SMASHと量子化学

石村　和也
（分子科学研究所　計算分子科学研究拠点(TCCI)）

【配信講義】CMSI計算科学技術特論C 第11回
2015年12月10日

本日の内容

- 自己紹介
- SMASHプログラムについて
- 量子化学分野のプログラム紹介
- 開発を始めた経緯
- SMASHのアルゴリズムとパフォーマンス
- プログラム公開
- 今後の開発の課題
- まとめ

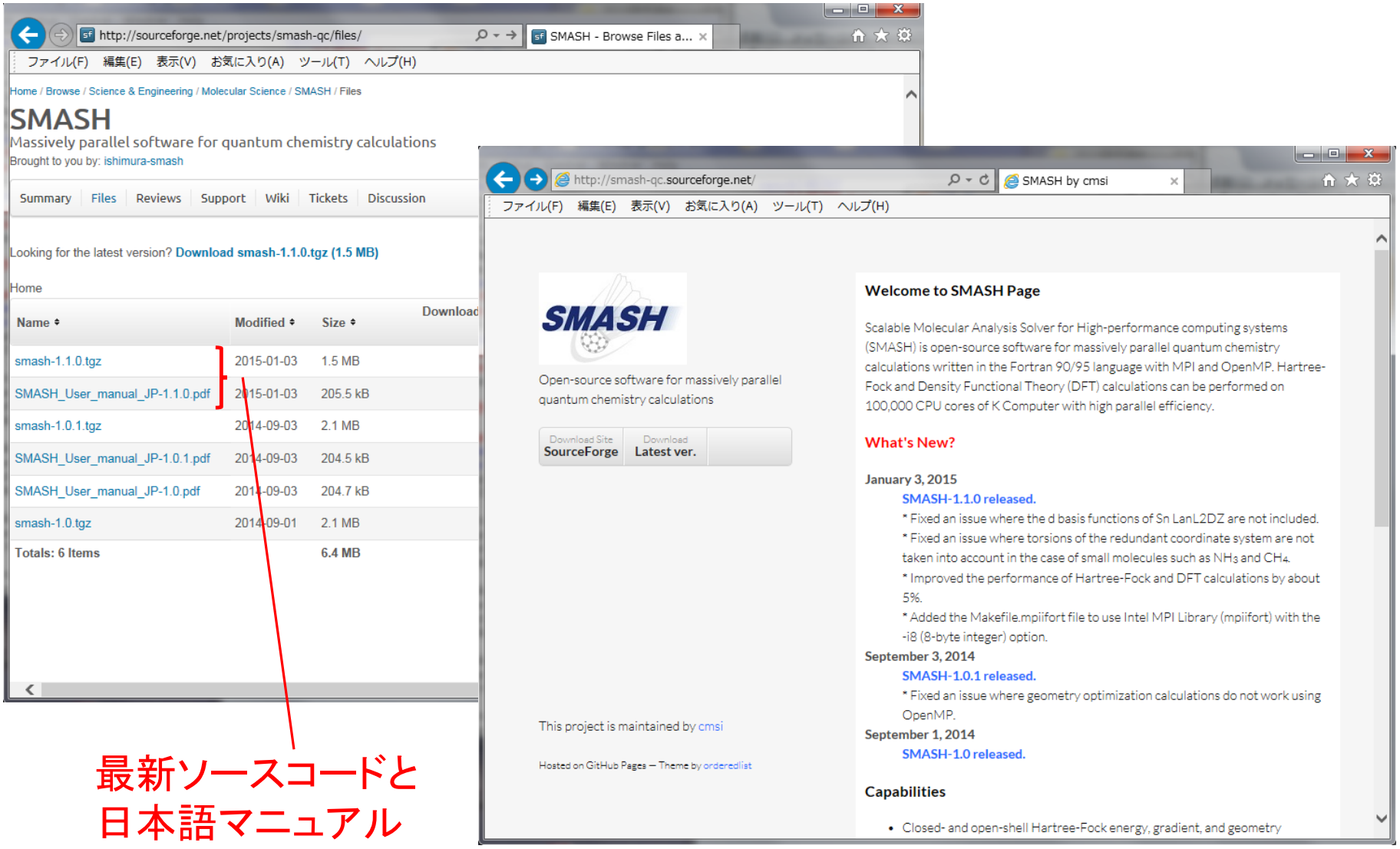
自己紹介

- 計算機は大学の授業で初めて触った
- 学部3年の前半までは実験に進む予定だったが、後半で受けた授業で理論・計算に興味を持ち、方針転換をした
- 学部4年で研究室に所属してから、本格的に計算機を使い始めた
- 学生時代、プログラム開発は量子化学計算プログラムGAMESS、HONDOをベースに行っていた
- 学位取得までは、PCクラスタ程度の高速化・並列化までしか行っていなかった
- その後、スパコンの利用を始めた(Cray XT4, XT5 (GAMESS) -> 京コンピュータ (GELLAN))
- CMSI・TCCIに所属してから(2012年4月)は、SMASHプログラムの開発を始めた

