量子化学計算の大規模化１

石村　和也

(分子科学研究所　計算分子科学研究拠点(TCCI))

ishimura@ims.ac.jp

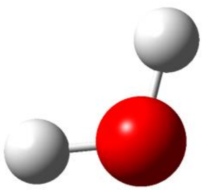
2013年度計算科学技術特論A 第14回
2013年7月18日

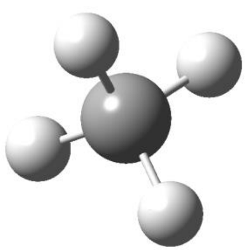
本日の内容

- 量子化学計算とは
- 分子科学分野におけるスパコン利用の歴史
- 量子化学計算方法とコスト
- 大規模計算を行うためには
- 高速化例

- 来週は(並列化+高速化)、今後のプログラムの方向性について

元素と分子





周期表

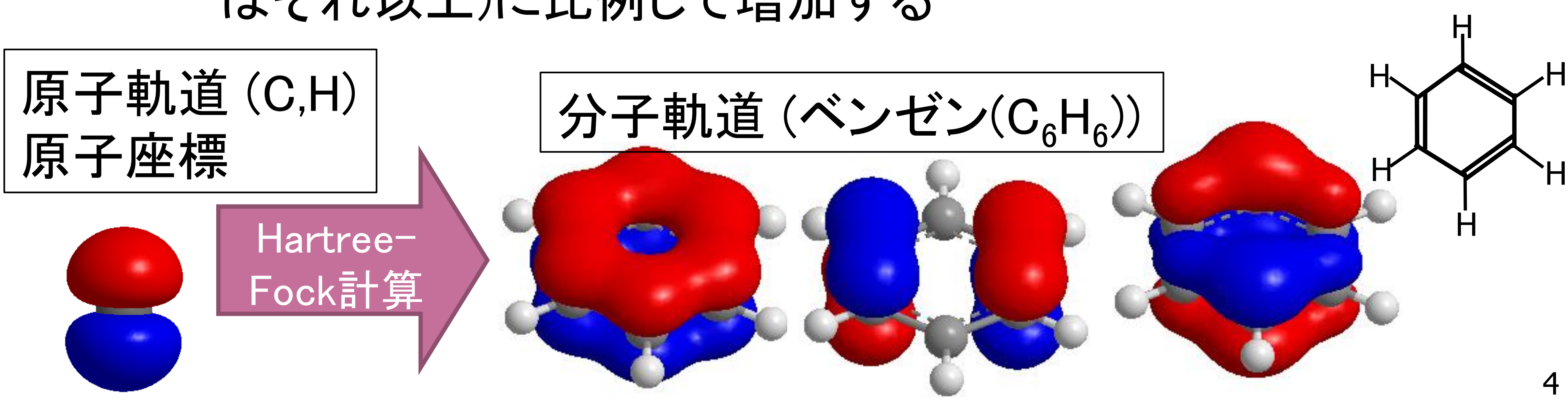
1 H																	2 He
3 Li	4 Be											5 B	6 C	7 N	8 O	9 F	10 Ne
11 Na	12 Mg											13 Al	14 Si	15 P	16 S	17 Cl	18 Ar
19 K	20 Ca	21 Sc	22 Ti	23 V	24 Cr	25 Mn	26 Fe	27 Co	28 Ni	29 Cu	30 Zn	31 Ga	32 Ge	33 As	34 Se	35 Br	36 Kr
37 Rb	38 Sr	39 Y	40 Zr	41 Nb	42 Mo	43 Tc	44 Ru	45 Rh	46 Pd	47 Ag	48 Cd	49 In	50 Sn	51 Sb	52 Te	53 I	54 Xe
55 Cs	56 Ba	*1	72 Hf	73 Ta	74 W	75 Re	76 Os	77 Ir	78 Pt	79 Au	80 Hg	81 Tl	82 Pb	83 Bi	84 Po	85 At	86 Rn
87 Fr	88 Ra	*2	104 Rf	105 Db	106 Sg	107 Bh	108 Hs	109 Mt	110 Ds	111 Rg	112 Cn	113 Uut	114 Uuq	115 Uup	116 Uuh	117 Uus	118 Uuo

*1 Lanthanoid	57 La	58 Ce	59 Pr	60 Nd	61 Pm	62 Sm	63 Eu	64 Gd	65 Tb	66 Dy	67 Ho	68 Er	69 Tm	70 Yb	71 Lu
*2 Actinoid	89 Ac	90 Th	91 Pa	92 U	93 Np	94 Pu	95 Am	96 Cm	97 Bk	98 Cf	99 Es	100 Fm	101 Md	102 No	103 Lr

