量子化学計算の大規模化2

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15回
2013年7月25日

本日の内容

- 原子軌道2電子積分計算の高速化
- 高速化・並列化の復習
- Hartree-Fock計算並列化
- MP2エネルギー計算並列化
- MP2エネルギー微分計算並列化
- 今後のプログラムについて

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

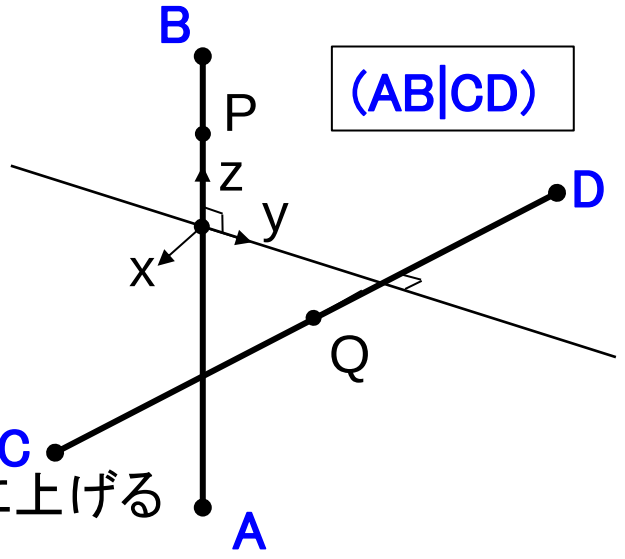
(*sp*,*sp*|*sp*,*sp*)の計算コスト

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
  $y_{CD}=0$

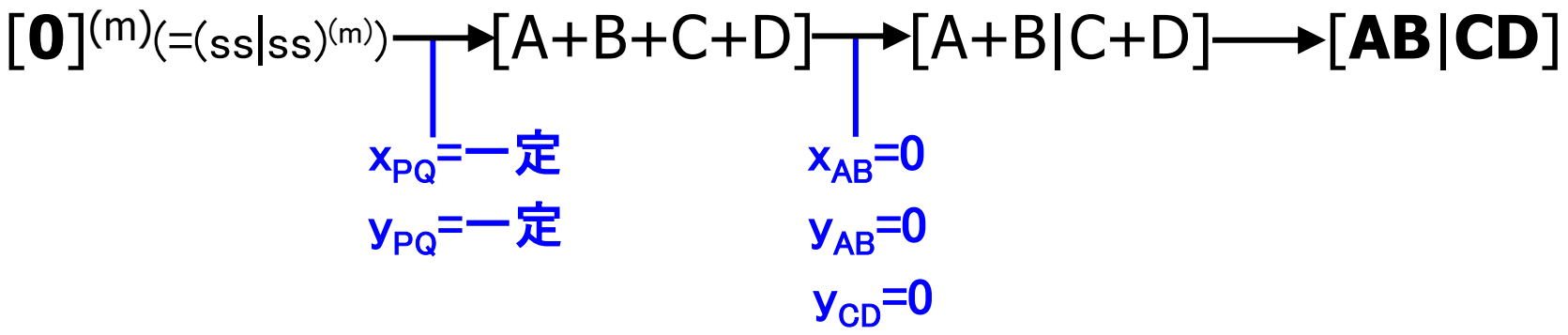


- **McMurchie–Davidson法**
  漸化式を用いて(ss|ss)型から角運動量を効率的に上げる
  $[0]^{(m)}(=(ss|ss)^{(m)})$–> (r) –> (p|q) –> (AB|CD)

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



- 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 (STO-3G)	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の導出、そのソース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**

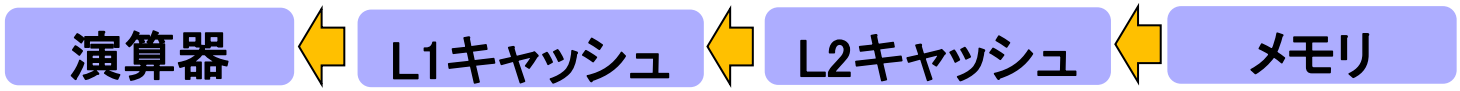
- 2 - 4割計算時間を削減
- 2005年からGAMESSにデフォルトルーチンとして正式導入
- 演算の約8割はdoループ内で行われるため、現在のCPUに適した方法
- 座標軸を元に戻す変換行列に6dから5dへの変換を組み込むことが可能(GAMESSには未実装)

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

高速化1（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）
- データアクセスを考慮したアルゴリズム・プログラム開発
  - キャッシュミスの削減など
  - 多次元配列の取り扱い 例：A(ix, iy, iz) or A(iz, iy, ix)

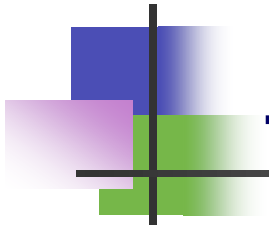


相対的な速度　100　10　1

高速化2（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1

- 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2(MPI)

