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

2015年度計算科学技術特論A 第14回
2015年7月16日

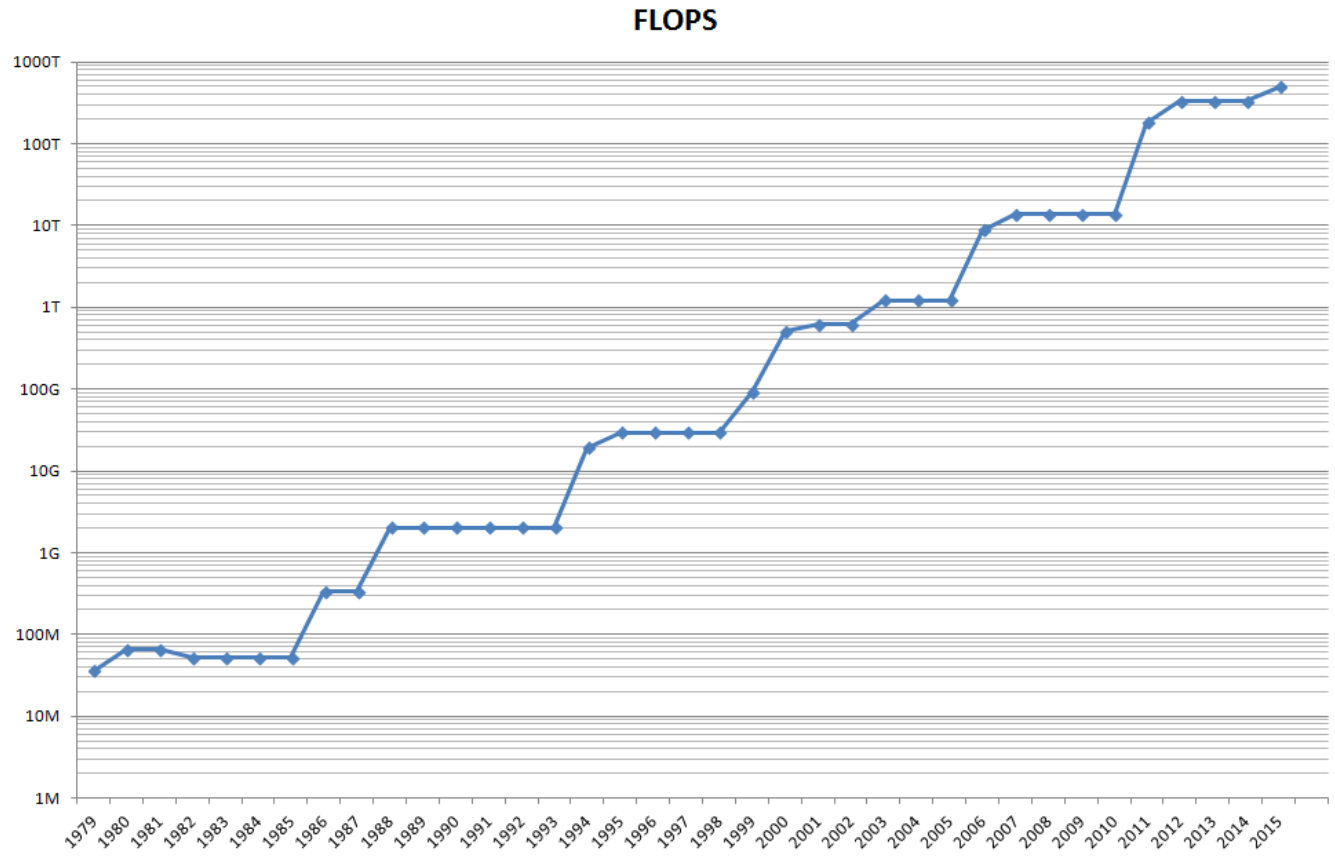
講義の内容

- 本日(第14回)
  - 分子科学分野におけるスパコン利用の歴史
  - 量子化学計算とは
  - 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(第15回)
  - 高速化・並列化例
  - 新たなオープンソースソフトウェアの開発
  - メニーコア時代に向けた開発

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/

演算性能はおおよそ10年で100倍のペースで向上



年	理論総演算性能 (GFLOPS)
1979	0.036
1989	2
1999	92
2009	13,606
2015	492,529

分子科学分野スパコンの変遷2

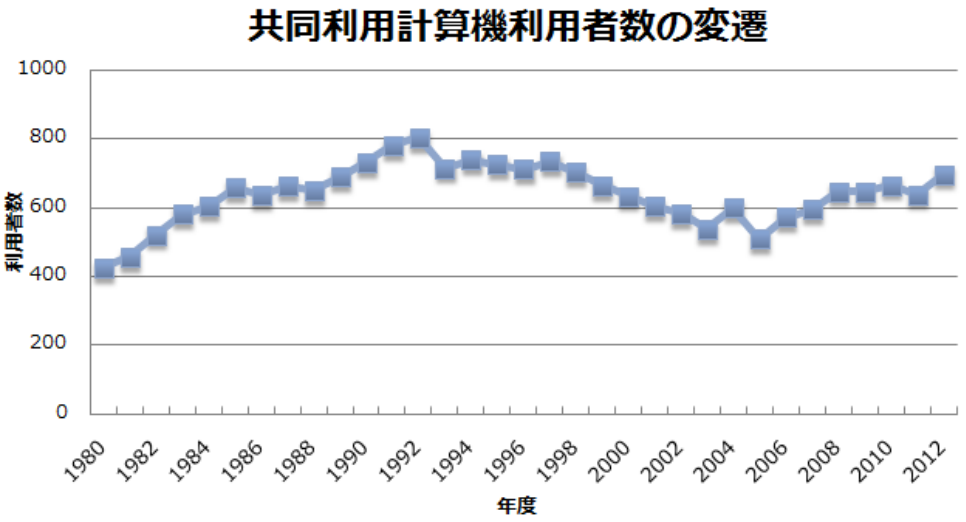
- 自然科学研究機構 岡崎共通研究施設 計算科学研究センター (旧分子科学研究所 電子計算機センター)におけるCPU能力の変遷　https://ccportal.ims.ac.jp/

