Ability
クロスアビリティ

Result : Acceleration of J-matrix

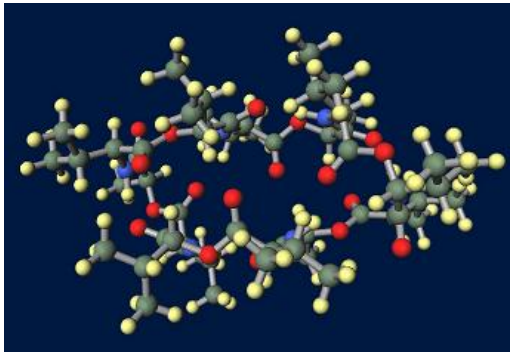
valinomycineなど通常分子サイズ（オーダーNが大きい）にoptimizeされたカーネル（L1/L2 cacheが効かない）

CPU(Core2Quad 1core) : 8.267 sec

GPU(GTX470 x 1) : 0.243 sec

Gaussian03 on 1Process-1CPU

vs 1Process-1GPU / 3-21G basis set



※ Gaussian is compiled with intel Fortran compiler / mkl

gaussianは本来はcontractedだが、uncontractedで計算

FMMで計算している部分は除く

X-Ability
クロスアビリティ

Hartree-Fock Exchange K

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

$$D(r_1,r_2)=\sum_{cd}D_{cd}\chi_c(r_1)\chi_d(r_2) \qquad \rho(r_2)=\sum_{cd}D_{cd}\chi_c(r_2)\chi_d(r_2)$$

1. Hermite Gauss基底に利点なし：密度行列をp,qの形で書いても意味無し（J-matrixと違う）
2. 2電子積分 $(ac|bd)D_{cd}$ で計算（下記アルゴリズムが望ましい）

静電ポテンシャルJと同時計算?

対称性$(ac|bd)=(ca|bd)=(bd|ac)=\ldots$計算量1/8

最も効率の良いアルゴリズムはGPUに未実装

CPUでの実装も非効率?

X-Ability
クロスアビリティ

Procedure of K-matrix

① Shell pair $|ab]$作る

② $|ab]$と$|cd]$並び替える

$|ab]$: $[ab|ab]^{1/2}$の減少順

$|cd]$: cのloopに関して$[cd|cd]^{1/2}$の減少順

③ GPUにp, q, D_qを送る

④ GPUで $K_{ac} = \sum_{b,d} [ab|cd] D_{bd}$ を並列計算

各threadが1組の$[ab|cd]$を計算

積分対称性はフルに活用する　※総当り戦アルゴリズムの活用

⑤ 積算後の K_{ac}をhostに返す

通信量：計算量の10%以下

X-Ability
クロスアビリティ

K-matrix : Pseudocode(CPU)

$$
\begin{aligned}
&do\ a\\
&\ do\ c\\
&\ \ K_{ac} = 0\\
&\ \ do\ b = a, n\\
&\ \ \ do\ d = c, n\\
&\ \ \ \ if\left(ab \succ cd\right) then\\
&\ \ \ \ \ K_{ac} += \left\langle ab \middle| cd \right\rangle D_{bd}\\
&\ \ \ \ end\ \ if\\
&\ \ \ enddo\\
&\ \ enddo\\
&\ enddo\\
&enddo
\end{aligned}
$$

対称性の活用

$$
\begin{aligned}
K_{13} &+= \langle\ |\ \rangle D_{24}\\
K_{23} &+= \langle\ |\ \rangle D_{14}\\
K_{14} &+= \langle\ |\ \rangle D_{23}\\
K_{24} &+= \langle\ |\ \rangle D_{13}
\end{aligned}
$$

対称性の定義

$$
\begin{aligned}
&\langle 12 | 34 \rangle\\
&= \langle 21 | 34 \rangle\\
&= \langle 12 | 43 \rangle\\
&= \langle 21 | 43 \rangle\\
&= \langle 34 | 12 \rangle\\
&= \langle 43 | 12 \rangle\\
&= \langle 34 | 21 \rangle\\
&= \langle 43 | 21 \rangle
\end{aligned}
$$

総当り戦アルゴリズム(GPU)

A,B,C,Dは同時に2試合A-B, C-Dには出れない。これを活用し、異なるブロックで同じaやbが重複しないようにする。対戦日に対応するKがあり、最後に足す

Shell pair : 試合、Shell : チーム、K : 対戦日

X-Ability
クロスアビリティ

Picture of K-matrix on GPU (1)