- MPI通信時間は、データサイズによって原因が異なり、対応策も異なる
  - 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
- チューニング、デバッグはOpenMPよりMPIの方が楽な場合が多い(原因を特定しやすい)
  - 計算負荷分散は、MPIランクを用いた部分のみ
  - データが書き換わるのは、MPIルーチンでの送受信のみ

並列化3(OpenMP)

- !$OMP parallelや!$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書き換え
  - 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

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	$O(N^5 \sim)$	$O(N^4 \sim)$

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

Hartree-Fock計算

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

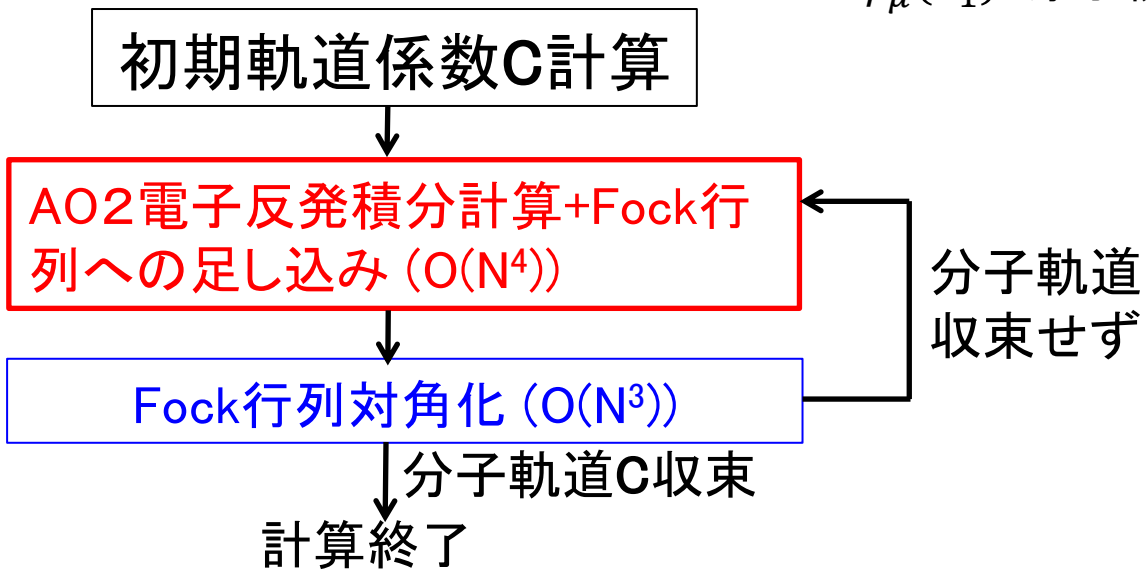
F: Fock行列，C: 分子軌道係数
S: 基底重なり行列，ε: 分子軌道エネルギー

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



MPI/OpenMP並列アルゴリズム1

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

```
!$OMP parallel do schedule(dynamic,1) reduction(+:Fock)
do μ=n, 1, -1                              <----- OpenMPによる振り分け
  do ν=1, μ
    μν=μ*(μ+1)/2+ν
    λstart=mod(μν+mpi_rank,nproc)+1
    do λ=λstart, μ ,nproc                  <----- MPIランクによる振り分け
      do σ=1, λ
        AO2電子積分(μν|λσ)計算+Fock行列に足し込み
      enddo
    enddo
  enddo
enddo
!$OMP end parallel do
call mpi_allreduce(Fock)
```

MPI/OpenMP並列アルゴリズム2

- GAMESSプログラムを基に実装
- 1番目のループをOpenMP、3番目のループをMPIで並列化
  - MPIランクはプロセスに固有の値なので、MPIランクによるプロセス間の分散箇所が決まれば、OpenMP並列が後でも先でも、各プロセスに割り当てられる仕事量は変わらない
- MPIランクとmod計算だけで、IF文を使わずにノード間分散
  - MPIランクによる分散までの演算はすべてのプロセスが実行
  - ノード数が非常に多くなるとIF文のコストが無視できなくなる
- MPI通信はOpenMP領域外で実行

MPI/OpenMP並列アルゴリズム3

- OpenMPの分散を最外ループで行うことにより、スレッド生成などのオーバーヘッドを削減
- 演算量の多いインデックスから動的に割り振ることでノード内の負荷分散を均等化
- すべての変数について、計算機内で共有するか(shared)、別々の値を持つか(private) を分類
  - privateにすべきcommon変数は、threadprivateを使わずにサブルーチンの引数に変更
  - 引数の数を減らすため、x、y、zなどのスカラ変数をxyz配列に書き換え

初期軌道計算などの高速化・並列化

- 初期軌道計算のハイブリッド並列化
- 軌道の射影の高速化・並列化

$$\mathbf{C}_1 = \mathbf{S}_{11}^{-1}\mathbf{S}_{12}\mathbf{C}_2\left[\mathbf{C}_2^t\mathbf{S}_{12}^t\mathbf{S}_{11}^{-1}\mathbf{S}_{12}\mathbf{C}_2\right]^{-1/2}$$

