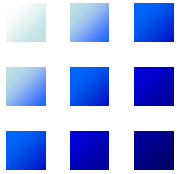


量子化学計算の大規模化2

石村　和也
（株式会社クロスアビリティ）

2019年度計算科学技術特論A 第15回
2019年7月25日

本日の内容

- 高速化・並列化例
  - 2次の摂動(MP2)計算 (MPI並列)
  - Hartree-Fock計算 (MPI/OpenMP並列)
- 新たなオープンソースソフトウェアの開発
  - 大規模並列量子化学計算プログラムSMASH
- メニーコア時代に向けた開発
  - OpenMP並列効率のさらなる向上
  - メモリ使用量削減
  - 通信時間削減
- まとめ
- 参考資料

高速化

- 演算量削減
  - 演算量の少ない1,2電子積分計算アルゴリズムの開発
  - 効果的なカットオフの導入
  - 対称性の導入
- 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
- 収束回数の削減
  - DIIS、SOSCF法などによるSCFサイクル数削減
  - 初期Hessian改良による構造最適化回数の削減
- 実行性能の向上
  - SIMD演算の利用
  - 時間のかかる演算回数を削減
  - 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
  - BLAS, LAPACKなど数学ライブラリの利用

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
- 高速化と並列化の両立

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

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

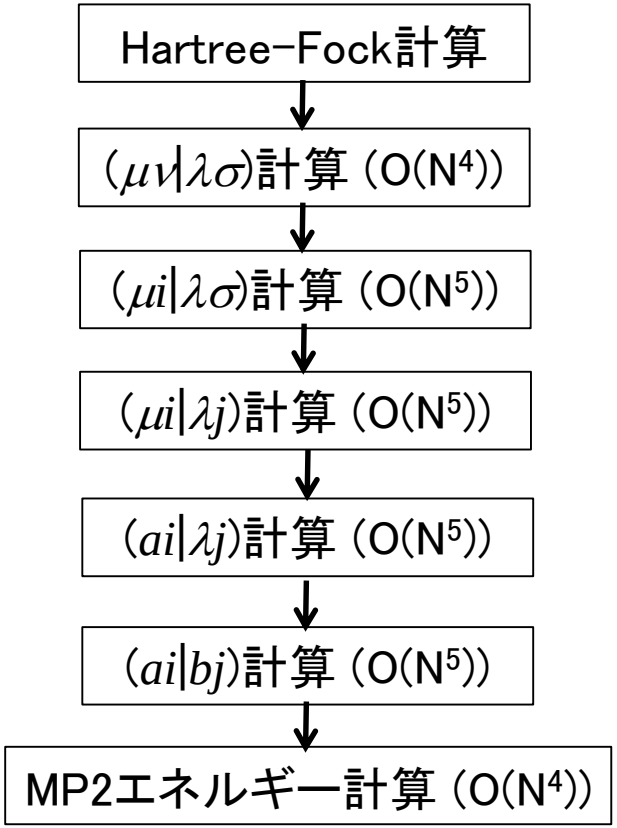
$$\underline{(ai|bj)} = \sum_{\mu\nu\lambda\sigma}^{AO} C_{\mu a} C_{\nu i} C_{\lambda b} C_{\sigma j} \underline{(\mu\nu|\lambda\sigma)}$$

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

i, j: 占有軌道, a, b: 非占有軌道



従来のMP2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:複数プロセスにある部分的なMO2電子積分の足し合わせ
    総通信量がプロセス数に依存
  - MO:同じAO2電子積分を複数のプロセスで計算 or Broadcast
    不完全な計算負荷分散 or 総通信量がプロセス数に依存

    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
  - 完全な計算負荷分散 and 総通信量がプロセス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.

- 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

```
μ (AO index)  各ノードに分散
  do λ,σ
    AO積分計算 (μν|λσ)   [ν, μλσ]   (all ν)
    第1変換    (μi|λσ)   [i, μλσ]   (all i)
    第2変換    (μi|λj)   [ij, λ, μ]  (i≧j)
  end do λ,σ
  第3変換      (μi|bj)   [b, ij]     (all b)
  (μi|bj)をディスクに書込み  [b, ij, μ]
end do μ
ij (MO index)  各ノードに分散
  (μi|bj)をディスクから読込み + MPI_isend,irecv
  第4変換      (ai|bj)   [b, a]      (all a,b)
  MP2エネルギー計算
end do ij
```

$\mu, \nu, \lambda, \sigma$: 原子基底関数,　　i, j: 占有軌道,　a, b: 非占有軌道

BLASルーチンの使い方

- BLAS level3のDGEMM(倍精度行列−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(データの並び替え)
  - DGEMMを用いて転置した行列での演算
  - 通信におけるデータソーティングをシンプルにするため、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

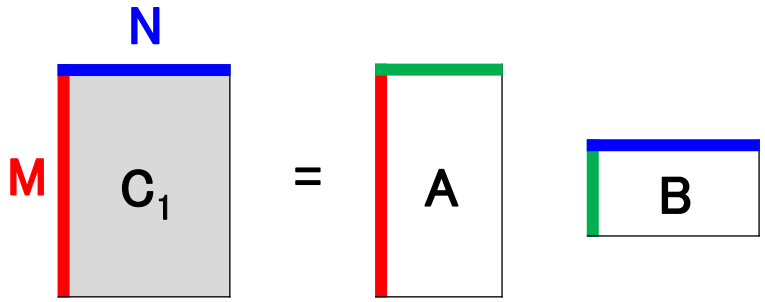
第2変換 (μi|λj)	[ij, λ, μ]	(i≧j)
第3変換 (μi|bj)	[b, ij, μ]	(all b)

ソースコード

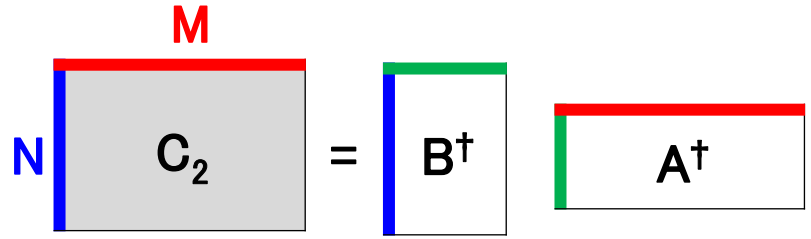
```
do μ
   call dgemm( 'T' ,' T' ,...,C,...,(μi|λj),...,(μi|bj),..)
enddo
```

DGEMMでの行列転置

- 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,...)