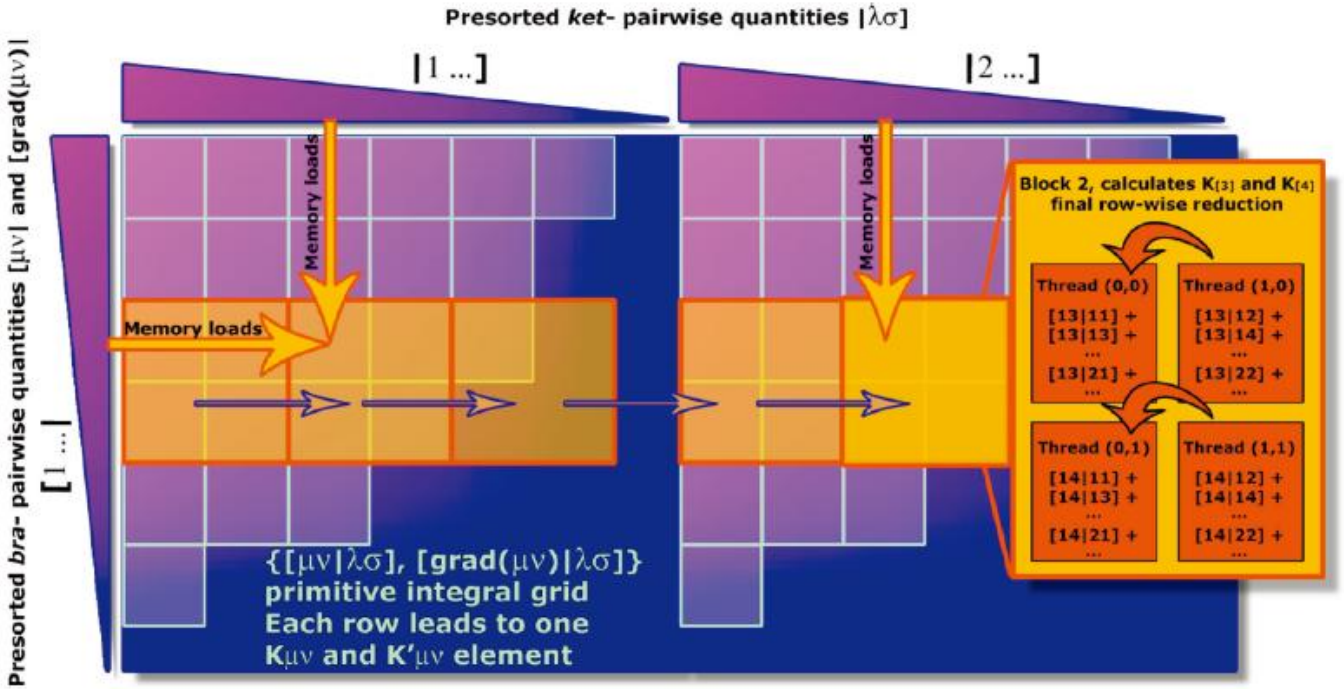
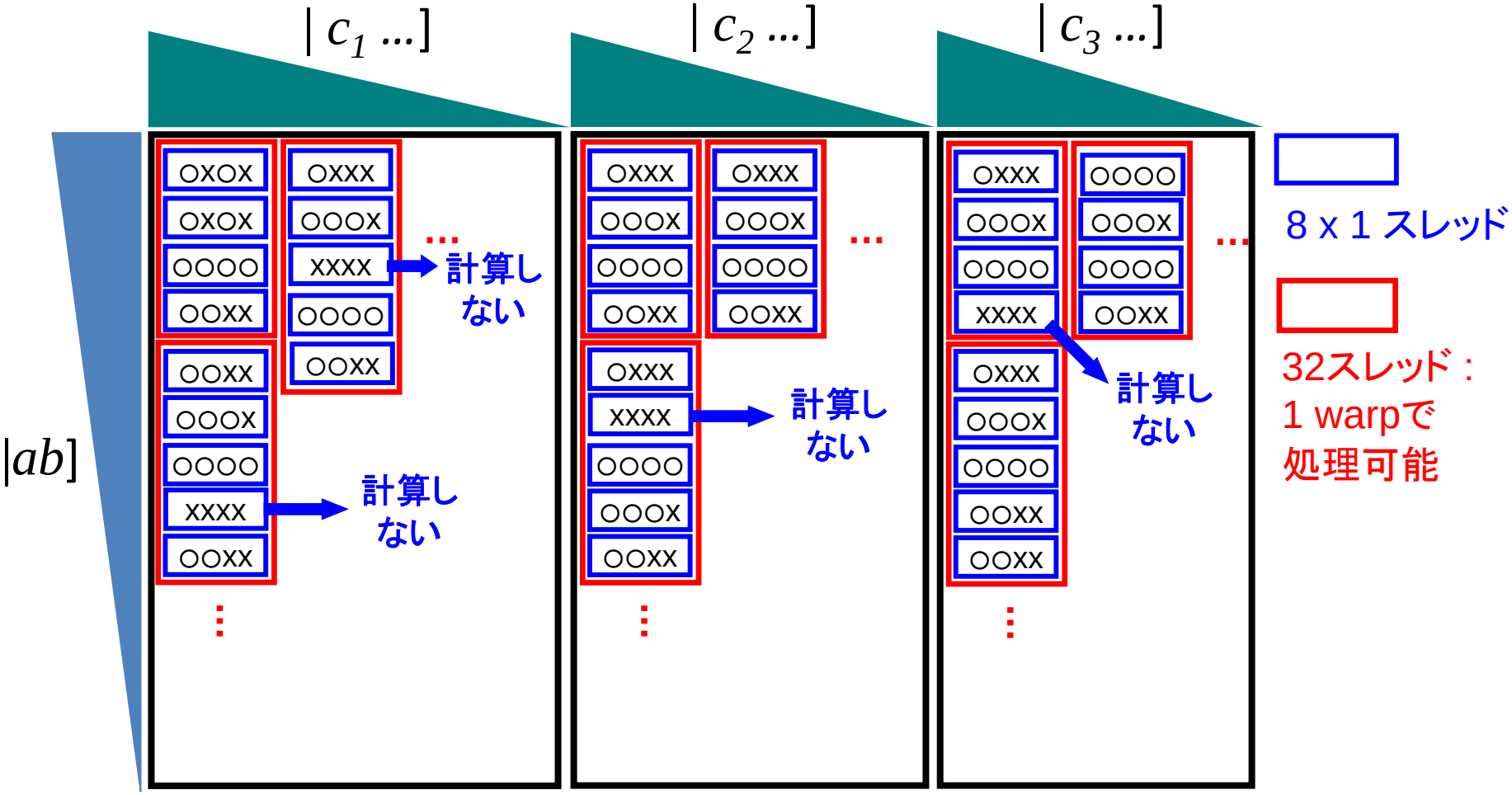