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1年間で、ダウンロード数は309(海外からは約20%)

SMASH ウェブサイト



最新ソースコードと
日本語マニュアル

量子化学計算プログラム

- Gaussian (Linda+OpenMP)
- GAMESS (MPI(DDI)、無料(登録必要)、二次配布禁止)
- NWChem (MPI+OpenMP, Educational Community License version 2.0)
- Molpro (MPI+OpenMP)
- ACES (MPI+OpenMP, GPL)
- NTChem (MPI+OpenMP)
- GELLAN (MPI+OpenMP)
- ProteinDF (MPI+OpenMP)
- OpenFMO(MPI+OpenMP)
- 他にも多数

- サポートしている計算方法・収束方法・解析手法の数、シンプルな入力形式、サンプルインプットとマニュアルの整備、GUI対応など、ユーザー視点ではGaussianが最も使いやすい

Gaussianについて

- Gaussian09のソースコードは180万行以上
- 非常に多くの研究者が開発に関わっている
- M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, B. Mennucci, G. A. Petersson, H. Nakatsuji, M. Caricato, X. Li, H. P. Hratchian, A. F. Izmaylov, J. Bloino, G. Zheng, J. L. Sonnenberg, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. Bearpark, J. J. Heyd, E. Brothers, K. N. Kudin, V. N. Staroverov, T. Keith, R. Kobayashi, J. Normand, K. Raghavachari, A. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, N. Rega, J. M. Millam, M. Klene, J. E. Knox, J. B. Cross, V. Bakken, C. Adamo, J. Jaramillo, R. Gomperts, R. E. Stratmann, O. Yazyev, A. J. Austin, R. Cammi, C. Pomelli, J. W. Ochterski, R. L. Martin, K. Morokuma, V. G. Zakrzewski, G. A. Voth, P. Salvador, J. J. Dannenberg, S. Dapprich, A. D. Daniels, O. Farkas, J. B. Foresman, J. V. Ortiz, J. Cioslowski, and D. J. Fox, Gaussian, Inc., Wallingford CT, 2013.
- MPI版開発の噂は聞くが、詳しいことはわからない
- これだけ大きなプログラムになると、大規模並列化の対応は非常に困難だと思われる
- GUI GaussViewは非常に便利(入力作成、出力表示)
- 分子研計算センターのスパコンには、キューイングシステムの通常のジョブ投入コマンド(jsub)以外に、Gaussian用の投入コマンドg09subも用意 されている(実験研究者の利用も多い)

SMASH開発前の取り組み

(高速化)新たな原子軌道2電子積分計算アルゴリズム及び
(並列化)Hartree-Fock計算のMPI/OpenMPハイブリッド計算アルゴリズムの開発と
GAMESSプログラムへの実装

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

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

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

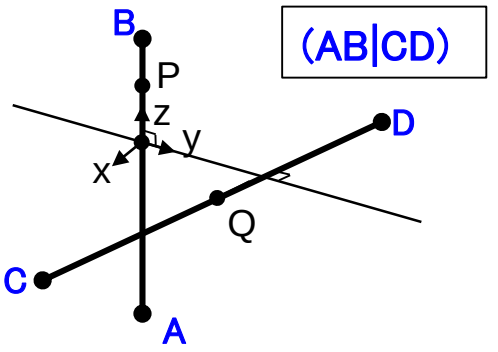
計算終了

新たな2電子積分計算アルゴリズム作成

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

- 原子軌道2電子積分は、大半の量子化学計算で必要

$$(\mathrm{AB}|\mathrm{CD}) = \int \mathrm{d}\mathbf{r}_1 \mathrm{d}\mathbf{r}_2 \chi_\mathrm{A}(\mathbf{r}_1)\chi_\mathrm{B}(\mathbf{r}_1)\frac{1}{\mathrm{r}_{12}}\chi_\mathrm{C}(\mathbf{r}_2)\chi_\mathrm{D}(\mathbf{r}_2)$$