- 原子の種類(元素)は110程度
- 3000万以上の分子が確認されている（例：H_2O, CH_4）
- 原子間の結合は電子が担っている

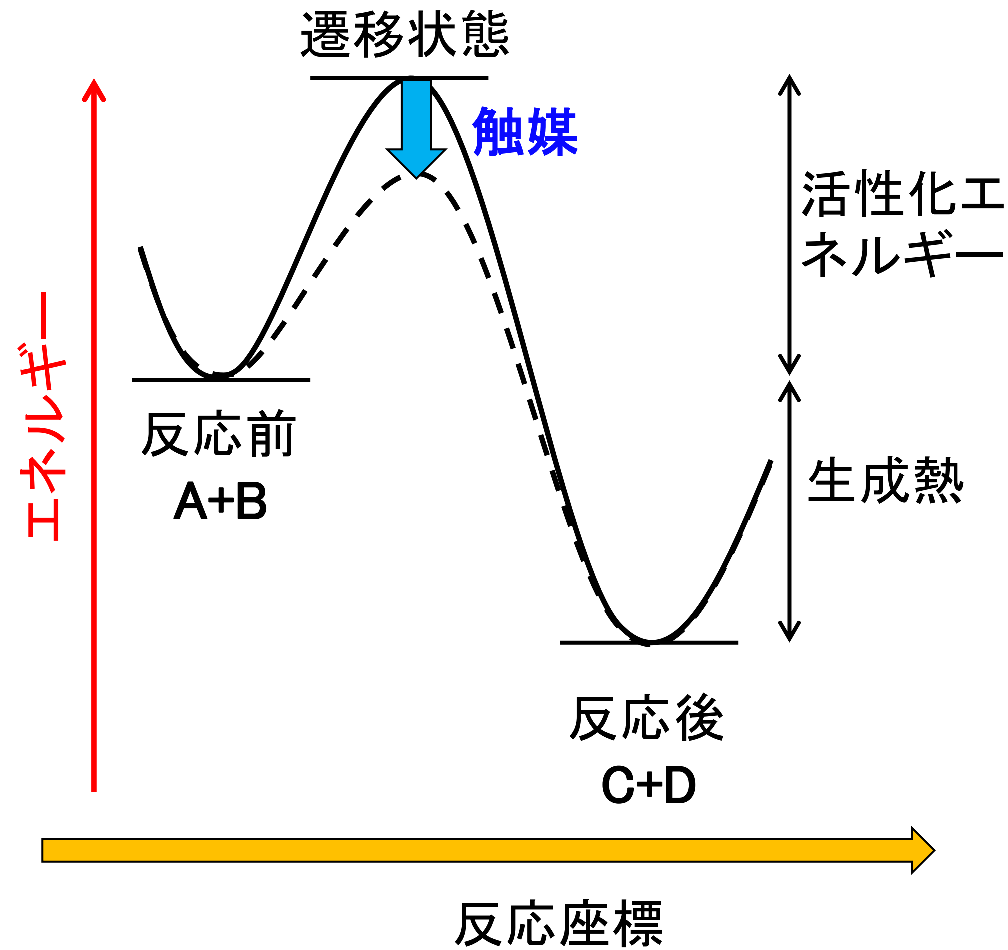
量子化学計算とは

- 量子化学計算:分子の電子分布を計算し、分子の構造、反応性、物性などを解析・予測する
- 入力:原子の電子分布(原子軌道)、原子座標
- 出力:分子の電子分布(分子軌道)
- 計算量は計算方法により異なり、原子数の3乗から7乗(もしくはそれ以上)に比例して増加する

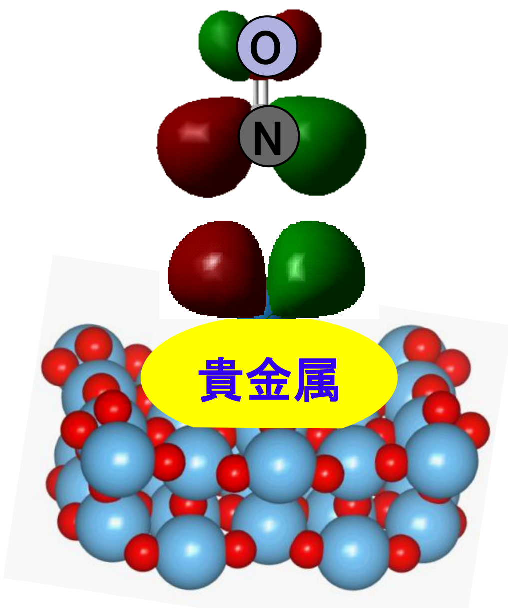


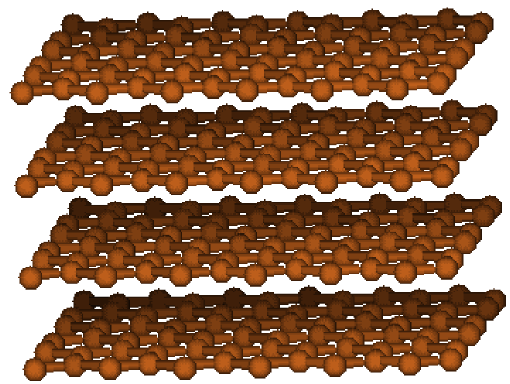
量子化学計算で得られるもの

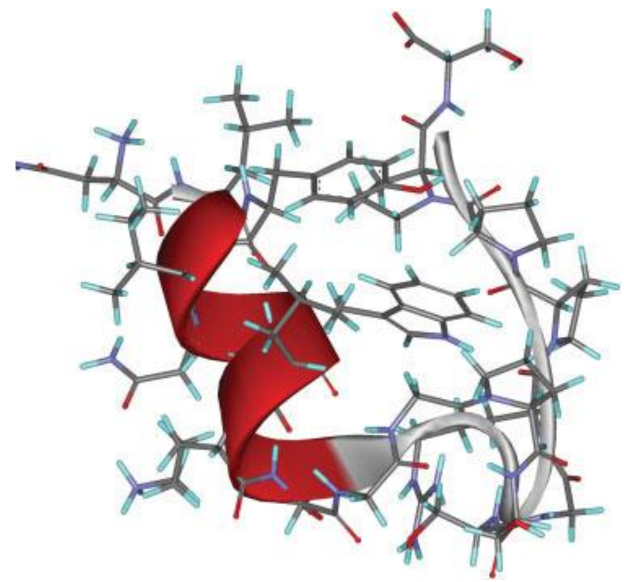
- 分子のエネルギー
- 安定構造、遷移状態構造
- 化学反応エネルギー
- 光吸収、発光スペクトル
- 振動スペクトル
- NMR(核磁気共鳴)スペクトル
- 各原子の電荷
- 溶媒効果
- 結合軌道解析など



量子化学計算適用例







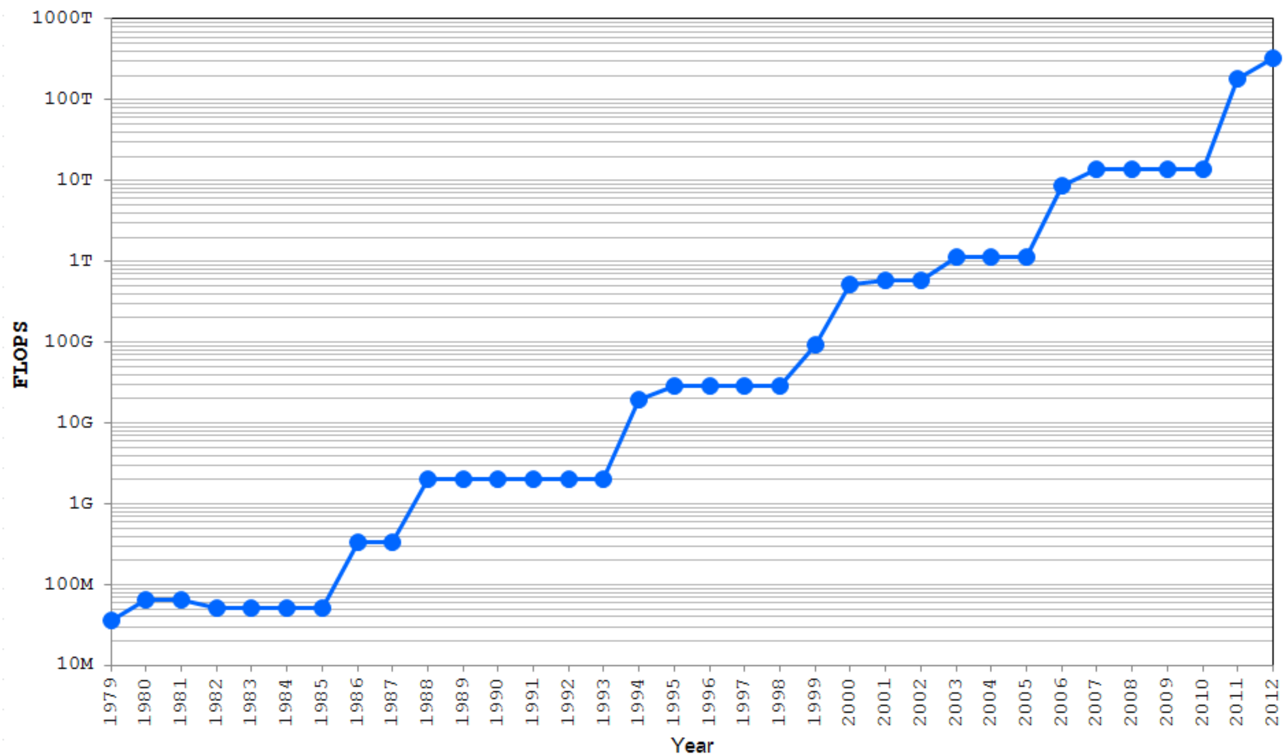
反応分子と触媒表面のくっつきやすさ(相互作用)を計算して、より反応が進む金属を調べる