- 並列計算は必要不可欠で、大規模並列計算も当たり前の時代になりつつある
- 2020年代前半には、京コンピュータ並のマシンが導入されている可能性が高い

年	機種	理論総演算性能 (GFLOPS)	
1979	HITACHI M-180 (2台)	0.036	
2000	IBM SP2 (Wide24 台)	7	
	IBM SP2 (Thin24 台)	3	
	NEC SX-5 (8CPU)	64	
	Fujitsu VPP5000 (30PE)	288	
	SGI SGI2800 (256CPU)	153	
	合計	514	←Core i7　5775C (3.3GHz,4コア) 2.5台分
2015	Fujitsu PRIMERGY RX300S7 (5472core)	126,950	
	(+ NVIDIA Tesla M2090 32台)	21,280	
	Fujitsu PRIMEHPC FX10 (1536core)	20,152	
	SGI UV2000 (1024core)	21,299	
	Fujitsu PRIMERGY CX2550M1 (7280core)	302,848	
	合計	492,529	← 京コンピュータの20分の1
	地球シミュレータ(2002)	35,860	
	京コンピュータ(2011)	10,510,000	

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
  - 2015年時点では、クラスタ専有利用枠は最大4096コア、7日間x12回



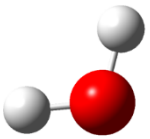
元素と分子

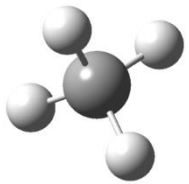
周期表

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
- 現在9000万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**