- Pople-Hehre法

  座標軸を回転させることにより演算量を減らす

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

  x_{PQ} = constant, y_{PQ}= constant

- McMurchie-Davidson法

  漸化式を用いて(ss|ss)型から軌道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

- (ss|ss)から(dd|dd)までの21種類の積分計算ルーチン約2万行のコードをFortranとPerlで自動作成した
- GAMESSに実装したところ2-4割計算時間短縮でき、2005年から正式に導入されている

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

- MPIはすでにGAMESSに導入されていたので、追加のコストはほとんどなかった
- OpenMPは導入されていなかったので、大幅なコードの書き換えを行った
  - OpenMP領域の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

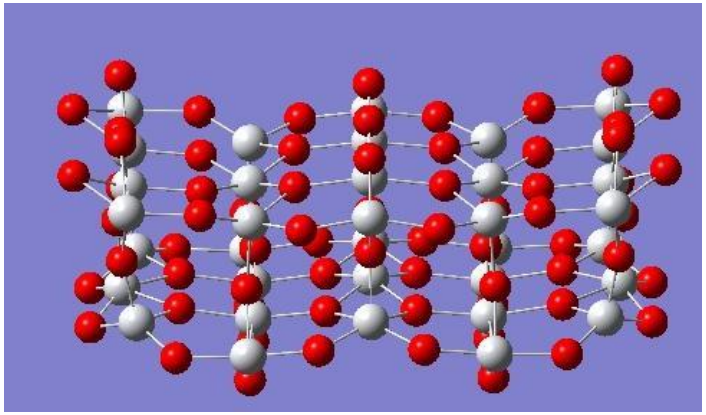
D. Cremer and J. Gauss, *J. Comput. Chem.* **7** 274 (1986).

- 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エネルギー並列計算条件

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)



Hartree-Fockエネルギー並列計算結果

全計算時間の
並列加速率

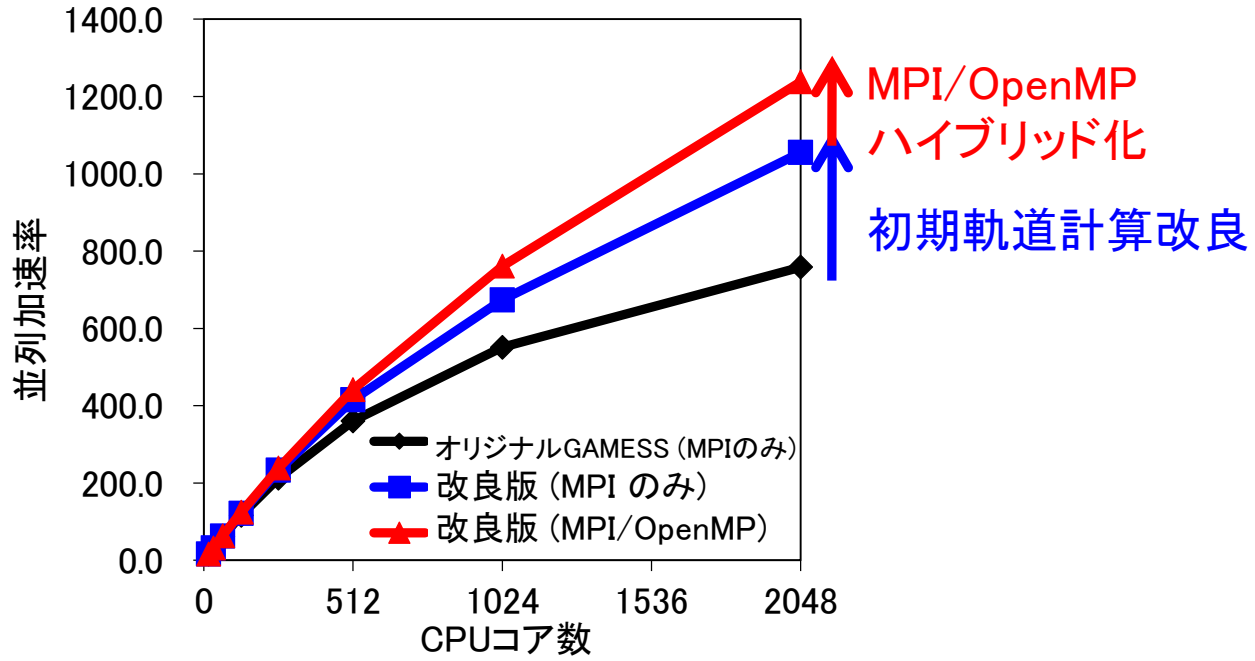


Table 全計算時間(秒)と並列加速率(カッコ内)

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)**

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(プログラム構造、変数の取り方)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方針