行列積の多用で高速化・並列化

$\mathbf{C}_1$: 初期軌道係数
$\mathbf{C}_2$: 拡張Huckel法による軌道係数
$\mathbf{S}_{11}$: 実際の計算で用いる基底の重なり積分
$\mathbf{S}_{12}$: 拡張Huckel法で用いた基底と実際の計算で用いる基底との重なり積分

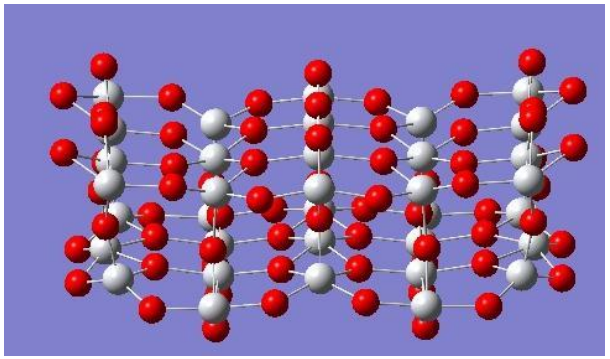
D. Cremer and J. Gauss, *J. Comput. Chem.* **7** 274 (1986).

- SCF計算の途中は対角化をせずに、Newton-Raphsonを基にしたSecond-Order SCF法を採用

Hartree-Fock計算条件

計算機: Cray XT5 2048 CPUコア
(Opteron 2.4GHz, Shanghai, 8cores/node)

コンパイラ: PGI fortran compiler-8.0.2
BLAS,LAPACKライブラリ: XT-Libsci-10.3.3.5
MPIライブラリ: XT-mpt-3.1.2.1 (MPICH2-1.0.6p1)
プログラム: GAMESS
分子: TiO2クラスター($Ti_{35}O_{70}$)
(6-31G, 1645 functions, 30 SCF cycles)



TiO_2クラスター計算結果

全計算時間の
並列加速率

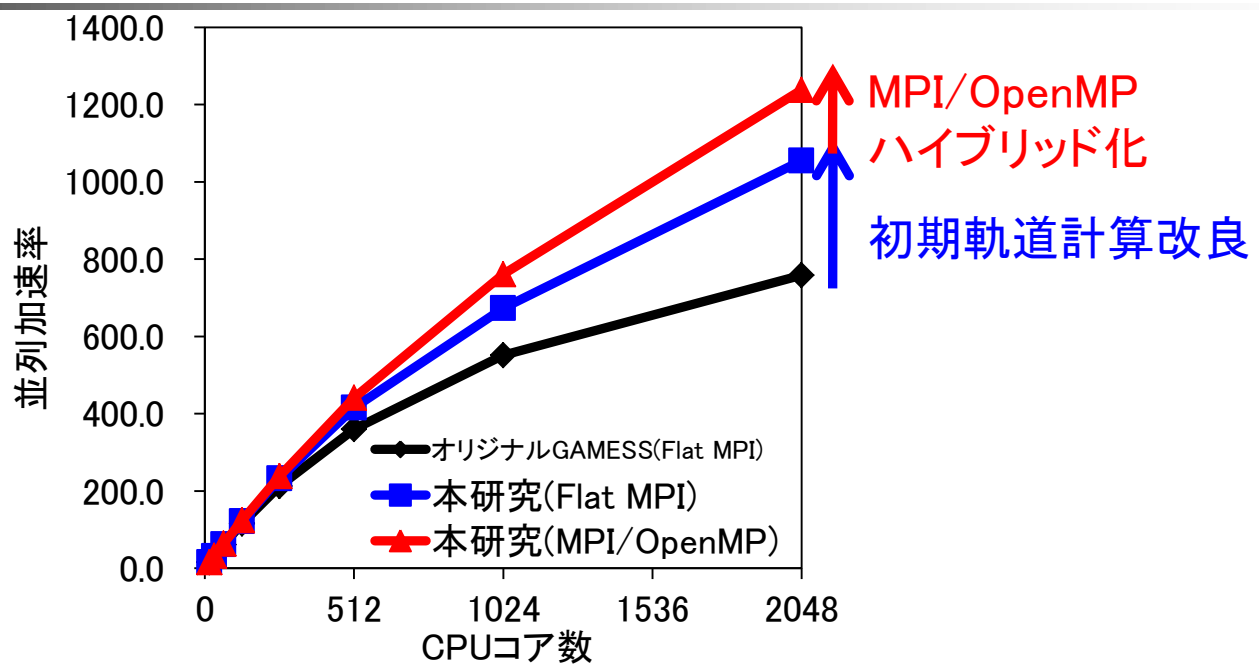


Table 全計算時間(秒)と並列加速率(カッコ内)

CPUコア数		16	256	1024	2048
オリジナル GAMESS	MPIのみ	18176.4 (16.0)	1368.6 (212.5)	527.6 (551.2)	383.5 **(758.3)**
改良版	MPIのみ	18045.6 (16.0)	1241.2 (232.6)	428.7 (673.5)	273.7 **(1054.9)**
改良版	MPI/OpenMP	18121.6 (16.0)	1214.6 (238.7)	381.1 (760.8)	234.2 **(1238.0)**