Figure 2. $K_{\mu\nu}$ and $K'_{\mu\nu}$ calculation algorithm. Similar to the *J* calculation shown in Figure 1, but in this case all *ket*-pairs are grouped into *N* segments according to the λ_i index. Two such segments are represented. The GPU thread block scans all the segments sequentially. Once integrals which make contributions smaller than 10^{-11} au are reached, the scan of the particular segment is aborted, and the block proceeds from the next segment.

Kを最初にやったMartinezは、interblock通信(e.g. Atomic Add)をしたくなかったので、acを1つのblockに担当させて、そのblock内で全部bdをやるという方法をとった。その結果、shell pairの対称性は使えなくった（全体の1/2しか削減で）。

X-Ability
クロスアビリティ

Picture of K-matrix on GPU(2)



我々は対称性をフルに使うことを考え、$|ab]$をblock単位とするループを採用した。計算スレッドはサイズ依存性があるがアミノ酸なら〇：X = 8：2

X-Ability
クロスアビリティ

Picture of K-matrix on GPU(3)

① Martinezは対称性を犠牲にしてblock内でaとcを固定した。

② K_{ac}、K_{ad}、K_{bc}、K_{bd}の片方の添字がaあるいはbなので、block間でaが重複しておらず、かつ、bが重複していなければmatrixの中で独立した要素にアクセスできた

③ それを最大限活用する単位をblockに担当させた。

④ １個のK-matrixに対応するshell pairを適切に配分することを考えた結果、全てのpairがでてきて、かつ全てのプレイヤーがダブルヘッダーはできない総当り戦アルゴリズムが最適だと考えた。

X-Ability
クロスアビリティ

Implementation of K-matrix

- Atomic addをやらないように総当り戦のアルゴリズムを採用した
- _syncthreads()のコストを減らすためにwarp単位(4 x 8)でSIMT演算を行った
- 密度行列のスクリーニング基準を減らすためD_{MAX}を採用した

X-Ability
クロスアビリティ

Result : Acceleration of K-matrix(1)

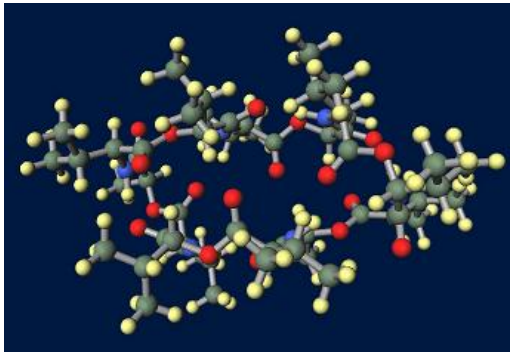
valinomycineなど通常分子サイズ（オーダーNが大きい）にoptimizeされたカーネル（L1/L2 cacheが効かない）

CPU(Core2Quad 1core) : 59.995 sec

GPU(GTX470 x 1) : 3.903 sec

Gaussian03 on 1Process-1CPU

vs 1Process-1GPU / 3-21G basis set



※ Gaussian is compiled with intel Fortran compiler / mkl

1回のK行列の(sp,sp | sp,sp)の計算のみの時間測定

gaussianは本来はcontractedだが、uncontractedで計算

gaussianは本来J/Kまとめて計算するが、ここではKのみ計算

(sp,sp | sp,sp)のみ

X-Ability
クロスアビリティ

Summary of Density Functional Theory

電子の波動関数（軌道）

$$[-\nabla^2/2 + v_{eff}(r)]\psi_i(r) = \varepsilon_i \psi_i(r)$$

運動エネルギー

原子核と電子からの静電ポテンシャル+交換相関ポテンシャル

固有値問題
行列対角化で解く

$$\psi_i(r) = \sum_k C_k^{(i)} \chi_k(r), \quad \rho(r) = 2\sum_i \psi_i(r)^2$$ 基底関数展開

行列要素計算が律速

静電ポテンシャル $J_{ab} = \sum_{cd} (ab|cd) D_{cd}, \quad D_{cd} = 2\sum_i C_c^{(i)} C_d^{(i)}$

2電子積分 $(ab|cd) = \int \frac{\chi_a(r)\chi_b(r)\chi_c(r')\chi_d(r')}{|r-r'|} dr'dr$

