ABINIT-MP

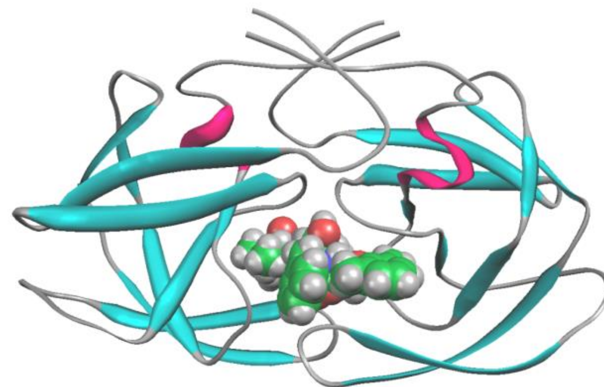
FMOプログラムABINIT-MPの「不老」Type I 向け改良について

〇望月祐志[1,2]（ABINIT-MP取り纏め役）
中野達也[3]、坂倉耕太[4]、渡邊啓正[5]、土居英男[1]
片桐孝洋[6]
1:立教大、2:東大生研、3:国立衛研、4:FOCUS
5:HPCシステムズ、6:名大

謝辞：JHPCN jh210036-NAH&jh2200010課題 / 富士通SS研-A64FXアプリWG

FMO法とABINIT-MPプログラム

フラグメント分子軌道(FMO)法



【HIVプロテアーゼとロピナビル】

◇巨大分子系

生体高分子や凝集系では一般的

⇒ タンパク質、DNA （水和状態）

数千～数万原子、数千～数十万軌道

◇分割&統合系のアプローチの一つ

北浦らが23年程前に2体展開で提案

⇒ フラグメントとその対で系のエネルギーを評価 （FMO2）

⇒ 環境静電ポテンシャル（ESP）、直接結合切断（BDA）

⇒ 階層的な並列処理（フラグメントリスト&内部処理）

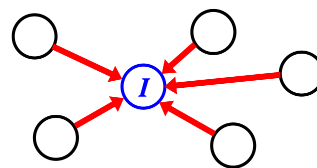
⇒ 定量性を高める電子相関の導入も直截

フラグメント間の相互作用エネルギー（IFIE）

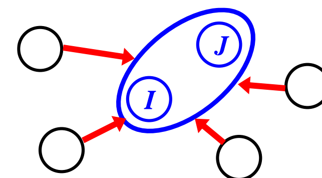
⇒ 計算対象の解析ツール

⇒ 生物物理や創薬に向く

⇒ 材料系にも適用可能



モノマー （アミノ酸単位など）



ダイマー

FMO計算のためのプログラム

◇GAMESS-US [米国Gordonグループ]; Fedorov、Gordon、北浦ら

GAUSSIANに抗し得る有力なフリーソフト、世界規模 (Fortran)

⇒ 様々な機能をFMO化、多彩な計算、GDDI並列

◇ABINIT-MP; 望月、中野ら

実用機能は十分、東大系PJ・CREST-PJなどで開発 (Fortran)

⇒ IOレス、MPI、OpenMP/MPI混成並列、スパコンと好相性

◇PAICS; 石川

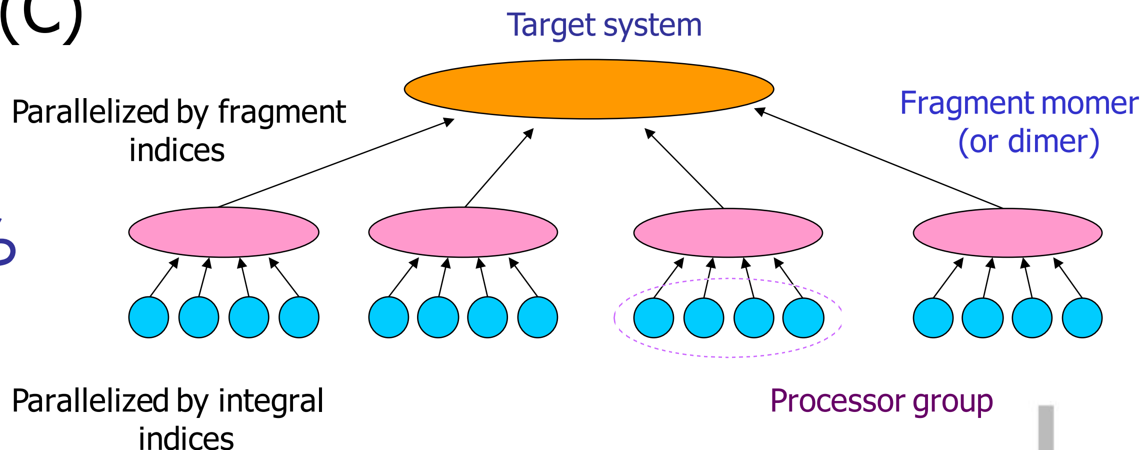
FMO-MP2(RI)に特化 (C)

⇒ MPI並列

◇OpenFMO; 稲富、鬼頭ら

FMO-HF (C)

⇒ 超並列指向



Ref.; "The ABINIT-MP program", Y. Mochizuki et al. in "*Recent Advances of the Fragment Molecular Method*" (Springer, 2021).