TiO_2クラスター計算の解析

Table Fock行列計算時間(秒)と並列加速率(カッコ内)

CPUコア数		16	256	1024	2048
オリジナル GAMESS	MPIのみ	17881.8 (16.0)	1175.2 (243.5)	334.0 (856.6)	188.6 (1517.0)
改良版	MPIのみ	17953.5 (16.0)	1175.2 (244.4)	360.0 (797.9)	203.1 (1414.4)
改良版	MPI/OpenMP	17777.6 (16.0)	1150.4 (247.3)	316.4 (899.0)	174.8 (1627.2)

Table初期軌道計算時間(秒)と全計算に占める割合(カッコ内)

CPUコア数		16	256	1024	2048
オリジナル GAMESS	MPIのみ	166.2 (0.9%)	143.6 (10.5%)	143.6 (27.2%)	143.8 **(37.5%)**
改良版	MPIのみ	20.2 (0.1%)	18.6 (1.5%)	18.9 (4.4%)	19.2 (7.0%)
改良版	MPI/OpenMP	18.6 (0.1%)	13.2 (1.1%)	13.6 (3.6%)	13.8 (5.9%)

HF計算ハイブリッド並列化のまとめ

- 使用CPUコア数が多くなると、ハイブリッド並列化の効果が顕著に出た
- 元々1%以下の初期軌道計算が、2048コアでは約4割を占めた
  - すべての計算を高速化・並列化する必要がある
- OpenMP導入のため、GAMESSの大幅な書き換えを行った
  - 多くのCommon変数をサブルーチンの引数に変更したため、Hartree-Fock及びDFT計算しか実行できないコードになった
  - common変数が多く使われているプログラムにOpenMPを導入して、計算時間削減と並列化効率向上を両立するのは大変

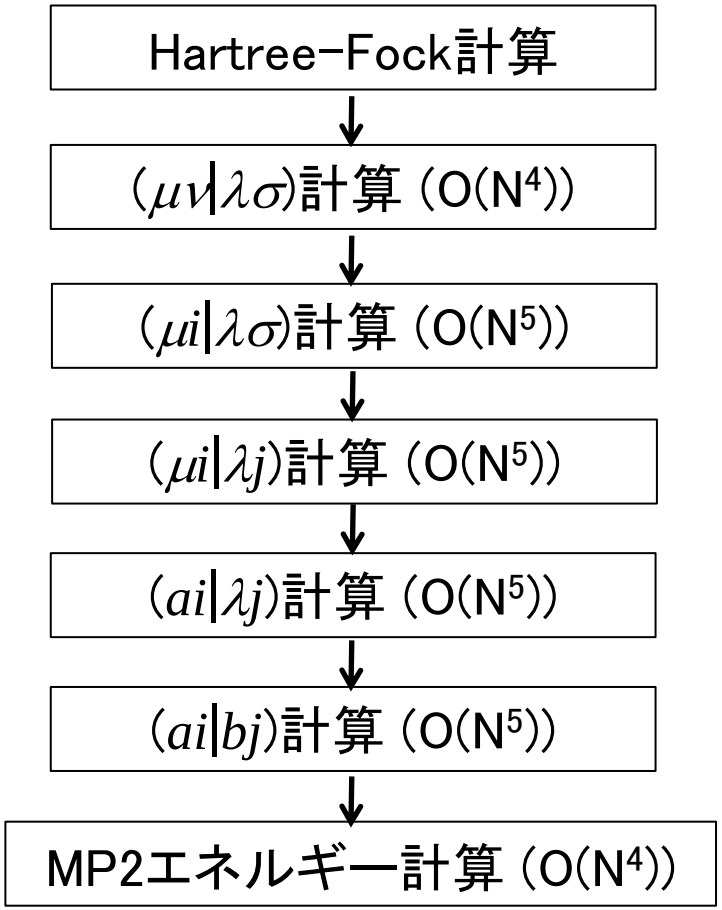
2次の摂動(MP2)法

- 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
i,j: 占有分子軌道, a,b: 非占有分子軌道
μ,ν,λ,σ: 原子軌道



これまでの並列計算アルゴリズム

- AOもしくはMOインデックス分散
  - AO:複数ノードにある部分的なMO2電子積分の足し合わせ
  - MO:同じAO2電子積分を複数のノードで計算 or Broadcast

R. A. Whiteside, J. S. Binkley, M. E. Colvin, H. F. Schaefer (1987) (32CPU)

I. M. B. Nielsen, C. L. Janssen (2000) (MPI + pthreads)

- Global arrays, ARMCI, DDI
  - 分散メモリを仮想的に共有メモリのように扱うライブラリ
- 前半AO、後半MOインデックス分散
  - 通信量がノード数に関わらずほぼ一定