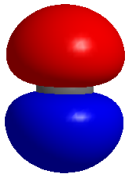


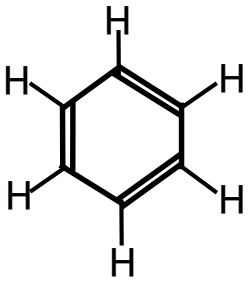


量子化学計算とは

- 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

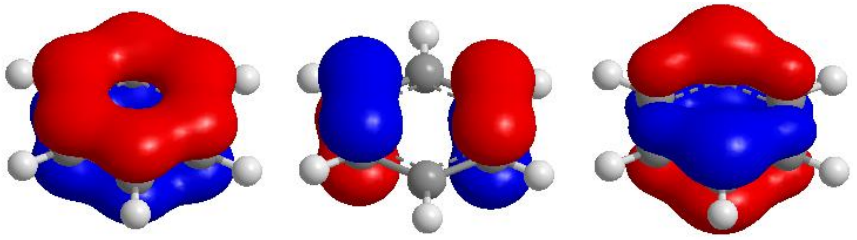
原子軌道（C,H）
原子座標





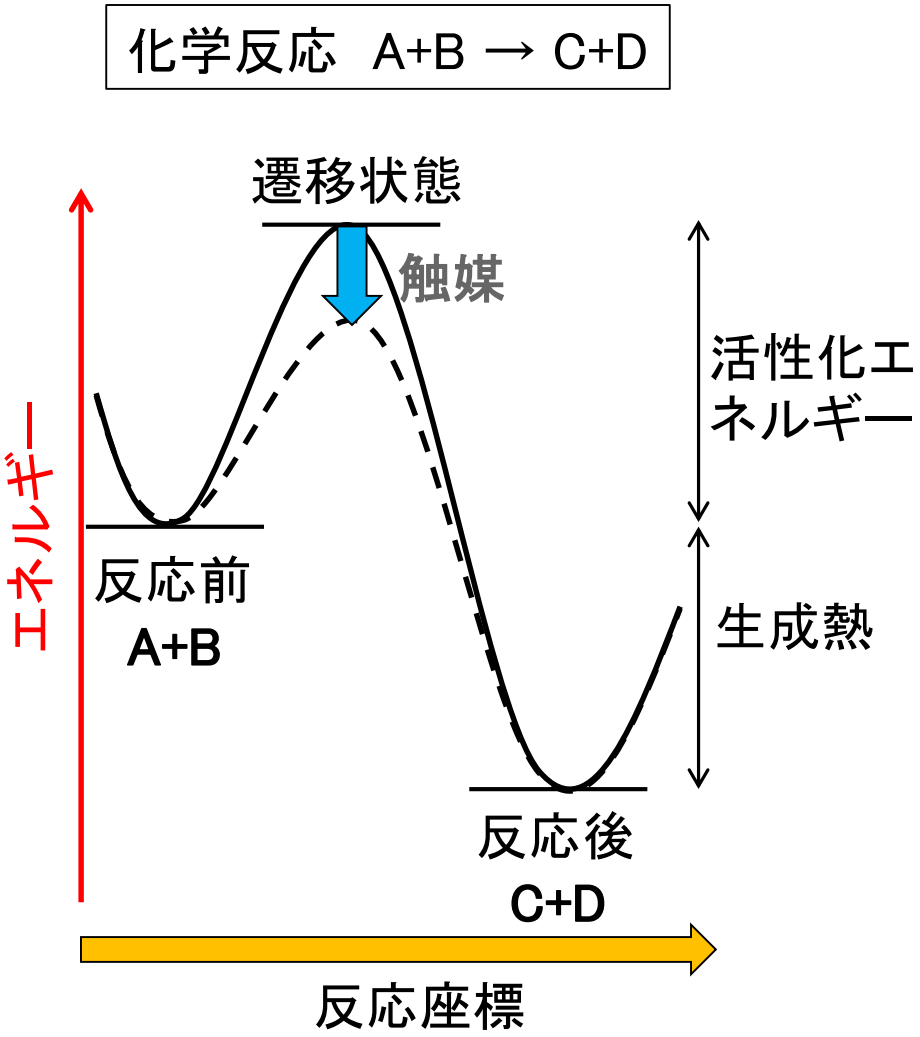
Hartree-Fock計算

分子軌道（ベンゼン(C_6H_6)）

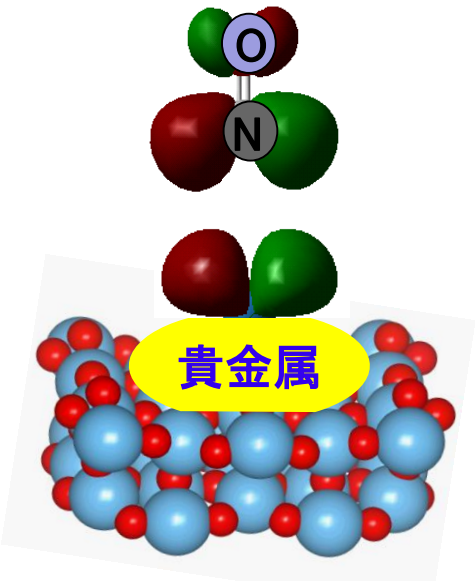


量子化学計算で得られるもの

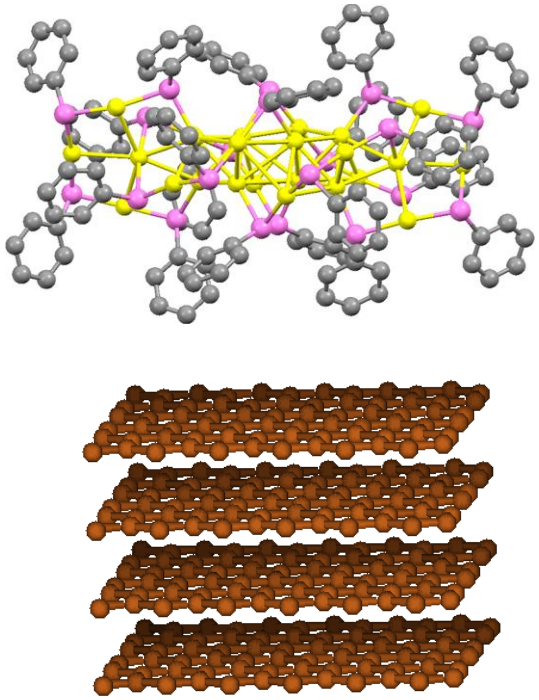
- 分子のエネルギー
- 安定構造、遷移状態構造
- 化学反応エネルギー
- 光吸収、発光スペクトル
- 振動スペクトル
- NMR(核磁気共鳴)スペクトル
- 各原子の電荷
- 溶媒効果
- 結合軌道解析など



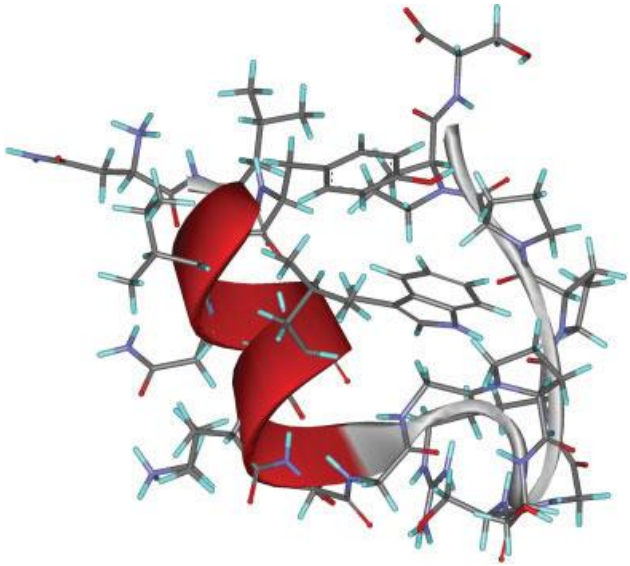
量子化学計算適用例



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・クラスターの計算(反応性、安定性)が可能になり、取り扱える対象が大幅に拡大



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

量子化学計算方法とコスト

- 演算内容から分類

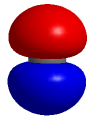
N: 基底(or電子)数

計算方法	演算量	データ量	通信量
Hartree-Fock(SCF), DFT法 2電子積分計算(キャッシュ内演算) 密対称行列の対角化	$O(N^4)$ (カットオフで$O(N^3)$程度)	$O(N^2)$	$O(N^2)$
摂動(MP2,MP3,..)法,結合クラスター(CCSD, CCSD(T),..)法 密行列-行列積	$O(N^5\sim)$	$O(N^4\sim)$	$O(N^4\sim)$
配置間相互作用(CIS, CISD,...)法 疎行列の対角化	$O(N^5\sim)$	$O(N^4\sim)$	$O(N^4\sim)$