ABINIT-MPの主な機能（オープンシリーズとして整備中）

・エネルギー
- → FMO4: HF, MP2
- → FMO2: HF～CCSD(T), LRD
- → FMO2: CIS/CIS(D)

・エネルギー微分
- → FMO4: HF, MP2
- → FMO2: MP2構造最適化, MD

・その他機能
- → SCIFIE, PB, sp2-BDA, α(ω)
- → 電子密度生成, CAFI, FILM

・並列化環境（PC~スパコン）
- → MPI, OpenMP/MPI混成
- → 最深部はBLAS処理

http://www.cenav.org/abinit-mp-open_ver-2-rev-4/

レポート　ニュース　解析事例データベース　ライブラリ　Q&A　コミュニティー

ABINIT-MP Openシリーズ (Ver.2 Rev.4)

※2020年6月版(Ver.1 Rev.22)に関するページはこちらです

はじめに

フラグメント分子軌道（FMO）計算のプログラムABINIT-MP [1,2]のOpenシリーズは、2021年度にVer. 2系に移行しました。Ver. 2系では、これまで通りの機能の拡充に加え、常用されるメラープレセット2次摂動（MP2）レベルのジョブの「富岳」（理研）などの富士通A64FX系スーパーコンピュータでの数倍を睨んだ高速化、ならびに将来的には数万フラグメントの系を扱える大規模化対応が進められています。

Open Ver. 1 （ポスト「京」のPJで整備）
- ・Rev. 5 (2016年12月)
- ・Rev. 10 (2018年2月)
- ・Rev. 15 (2019年3月)
- ・Rev. 22 (2020年6月); 当面は併存

Open Ver. 2 （「富岳」の時代に移行）
- ・Rev. 4 (2021年9月)
- ・Rev. 8 (2023年3月を予定)

（注記：Ver. 2系では、BioStation Viewerへのデータファイルの書出しを廃止した）

基本のHF計算

$\mathbf{F}^x\mathbf{C}^x=\mathbf{S}^x\mathbf{C}^x\boldsymbol{\varepsilon}^x$　　$\mathbf{F}^x=\mathbf{H}^x+\mathbf{G}^x$　⇔ HFの一般化固有値問題（要反復計算）

$H^x_{\mu\nu}=H^{core\ x}_{\mu\nu}+V^x_{\mu\nu}+\sum_k B_k\langle\mu|\theta_k\rangle\langle\theta_k|\nu\rangle$　　$V^x_{\mu\nu}=\sum_{K\neq x}\left(u^K_{\mu\nu}+v^K_{\mu\nu}\right)$　⇔ 1電子部分の修飾

$u^K_{\mu\nu}=\sum_{A\in K}\langle\mu|\left(-Z_A/|\mathbf{r}-\mathbf{A}|\right)|\nu\rangle$　　$v^K_{\mu\nu}=\sum_{\lambda\sigma\in K}P^K_{\lambda\sigma}(\mu\nu|\lambda\sigma)$　⇔ 環境静電ポテンシャル(ESP)

$P_{\mu\nu}=2\sum_{i=1}^{occ}C^*_{\mu i}C_{\nu i}$　　$G^x_{\mu\nu}=\sum_{\lambda\sigma\in x}P^x_{\lambda\sigma}\left[(\mu\nu|\lambda\sigma)-\frac{1}{2}(\mu\sigma|\lambda\nu)\right]$　⇔ 2電子部分　O(N^4)（並列処理）

高速化のための幾つかの工夫

$V^L_{\mu\nu}\cong\sum_{\lambda\in L}(\mathbf{P}^L\mathbf{S}^L)_{\lambda\lambda}(\mu\nu,\lambda\lambda)$　for $R_{\min}(X,L)\geq L_{\text{aoc}}$　⇔ ESP-AOC近似（実用精度高し）

$V^L_{\mu\nu}\cong\sum_{A\in L}\langle\mu|\left(Q_A/|\mathbf{r}-\mathbf{A}|\right)|\nu\rangle$　for $R_{\min}(X,L)\geq L_{ptc}$　　$Q_A=\sum_{\lambda\in A}(\mathbf{P}^L\mathbf{S}^L)_{\lambda\lambda}$　⇔ ESP-PTC近似（速い）

$E'_{IJ}\cong E'_I+E'_J+\mathrm{Tr}\left(\mathbf{P}^I\mathbf{u}^J\right)+\mathrm{Tr}\left(\mathbf{P}^J\mathbf{u}^I\right)+\sum_{\mu\nu\in I}\sum_{\lambda\sigma\in J}\mathbf{P}^I_{\mu\nu}\mathbf{P}^J_{\lambda\sigma}(\mu\nu|\lambda\sigma)$　⇔ Dimer-ES近似（HF計算無）