J. Baker, P. Pulay (2002) (通信のためのデータソートに時間がかかる)

新規MP2エネルギー並列計算アルゴリズム

K. Ishimura, P. Pulay, S. Nagase, J Comput Chem 2006, 27, 407.

```
μ (AO index)  各ノードに分散
   do λ,σ
      calculate (μν|λσ)   [ν, μλσ]   (all ν)    AO 積分計算
      calculate (μi|λσ)   [i, μλσ]   (all i)    第1変換
      calculate (μi|λj)   [ij, λ, μ]  (i≧j)     第2変換
   end do λ,σ
   calculate (μi|bj)      [b, ij]     (all b)    第3変換
   (μi|bj)をディスクに書き込み [b, ij, μ]
end do μ
i,j (MO index)  各ノードに分散
   (μi|bj)をディクスから読み込み + MPI_isend,irecv
   calculate (ai|bj)      [b, a]      (all a,b)  第4変換
   calculate MP2 energy
end do i,j
```

- MPIのみで並列化、前半はAO、後半はMOインデックスを分散
- シンプルなデータソーティング
- ノード数にかかわらず、全体の通信量はほぼ一定

BLASルーチンの使い方

- DGEMM(行列−行列積)を用いた演算と多次元配列のインデックス入れ替え、さらにゼロクリア
  - DGEMMで行列を転置して演算
  - 通信回数を減らすため、ノード間で分散されているインデックスを外側に移動
  - 例：第3積分変換 $(\mu i|bj) = \sum_{\lambda}^{AO} C_{\lambda b}(\mu i|\lambda j)$

配列の並び

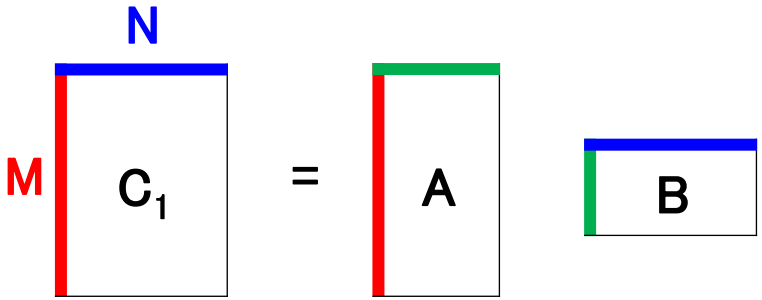
calculate (μi|λj)	[ij, λ, μ]	(i≧j)	第2変換
calculate (μi|bj)	[b, ij]	(all b)	第3変換

ソースコード

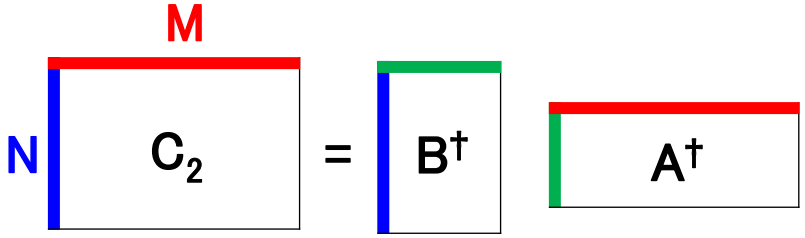
```
do μ
   call dgemm( 'T' ,' T' ,...,C,...,(μi|λj),...,(μi|bj),.,zero,..)
enddo
```

DGEMMでの行列転置1

- A(M,K), B(K,N)の場合、C=ABの計算でCの配列は(M,N)と(N,M)の2通り可能
  - 転置しない場合：DGEMM(‘N’ ,’ N’ ,...,A,...,B,...,C,...)

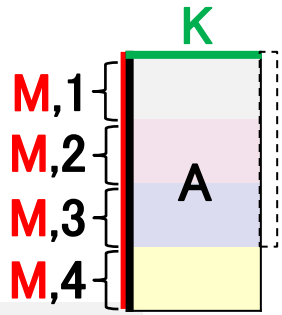


  - 転置する場合：DGEMM(‘T’ ,’ T’ ,...,B,...,A,...,C,...)



DGEMMでの行列転置2

- 3次元配列A(M,L,K), B(K,N)の場合、C=ABの計算でCの配列は(M,L,N)と(N,M,L)だけでなく、(M,N,L)や(N,L,M)も可能
  - (M,L,N):DGEMM('N','N',...,A,...,B,...,C,...), A(M*L,K), C(M*L,N)
  - (N,M,L):DGEMM('T','T',...,B,...,A,...,C,...), A(M*L,K), C(N,M*L)
  - (M,N,L):Lのループ内でDGEMM利用, A(1:M,I,1:K)より高速、
    A(M,L,K+1)と確保した方が安全




```
do I
   call DGEMM('N','N',M,N,K,one,A(1,I,1),M*L,B,K,zero,C(1,1,I),M)
enddo
```

  - (N,L,M): A(M,L,K+1) ,C(N,L,M+1)と確保した方が安全

```
do I
   call DGEMM('T','T',N,M,K,one,B,K,A(1,I,1),M*L,zero,C(1,I,1),N*L)
enddo
```