MP2エネルギーテスト計算

計算機: Pentium4 3.0GHz
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**

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)
- AO積分計算 (μν|λσ)
- 第1変換 (μν|λj)
- 第2変換 (μi|λj)
- 第3変換 (ai|λj)
- ディスク書込み (ai|λj)

Enddo λ

do a(MP2エネルギー、アンプリチュード計算)
- ディスク読込みと送受信 (ai|λj)
- 第4変換 (ai|bj), (ai|kj)
- MP2エネルギーとアンプリチュード計算
- 各項計算 $t_{ij}^{ab}, P_{ij}^{(2)}, P_{ab}^{(2)}, W_{ij}^{(2)}[I], W_{ab}^{(2)}[I], W_{ai}^{(2)}[I], t_{ij}^{a\sigma}$
- 送受信とディスク書込み $t_{ij}^{a\sigma}$

enddo a

do σ (MP2ラグランジアン計算)
- ディスクから読込み $t_{ij}^{a\sigma}$
- 計算とディスク書込み $t_{ij}^{\nu\sigma}$
- AO積分計算 (μν|λσ)
- 第1変換 (μν|jσ)
- 各項計算 $L_{\mu\nu}^{1,2}, L_{\mu i}^{4}$

enddo σ

CPHF計算

do μ (AO積分の微分計算)
- 微分項計算 $H_{\mu\nu}^{x}, S_{\mu\nu}^{x}$
- ディスク読込み $t_{ij}^{\nu\sigma}$ と計算$t_{\mu\lambda}^{\nu\sigma}$
- 微分項計算 $(\mu\nu|\lambda\sigma)^{x}$

enddo μ

赤字のインデックスをMPI並列化

$\mu, \nu, \lambda, \sigma$: 原子基底関数, i, j, k: 占有軌道, a, b: 非占有軌道

MP2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、メモリ使用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
  - 積分変換をしないために出てくる分子軌道係数Cの処理を、あらかじめ行う($P_{\lambda\sigma} \leftarrow P_{bc}$)

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

$$= \sum_{\boldsymbol{\lambda\sigma}}^{AO} P_{\boldsymbol{\lambda\sigma}}(ia|\boldsymbol{\lambda\sigma})$$

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

$$P_{\lambda\sigma} = \sum_{bc} C_{\lambda b} C_{\sigma c} P_{bc}$$

MP2エネルギー微分テスト計算

計算機: Pentium4 3.0GHz
プログラム：GAMESS
分子: Taxol($C_{47}H_{51}NO_{14}$)

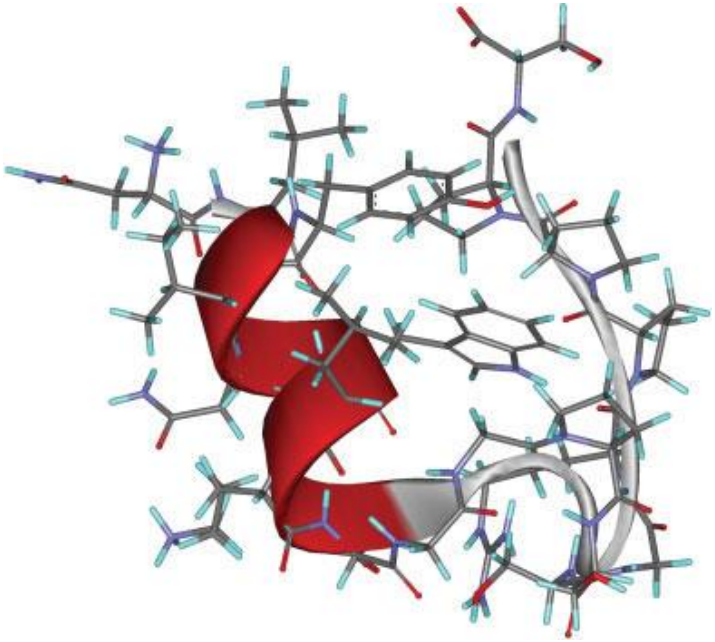
	6-31G	6-31G(d)
基底次元数	660	1032
占有軌道数	164	164
非占有軌道数	434	806
1プロセス当たりメモリ使用量	**0.78GB**	**1.84GB**
1プロセス当たりディスク使用量	**147GB/n_{proc}**	**426GB/n_{proc}**
全体の通信量	**147GB**	**426GB**

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
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

FMO-MP2法

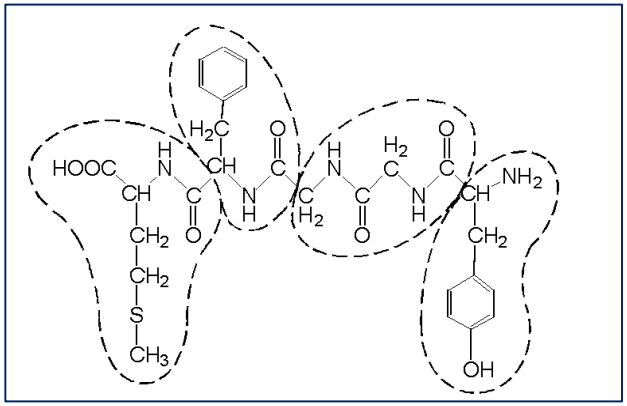
D. Fedorov, K. Ishimura, T. Ishida, K. Kitaura, P. Pulay, S. Nagase, J. Comput. Chem. 2007, 28, 1476.