ナノサイズの分子の計算(反応性、安定性)が可能になり、新たな材料設計

新たな理論や並列アルゴリズムの開発によりタンパク質の計算も可能になった

分子科学分野スパコンの変遷1

- 自然科学研究機構 岡崎共通研究施設 計算科学研究センター（旧分子科学研究所電子計算機センター)におけるCPU能力の変遷　https://ccportal.ims.ac.jp/



年	理論総演算性能（MFLOPS）
1979	36
1989	2,016
1999	92,944
2009	13,606,400
2012	326,283,520

Core i7-4770S　99,200
(4core, 3.1GHz)

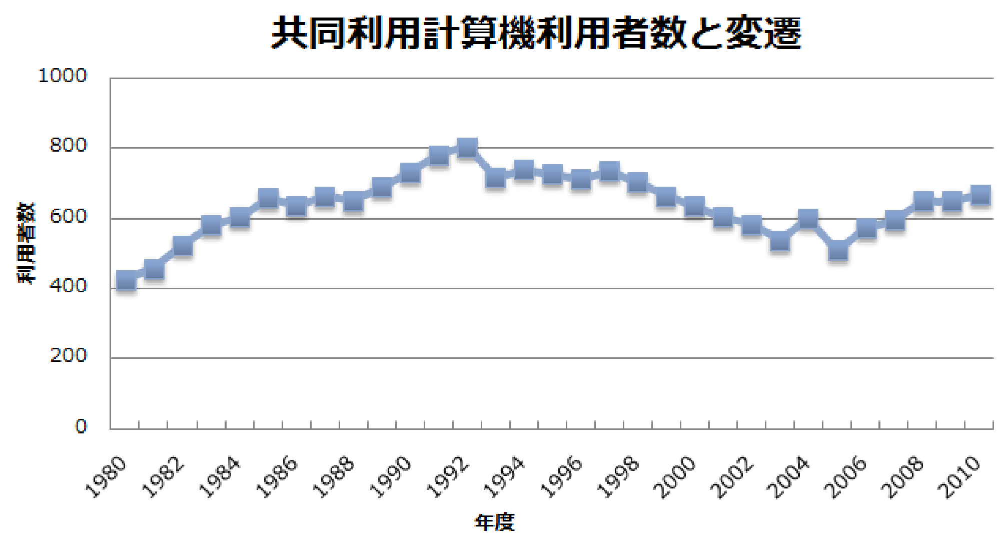
分子科学分野スパコンの変遷2

- 自然科学研究機構 岡崎共通研究施設 計算科学研究センター (旧分子科学研究所電子計算機センター)におけるCPU能力の変遷

年	機種	理論総演算性能 (MFLOPS)
1979	HITACHI M-180(2台)	36
1999	IBM SP2(Wide24 台)	288.0×24
	IBM SP2(Thin24 台)	118.0×24
	NEC SX-5 (8CPU)	64,000
	NEC SX-3/34R (3CPU)	19,200
	合計	92,944
2000(追加)	SGI SGI2800 (256CPU)	153,000
2012	Fujitsu PRIMERGY RX300S7(5472core)	126,950,400
	(+ NVIDIA Tesla M2090 32台)	21,280,000
	Fujitsu PRIMEHPC FX10(1536core)	20,152,320
	SGI UV2000(1024core)	21,299,200
	Fujitsu PRIMERGY CX250S1(5888core)	136,601,600
	合計	326,283,520
	ASCI Red (Intel, 9632cores) (1999)	3,207,000
	地球シミュレータ(2002)	35,860,000
	京コンピュータ(2011)	10,510,000,000

分子科学分野スパコン利用の変遷

- 設立当初から多くの研究者がスパコンを利用
- システムの一部を長時間占有する大規模計算が近年増加



量子化学計算方法とコスト

- 演算内容から分類

N: 基底(or原子)数

計算方法	計算コスト	データ量
Hartree-Fock, 密度汎関数(DFT)法 2電子クーロン反発計算が中心(キャッシュ内演算) 密対称行列の対角化	$O(N^3 \sim N^4)$	$O(N^2)$
摂動法、結合クラスター法 密行列-行列積	$O(N^5 \sim)$	$O(N^4 \sim)$
配置間相互作用法 疎行列の対角化	$O(N^5 \sim)$	$O(N^4 \sim)$

基底の数が2倍になると、計算量はN^3:8倍、N^5:32倍
基底の数が10倍になると、計算量はN^3:1,000倍、N^5:100,000倍

Hartree−Fock法

$$\mathbf{FC} = \boldsymbol{\varepsilon}\mathbf{SC}$$

F: Fock行列, **C**: 分子軌道係数
S: 基底重なり行列, ε: 分子軌道エネルギー

Fock行列

$$F_{\mu\nu} = H_{\mu\nu} + \sum_{i,\lambda,\sigma} C_{\lambda i} C_{\sigma i} \underbrace{\{2(\mu\nu|\lambda\sigma) - (\mu\lambda|\nu\sigma)\}}_{\text{原子軌道(AO)2電子積分}}$$

$$(\mu\nu|\lambda\sigma) = \int d\boldsymbol{r}_1 \int d\boldsymbol{r}_2 \phi_\mu(\boldsymbol{r}_1)\phi_\nu(\boldsymbol{r}_1)\frac{1}{r_{12}}\phi_\lambda(\boldsymbol{r}_2)\phi_\sigma(\boldsymbol{r}_2)$$

$\phi_\mu(\boldsymbol{r}_1)$: 原子軌道Gauss関数

初期軌道係数C計算

↓

AO2電子反発積分計算+Fock行列への足し込み (O(N⁴))