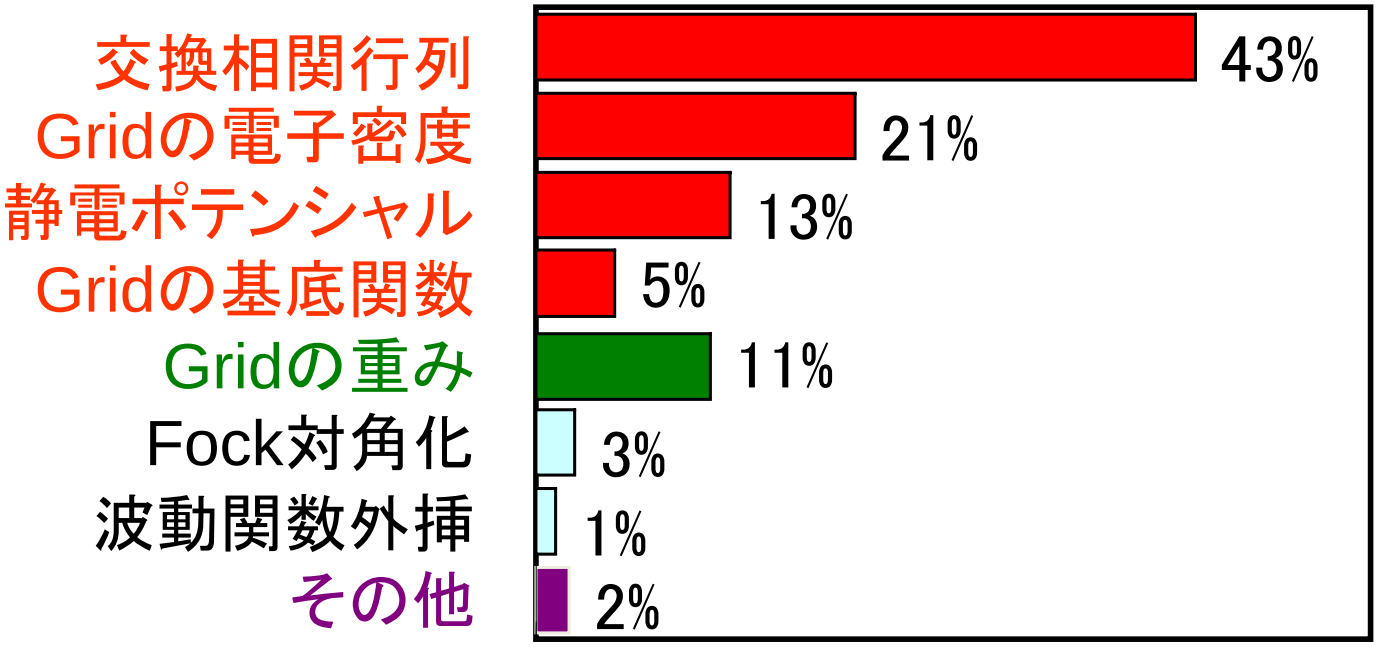
交換相関ポテンシャル $v_{kl} = \int \chi_k(r)\chi_l(r) f(\rho(r), \nabla\rho(r)) dr$

X-Ability
クロスアビリティ

Computation Time of DFT(Gaussian)

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

アクセラレータで加速 / CPUで加速 / 加速可能



メモリー量、精度、並列化方法、通信量を検討する

X-Ability
クロスアビリティ

Exchange Correlation Term

$$E_{xc} = \int \varepsilon(\rho(r), \gamma(r))dr \approx \sum_i w_i \varepsilon(\rho(r_i), \gamma(r_i)), \quad \gamma(r) = \nabla\rho(r) \cdot \nabla\rho(r)$$

交換相関エネルギー：運動＋静電以外の残り

交換相関ポテンシャル

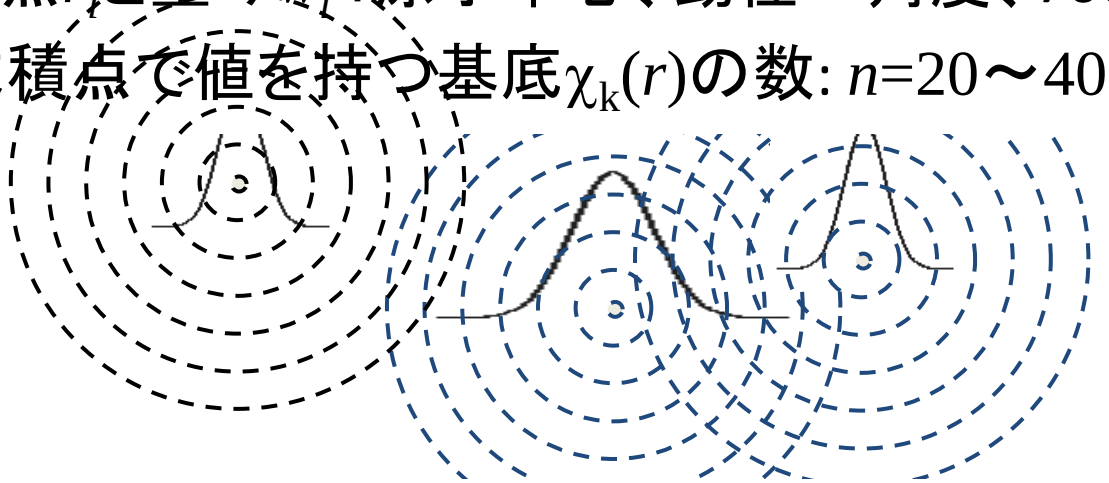
$$\langle \chi_k | v_{xc} | \chi_l \rangle = \langle \chi_k | \frac{\delta E_{xc}}{\delta \rho} | \chi_l \rangle = \int \chi_k \frac{\partial \varepsilon}{\partial \rho} \chi_l + \frac{\partial \varepsilon}{\partial \gamma} \nabla\rho \cdot \nabla(\chi_k \chi_l) dr$$