- 開発したMP2ルーチンとFMOルーチンの組み合わせ
- 分子：Trp-cage設計蛋白質TC5b ($C_{98}H_{150}N_{27}O_{29}$)
- GAMESSプログラムに実装



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

メモリ使用量：FMO-MP2 数百MB/プロセス
分割無しMP2　4.7GB/プロセス



FMO法の分割イメージ

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)

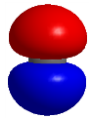
	FMO2	FMO3
6-31(+)G*	2.1	-0.2
6-311(+)G*	-2.9	-0.5

計算時間(hour)

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

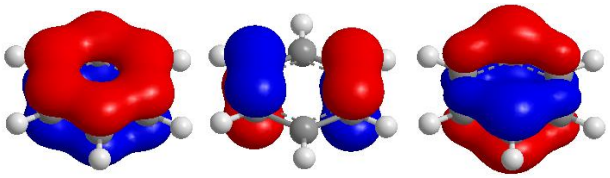
- 困難であった巨大分子FMO-MP2計算の精度確認が可能になった
- FMO-MP2の高速化・高並列化

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(N⁴), カットオフでO(N³)程度)

↓

Fock行列対角化 (O(N³))

分子軌道C収束

↓

計算終了

分子軌道C収束せず

MPI/OpenMP並列アルゴリズム1

K. Ishimura, K. Kuramoto, Y. Ikuta, S. Hyodo, J. Chem. Theory Comp. 2010, 6, 1075.

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

並列化前

```
do μ=1, nbasis
 do ν=1, μ
  do λ=1, μ
   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
```

MPI/OpenMPハイブリッド並列化後

```
!$OMP parallel do schedule(dynamic,1) reduction(+:Fock)
do μ=nbasis, 1, -1                    <-- OpenMPによる振り分け
  do ν=1, μ
    μν=μ(μ+1)/2+ν
    λstart=mod(μν+mpi_rank,nproc)+1
    do λ=λstart, μ ,nproc             <-- MPIランクによる振り分け
      do σ=1, λ
        AO2電子積分(μν|λσ)計算+Fock行列に足し込み
      enddo
    enddo
  enddo
enddo
!$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は変わらない
- MPIランクとmod計算だけで、IF文を使わずにプロセス間分散
  - 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
- MPI通信はOpenMP領域外で実行

```
!$OMP parallel do schedule(dynamic,1) reduction(+:Fock)
do μ=nbasis, 1, -1                <-- OpenMPによる振り分け
  do ν=1, μ
    μν=μ(μ+1)/2+ν
    λstart=mod(μν+mpi_rank,nproc)+1
    do λ=λstart, μ ,nproc         <-- MPIランクによる振り分け
      do σ=1, λ
        AO2電子積分(μν|λσ)計算+Fock行列に足し込み
      enddo
    enddo
  enddo
enddo
!$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、スレッド生成や分散などのオーバーヘッド(余分なコスト)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プロセス内の負荷分散を均等化
- すべての変数について、スレッド間で共有するか(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

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

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-行列積の多用で高速化・並列化

C_1: 初期軌道係数
C_2: 拡張Huckel法による軌道係数
S_{11}: 実際の計算で用いる基底の重なり積分
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

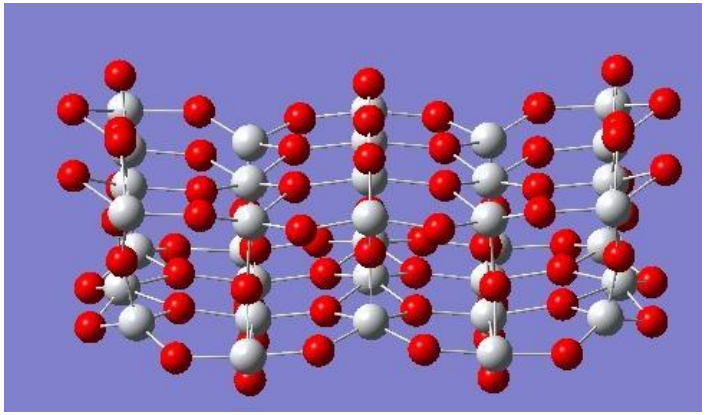
D. Cremer and J. Gauss, *J. Comput. Chem.* **7** 274 (1986).

- 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
  - 対角化はSCFの最初と最後の2回のみ

Hartree-Fockエネルギーテスト計算条件

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

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

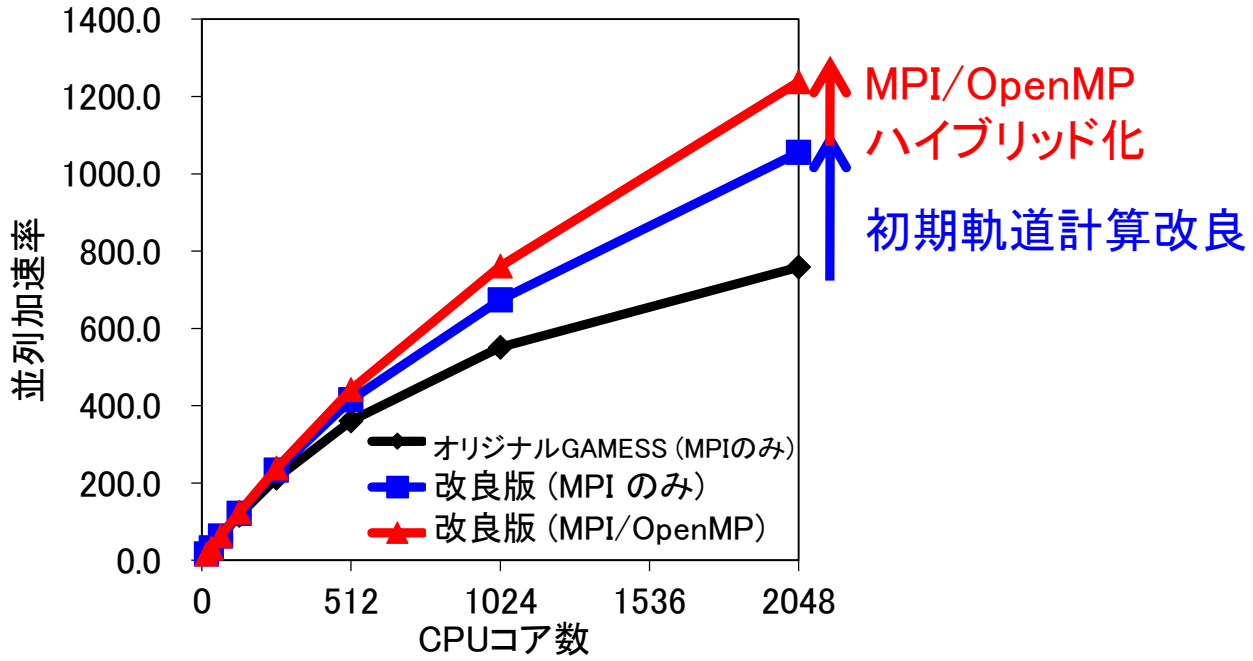


Table 全計算時間(秒)と並列加速率(カッコ内)