↓

Fock行列対角化 (O(N³))

分子軌道収束せず（AO2電子反発積分計算へ戻る）

↓ 分子軌道C収束

計算終了

2次の摂動(MP2)法

- 積分変換(密行列–行列積)計算が中心

$$E_{MP2} = \sum_{ij}^{occ} \sum_{ab}^{vir} \frac{(ai|bj)\{2(ai|bj) - (aj|bi)\}}{\varepsilon_i + \varepsilon_j - \varepsilon_a - \varepsilon_b}$$

$$(ai|bj) = \sum_{\mu\nu\lambda\sigma}^{AO} C_{\mu a} C_{\nu i} C_{\lambda b} C_{\sigma j} (\mu\nu|\lambda\sigma)$$

ε_i: 軌道エネルギー, $C_{\mu a}$: 分子軌道係数

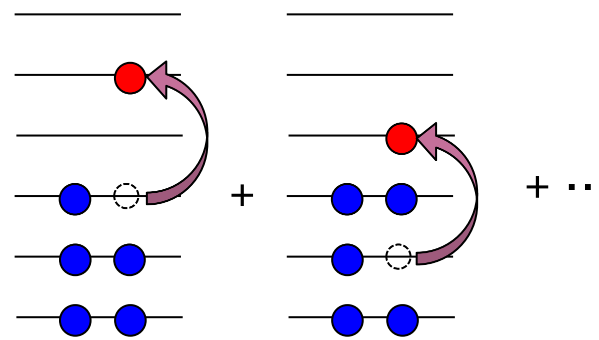
例) 4番目の積分変換のコスト: O^2V^2N
C_{60}(6-31G(d))の場合: O:180, V:720, N:900
演算量O^2V^2N =15000GFlop
Core i7(4core, 3.1GHz)では、約150秒
データ量$1/2 O^2V^2 \times$ 8Byte=67GB

Hartree-Fock計算
↓
$(\mu\nu|\lambda\sigma)$計算 (O(N^4))
↓
$(\mu i|\lambda\sigma)$計算 (O(N^5))
↓
$(\mu i|\lambda j)$計算 (O(N^5))
↓
$(ai|\lambda j)$計算 (O(N^5))
↓
$(ai|bj)$計算 (O(N^5))
↓
MP2エネルギー計算 (O(N^4))

配置換相互作用(CI)法

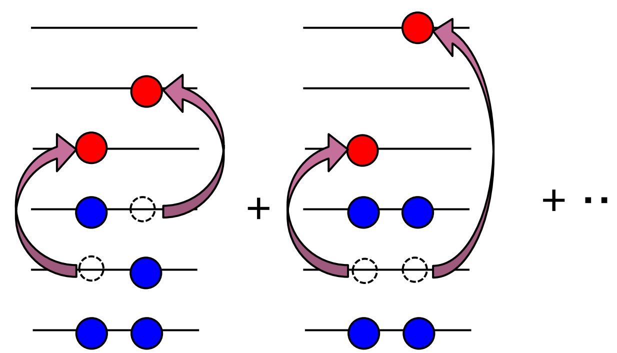
- CIS法：1電子励起配置の線形結合

$$\Psi_{CIS} = \sum_{i}^{occ} \sum_{a}^{vir} c_{ia} \Phi_i^a$$



- CISD法：1,2電子励起配置の線形結合

$$\Psi_{CISD} = c_{HF}\Phi_{HF} + \sum_{i}^{occ} \sum_{a}^{vir} c_{ia} \Phi_i^a + \sum_{ij}^{occ} \sum_{ab}^{vir} c_{ijab} \Phi_{ij}^{ab}$$



例) C_{60}(6−31G(d))の場合：O:180, V:720, N:900

CISの配置の数：O × V= 1.3×10^5

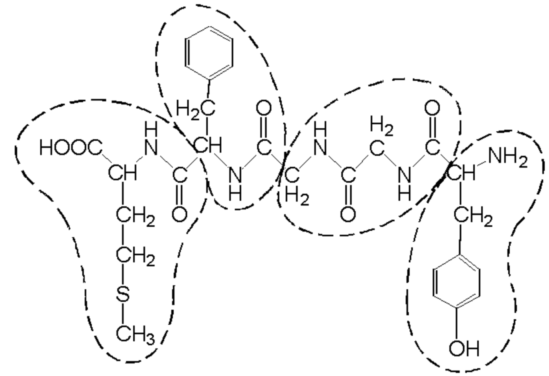
CISDの配置の数: $O^2 \times V^2 = 1.7 \times 10^{10}$

配置の数が疎行列の次元数

大規模計算を行うためには

- **近似の導入**
  - FMO, DC, ONIOM, QM/MMなど分割法
  - ECP, Frozen core, 局在化軌道など化学的知見の利用
- **高速化**(青字:プログラミングが特に重要になる内容)
  - 演算量の削減
  - 収束回数の削減
  - 実行性能の向上
- **並列化**
  - 計算機間の並列化
  - 計算機内の並列化
  - データの分散