$$\approx \sum_i w_i \chi_k(r_i) \left[\frac{\partial \varepsilon(r_i)}{\partial \rho} \chi_l(r_i) + \frac{2\partial \varepsilon(r_i)}{\partial \gamma} \nabla\rho(r_i) \cdot \nabla\chi_l(r_i) \right]$$

ε, v_{xc}: 複雑な関数、解析積分不可能

求積点r_iと重みw_i：原子中心、動径×角度、7000点/原子

求積点で値を持つ基底$\chi_k(r)$の数: n=20～400



X-Ability
クロスアビリティ

Procedure of Exchange Correlation

$$E_{xc} = \sum_i w_i \varepsilon(\rho(r_i), \gamma(r_i)), \quad \gamma(r) = \nabla\rho(r) \cdot \nabla\rho(r)$$

$$\langle \chi_k | v_{xc} | \chi_l \rangle = \sum_i w_i \chi_k(r_i) \left[\frac{\partial \varepsilon(r_i)}{\partial \rho} \chi_l(r_i) + \frac{2\partial \varepsilon(r_i)}{\partial \gamma} \nabla\rho(r_i) \cdot \nabla\chi_l(r_i) \right]$$

① 求積点r_iと重みw_iの生成

② 求積点上の電子密度$\rho(r_i)$と$\nabla\rho(r_i)$

$$\rho(r_i) = \sum_{kl} D_{kl} \chi_k(r_i) \chi_l(r_i)$$

求積点r_iで並列化

密度行列Dを送り、$\rho(r_i)$と$\nabla\rho(r_i)$を回収

③ E_{xc},求積点上のポテンシャル f_i, $\mathbf{g_i}$

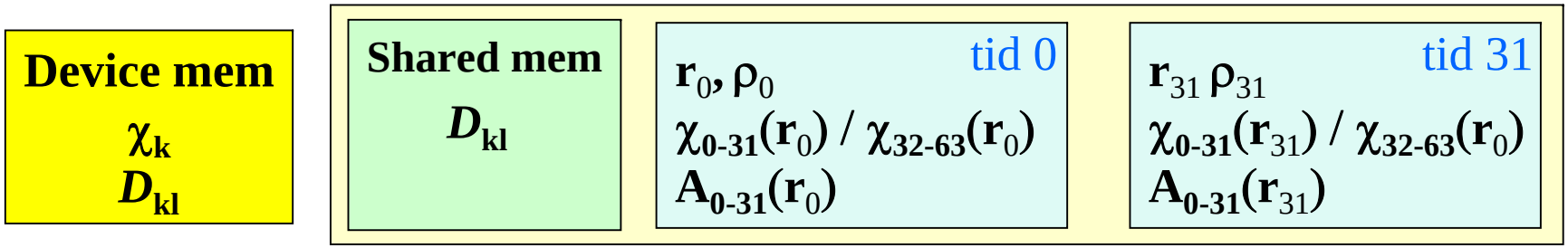
$$f_i = w_i \frac{\partial \varepsilon(r_i)}{\partial \rho}, \quad \mathbf{g}_i = 2w_i \frac{\partial \varepsilon(r_i)}{\partial \gamma} \nabla\rho(r_i)$$

④ v_{xc} 行列： $v_{kl} = \sum_i \{ f_i \chi_k(r_i) + \mathbf{g}_i \cdot \nabla\chi_k(r_i) \} \chi_l(r_i)$

f_i, $\mathbf{g_i}$を送り、v_{kl} 行列を回収

X-Ability
クロスアビリティ

Density Matrix : Parallelization



求積点r_i、基底関数χ_kを、32個単位にblock化

128 threadsで①～③を反復

① r_0～r_{31}並列に $\chi_0(r_i)$～$\chi_{31}(r_i)$を計算 $\chi_l(r_i) = \sum_n C_n e^{-\alpha_n (r_i - R_n)^2}$