MP2エネルギー計算条件

計算機: Pentium4 3.0GHz
ネットワーク: Gigabit Ethernet
プログラム：GAMESS
分子: Taxol($C_{47}H_{51}NO_{14}$)

	6-31G(d)	6-311G(d,p)
基底次元数	1032	1484
占有軌道数	164	164
非占有軌道数	806	1258
1プロセス当たりメモリー使用量	0.67GB	0.96GB
1プロセス当たりディスク使用量	$90GB/n_{proc}$	$202GB/n_{proc}$
全体の通信量	90GB	202GB

MP2エネルギー計算結果

CPU数	1	2	4	8	16
6-31G(d) （1032 AOs）					
実時間 (hour)	10.2	5.08	2.54	1.31	0.64
Speed-up	1.0	2.0	4.0	7.8	15.8
6-311G(d,p) （1484 AOs）					
実時間 (hour)	31.6	16.3	8.06	4.05	2.05
Speed-up	1.0	1.9	3.9	7.8	15.4

MP2エネルギー微分計算式

$$E^x_{MP2} = -2\sum_{pq}^{all}\left[H^x_{pq} + \sum_{k}^{oall} 2(pq|kk)^x - (pk|qk)^x \right] P^{(2)}_{pq}$$

$$-2\sum_{i}\sum_{j}^{oall} S^x_{ij} W^{(2)}_{ij}[I] - 2\sum_{a}\sum_{b}^{vall} S^x_{ab} W^{(2)}_{ab}[I] - 4\sum_{a}\sum_{i}^{oall} S^x_{ai} W^{(2)}_{ai}[I]$$

$$+\sum_{ij}^{oall} S^x_{ij} W^{(2)}_{ij}[II] - \sum_{ab}^{vall} S^x_{ab} W^{(2)}_{ab}[II] - 4\sum_{i}^{oall}\sum_{a}^{vall} S^x_{ai} W^{(2)}_{ai}[II] + \sum_{ij}^{oall} S^x_{ij} W^x_{ij}[III] + 2\sum_{ij}\sum_{ab} t^{ab}_{ij}(ia|jb)^x$$

$$P^{(2)}_{ij} = \sum_{k}\sum_{ab} t^{ab}_{ik} \frac{(ja|kb)}{D^{ab}_{jk}}, \quad P^{(2)}_{iJ} = -\sum_{k}\sum_{ab} t^{ab}_{ik} \frac{(Ja|kb)}{\varepsilon_i - \varepsilon_J}, \quad P^{(2)}_{ab} = \sum_{ij}\sum_{c} t^{ac}_{ij} \frac{(ib|jc)}{D^{bc}_{ij}}, \quad P^{(2)}_{aB} = \sum_{ij}\sum_{c} t^{ac}_{ij} \frac{(iB|jc)}{\varepsilon_a - \varepsilon_B}$$

$$W^{(2)}_{ij}[I] = \sum_{k}\sum_{ab} t^{ab}_{ik}(ja|kb), \quad W^{(2)}_{ab}[I] = \sum_{ij}\sum_{c} t^{ac}_{ij}(ib|jc), \quad W^{(2)}_{ai}[I] = \sum_{jk}\sum_{b} t^{ab}_{jk}(ij|kb)$$

$$W^{(2)}_{ij}[II] = P^{(2)}_{ij}(\varepsilon_i + \varepsilon_j), \quad W^{(2)}_{ab}[II] = P^{(2)}_{ab}(\varepsilon_a + \varepsilon_b), \quad W^{(2)}_{ai}[II] = P^{(2)}_{ai}\varepsilon_i, W^{(2)}_{ij}[III] = \sum_{pq}^{MO} P^{(2)}_{pq} A_{pqij}$$

$$L_{ai} = \sum_{jk}^{oall} P^{(2)}_{jk}\{4(ia|jk) - 2(ik|aj)\} - \sum_{bc}^{vall} P^{(2)}_{bc}\{4(ia|bc) - 2(ib|ac)\} + 2N_a\sum_{jk}\sum_{b} t^{ab}_{jk}(ij|kl) - 2N_i\sum_{j}\sum_{bc} t^{bc}_{ij}(ab|jc)$$

新規MP2エネルギー微分並列計算アルゴリズム

K. Ishimura, P. Pulay, S. Nagase, J. Comput. Chem., 2007, 28, 2034.

do λ (積分変換計算)
- generate AO integrals (μν|λσ)
- calculate (μν|λj)
- calculate (μi|λj)
- calculate (ai|λj)
- save (ai|λj) on disk

enddo λ

do a(MP2エネルギー、アンプリチュード計算)
- read and transfer (ai|λj)
- calculate (ai|bj), (ai|kj)
- calculate MP2 energy and amplitude
- generate $t_{ij}^{ab}, P_{ij}^{(2)}, P_{ab}^{(2)}, W_{ij}^{(2)}[I], W_{ab}^{(2)}[I], W_{ai}^{(2)}[I], t_{ij}^{a\sigma}$
- transfer and save $t_{ij}^{a\sigma}$