近似の導入1



- Fragment MO(FMO)法

K. Kitaura, E. Ikeo, T. Asada, T. Nakano, and M. Uebayasi, Chem. Phys. Lett., 313 (1999) 701-706.

  - 大きなタンパク質をアミノ酸残基(もしくは複数の残基)ごとに分割して、1量体と2量体のエネルギーからタンパク質全体のエネルギーを計算

$$E_{FMO2} = \sum_{I} E_I + \sum_{I>J} \left(E_{IJ} - E_I - E_J \right)$$

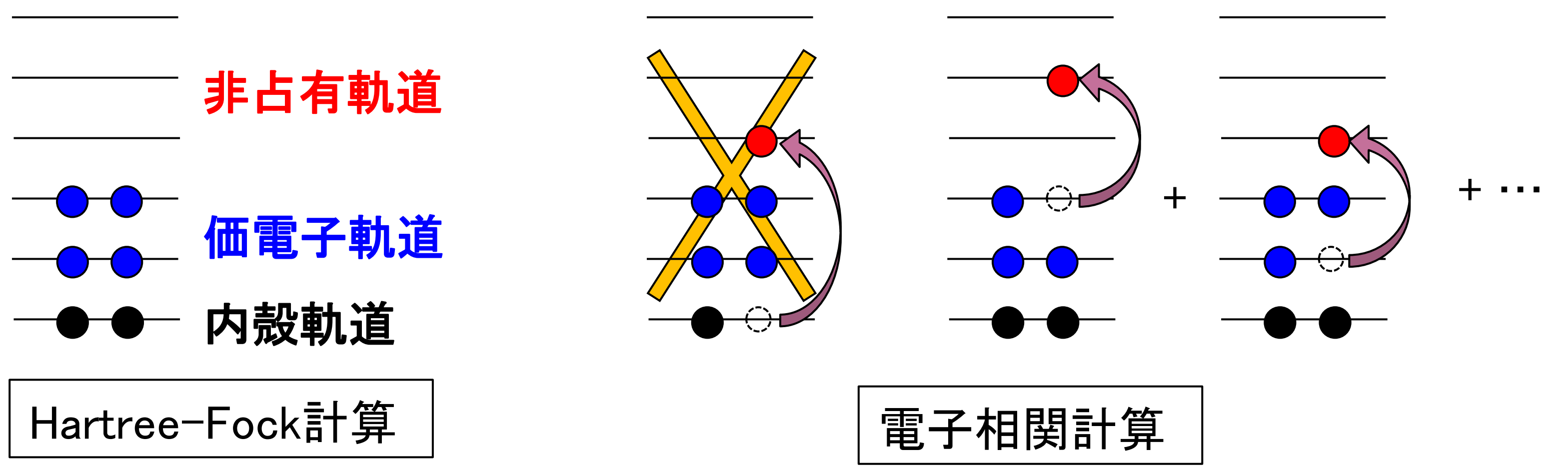
E_I: I番目の1量体エネルギー

E_{IJ}: I番目とJ番目の2量体エネルギー

  - より正確なエネルギーを求める場合は、3、4量体の計算を行う
  - エネルギー計算がアミノ酸残基ごとの相互作用エネルギー解析にもなっている
  - 現在、京数万ノードを使った計算が行われている
- 他に、DC法、ONIOM法、QM/MM法などの分割法がある

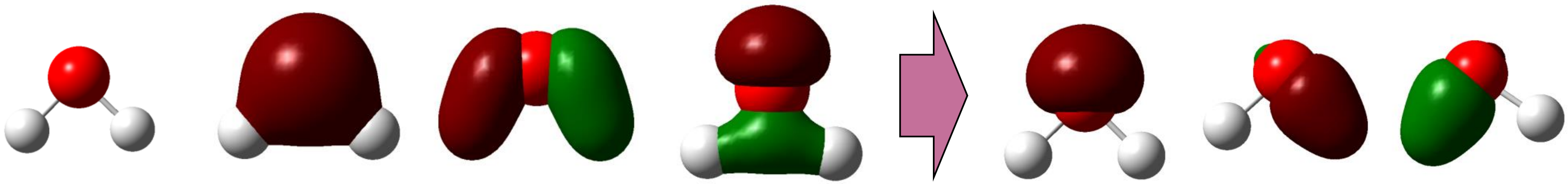
近似の導入2

- Effective core potential(ECP)法
  - 原子間の結合は価電子が重要であるため、内殻電子をポテンシャルに置き換え
  - Hay-Wadt(LANL2), SBKJC, Stuttgart, ...
- Frozen core近似
  - 電子相関計算において、内殻軌道からの励起配置を考慮しない



近似の導入3

- Localized MO法
  - 電子相関計算において、通常分子全体に広がっている分子軌道を局在化させて近くの軌道間の相関のみ考慮



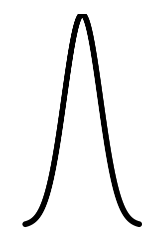
通常のCanonical MOs　　　　Localized MOs

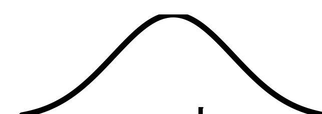
- Resolution of the identity(RI)法, 密度フィッティング法
  - 補助基底を導入して、4中心積分を3中心積分などの積で表現
- Fast Multipole法(FMM)など他にも様々な方法がある

高速化1(演算量の削減)

- 演算量の少ない1,2電子積分計算アルゴリズム開発
- Schwarz inequalityを用いたカットオフ
  - AO2電子積分$(\mu\nu|\lambda\sigma)$計算実行をシェルごとに判断
  - 重原子の価電子用関数、Diffuse関数などexp(−αr^2)のαが小さい場合、原子から遠くまで値があるためあまりスキップされない

```
do M=1, Nbasis
  do N=1, M
    do Λ=1, N
      do Σ=1, Λ
        if({(MN|MN)*(ΛΣ|ΛΣ)}^1/2≧threshold) (μν|λσ)ブロック計算
      enddo
    enddo
  enddo
enddo
```


α:大


α:小

高速化2(演算量の削減)

- 対称性の利用 $(\mu\nu|\lambda\sigma)$ = $(\mu\nu|\sigma\lambda)$ = $(\lambda\sigma|\mu\nu)$ = $(\lambda\sigma|\nu\mu)$ = ...
  - $\mu \Leftrightarrow \nu$、$\lambda \Leftrightarrow \sigma$、$\mu\nu \Leftrightarrow \lambda\sigma$の入れ替えが可能

$$(\mu\nu|\lambda\sigma) = \int d\boldsymbol{r}_1 \int d\boldsymbol{r}_2 \phi_\mu{}^*(\boldsymbol{r}_1)\phi_\nu(\boldsymbol{r}_1)\frac{1}{r_{12}}\phi_\lambda{}^*(\boldsymbol{r}_2)\phi_\sigma(\boldsymbol{r}_2)$$

$\phi_\mu(\boldsymbol{r}_1)$: 原子軌道Gauss関数

  - 具体的には

$$F_{\mu\nu} = H_{\mu\nu} + \frac{1}{2}\sum_{\lambda,\sigma} D_{\lambda\sigma}\{2(\mu\nu|\lambda\sigma) - (\mu\lambda|\nu\sigma)\}$$

$$F_{\mu\nu} = F_{\mu\nu} + D_{\lambda\sigma}(\mu\nu|\lambda\sigma) + D_{\sigma\lambda}(\mu\nu|\sigma\lambda)$$

$$F_{\lambda\sigma} = F_{\lambda\sigma} + D_{\mu\nu}(\lambda\sigma|\mu\nu) + D_{\nu\mu}(\lambda\sigma|\nu\mu)$$

$$F_{\mu\lambda} = F_{\mu\lambda} - \frac{1}{2}D_{\nu\sigma}(\mu\nu|\lambda\sigma)$$

$$F_{\nu\sigma} = F_{\nu\sigma} - \frac{1}{2}D_{\mu\lambda}(\lambda\sigma|\mu\nu)$$

高速化3(収束回数の削減)

- SCFの収束回数削減
  - Direct inversion of the iterative subspace(DIIS)法
  - Second-Order SCF(SOSCF)法
  - Level shift法 (HOMO-LUMOギャップが小さい時に有効)
  - 小さな基底で分子軌道計算　-> それを初期軌道にして大きな基底での計算 (DFT計算で重要)
- 構造最適化回数削減
  - Newton-Raphson法,
  - Hessianのアップデート: BFGS法
  - 効率的な座標系 : Redundant coordinate, Delocalized coordinate(DLC)