② 32×32行列D_{kl}を読み、行列-ベクトル積

$$A_K(r_i) = \sum_l D_{kl}\chi_l(r_i)$$

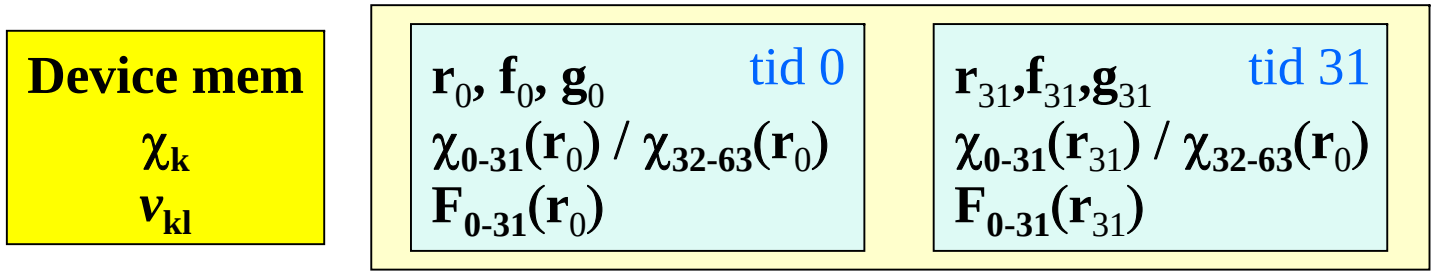
③ r_0～r_{31}並列に $\chi_{32}(r_i)$～$\chi_{63}(r_i)$を計算