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

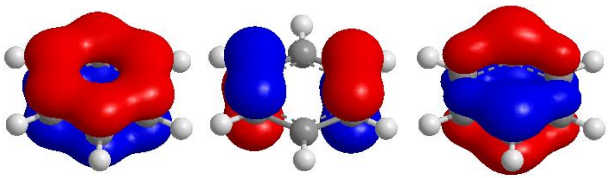
- 複素数演算: 重原子を含むときは相対論効果が重要になる。複素数の演算が必要になる場合がある。
- 高精度演算 : 将来の巨大系において、倍精度では有効桁数が不足する可能性あり。ポスト京の次では4倍精度が必要になるかも。

Hartree-Fock法



原子軌道Gauss関数

Hartree-Fock計算



原子軌道の線形結合係数(分子軌道係数)を求める

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

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 $F_{\mu\nu} = H_{\mu\nu} + \sum_{i,\lambda,\sigma} C_{\lambda i} C_{\sigma i} \{2(\mu\nu|\lambda\sigma) - (\mu\lambda|\nu\sigma)\}$

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(N^4))

↓

Fock行列対角化 (O(N^3))

分子軌道収束せず

↓ 分子軌道C収束

計算終了

2次の摂動(MP2)法

- Hartree-Fock計算で分子のエネルギーの約99%を求めることができるが、定量的な議論を行うためには残り1%の電子相関エネルギーが重要
- MP2法は最も簡便な電子相関計算方法
- 積分変換(密行列-行列積)計算が中心

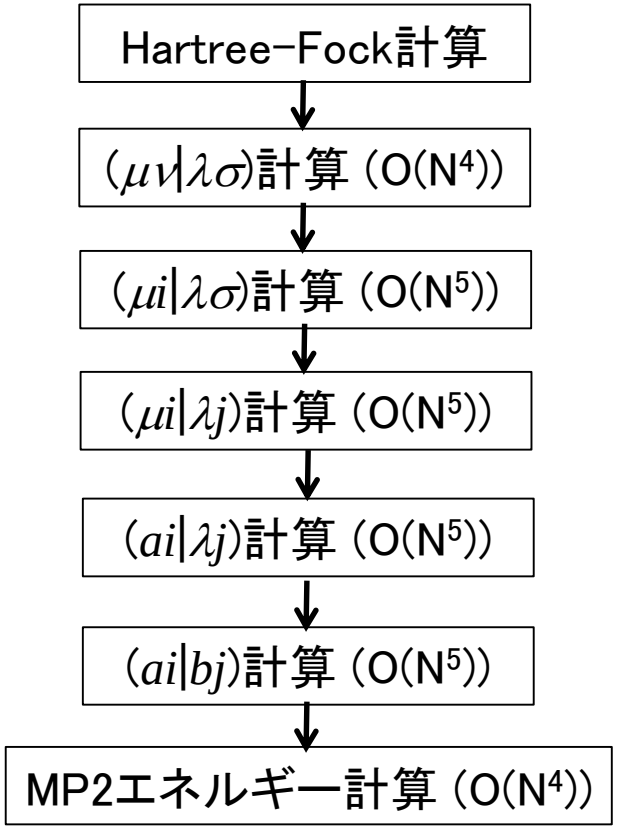
$$E_{MP2} = \sum_{ij}^{occ}\sum_{ab}^{vir} \frac{(ai|bj)\{2(ai|bj) - (aj|bi)\}}{\varepsilon_i + \varepsilon_j - \varepsilon_a - \varepsilon_b}$$

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

分子軌道(MO) 2電子積分　　原子軌道(AO) 2電子積分

ε_i: 軌道エネルギー, $C_{\mu a}$: 分子軌道係数

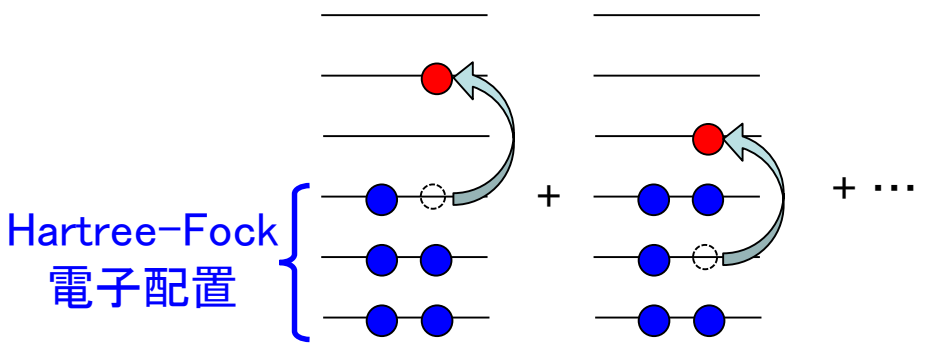
例)・ C_{60}(6-31G(d))の場合：O:120, V:720, N:840
・データ量$1/2O^2V^2 \times 8$Byte = 25GB
・Xeon E5-2698(32コア, 2.3GHz)1ノード,
SMASH program,対称性無しで約771秒



配置間相互作用(CI)法

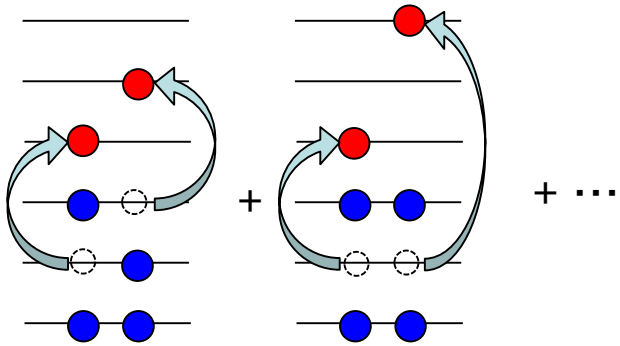
- 主な計算内容は、疎行列の行列要素とその対角化
- CIS法：1電子励起配置の線形結合