高速化4(実行性能の向上)

- SIMD演算の利用
  - 1つの演算命令で複数のデータを処理
  - do、forループを多用
- 時間のかかる演算回数を削減
  - 割り算、組み込み関数の利用回数削減（まとめて演算する、ループ内で同じ演算はループの外に出す）
- データアクセスを効率的にしたアルゴリズム・プログラム開発
  - 連続したデータアクセスでキャッシュミスの削減
  - 多次元配列の取り扱い 例：A(ix, iy, iz) or A(iz, iy, ix)

演算器 ← **L1キャッシュ** ← **L2キャッシュ** ← **メモリ**

相対的な速度 100 10 1

高速化5(実行性能の向上)

- コンパイラの最適化オプションの設定
  - 基本的にはコンパイラが最適化しやすいようにコードを書く必要がある
- BLAS, LAPACKなど数学ライブラリの利用
  - BLASライブラリはCPUの性能を引き出してくれるが、小さい配列(100次元程度)の場合、サブルーチンコールのオーバヘッドの方が大きくなる可能性がある
  - BLAS2を数多く使うより、BLAS3でまとめて実行

	演算量	データ量
BLAS2 (行列−ベクトル)	$O(N^2)$	$O(N^2)$
BLAS3 (行列−行列)	$O(N^3)$	$O(N^2)$

並列化

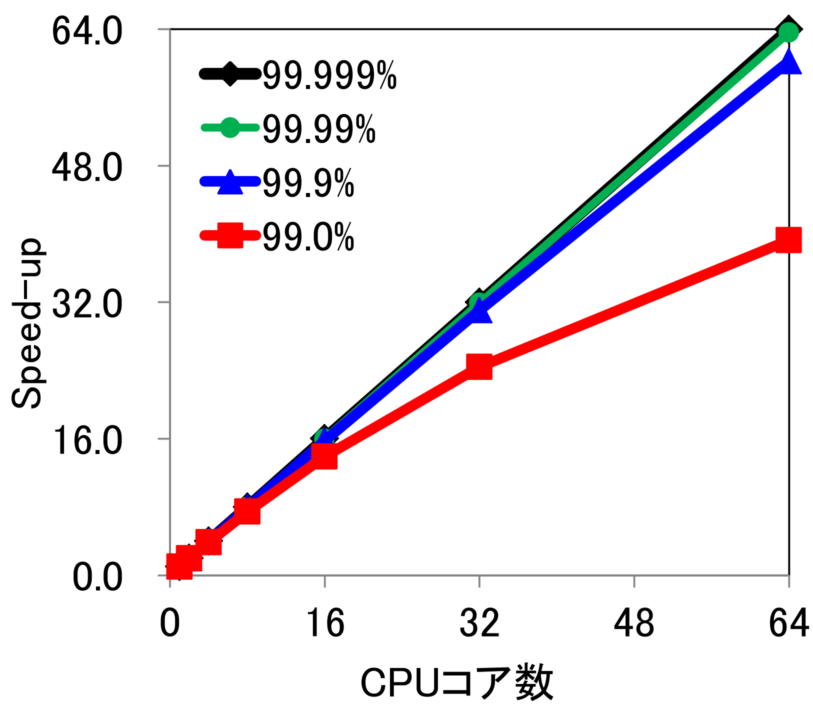
- 均等な計算負荷分散
  - 式の変形
  - 多重ループの順番の工夫
- 大容量データの分散
  - 京のメモリは1ノード16GB, 8万ノードでは1PB以上
  - 中間データ保存用としてディスクはあまり期待できない
- 通信量、通信回数の削減
  - 並列計算プログラムのチューニングで最後に残る問題は通信関係(特にレイテンシ)が多い
- ノード間(MPI)、ノード内(OpenMP)それぞれで分散

並列化率と並列加速率(Speed-Up)

計算負荷が均等に分散されている場合

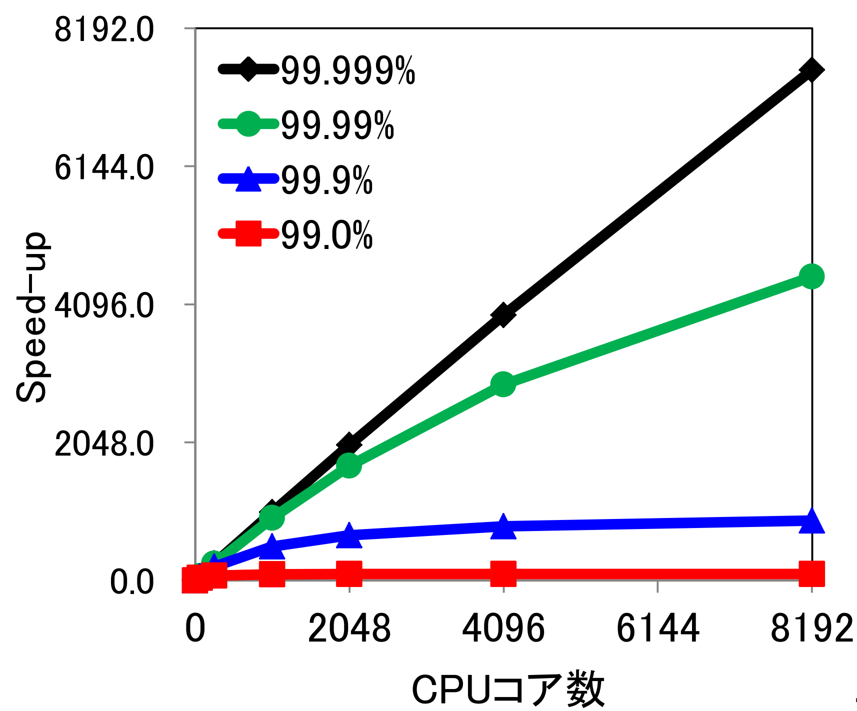
数十コア(PCクラスター)

並列化率**99%**でもある程度速くなる



数千コア(スーパーコンピューター)

並列化率**99.9%**でも不十分



並列化手法

難易度

OpenMP(ノード内)

使い始めるのは簡単だが、性能を引き出すのは大変な場合が多い

MPI(ノード間、ノード内)

使い始めは大変だが、性能を出すのはOpenMPよりも容易

BLAS,LAPACKライブラリ利用(ノード内)

行列・ベクトル演算を置き換え、行列×行列は効果大

Intel MKL

AMD ACML

コンパイラの自動並列化(ノード内)