ベクトル内積 $\rho(r_i) += \sum_k \chi_k(r_i) A_k(r_i)$

②だけをcuBLASで高速化はできない

X-Ability
クロスアビリティ

Exchange Correlation Potential Matrix



求積点r_i、基底関数χ_kを、32個単位にblock化

128 threadsで①～④を反復

① r_0～r_{31}並列に $\chi_0(r_i)$～$\chi_{31}(r_i)$, $F_0(r_i)$～$F_{31}(r_i)$を計算

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

② $F_k(r_i)$を転置

③ r_0～r_{31}並列に $\chi_{32}(r_i)$～$\chi_{63}(r_i)$を計算

④ 32×32行列積　$v_{kl} += \sum_i F_k(r_i)\chi_l(r_i)$

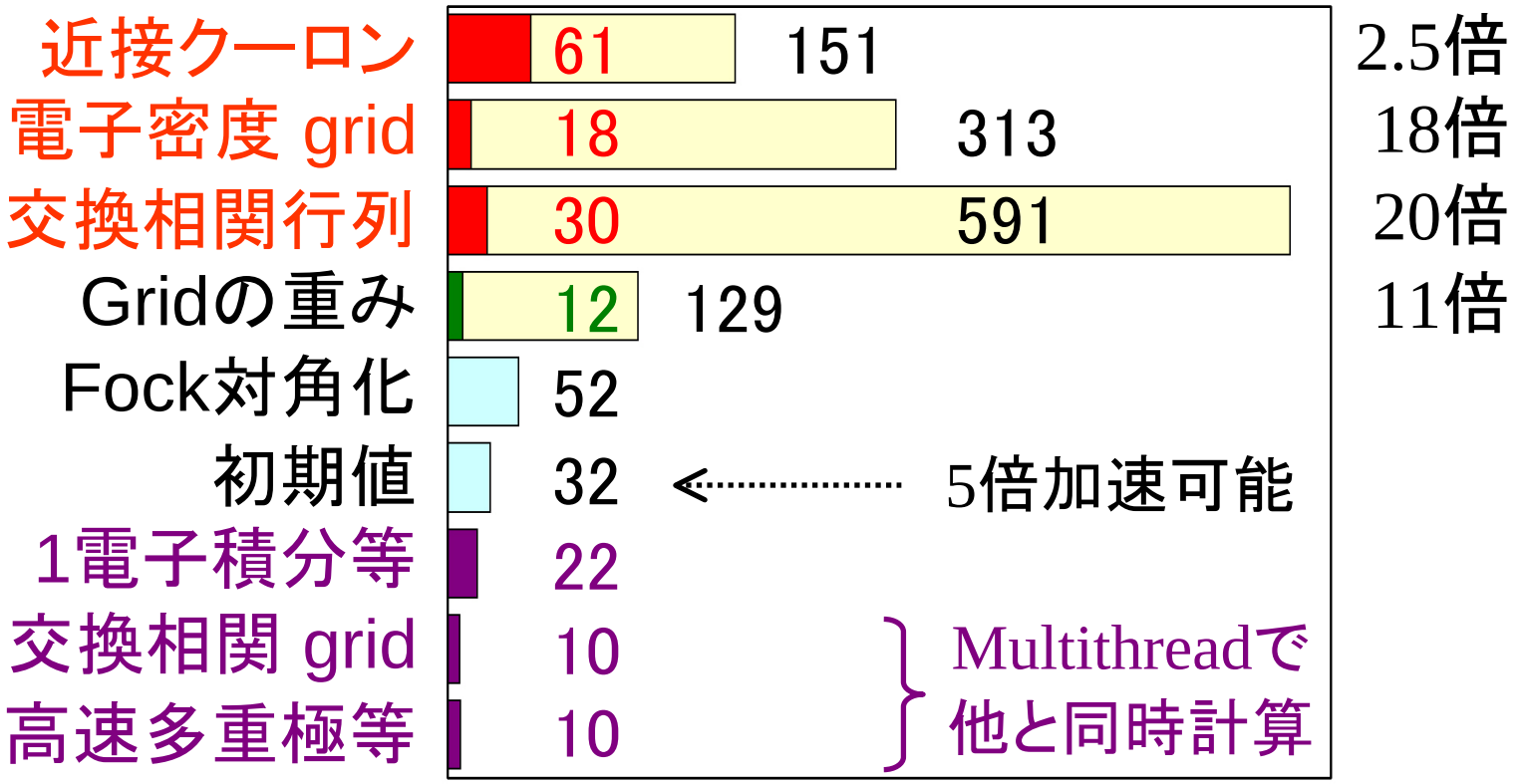
④だけをcuBLASで高速化はできない

X-Ability
クロスアビリティ

Sample of Acceleration on GPU

Core2 Quad 2.66GHz (4core) + G80

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



全体で5倍程度の加速

プログラムの工夫で8倍程度に

X-Ability
クロスアビリティ

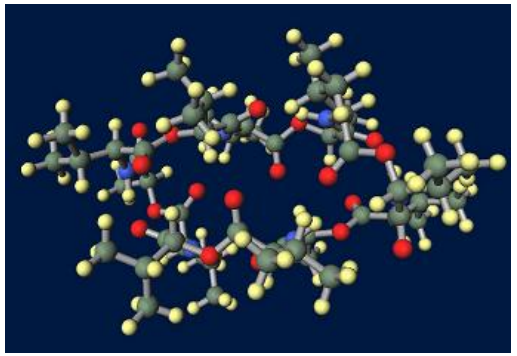
Result : Acceleration of Gaussian03

- ベンチマーク（擬似test397 : DFTエネルギー勾配）

Valinomycine($C_{54}H_{90}N_6O_{18}$)での計算時間の比較 （3-21G基底、blyp汎関数）

		時間[秒]	エネルギー[a.u.]
CPUのみ	Gaussian 03 rev B.01	289.93	-3772.609959
CPUのみ	GAMESS-US 2010 R3	3603.42	-3772.609882
Ufimtsevら	TeraChem beta3(1GPU)	192.76	-3772.608483
我々	Gaussian + XA-CUDA-QM (1GPU)	124.90	-3772.609078
我々	Gaussian + XA-CUDA-QM (2GPU)	113.80	-3772.609077

CPU:Intel Xeon E5540 2.53 GHz 8 core
GPU:Tesla C1060 x 2
Intel Fortran Compiler 11.1/CUDA 2.3/MKL 10.2



X-Ability
クロスアビリティ

Accuracy of Quantum Chemistry

- 1 hartree = 627.51 kcal/mol (unit : a.u.)
  - 水素結合 5 kcal/mol
  - Vdw結合 0.5 kcal/mol
  - コンフォメーション解析が必要な精度 0.1-0.3 kcal/mol
  - Gaussian + XA-CUDA-QMは高速なGPU単精度ユニットを活用すべく、10^{-4} hartreeまで保証する仕様だが、627.51 x 10^{-4} = 0.06 kcal/molなので、コンフォメーション解析も仕様を満たす
  - エネルギー絶対値が重要なら（特にマテリアル分野）10^{-6} くらいの議論が必要かもしれない
  - 沢山計算して相対値を議論するような用途であれば上記の精度仕様はむしろ過剰かもしれない

X-Ability
クロスアビリティ

Second Order Perturbation Energy

平均場近似への補正（分散力）
計算量O(N^5)、d, f...関数必要

RI-MP2：補助基底P,QでO(N^3)

① $(P|Q)^{-1/2}$, $(\mu\nu|P)\rightarrow(ia|P)$

② $B_{ia,Q} = \sum_P (ia|P)(P|Q)^{-1/2}$

③ $(ia|jb) = \sum C_{\mu i} C_{\nu a} C_{\lambda j} C_{\sigma b} (\mu\nu|\lambda\sigma) \approx \sum_Q B_{ia,Q} B_{jb,Q}$ （cuBLAS）

$(\mu\nu|\lambda\sigma) \approx \sum_{PQ} (\mu\nu|P)(P|Q)^{-1}(Q|\lambda\sigma)$

④ $E^{(2)} = \sum_{ijab} \frac{|(ia|jb) - (ib|ja)|^2}{\varepsilon_i + \varepsilon_j - \varepsilon_a - \varepsilon_b}$

$(ia|jb)$：分子軌道基底の2電子積分
ε_i：分子軌道エネルギー

C_{10}～C_{20}で2～4倍の加速

X-Ability
クロスアビリティ

What is Fragment MO method?