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

- Fock行列計算では、MPI/OpenMPが最も速く並列加速率も高い
- オリジナルの初期軌道計算時間は、2048コアで全体の40%弱を占める

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%)

Hartree-Fock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させるのは大変

- ✓ GAMESSに実装しても、特定の計算しか実行できないためそのソースコードを公開することができない
- ✓ MPIとOpenMPのハイブリッド並列をさらに進めるには、設計段階からしっかりと考慮した新たなプログラムが必要

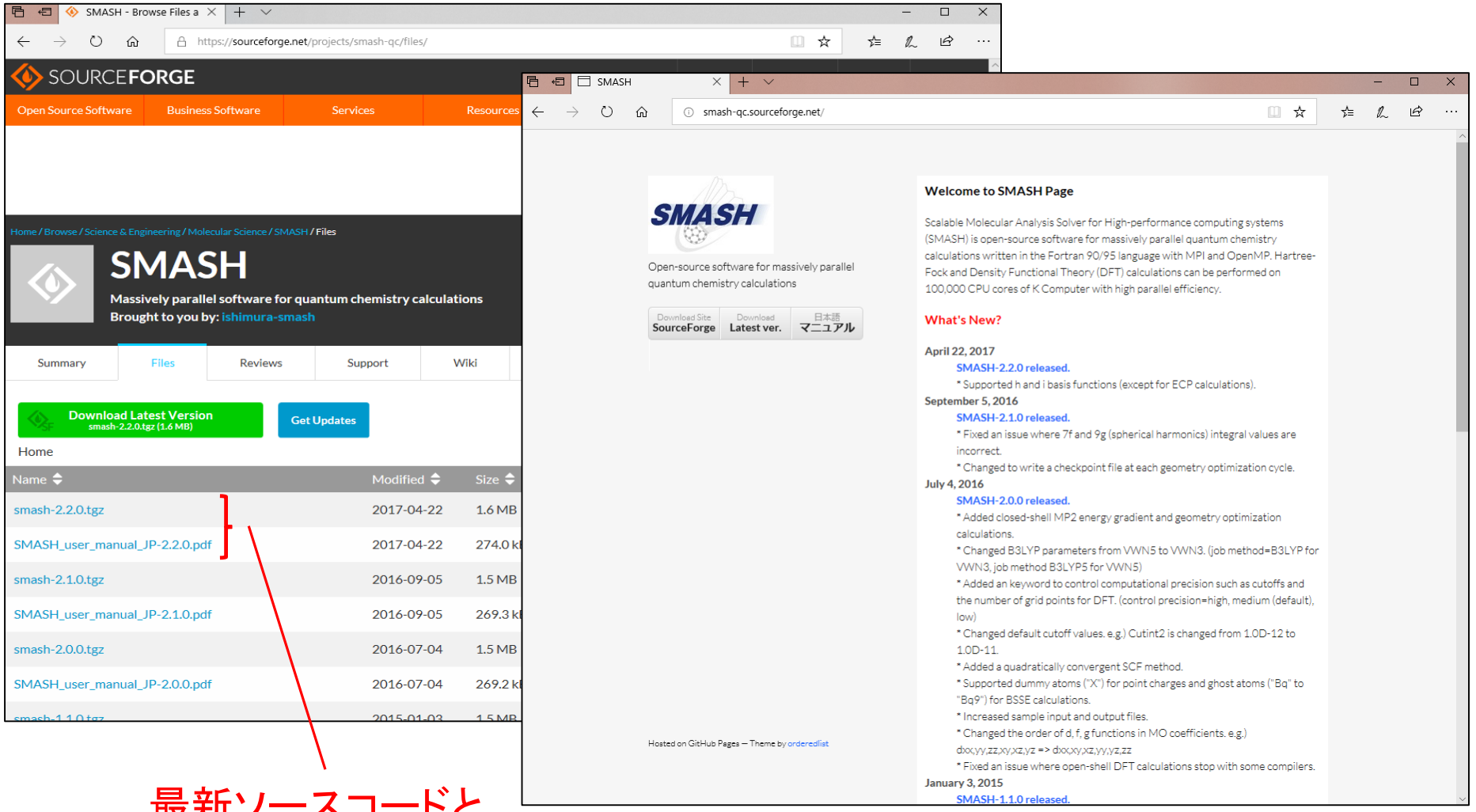
新たなオープンソースソフトウェアの開発

- キーワードはシンプル(実行方法、入力形式、ライセンス、開発)
- MPI/OpenMPハイブリッド並列化と高速計算アルゴリズムを組み込み、一つのプログラムでスカラ型CPU搭載計算機をカバー
  - 演算量削減、計算負荷とデータの均等な分散、通信の最適化
  - 今後ノード当たりのコア数はさらに増加
  - 京、富岳も研究室レベルの計算機と基本的な構成は同じ
- よく用いられるルーチンのライブラリ化・モジュール化
- オープンソースライセンスで配布
  - ますます複雑になる計算機を理解した上で、最適な式、アルゴリズム、近似を開発、選択してプログラムを作る必要があり、開発コスト削減は不可欠
  - 複数の人が同じような開発やチューニングをしない仕組み作り
  - 技術・ノウハウの共有
  - 計算機科学との連携

SMASHプログラム

- 大規模並列量子化学計算プログラムSMASH (Scalable Molecular Analysis Solver for High performance computing systems)
- オープンソースライセンス(Apache 2.0)
- http://smash-qc.sourceforge.net/
- 2014年9月1日公開
- 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)
- エネルギー微分、構造最適化計算を重点的に整備
- 現時点で、Hartree-Fock, DFT(B3LYP), MP2計算が可能
- MPI/OpenMPハイブリッド並列を設計段階から考慮したアルゴリズム及びプログラム開発 (Module変数、サブルーチン引数の仕分け)
- 言語はFortran90/95
- 1,2電子積分など頻繁に使う計算ルーチンのライブラリ化で開発コスト削減
- 電子相関計算の大容量データをディスクではなくメモリ上に分散保存
- 2014年9月からの約5年間で、ダウンロード数は約1800