コンパイルするときにオプションを付けるだけ

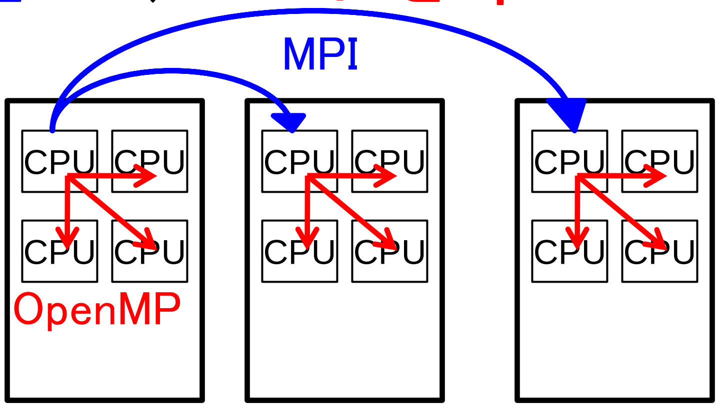
ループの計算を分散、あまり効果が得られないことが多い

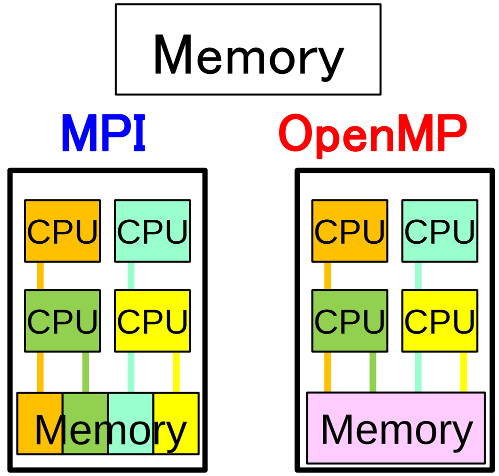
pgf90 –Mconcur

ifort –parallel

MPI+OpenMPハイブリッド並列化

ノード間をMPI、ノード内をOpenMPで並列化





メリット

並列化効率の向上　ノード内ではOpenMPで動的に負荷分散
MPIプロセス数削減による計算負荷バランスの向上
MPIプロセス数削減による通信の効率化

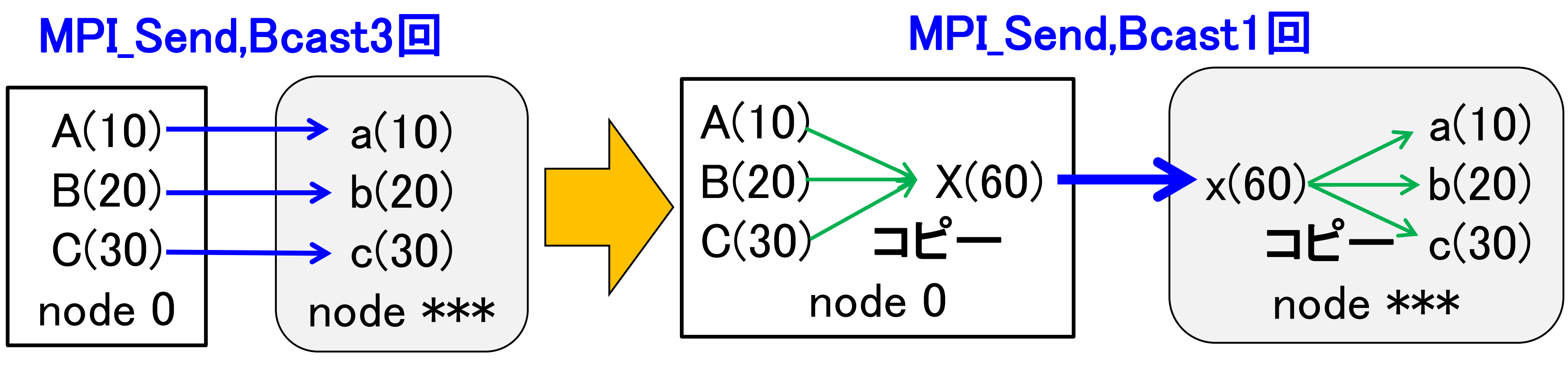
メモリの有効利用　OpenMPによるノード内メモリの共有

デメリット

アルゴリズム、プログラムが複雑になる

MPI通信のチューニング

- 通信時間はデータサイズによって主要因が異なり、対応策が異なる
  - 小さいデータ：レイテンシ(遅延)時間 → 送受信**回数**削減
  - 大きなデータ(約1MB以上)：バンド幅 → 送受信**データ量**削減
- 小さいデータの場合：何度も送受信するより、配列にデータを集めて一度に送受信
  - 現在のMPI通信のレイテンシ1μ秒程度 (Core i7(4core, 3.1GHz)では10万回の演算に相当)



OpenMPの変数

- OpenMP並列領域では、すべての変数をノード内で共有する変数(shared)と各スレッドが別々の値を持つ変数(private)に分類
  - 既存のコードにOpenMPを導入する場合、privateにすべき変数の指定忘れによるバグに注意
  - デフォルトはshared
  - common、module変数はshared
  - do変数はprivate
  - OpenMP領域で呼ばれた関数やサブルーチン内で新規に使われる変数はprivate

```
 use module_A: cmsi
 real*8 : tcci
!$OMP parallel do
  do j = 1,n
    call test(j, tcci)
  enddo
…

Subroutine test(j, tcci)
 use module_A: cmsi
 integer : ims
```

OpenMPのオーバーヘッド

- !$OMP parallelや!$OMP do (特にschedule(dynamic))のオーバーヘッドは、OpenMP領域の計算量が少ないと無視できない → できるだけ多くの計算(領域)をまとめてOpenMP並列化
- 排他的処理criticalやatomicを多用すると、待ち時間が増加し効率が低下することが多い → できるだけ上書きをしないコードにする or スレッドごとに変数を用意(private, reduction)
- Common、module変数をprivate変数にする場合、threadprivateは便利だが、多用するとオーバーヘッドが無視できなくなることがある → common、module変数からサブルーチン、関数の引数に変更

高速化と並列化の重要性と難しさ

- ノードあたりのCPUコア数はますます増加すると予想される
- 高速化と並列化は、スパコンだけではなく研究室レベルのPCクラスタでも必須になりつつある
- スカラCPU搭載スパコンもPCクラスターも、開発方針、ソースコードはほぼ同じ
- どの程度力を入れるかは、目的によって異なる

- 分野によっては、式の導出からやり直す必要がある
- 京コンピュータだけのチューニングは多くない
- 既存のコードの改良では性能を出すのが難しい場合がある

原子軌道2電子積分計算アルゴリズム

$$(\mu\nu|\lambda\sigma) = \int d\boldsymbol{r}_1 \int d\boldsymbol{r}_2 \phi_\mu{}^*(\boldsymbol{r}_1)\phi_\nu(\boldsymbol{r}_1)\frac{1}{r_{12}}\phi_\lambda{}^*(\boldsymbol{r}_2)\phi_\sigma(\boldsymbol{r}_2)$$