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



- CISD法：HF,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:120, V:720, N:840
CISの配置の数：$O \times V = 1.0 \times 10^5$
CISDの配置の数：$O^2 \times V^2 = 1.0 \times 10^{10}$
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

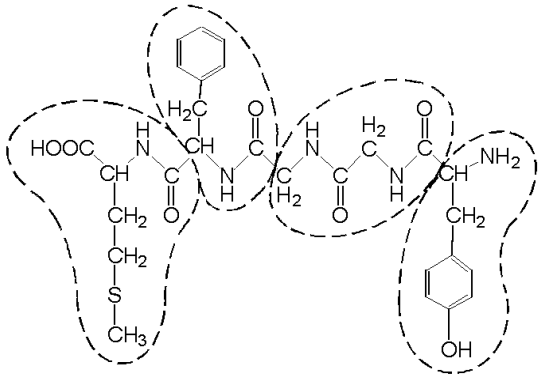
K. Kitaura, E. Ikeo, T. Asada, T. Nakano, and M. Uebayasi, Chem. Phys. Lett., 313 (1999) 701–706.

  - 巨大な分子(例えばタンパク質)をフラグメント(アミノ酸1残基もしくは複数残基)ごとに分割して、1量体と2量体のエネルギーから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

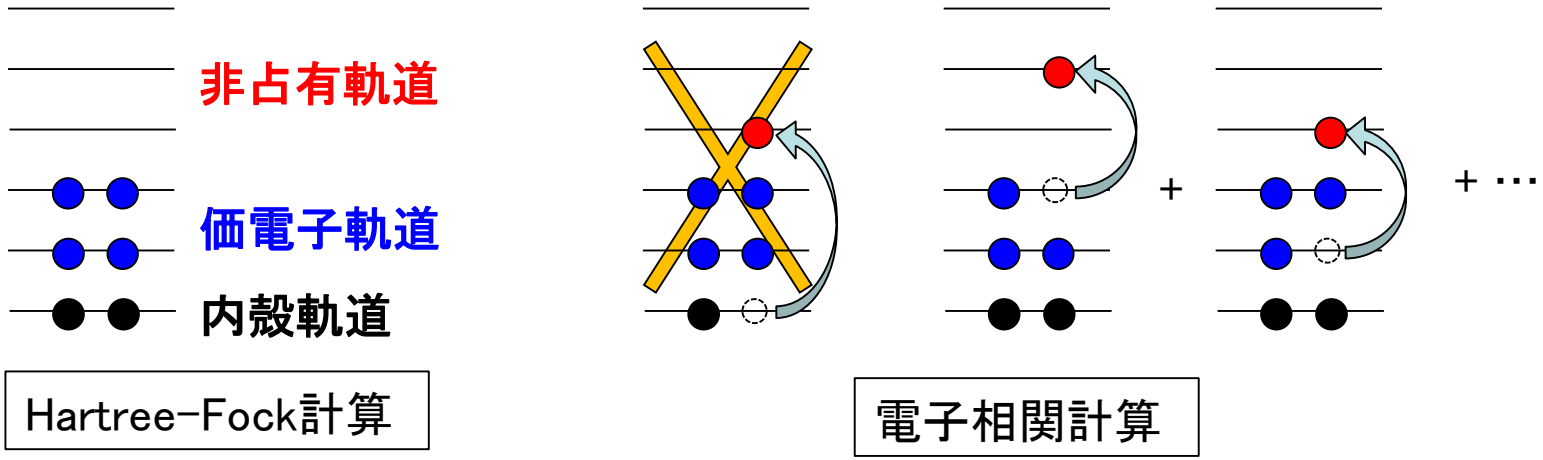
E_{IJ}: I番目とJ番目の2量体エネルギー



  - 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

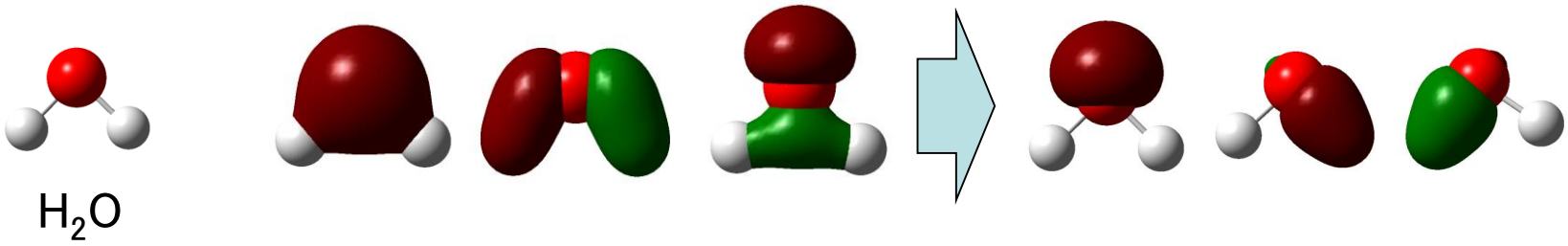
近似の導入2

- 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



- 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することで計算量やデータ量を削減
- Fast Multipole法(FMM)など他にも様々な方法がある

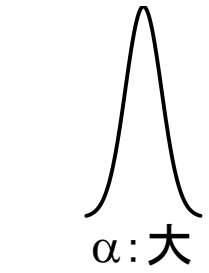
高速化1(演算量の削減)