- 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実装し、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、おそらくポスト京も研究室レベルの計算機と基本的な構成は似ているはず
- 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プログラミングルール(2015年12月10日時点)

1. 言語はFortran90/95で、並列化はMPIとOpenMPを利用する。
2. コンパイラの診断オプションを使ってコードをチェックする。
   - ex) ifort -warn all -check all
   - gfortran -Wall -fbounds-check
3. プロセス間のデータ通信はsrc/parallel.F90のMPIラッパーを利用する。直接MPIルーチンを呼ばない。
4. MPIを使わずにコンパイルする場合はマクロ(-DnoMPI)を利用する。
5. implicit noneを利用し、すべての変数を定義する。整数はi-nで始まる変数にする。
6. 配列をallocate文で確保する場合、call memsetで利用メモリ量をカウントする。開放するときは、call memunsetで開放する量を差し引く。
7. module変数は、parameter、threshold、inputデータなど計算中変更しないもののみとする。(例外:座標はmodule変数とする)(将来的には構造体を利用して引数渡しにして、module変数を削除する予定)
8. プログラムを止める場合、エラーメッセージを出力してcall iabortを書く。
9. inputの読み込み、checkpointファイルの読み書きを行うサブルーチンはfileio.F90に書く。
10. ディスクへの書き込みは、標準出力とcheckpointファイルの書き込みのみ。
11. 引数で渡された配列サイズは明示する

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
- 微分計算では、適切な係数と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

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

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

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

MPIプロセス並列ラッパールーチン

- MPIのラッパールーチンをparallel.F90まとめている
- use mpi、include ”mpif.h”はparallel.F90内だけで使用する
- ルーチン名は、para_＊
- マクロを使い、MPIを使わない場合でもmakeできるようにしている (-DnoMPI)
- MPIを使わない場合、何もしないか、必要ならデータコピーを行う
- 整数の送受信では、整数のサイズをmoduleのmodparallelにあるinterface checkintsizeで自動判定している
- 例えば、MPI_Bcastのラッパーは、para_bcastr(実数)、para_bcasti(整数)、para_bcastc(文字)、para_bcastl(ロジカル)の4つ

MPI communicator

- MPIコミュニケータを2種類用意している (mpi_comm1, mpi_comm2)
- mpi_comm1, nproc1, myrank1
  - mpi_comm1 = MPI_COMM_WORLD
  - インプットデータの読み込みと送受信、Fock行列計算の分散、MP2計算の分散など、大半の演算分散とプロセス間通信で利用
- mpi_comm2, nproc2, myrank2
  - DIIS、SOSCF、行列対角化、基底直交化変換の行列積など、全体で分散させるには演算量が少ない場合に、ある程度のプロセス数ごとに同じ計算をするために用意している
  - 公開版では今のところmpi_comm2 = mpi_comm1

入力形式

- できるだけシンプルに、Gaussianに近い形式にしている
- subroutine readinputの冒頭でインプットファイルを右のように変換したファイル作成し、この変換ファイルの内容を読み込んでいる
- 変換されたファイルがインプットの場合は、何もしない

インプットファイル

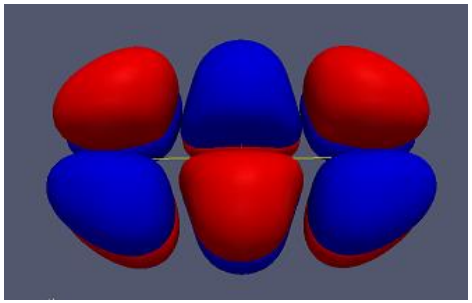
```
job method=mp2 basis=gen
control check=test.chk
scf dconv=1.0d-4
geom
 O   0.000    0.000    0.142
 H   0.000    0.756   -0.462
 H   0.000   -0.756   -0.462

basis
O H
6-31G(d)
****
```

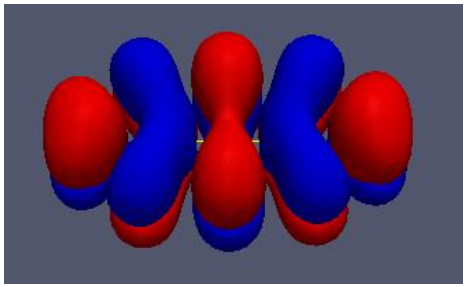
input.dat(プロセス番号)

```
&JOB METHOD='MP2' BASIS='GEN' /
&CONTROL CHECK='test.chk'  /
&SCF DCONV=1.0D-4 /
GEOM
O   0.000    0.000    0.142
H   0.000    0.756   -0.462
H   0.000   -0.756   -0.462

BASIS
O H
6-31G(D)
****
```