enddo a

do σ (MP2ラグランジアン計算)
- read $t_{ij}^{a\sigma}$ from disk
- calculate and save $t_{ij}^{\nu\sigma}$
- generate AO integrals (μν|λσ)
- calculate (μν|jσ)
- calculate $L_{\mu\nu}^{1,2}, L_{\mu i}^{4}$

enddo σ

Solve CPHF equation

do μ (AO積分の微分計算)
- calculate $H_{\mu\nu}^{x}, S_{\mu\nu}^{x}$
- read $t_{ij}^{\nu\sigma}$ and calculate $t_{\mu\lambda}^{\nu\sigma}$
- calculate $(\mu\nu|\lambda\sigma)^{x}$ terms

enddo μ

エネルギー微分計算アルゴリズムのポイント

- 全ての計算の負荷分散+データ分散+高速化+通信量削減
- ノード数にかかわらず、全体の通信量はほぼ一定
- 演算量、使用メモリ量が少なくなるように式を展開
  - 例えば$(ia|bc)$ (i:占有軌道、a,b,c:非占有軌道)の積分変換はコストがかかるので、全てを変換せずに計算

$$L'_{ai} = \sum_{bc} P_{bc}(ia|bc) = \sum_{bc}^{AO}\sum_{\lambda\sigma}^{AO} C_{\lambda b} C_{\sigma c} P_{bc}(ia|\lambda\sigma)$$

O(N^5)の演算2回をO(N^3)2回に

$$= \sum_{\lambda\sigma}^{AO} P_{\lambda\sigma}(ia|\lambda\sigma) \qquad P_{\lambda\sigma} = \sum_{bc} C_{\lambda b} C_{\sigma c} P_{bc}$$

さらにO(N^5)1回をO(N^3)1回に

$$= \sum_{\nu}^{AO} C_{\nu a} \underbrace{\sum_{\lambda\sigma}^{AO} P_{\lambda\sigma}(i\nu|\lambda\sigma)}_{X_{i\nu}}$$

MP2エネルギー微分計算条件

計算機: Pentium4 3.0GHz
ネットワーク: Gigabit Ethernet
プログラム：GAMESS
分子: Taxol($C_{47}H_{51}NO_{14}$)

	6-31G	6-31G(d)
基底次元数	660	1032
占有軌道数	164	164
非占有軌道数	434	806
1プロセス当たりメモリー使用量	0.78GB	1.84GB
1プロセス当たりディスク使用量	147GB/n_{proc}	426GB/n_{proc}
全体の通信量	147GB	426GB

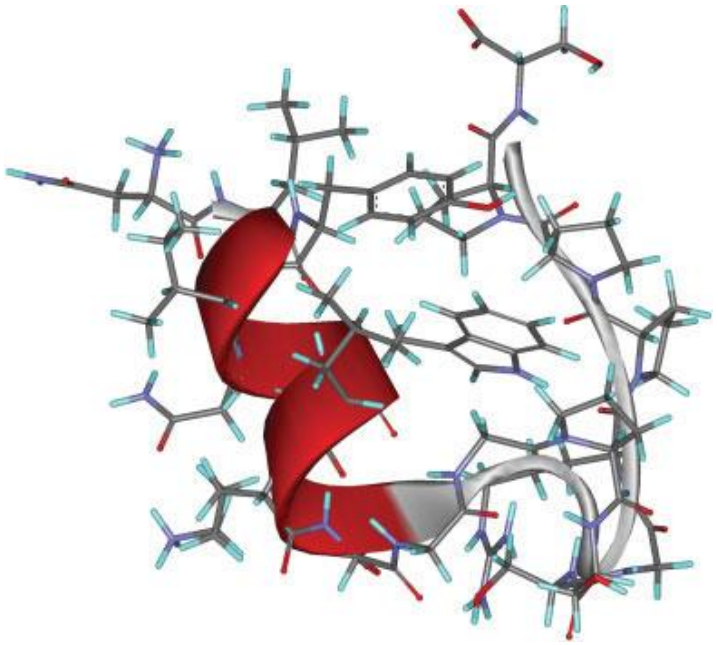
MP2エネルギー微分計算結果

CPU数	1	2	4	8	16	32
6-31G (660基底)						
実時間 (hour)	17.4	8.09	3.82	1.92	0.97	0.53
Speed-up	1.0	2.1	4.5	9.0	17.9	33.0
CPU使用率(%)	69.8	76.9	81.2	78.1	76.7	72.9
6-31G(d) (1032基底)						
実時間 (hour)		31.1	15.1	7.57	3.86	2.05
Speed-up		2.0	4.1	8.2	16.1	30.4
CPU使用率(%)		80.2	85.6	82.3	78.6	75.8

FMO-MP2法

D. Fedorov, K. Ishimura, T. Ishida, K. Kitaura, P. Pulay, S. Nagase, J. Comput. Chem. 2007, 28, 1476.

- 開発したMP2プログラムとFMOプログラムの組み合わせ
- 分子：Trp-cage設計蛋白質TC5b ($C_{98}H_{150}N_{27}O_{29}$)