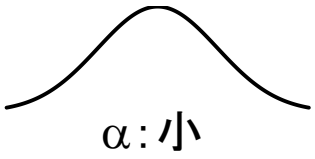
- 演算量の少ない1,2電子積分計算アルゴリズム開発
- Schwarz inequalityを用いたカットオフ
  - AO2電子積分$(\mu\nu|\lambda\sigma)$計算実行を基底シェルごとに判断
  - 内殻電子の基底$\exp(-\alpha r^2)$のαが大きい場合、異なる原子間では重なりが小さくなり、大半がスキップされる
  - 密度行列 $(D_{\mu\nu} = 2\sum_i^{occ} C_{\mu i}C_{\nu i})$の差も掛けた上で判定すると、収束に近づくにつれてより多くの計算がスキップされる

Fock行列作成4重ループ

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

$\exp(-\alpha r^2)$





高速化2(演算量の削減)

- 対称性の利用
  - 基底が実数の場合のAO2電子積分: $(\mu\nu|\lambda\sigma) = (\mu\nu|\sigma\lambda) = (\lambda\sigma|\mu\nu) = (\lambda\sigma|\nu\mu) = \cdots$
  - $\mu \Leftrightarrow \nu$、$\lambda \Leftrightarrow \sigma$、$\mu\nu \Leftrightarrow \lambda\sigma$の入れ替えが可能で、計算量は1/8に

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

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

  - 具体的には、Fock行列計算において、$(\mu\nu|\lambda\sigma)$1つで複数のAO2電子積分をカバー

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

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

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

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

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

$$\cdots$$

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 (分子の結合、角度、二面角の情報から初期Hessianを改良)

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
  - キャッシュ内のデータを再利用し、メモリアクセスを削減
  - 多次元配列の取り扱い 例 : A(x, y, z) or A(z, y, x)

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、わかりやすいシンプルなコードを書く必要がある
  - データの依存関係や多重ループ内にIF文があるなど複雑になると、最適化されない場合が多い
- 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

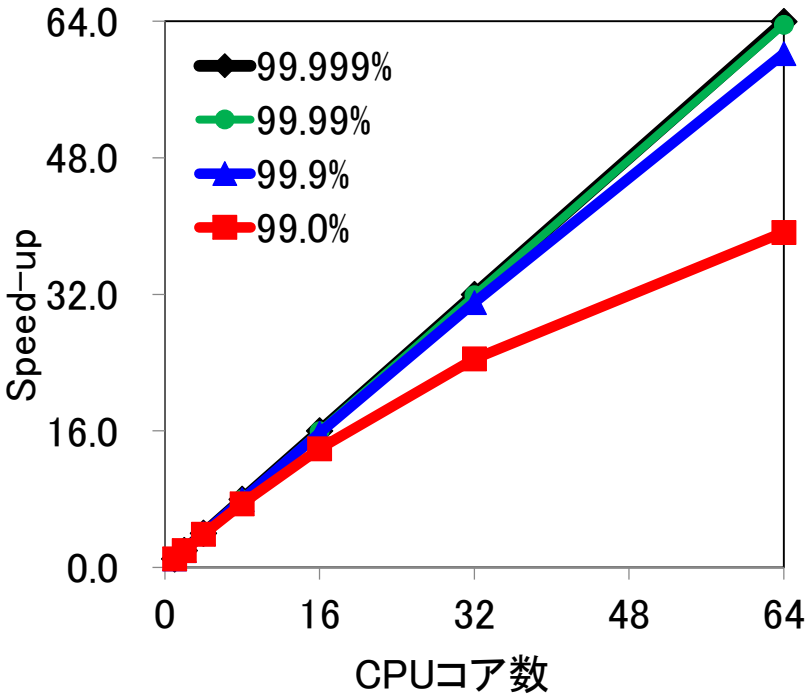
- ノード間(MPI)、ノード内(OpenMP)それぞれで並列化
  - 式の変形
  - 多重ループの順番の工夫
- 均等な計算負荷分散
  - ノード間で分散、さらにノード内でも分散
- 大容量データの分散
  - 京のメモリは1ノード16GB, 8万ノードでは1PB以上
  - ノード間では分散、ノード内では共有
  - 中間データ保存用としてディスクはあまり期待できない
- 通信量、通信回数の削減
  - 並列計算プログラムのチューニングで最後に残る問題は通信関係(特にレイテンシ)が多い

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

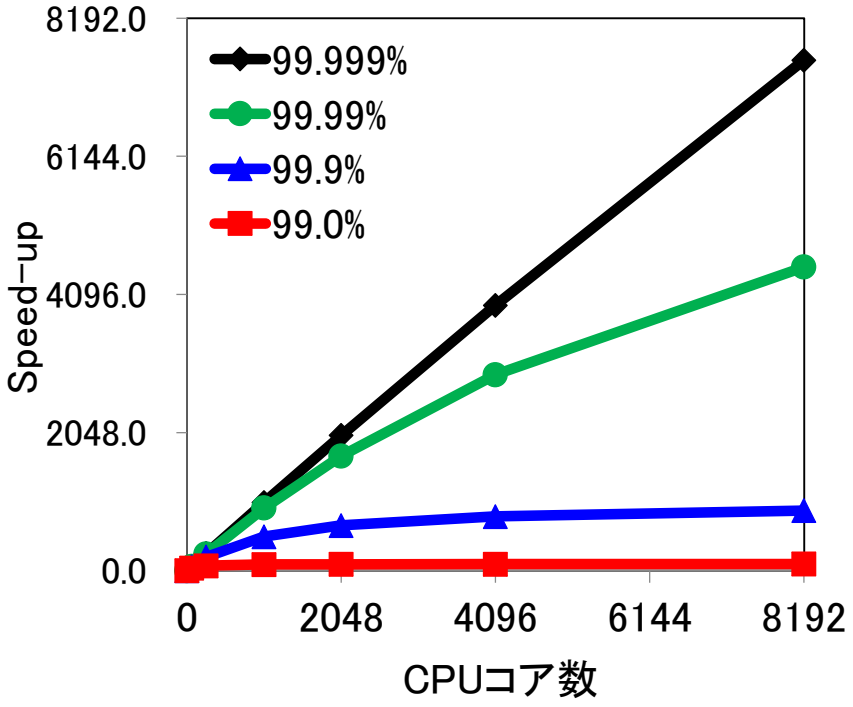
数十コア(PCクラスター)