可視化

- smash/vtkディレクトリにvtkファイルを作るコードを用意
  - ParaViewで分子軌道を表示





- 次のバージョンでは、Gaussian cubeファイルを作るプログラムを用意
  - 様々な量子化学計算用可視化ソフトウェアで分子軌道を表示

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)
```

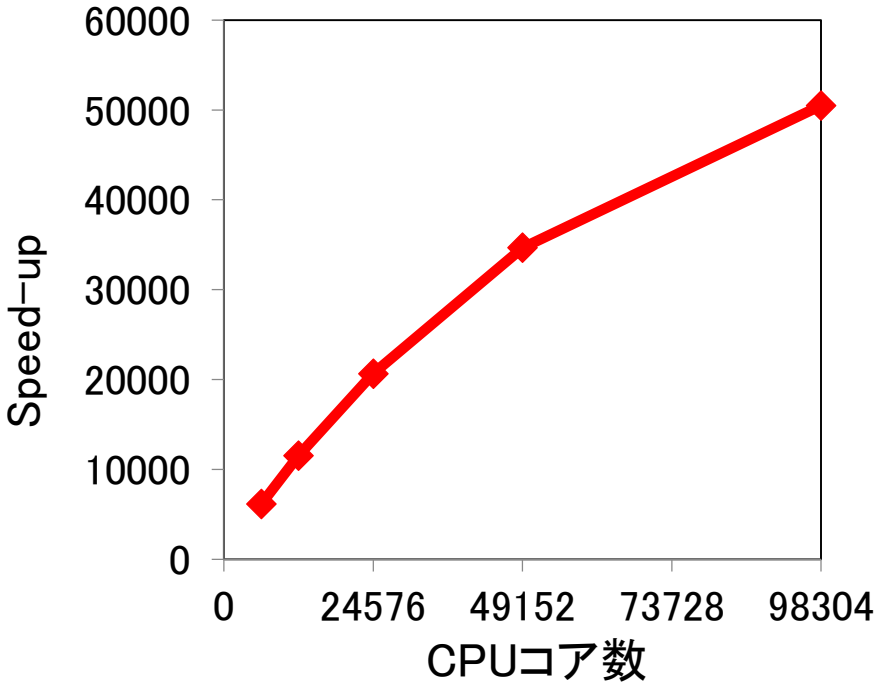
$$E_{MP2} = \sum_{ij}^{occ}\sum_{ab}^{vir} \frac{(ai|bj)\{2(ai|bj) - (aj|bi)\}}{\varepsilon_i + \varepsilon_j - \varepsilon_a - \varepsilon_b}$$

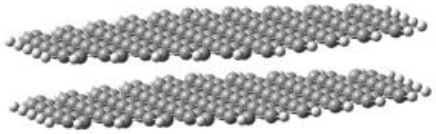
$$(ai|bj) = \sum_{\mu\nu\lambda\sigma}^{AO} C_{\mu a} C_{\nu i} C_{\lambda b} C_{\sigma j} (\mu\nu|\lambda\sigma)$$

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

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
- 密行列対角化(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

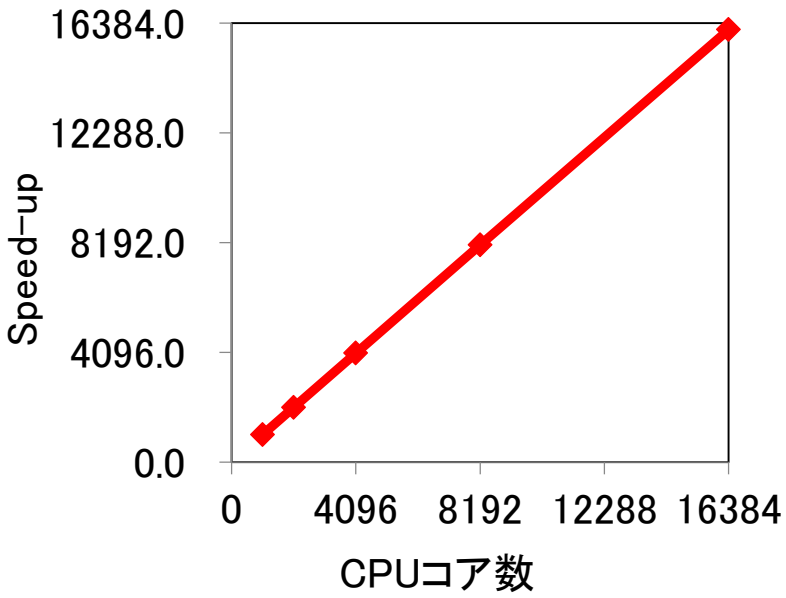
GAMESSとの比較

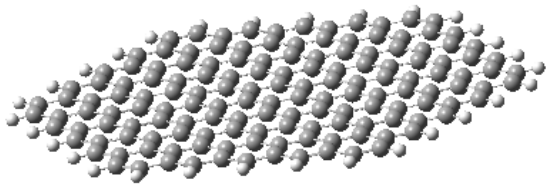
- Xeon E5649 2.53GHz 12core、1ノード利用
- Taxol($C_{47}H_{51}NO_{14}$)のHartree-Fock計算時間(sec)
- 同じ計算条件(積分Cutoffなど)

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%



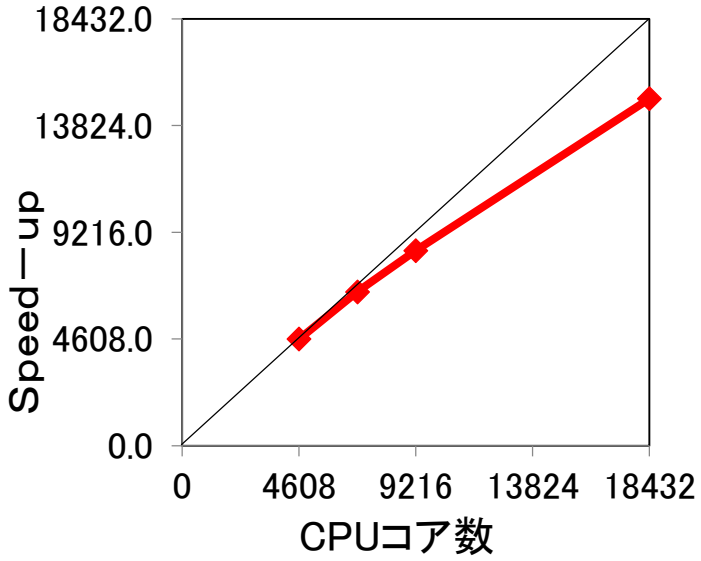


計算機：京コンピュータ
分子　：($C_{150}H_{30}$)
基底関数：cc-pVDZ (2250 functions)
計算方法：B3LYP

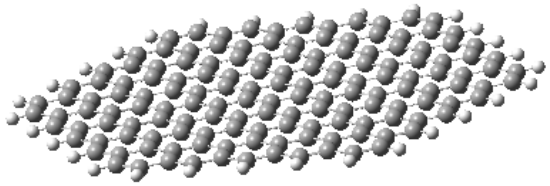
CPUコア数	1024	4096	8192	16384
微分計算時間(秒)	402.0	101.2	50.8	25.5
並列加速率	1024.0	4067.7	8103.3	16143.1

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