SMASH ウェブサイト



最新ソースコードと
日本語マニュアル

AO2電子積分ルーチンのインターフェイス

- 2電子積分ルーチン：データはすべて引数で受け渡し、ブラックボックス化
- call int2elec(twoeri, exijkl, coijkl, xyzijkl, nprimijkl, nangijkl, nbfijkl, maxdim, mxprsh, threshex)

twoeri	2電子積分計算値 (Output)
exijkl	primitive関数の指数 (Input)
coijkl	primitive関数の係数
xyzijkl	xyz座標
nprimijkl	primitive関数の数
nangijkl	軌道角運動量(s=0, p=1, d=2,...)
nbfijkl	基底関数の数(s=1, p=3, d=5or6,...)
maxdim	最大twoeriの次元数
mxprsh	最大primitive関数の数
threshex	$\exp(-x^2)$計算の閾値

プログラム開発の効率化

- 2電子積分ルーチンはHartree-Fock,DFT計算以外の高精度電子相関計算でも利用
- 2電子積分の微分は軌道角運動量の異なる2電子積分の線形結合になるため、微分計算で2電子積分ルーチンを再利用可能
- 適切な係数と軌道角運動量を引数で渡し、2電子積分ルーチンをcallした後適切な要素へ結果を足しこむだけで実装完了

call int2elec(twoeri, exijkl, coijkl, xyzijkl, nprimijkl, nangijkl, nbfijkl, maxdim, mxprsh, threshex)

例：$(\boldsymbol{p_x}s|ss)$の微分計算の場合

$$\partial(p_x s|ss)/\partial X_a = \boldsymbol{2\alpha_a}(\boldsymbol{d_{xx}}s|ss) - (\boldsymbol{s}s|ss)$$

$$\partial(p_x s|ss)/\partial Y_a = \boldsymbol{2\alpha_a}(\boldsymbol{d_{xy}}s|ss)$$

$$\partial(p_x s|ss)/\partial Z_a = \boldsymbol{2\alpha_a}(\boldsymbol{d_{xz}}s|ss)$$

MPI/OpenMPハイブリッド並列アルゴリズム

Hartree-Fock計算

```
!$OMP parallel do schedule(dynamic,1) &
!$OMP reduction(+:Fock)
do μ=nbasis, 1, -1              <- OpenMP
  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
      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)
```

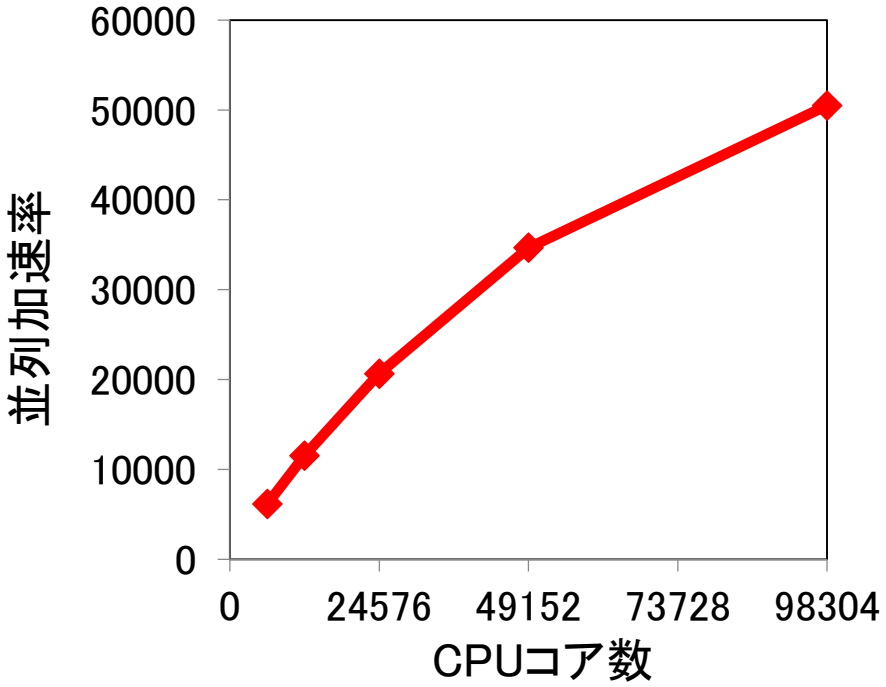
MP2計算

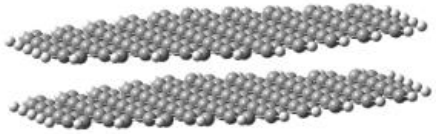
```
do μλ (AO index pair)    MPIランクによる振り分け
!$OMP parallel do schedule(dynamic,1)
  do σ
    AO積分計算      (μν|λσ)   (all ν)
    第1変換         (μi|λσ)    (all i)
  enddo
!$OMP end parallel do
  第2変換(dgemm)  (μi|λj)     (i≧j)
end do μλ
do ij (MO index pair)    MPIランクによる振り分け
  MPI_sendrecv (μi|λj)
  第3変換(dgemm)  (μi|bj)     (all b)
  第4変換(dgemm)  (ai|bj)     (all a)
  MP2エネルギー計算
end do ij
call mpi_reduce(MP2エネルギー)
```

- 膨大な中間データをすべてメモリ上に分散保存
- BLAS level3(dgemm)で効率的な演算とスレッド並列化を実現

B3LYPエネルギー並列計算性能

- 10万コアで5万倍のスピードアップ、実行性能13%
- 10万コアで360原子系のB3LYP計算が2分半
- 行列対角化(LAPACK,分割統治法)3回分の時間は約35秒
  →ScaLAPACK、EigenExaなどプロセス並列化されているライブラリ導入が必要





計算機：京コンピュータ
分子　：$(C_{150}H_{30})_2$ (360原子)
基底関数：cc-pVDZ
(4500基底)
計算方法：B3LYP
SCFサイクル数：16

CPUコア数	6144	12288	24576	49152	98304
実行時間 (秒)	1267.4	674.5	377.0	224.6	154.2

Hartree-Fockエネルギー1ノード計算性能

- 1ノードでは、GAMESSよりHartree-Fock計算時間を最大40%削減
- SMASHではS関数とP関数を別々に計算するため、SP型の基底ではGAMESSとの差は小さい