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

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
今後ノードあたりのコア数はさらに増えると予想され、ますます重要になる

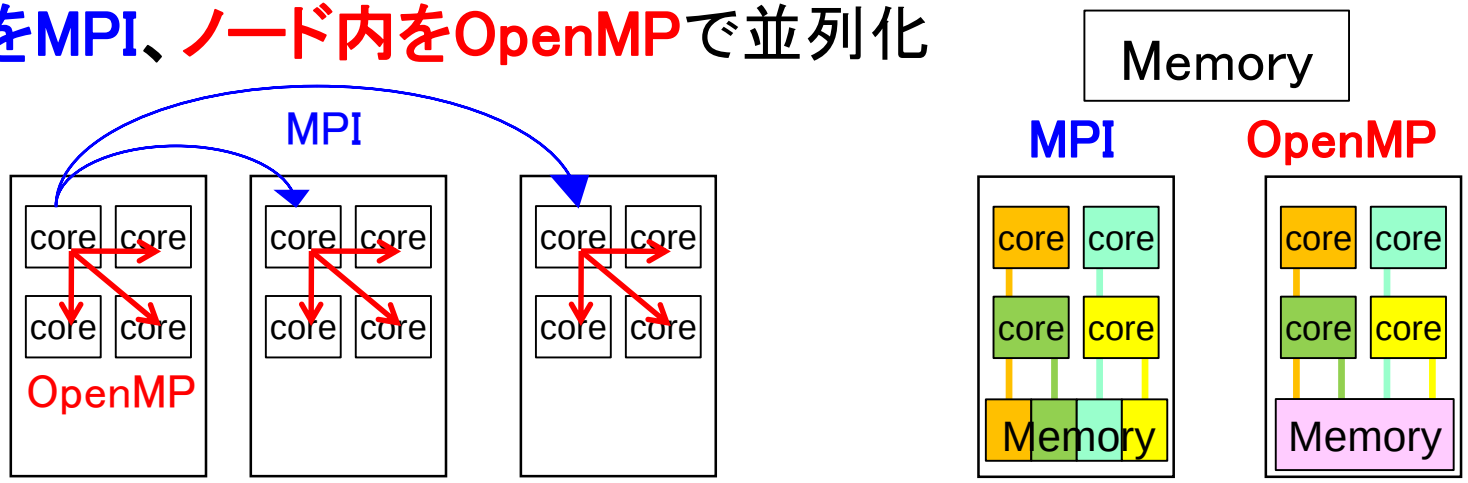
MPI(ノード間、ノード内)
使い始めは大変だが、性能を出すのはOpenMPよりも容易

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

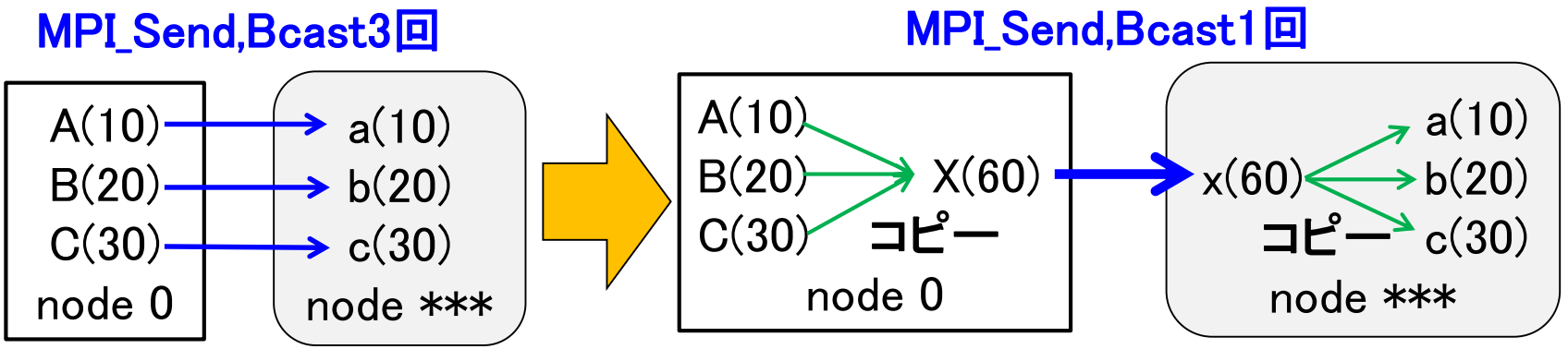
メモリの有効利用　**OpenMPによるノード内メモリの共有**

デメリット

アルゴリズム、プログラムが複雑になり、開発コスト上昇

MPI通信の最適化

- 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通信のレイテンシは0.1μ秒以上 (Xeon E5-2698(32コア, 2.3GHz)では約10万回の演算に相当)



OpenMPの変数(Fortran)

- 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))のコストは、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
  - → 多重ループの分散では、外側のループを並列化
- 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
  - → できるだけ上書きをしないコードにする
  - → 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、OpenMP領域で頻繁に呼ばれる関数やサブルーチンで利用するとそのコストが大きくなる場合がある
  - → 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
- 京コンピュータはスカラCPUであるため、京だけのチューニングは多くない
- 既存のソースコードの改良では性能を出すのが難しい場合がある
- 開発コストは増える一方なので、オープンソースでの公開と共有で分野全体でのコスト削減を考える段階に来ている