- 省メモリ版は次バージョンで公開
- 現在、MP2エネルギー微分並列計算アルゴリズムを開発中



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



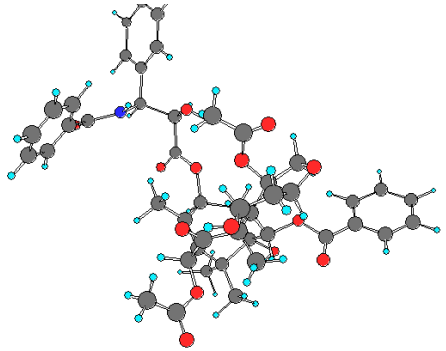
CPUコア数	4608	6912	9216	18432
MP2計算時間(秒)	152.5	105.7	83.4	46.9
並列加速率	4608.0	6648.2	8425.9	14983.4

構造最適化回数の削減

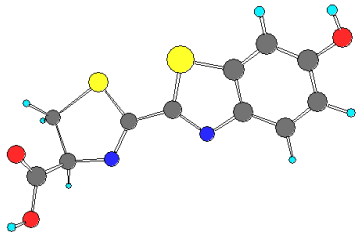
- 分子の結合、角度、二面角の情報を使うRedundant座標と力場パラメータを使い、初期ヘシアンを改良することで最適化回数が1/5から1/6になった
- Cartesian座標での初期ヘシアンは単位行列を利用
- 2サイクル目以降のヘシアンはBFGS法で更新
- 実用的な計算では、最適化回数削減は非常に重要

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($C_{11}H_8N_2O_3S_2$)

プログラム公開

- どの段階で公開するかは難しい
- CMSI、特にMateriAppsからのプレッシャーのおかげでSMASHは公開のタイミングを得た
- オープンソースライセンスは開発当初から考えていた
  - CMSIでの開発
  - 技術の共有、開発の効率化
- Webサイトは、MateriApps LIVE!を参考にした（GitHubのgh-pagesを使った）
- 入力形式はシンプルにして、サンプルインプットを複数用意することで、マニュアルは最小限にした
- Webサイト、同封マニュアルは英語だが、国内ユーザーのハードルを少しでも下げるために日本語マニュアルも用意した

公開による広がり

- 公開から1年間でダウンロード数は309 (海外から約20%)
- SMASHの一部ルーチンが他のプログラムへ移植され始めている
- SMASHを用いた共同研究がいくつか始まっている
- 公開前に試していない環境(OS、コンパイラ)でのトラブルの連絡をいくつか受け取り、それらを修正して新バージョンを公開した
- 巨大な分子の計算をしている人から、SCF収束の問題の連絡が複数あり、現在対応中である
- ソースコードを含めた公開で、情報共有が量子化学分野を超えて広がっていると思われる
- 計算機科学分野からの問い合わせもあった
- 計算できる方法がまだ少ないため、問い合わせはあまり多くなく、現時点では一人で対応できている

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幅の増加
- 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)
```