大きな絶縁体のab initio法 （タンパク質）

1. 混成軌道で分子を*N*分割

   各フラグメントは閉殻

   フラグメント間相互作用を小さく

2. Self-consistent charge (SCC)

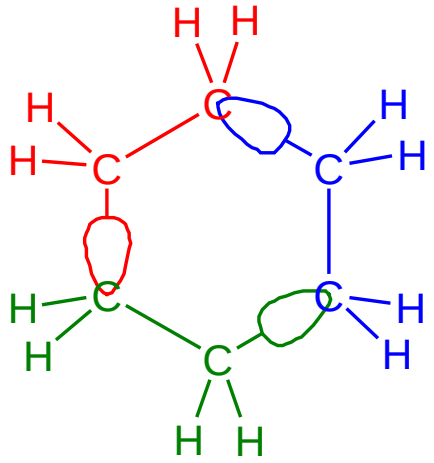
   静電場でのフラグメントのE_I

3. SCC下でフラグメント対のE_{IJ}

4. 全エネルギー $E = \sum_{I>J} E_{IJ} - (N-2)\sum_{I} E_I$

   ◎並列化、実装容易

   ×陰イオン、半導体、励起状態、構造変化



$(C_2H_4)_3$

X-Ability
クロスアビリティ

Acceleration of Environmental Static Potential

- ERI J-matrixをベースに加速
  - cdは他のフラグメントの基底関数

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

- 1フラグメント1アミノ酸残基のような小さいサイズの問題に対して（デフォルトでは基底が180以下）Direct SCFではなくなるので、ERIがボトルネックではなくなり、環境静電ポテンシャルの計算の割合が大きくなるためここを加速する必要がある。
- 環境静電ポテンシャルのみのGPU加速に相当するベンチマークは正確に計測できないが（CPUとのハイブリッドのため）これがFMO加速の肝
  - FMMで加速できるかもしれない

X-Ability
クロスアビリティ

Result : Acceleration of GAMESS-US FMO

FMO2のFK5リガンド分子の各フラグメントのmonomer特性

	E (CPU)	E (GPU)	DX	DY	DZ
1(FK5001 L1)	-402.317404135	-402.317404147	-0.20994	4.36026	1.16317
2(FK5002 L1)	-459.054761634	-459.054761645	6.7929	0.2928	-3.10568
3(FK5003 L1)	-466.843941102	-466.843941089	1.02059	-1.28229	-2.07834
4(FK5004 L1)	-293.669434284	-293.669434291	-1.14973	-1.54089	1.95314
5(FK5005 L1)	-407.321099998	-407.321099990	-2.14597	1.88929	-3.06977
6(FK5006 L1)	-537.570750054	-537.570750051	-2.58186	2.29468	-1.05552

GAMESS-US 2009 FMO2 (gfortran + ATLAS), RHF/6-31G, CPU: Core i7 860 2.80GHz×4 core, GPU: GTX470×1

主に環境静電ポテンシャル(ESP)を加速した。FMOエネルギー：-2657.853462811 (CPU), -2657.853429457(GPU), 誤差は3.3×10^{-5} a.u. で事実上無視できる。現時点での計算時間はdimer計算も含め<u>CPU 986.7秒に対しGPU 378.3秒だった。</u>

X-Ability
クロスアビリティ

Result : Acceleration of K-matrix(2)

Amino acidにoptimizeされたカーネル（L1/L2 cacheが効く）

前述したカーネルよりも大きなサイズでは低速

FMOなどには有効

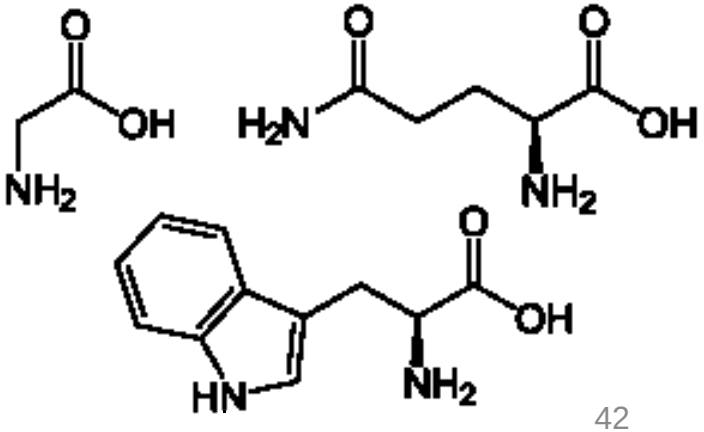
	Core 2	Xeon	GTX470	Tesla C1060
glycine	0.161	0.171	0.013	0.068
glutamine	1.425	1.44	0.098	0.578
tryptphan	4.415	4.393	0.275	1.512

GAMESS-US on 1Process-1CPU

vs 1Process-1GPU / 6-31G basis set

※ GAMESS is compiled with gfortran + ATLAS

※ 8 x 4 -> 32 x 1でL1/L2 cacheを積極的に活用



X-Ability
クロスアビリティ

Conclusion

① 静電ポテンシャル、Hartree-Fock交換項、交換相関ポテンシャルの計算を、GPUで一桁オーダー高速化した。

② Full CPUに対し、Gaussian03のDFTエネルギー勾配を2.7倍に加速することに成功した。

③ Coulomb Potential（J-matrix）のアルゴリズムを流用した環境静電ポテンシャルの高速化により、1フラグメント1アミノ酸残基という小さい問題サイズのGAMESS-US FMOを加速した。エネルギー精度も高く、双極子モーメントなどの物理量もほぼ一致した。

④ データは不規則、小サイズになりがちである。SIMD長は数十以下が良い。行列積だけの加速は役立たない。Kernelは複雑で、種類も多い。

⑤ 特にHartree-Fock交換項については、レジスタ不足で単純な並列化が難しい場合もある。

⑥ 短縮Gauss基底を使う場合、ホストーボード間の通信速度は十分。