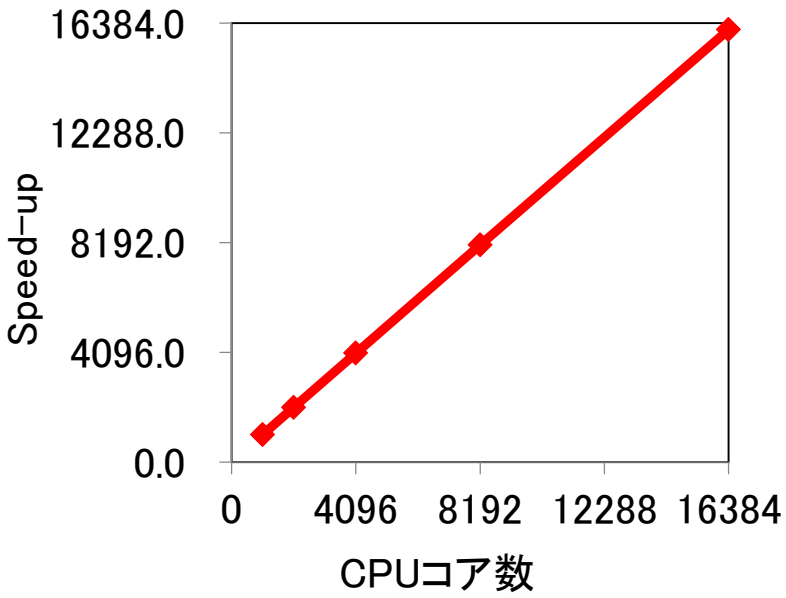
GAMESSとの比較

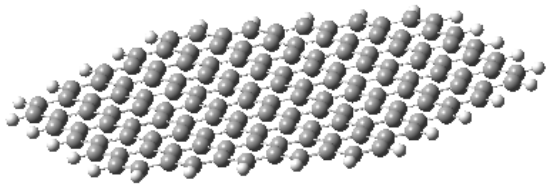
- Xeon E5649 2.53GHz 12core、1ノード利用
- Taxol($C_{47}H_{51}NO_{14}$)のHartree-Fock計算時間(sec)
- 同じ計算条件(積分Cutoffなど)

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

B3LYPエネルギー微分並列計算性能

- エネルギー計算と異なり対角化計算が無いため、並列化効率はほぼ100%





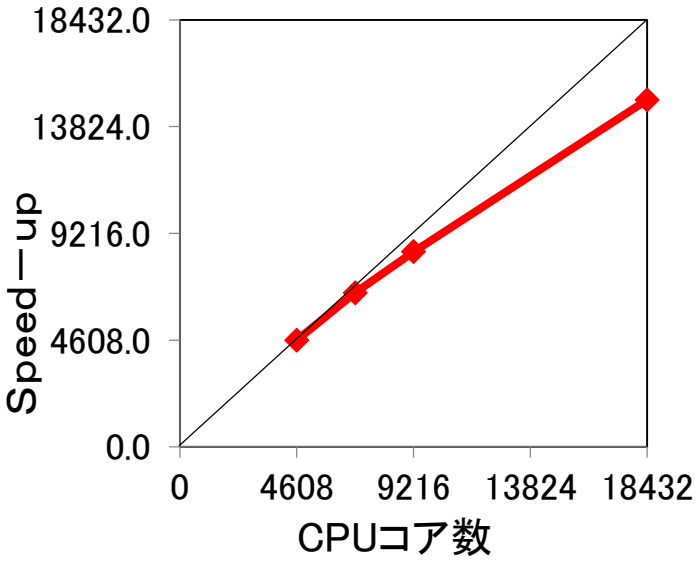
計算機：京コンピュータ
分子　：($C_{150}H_{30}$)
基底関数：cc-pVDZ (2250 functions)
計算方法：B3LYP

Table B3LYPエネルギー1次微分計算時間と並列加速率

CPUコア数	1024	4096	8192	16384
微分計算のみ計算時間(秒)	402.0	101.2	50.8	25.5
並列加速率	1024.0	4067.7	8103.3	16143.1

MP2エネルギー計算性能

- 通信量と回数を削減し、1万CPUコアでも高い並列性能を実現



計算機：京コンピュータ
分子　：$C_{150}H_{30}$ (180原子)
基底関数：6-31G(d) (2160基底)
計算方法：MP2

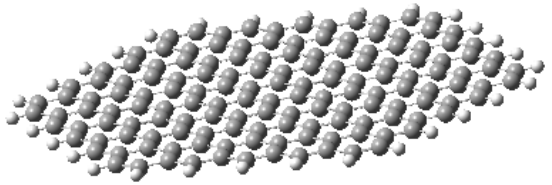


Table MP2エネルギー計算時間(秒)と並列加速率(カッコ内)

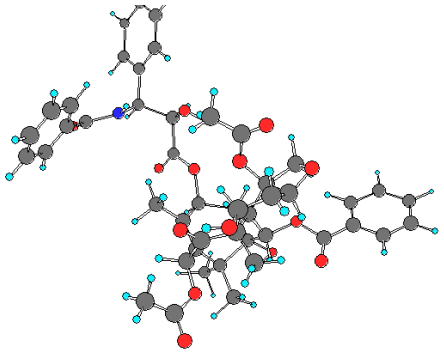
CPUコア数	4608	6912	9216	18432
MP2計算時間(秒)	152.5	105.7	83.4	46.9
Speed-up	4608.0	6648.2	8425.9	14983.4

構造最適化回数の削減

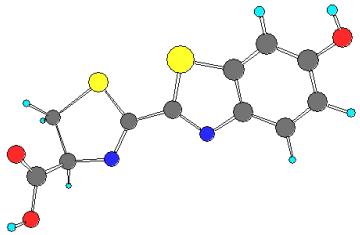
- Redundant座標と力場パラメータを使い、初期ヘシアンを改良することで最適化回数が1/5から1/6になった
- 2サイクル目以降のヘシアンはBFGS法で更新

Table B3LYP/cc-pVDZ構造最適化回数(初期構造HF/STO-3G)

	Cartesian 座標		Redundant 座標
Luciferin($C_{11}H_8N_2O_3S_2$)	63	⇨	11
Taxol ($C_{47}H_{51}NO_{14}$)	203		40



Taxol ($C_{47}H_{51}NO_{14}$)