基底関数：6-31(+)G*　2480基底
6-311(+)G*　3246基底
FMO計算：1フラグメントに2残基

使用メモリ：FMO-MP2 数百MB/プロセス
MP2　4.7GB/プロセス

FMO-MP2計算結果

FMOn　(n体相互作用まで計算)

MP2とのエネルギー誤差(kcal/mol)

	FMO2	FMO3
6-31(+)G*	2.1	-0.2
6-311(+)G*	-2.9	-0.5

計算時間(hour)

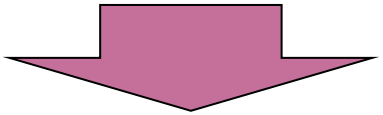
	FMO2	FMO3	MP2
6-31(+)G*	3.2	22.7	20.4
6-311(+)G*	7.6	56.8	46.5

・困難であった巨大分子FMO-MP2計算の精度確認が可能に

・FMO-MP2の高速化・高並列化

これからの大規模並列計算プログラム開発について（個人的な意見）

- 今後さらに増えるCPUコア数、GPGPUやIntel Phiなどアクセラレータに対応しようとすると、計算手法とプログラムの開発コストはますます増加
- 分野としてプログラム開発コストを減らすためには、複数の人が同じコードの開発やチューニングをしない仕組み作り



- よく用いられる計算はライブラリ化、モジュール化（コミュニティーコード）
- 自由に利用できるライセンス

新規プログラムの開発方針

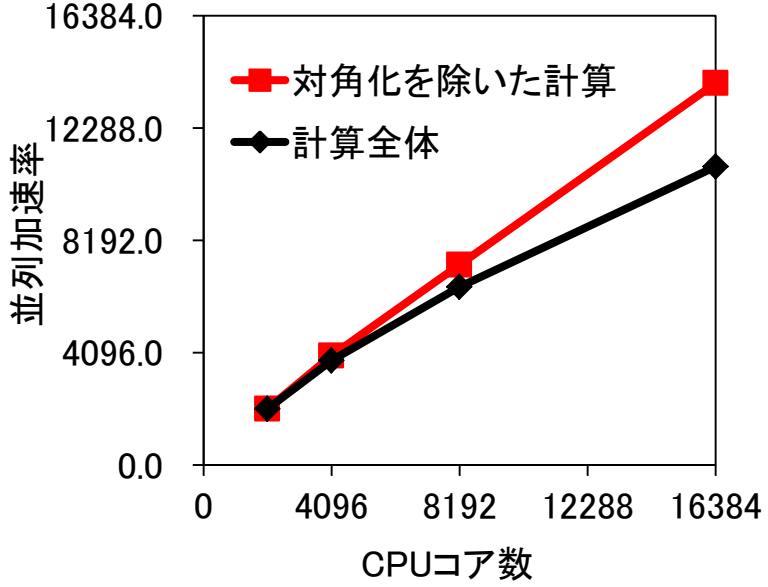
- MPI/OpenMPハイブリッド並列を設計段階から考慮したプログラム開発 (Module変数、サブルーチン引数の仕分け)
- 言語はFortran90
- 対象マシン:スカラ型CPUを搭載した計算機(PCクラスタから京コンピュータまで)
- オープンソースライセンス(Apache 2.0)
- 1,2電子積分など計算の一部を容易に抜き出せる構造
- 中間データの保存にディスクは使わず、メモリ上に分散保存
- 巨大分子、重原子を含む分子に対応できるようChebyshev展開次数を上げて積分計算の有効桁数向上

新規プログラムについて

- SMASH(Scalable Molecular Analysis Solver for High performance computing)
- 数万、数十万CPUコアでの効率的な実行が当面の目標
- エネルギー微分、構造最適化計算を重点的に整備
- 現在開発中

新規プログラムの性能

並列化効率 (京コンピュータ) ($C_{150}H_{30})_2$, cc-pVDZ (4500 基底)の計算時間(sec)



GAMESSとの比較 (Xeon E5649 2.53GHz 12core,1ノード利用)

taxol($C_{47}H_{51}NO_{14}$)の計算時間(sec)

基底関数	GAMESS	SMASH
6-31G(d) (1032基底)	706.4	666.6
cc-pVDZ (1185基底)	2279.9	1434.3

参考資料

- 量子化学
  - “分子軌道法” 藤永茂
  - “新しい量子化学 上・下” A.ザボ, N.S.オスランド著, 大野公男,阪井健男, 望月祐志訳
  - “Introduction to Computational Chemistry” Frank Jensen
  - “Molecular Electronic-structure Theory” Trygve Helgaker, Poul Jørgensen, Jeppe Olsen
- 計算機
  - “プロセッサを支える技術” Hisa Ando
- BLAS、LAPACK
  - /opt/intel/Compiler/(バージョン番号)/Documentation/ja_JP/mkl/mklman90.pdf もしくはmklman90_j.pdf （バージョンによって場所は違う）(Intel Compilerを持っている方)