原子軌道(AO)2電子積分計算アルゴリズム

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

$\phi_\mu(\boldsymbol{r})$: 基底関数(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)
- Obara-Saika (垂直漸化関係式(VRR))
- McMurchie-Davidson (Hermite Gaussianを利用した漸化式)
- Head-Gordon-Pople (水平漸化関係式(HRR))
- ACE (随伴座標展開)
- PRISM(適切なタイミングで短縮(contraction)を行う)
- 他にも数多くのアルゴリズムが提案されている

AO2電子積分計算のコスト

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

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

水素のSTO-3G
基底関数(K = 3)

```
H
S   3
  3.42525091    0.15432897
  0.62391373    0.53532814
  0.16885540    0.44463454
```

↑ α_i　　↑ c_i

2電子積分の演算量=xK^4+yK^2+z

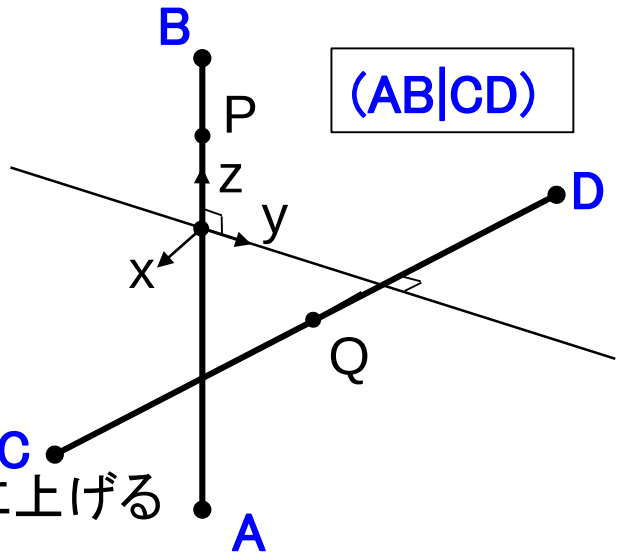
K(基底関数の短縮数)に依存している
(*sp*,*sp*|*sp*,*sp*)の計算コスト

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

高速化例(新規アルゴリズム開発)

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

- **Pople–Hehre (PH)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
  $y_{CD}=0$

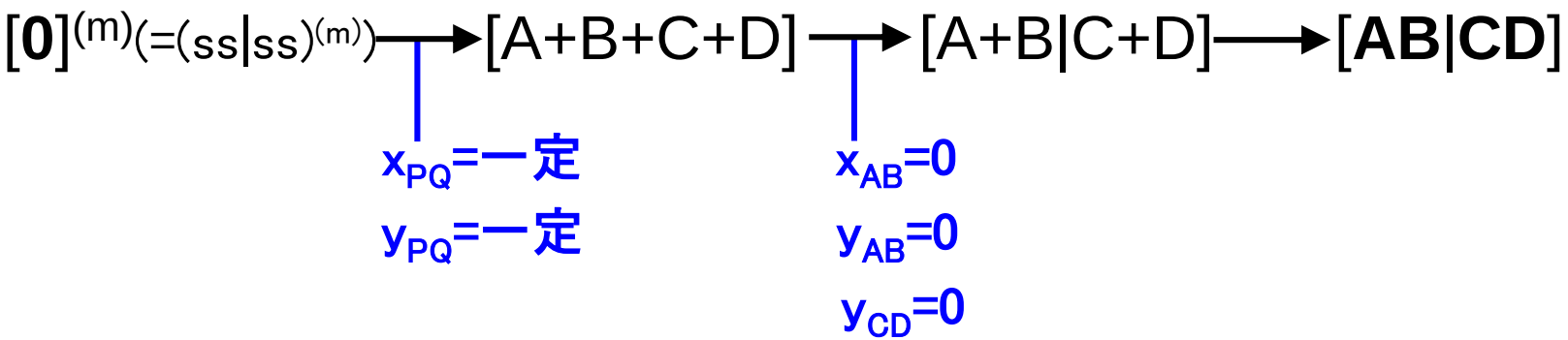


- **McMurchie–Davidson (MD)法**
  漸化式を用いて(ss|ss)型から角運動量を効率的に上げる
  $[0]^{(m)}(=(ss|ss)^{(m)})$–> (r) –> (p|q) –> (AB|CD)

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

新規アルゴリズムの特徴



- **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
- 座標回転のメリット(x_{AB}=0, y_{AB}=0など)を考慮した漸化式を導出、その計算コードを作る自動生成プログラムを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、expなど組み込み関数はできるだけまとめて演算
- 詳細は、SMASHプログラムのint2sp.F90, int2spd[1-4].F90、もしくはGAMESSプログラムのint2b.src (sp関数), int2[r-w].src (spd関数)を参照
- SMASHでは2電子積分ルーチンをライブラリ化し、引数でのみデータのやり取りを行っているため、容易に他のプログラムへ移植することができる

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
- d関数を含む場合、削減の割合は大きくなる
- 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

まとめ

- 分子科学分野では35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、分子のサイズが大きくなると計算量、データ量、通信量が急激に増加する
- 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
- 京コンピュータレベルの計算機が10年以内に全国のスパコンセンターに導入される可能性が高い
- 計算機はますます複雑になり(CPU内部もコア数も)、アルゴリズムとプログラム開発はさらに大変になるため、分野全体での開発コスト削減を考える必要がある
- 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、適切な近似の導入も重要である