Luciferin($C_{11}H_8N_2O_3S_2$)

メニーコア時代に向けた開発

- OpenMP並列効率の向上
  - ノードあたりのコア数は大幅に増えると予測される
  - 場合によっては、演算量を増やしても同じデータへのアクセス競合やReductionを削減した方が計算時間削減につながる
- 使用メモリ量の削減
  - ノードあたりの演算性能向上に比べて、メモリ量はそれほど増えない可能性が高い
- 通信時間の削減
  - 計算と通信のオーバーラップが重要になる（富士通FX100では1ノードに32演算コア+2アシスタントコア）
  - 通信レイテンシの向上はあまり望めないと予測される
- SIMD長の増加
- 開発コストの削減
  - モジュール化、ライブラリ化の推進
    - 分野全体での共通ルーチンの共有によるコスト削減
    - 情報とノウハウ共有
  - Pythonを用いた効率的な開発

OpenMP並列効率のさらなる向上1

- 振り分けるタスクの数を増やす（タスクの粒度を小さくする）
  例）Hartree-Fock計算

改良前

```
!$OMP parallel do schedule(dynamic,1) &
!$OMP reduction(+:Fock)
do μ=nbasis, 1, -1
  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
      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)
```

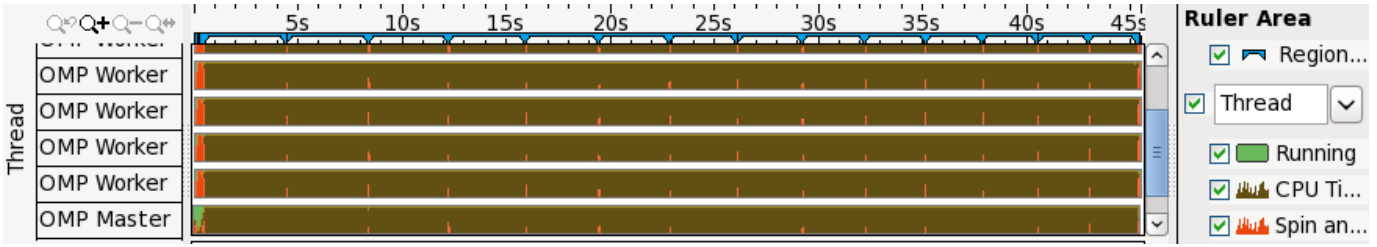
改良後

```
!$OMP parallel do schedule(dynamic,1) &
!$OMP reduction(+:Fock)
do μν=nbasis*(nbasis+1)/2, 1, -1
  μνからμとνを逆算

   λstart=mod(μν+mpi_rank,nproc)+1
    do λ=λstart, μ ,nproc  <- MPIランク
      do σ=1, λ
        AO2電子積分(μν|λσ)計算
        +Fock行列に足し込み
      enddo
    enddo

enddo
!$OMP end parallel do
call mpi_allreduce(Fock)
```

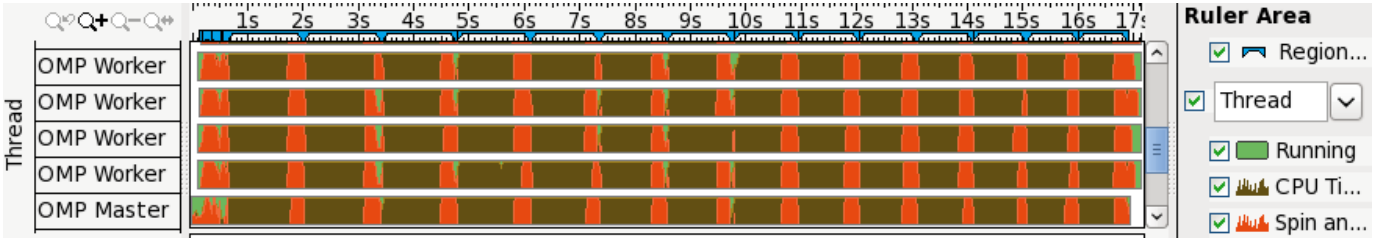
OpenMP並列効率のさらなる向上2

- 計算機：Fujitsu PRIMEGY CX2500 (Xeon E5-2697, 28コア/ノード)
- Intel VTune Amplifier2015を用いた解析
  - 計算条件：Luciferin($C_{11}H_8N_2O_3S_2$)、Hartree-Fock/cc-pVDZ(300基底)
  - 改良前は、8コアではスレッド待ち時間とオーバーヘッド(オレンジ色)はほとんどないが、28コアでは全計算時間の約1/4まで増加
  - 改良後の待ち時間とオーバーヘッドは、数%程度まで減少

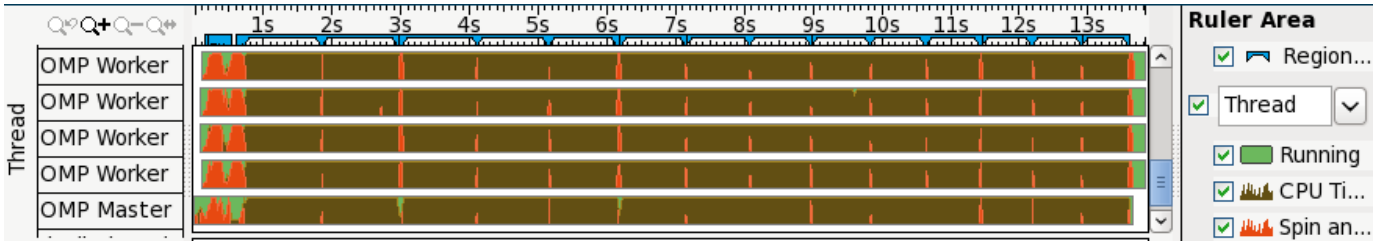
改良前
8コア



改良前
28コア



改良後
28コア



OpenMP並列効率のさらなる向上3

- 第1世代Xeon Phi(Knights Corner, 5110P, 60コア, 1011GFLOPS)
- Taxol($C_{47}H_{51}NO_{14}$),cc-pVDZのHartree-Fock計算時間(sec)
- Taxol分子では、60スレッド以上で並列加速率と計算時間に差が出ている