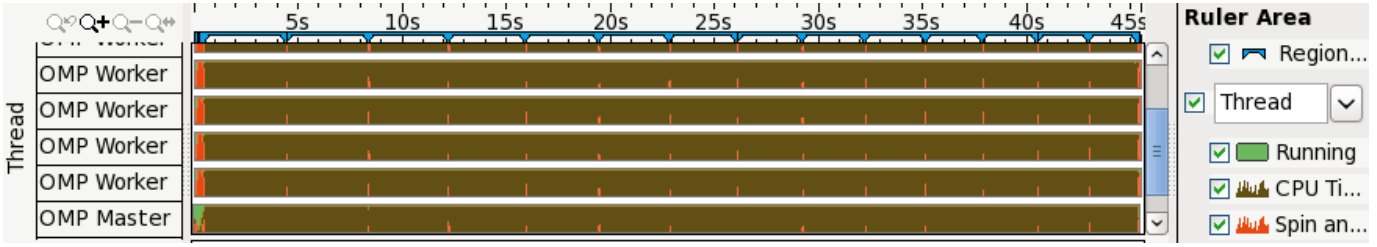
改良後

```
!$OMP parallel do schedule(dynamic,1) &
!$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)
```

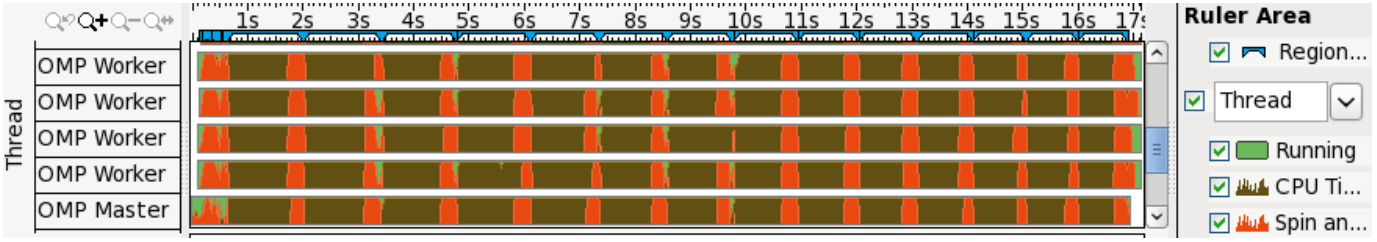
OpenMP並列効率のさらなる向上2

- 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

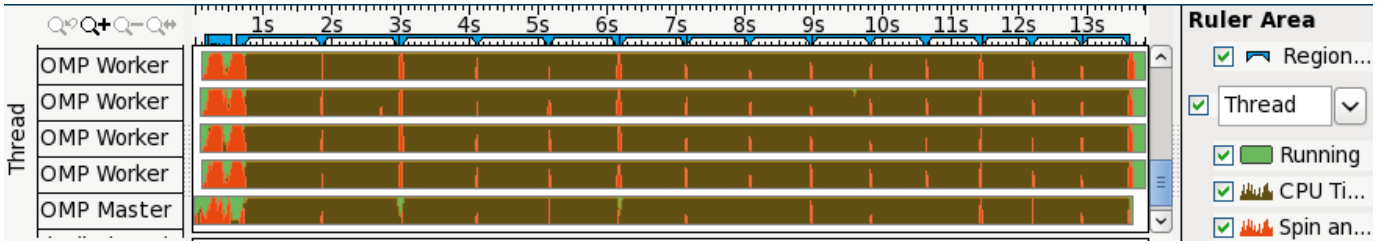
改良前
8コア



改良前
28コア



改良後
28コア



メモリ使用量削減1

- 演算量を増やしても、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

- 前ページの左のアルゴリズムでは、全プロセス合計の使用メモリ量は$O^2N^2/2$ (O:占有軌道数、N:基底関数次元数, $(\mu i|\lambda j)$配列)
- 右のメモリ使用量削減アルゴリズムでは、合計で$O^2N^2/(2n_{ij-block})$ ($n_{ij-block}$:ijペア分割数)
  - 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)
  - 分割しない場合、メモリ使用量は101GB
  - MP2エネルギー計算では、メモリ使用量削減のための追加コストは比較的小さい

計算時間(秒)とメモリ使用量(GB)

$n_{ij-block}$	1	2	3
Hartree-Fock計算時間	206.3	206.1	205.7
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

通信時間削減

- 前述のMP2アルゴリズムのプロセスあたりの通信量は$O^2N^2/(2n_{proc})$ (O:占有軌道数、N:基底関数次元数, n_{proc}:プロセス数, $(\mu i|\lambda j)$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, Infiniband FDR)
  - 今回の条件では、MP2計算のうち1割程度が通信時間
  - 現在はブロッキング通信MPI_sendrecvを使っているが、今後は演算と通信をオーバーラップさせるため、非ブロッキング通信MPI_isend,irecvに変更する必要がある

計算及び通信時間(秒)

n_{proc}	2	4
MP2全体	394.7	207.2
そのうち通信時間	33.2	24.5

人材育成

- ポスト京は京よりも取り組むべき内容が多く、モジュール化、ライブラリ化により分野内での共有で開発コスト削減を行うだけでは不十分
- 開発できる人材を増やすことも重要
- 特に、若手をどのように育てるか