$\phi_\mu(\boldsymbol{r}_1)$: 原子軌道Gauss関数

- Rys quadrature (Rys多項式を利用)
- Pople-Hehre (座標軸を回転させ演算量削減)
- McMurchie-Davidson (Hermite Gaussianを利用した漸化式)
- Obara-Saika (垂直漸化関係式(VRR))
- Head-Gordon-Pople (水平漸化関係式(HRR)+VRR)
- ACE (随伴座標展開)
- PRISM(適切なタイミングで短縮(contraction)を行う)

2電子積分計算のコスト

$$(\mu\nu|\lambda\sigma) = \int d\mathbf{r}_1 \int d\mathbf{r}_2 \phi_\mu{}^*(\mathbf{r}_1)\phi_\nu(\mathbf{r}_1)\frac{1}{r_{12}}\phi_\lambda{}^*(\mathbf{r}_2)\phi_\sigma(\mathbf{r}_2)$$

基底関数 $\phi_\mu(\mathbf{r}) = \sum_i^K c_i(x - A_x)^{n_x}(y - A_y)^{n_y}(z - A_z)^{n_z}\exp(-\alpha_i|\mathbf{r} - \mathbf{A}|^2)$

$$\exp(-\alpha_i|\mathbf{r} - \mathbf{A}|^2)\exp(-\alpha_j|\mathbf{r} - \mathbf{B}|^2) = \exp\left(-\frac{\alpha_i\alpha_j}{\alpha_i + \alpha_j}|\mathbf{A} - \mathbf{B}|^2\right)\exp(-(\alpha_i + \alpha_j)|\mathbf{r} - \mathbf{P}|^2)$$

2電子積分の演算量=xK^4+yK^2+z
K(基底関数の短縮数)に依存している

$(sp,sp|sp,sp)$の計算コスト

Method	PH	MD	HGP	DRK
x	220	1500	1400	1056
y	2300	1700	30	30
z	4000	0	800	800

新規アルゴリズム開発例

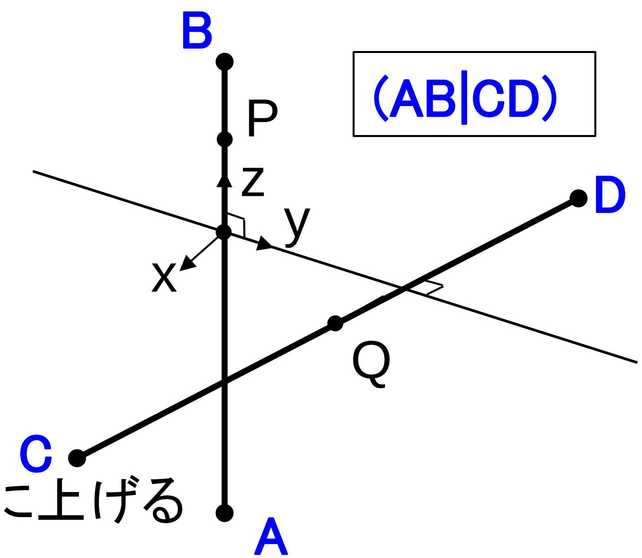
K. Ishimura, S. Nagase, Theoret Chem Acc, (2008) 120, 185-189.

- **Pople-Hehre法**
  座標軸を回転させることにより演算量を減らす

  $x_{AB}=0$　　　$x_{PQ}=$一定

  $y_{AB}=0$　　　$y_{PQ}=$一定

  $y_{CD}=0$



- **McMurchie-Davidson法**
  漸化式を用いて(ss|ss)型から角運動量を効率的に上げる

  $[0]^{(m)}(=(ss|ss)^{(m)})$-> (r) -> (p|q) -> (AB|CD)

2つの方法の組み合わせ

座標軸回転 → 漸化式を用いて角運動量を上げる → 座標軸再回転

アルゴリズムの特徴

$[\mathbf{0}]^{(m)}(=(ss|ss)^{(m)}) \longrightarrow [A+B+C+D] \longrightarrow [A+B|C+D] \longrightarrow [\mathbf{AB}|\mathbf{CD}]$

x_{PQ}=一定
y_{PQ}=一定

$x_{AB}=0$
$y_{AB}=0$
$y_{CD}=0$

- 演算量$=xK^4+yK^2+z$ (K:基底関数の短縮数)

  (sp,sp|sp,sp)型の場合

Method	PH	PH+MD
x	220	180
y	2300	1100
z	4000	5330

Method	PH	PH+MD
K=1	6520	6583
K=2	16720	12490
K=3	42520	29535

6-31G(d)やcc-pVDZなど適度な短縮数の基底関数で性能発揮

プログラム開発

- (ss|ss)から(dd|dd)までの21種類の積分計算ルーチンを作成
- $x_{AB}=0$, $y_{AB}=0$などを考慮した漸化式を再導出、その計算コードを作るプログラムをFortranとPerlで作成
- 文字列をFortranで出力し、Perlで整形
- 約2万行のコードを自動生成し、デバッグを含めた開発時間を短縮
- 自動生成以外のコードは、基本的にdoループで書けるよう配列データの並びを工夫し、さらに割り算やsqrtなど組み込み関数はできるだけまとめて演算
- 詳細は、GAMESSのint2b.src (sp関数), int2[r–w].src (spd関数)を参照

計算結果

GAMESSに実装して、Fock行列計算時間(sec)を測定
計算機：Pentimu4 3.0GHz

分子	Taxol($C_{47}H_{51}NO_{14}$)		Luciferin($C_{11}H_8N_2O_3S_2$)
基底関数	STO-3G (361 AOs)	6-31G(d) (1032 AOs)	aug-cc-pVDZ (550 AOs)
Original GAMESS (PH)	85.7	2015.2	2014.9
PH+MD	**69.9**	**1361.8**	**1154.5**

- 2－4割計算時間を削減
- 2005年からGAMESSにデフォルトルーチンとして正式導入
- 演算の約8割はdoループ内で行われるため、現在のCPUに適した方法
- 座標軸を元に戻す変換行列に6dから5dへの変換を組み込むことが可能（GAMESSには未実装）

まとめ

- 分子科学分野では30年以上前からスパコンを使い、それぞれの時代で可能な大規模計算を行ってきた
- 量子化学計算は、系のサイズが大きくなると計算量、データ量共に急激に増加する
- 高速化・並列化はスパコンだけではなく、研究室レベルのPCクラスタでも不可欠になりつつある
- プログラミングだけではなく、演算量の削減、収束回数の削減も重要である