	改良前		改良後	
スレッド数	計算時間	並列加速率	計算時間	並列加速率
15	9034.6	15.0	9057.9	15.0
30	4559.0	29.7	4565.8	29.8
60	2391.9	56.7	2302.6	59.0
120	2652.9	51.1	1592.9	85.3
240	2888.6	46.9	1333.5	101.9

参考：Xeon (E5-2698 v3, 32コア, 972GFLOPS)
32スレッドで345.9sec

メモリ使用量削減1

- 演算量を増やしてでも、使用メモリ量を削減
- MP2エネルギー計算では、AO2電子積分を複数回計算することで削減可

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

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

ε_i: 軌道エネルギー
$C_{\mu a}$: 分子軌道係数
i, j: 占有軌道
a, b: 非占有軌道

```
do μλ (AO index pair)
  do σ
    AO積分計算      (μν|λσ)   (all ν)
    第1変換         (μi|λσ)   (all i)
  enddo
  第2変換(dgemm)  (μi|λj)    (i≧j)
end do μλ
do ij (MO index pair)
  MPI_sendrecv (μi|λj)
  第3変換(dgemm)  (μi|bj)    (all b)
  第4変換(dgemm)  (ai|bj)    (all a)
  MP2エネルギー計算
end do ij
call mpi_reduce(MP2エネルギー)
```

```
do ij-block
  do μλ (AO index pair)
    do σ
      AO積分計算      (μν|λσ)   (all ν)
      第1変換         (μi|λσ)   (partial i)
    enddo
    第2変換(dgemm)  (μi|λj)    (partial j)
  end do μλ
  do partial-ij (MO index pair)
    MPI_sendrecv (μi|λj)
    第3変換(dgemm)  (μi|bj)    (all b)
    第4変換(dgemm)  (ai|bj)    (all a)
    MP2エネルギー計算
  end do ij
end do ij-block
call mpi_reduce(MP2エネルギー)
```

メモリ使用量削減2

- 全プロセス合計の使用メモリ量は$O^2N^2/2$ → $O^2N^2/(2n_{ij-block})$ (O:占有軌道数、N:基底関数次元数, $n_{ij-block}$:占有軌道分割数, (μi|λj)保存用配列)
  - 計算条件:Taxol ($C_{47}H_{51}NO_{14}$)、MP2/6-31G(d) (O=164軌道、N=970基底)
  - 計算機:Fujitsu PRIMEGY CX2500 (Xeon E5-2697, 28コア/ノード)
  - MP2エネルギー計算では、メモリ使用量削減のための追加演算コストは比較的小さい

計算時間(秒)とメモリ使用量(GB)

占有軌道分割数	1	2	3
MP2計算時間	765.7	943.4	1121.5
そのうちAO2電子積分と第1変換	618.8	803.6	981.2
メモリ使用量	101	51	34

計算時間
20%, 40%増加

メモリ使用量
1/2, 1/3へ減少

メモリ使用量削減3

- MP2エネルギー微分計算の並列加速率
  - 計算条件：$C_{150}H_{30}$、MP2/cc-pVDZ
  - 計算機：「京」
  - 使用ノード数が増えるほどトータルのメモリ量は増えて、占有軌道分割数が減るため、使用ノード数以上の並列加速率が得られた

計算時間(秒)と並列加速率

ノード数	192	384	768	1536
占有軌道分割数	3	2	1	1
MP2微分実行時間	4253.6	2002.0	743.6	382.0
並列加速率	192.0	407.9	1098.2	2137.9

通信時間削減1

- 演算と通信のオーバーラップにより、通信時間を削減
- ノンブロッキング通信MPI_Isend,MPI_Irecvを使用

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

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

ε_i: 軌道エネルギー
$C_{\mu a}$: 分子軌道係数
i, j: 占有軌道
a, b: 非占有軌道

```
do μλ (AO index pair)
  do σ
    AO積分計算      (μν|λσ)   (all ν)
    第1変換         (μi|λσ)    (all i)
  enddo
  第2変換(dgemm)  (μi|λj)     (i≧j)
end do μλ
do ij (MO index pair)
  MPI_sendrecv (μi|λj)
  第3変換(dgemm)   (μi|bj)     (all b)
  第4変換(dgemm)   (ai|bj)     (all a)
  MP2エネルギー計算
end do ij
call mpi_reduce(MP2エネルギー)
```

```
do μλ (AO index pair)
  do σ
    AO積分計算      (μν|λσ)   (all ν)
    第1変換         (μi|λσ)    (all i)
  enddo
  第2変換(dgemm)  (μi|λj)     (i≧j)
end do μλ
MPI_sendrecv (初めの(μi|λj))
do ij (MO index pair)
  MPI_isend, irecv (次の(μi|λj))
  第3変換(dgemm)   (μi|bj)     (all b)
  第4変換(dgemm)   (ai|bj)     (all a)
  MP2エネルギー計算
  MPI_Waitall (次の(μi|λj))
end do ij
call mpi_reduce(MP2エネルギー)
```

通信時間削減2

- 「京」96, 192ノードで$C_{150}H_{30}$(cc-pVDZ, 2250基底)のMP2エネルギー計算時間
  - 演算と通信をオーバーラップさせることで、通信時間を大幅に削減
  - 前述の占有軌道分割数は96ノードでは2、192ノードでは1

MP2実行時間と通信時間(秒)

	ノード数	96	192
改良前	MP2実行時間	1089.9	421.9
	(うち通信時間)	9.7	8.7
改良後	MP2実行時間	1082.2	416.7
	(うち通信時間)	5.2	3.4

通信時間半減

まとめ

- 並列計算アルゴリズム開発では、計算負荷分散、データ分散、通信コスト削減、高速化を同時に考慮して行う必要があり、計算式から考え直す場合もある
- MPI/OpenMPハイブリッド並列は、現在の計算機では必要不可欠
- 並列化効率向上のためには、初期値計算も含めたすべての計算を並列化する必要がある
- アルゴリズム・プログラム開発コストはますます上昇する可能性が高く、分野全体でのコスト削減と情報共有が重要になり、そのための手段としてオープンソースでの公開が挙げられる
- これからのメニーコア時代の計算機を効率的に使うためには、特にOpenMP並列効率、通信時間の削減、メモリ使用量の削減が重要になると考えられる

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
- 計算科学技術
  - “計算科学のためのHPC技術1・2” 下司雅章編
- BLAS、LAPACK
  - http://www.intel.co.jp/jp/software/products/