  - 論文・本を読んで自ら成長してもらうだけではなく、CMSIで行っているような授業や実習(合宿)で引き上げることも必要
  - 放っておいても数年あれば京レベルには到達するが、それではポスト京には間に合わない
  - 成果(アルゴリズムとプログラム開発)は、化学・物理分野以外に情報系にも積極的にアピールすべき

これまでの人材育成の取り組み1

- 2012年度CMSI若手技術交流会
  - 第6回 (2012年7月17日-19日　掛川)
    - CMSI研究員・企業研究者・学生28人、サポート富士通4人
    - 内容：FX10で各自のプログラムのコンパイルとチューニングもしくはフリーソフトのコンパイル
    - 成果：コンパイル・チューニングに関するwikiへの書き込み30件
      FX10もしくは京での各自のプログラムのコンパイルは、ほぼ成功
  - 第7回 (2013年2月14日-16日　金沢)
    - CMSI研究員・学生30人、サポートRIST4人と富士通4人
    - 内容：FX10で各自のプログラムの解析とチューニング、
      学生はFX10で各自のプログラムのコンパイル
    - 成果：チューニング成果のwikiへの書き込み18件
      計算時間を3割以上削減できた参加者　5名(2割)以上

これまでの人材育成の取り組み2

- 2013年度CMSI-TOKKUN!1-3, 2014年度TOKKUN!4-5
  - 京を含むHPCI一般利用枠申請に向けて毎回テーマを決めて、少人数(10名程度)で高度化作業 (実行性能向上、並列性能向上、実行・並列性能向上など)
  - 富士通から毎日2名応援
  - 毎回最後に成果発表と情報共有
  - 長時間、富士通の方と議論してアルゴリズムから見直すことができた参加者もいた
  - ソースコードを大画面に映して、全員でのチューニング作業も行った
  - 3組が2014年度京一般枠採択
- 2015年度TOKKUN!6
  - 主な対象を学生に変更して高度化支援
  - Intelマシンと解析ツールVtuneを使って参加者のプログラムの高速化と並列化
  - 参加者10名のうち学生5人
  - 5人が計算時間短縮に成功
- 毎回、全員が集まり情報共有を行う時間を設けた
- 現在、若手技術交流会とTOKKUN!の成果を1か所にまとめる作業を進めている

まとめ

- 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
- 公開により、様々な意見、批判、協力が届き、さらなる開発の原動力になっている
- 並列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
- 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
- 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、通信コストの削減、メモリ使用量の削減が重要になると考えられる
- 次世代の人材育成を効率的に行わなければならない