種々の工夫により、FMO2の計算コストのシステムサイズ依存性は2乗より低い

FMO-HF計算の流れ

モノマーの段階で
自己無撞着電荷
を課すのが特徴

分子をフラグメントに分割し、電子を割り当てる

↓

モノマーの初期電子密度を計算する

↓

与えられた電子密度を用いて、周囲のモノマーによる環境静電ポテンシャル下での、モノマーのエネルギーと電子密度を計算する

↓

与えられた電子密度と得られた電子密度の差が閾値よりも小さい

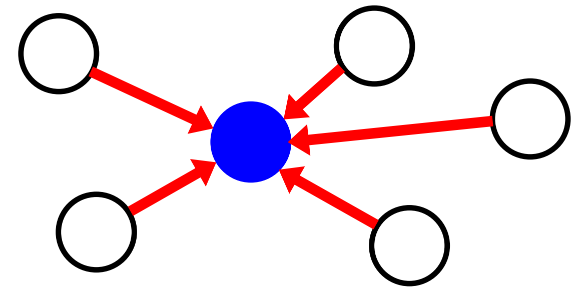
No → (前のステップへ戻る)

Yes ↓

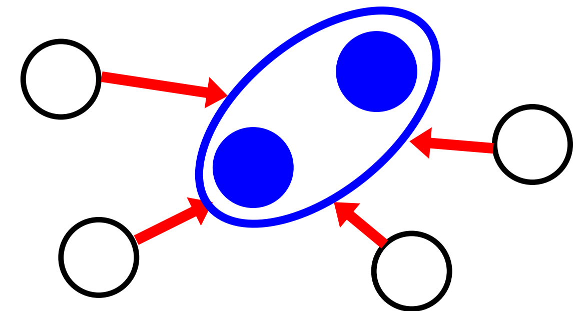
収束したモノマーの電子密度(Self-Consistent Charge; SCC)を用いて、周囲のモノマーによる環境静電ポテンシャル下での、ダイマーのエネルギーと電子密度を計算する

↓

分子の全エネルギーおよび全電子密度を計算



モノマー



ダイマー

Ref.; S. Obara et al., *J. Chem. Phys.* **84** (1986) 3963.

2電子積分の生成について#1

- 小原のVertical Recurrence Relation (VRR)がベース
- ジェネレータでF90コード群を自動生成 （微分も）

中野氏

TABLE I. Recurrence expressions[a] for the electron repulsion integrals over s and p Cartesian Gaussian functions.

$$(ss,ss)^{(0)} = (\zeta+\eta)^{-1/2} K(\zeta_a,\zeta_b,\mathbf{A},\mathbf{B}) K(\zeta_c,\zeta_d,\mathbf{C},\mathbf{D})\, F_0(T)$$

$$(p_i s,ss)^{(0)} = (P_i - A_i)(ss,ss)^{(0)} + (W_i - P_i)(ss,ss)^{(1)}$$

$$(p_i s,p_k s)^{(0)} = (Q_k - C_k)(p_i s,ss)^{(0)} + (W_k - Q_k)(p_i s,ss)^{(1)} + \frac{\delta_{ik}}{2(\zeta+\eta)}(ss,ss)^{(1)}$$

$$(p_i p_j,ss)^{(0)} = (P_j - B_j)(p_i s,ss)^{(0)} + (W_j - P_j)(p_i s,ss)^{(1)} + \frac{\delta_{ij}}{2\zeta}\{(ss,ss)^{(0)} - \frac{\rho}{\zeta}(ss,ss)^{(1)}\}$$

$$(p_i p_j,p_k s)^{(0)} = (Q_k - C_k)(p_i p_j,ss)^{(0)} + (W_k - Q_k)(p_i p_j,ss)^{(1)} + \frac{1}{2(\zeta+\eta)}\{\delta_{ik}(sp_j,ss)^{(1)} + \delta_{jk}(p_i s,ss)^{(1)}\}$$

$$(p_i p_j,p_k p_l)^{(0)} = (Q_l - D_l)(p_i p_j,p_k s)^{(0)} + (W_l - Q_l)(p_i p_j,p_k s)^{(1)} + \frac{1}{2(\zeta+\eta)}\{\delta_{il}(sp_j,p_k s)^{(1)} + \delta_{jl}(p_i s,p_k s)^{(1)}\} + \frac{\delta_{kl}}{2\eta}\{(p_i p_j,ss)^{(0)} - \frac{\rho}{\eta}(p_i p_j,ss)^{(1)}\}$$

$(i,j,k,l = x,y,z)$

[a] For the definition of the variables, see the text.

SCHEME I.

```
(First step)
    DO ICS = 1,n_CS     loops for the contracted shells
    DO JCS = 1,ICS
    DO IPS = 1,m_ICS    loops for the primitive shells
    DO JPS = 1,m_JCS
    The calculation of the parameters P, ζ, and K(ζ,ζ',R,R')
    for each pair of primitive shells
    CONTINUE

(Second Step)
    DO IPPS = 1,N_PPS   a loop for the first pair of primitive shells
    DO JPPS = 1, IPPS   a loop for the second pair of primitive shells
    The evaluation of ERI's
    CONTINUE
```

- Gauss型関数の角運動量の昇降を利用
- 並列化は短縮シェルの対のループで

$$G_{ijk:\alpha}(r) = N x^i y^k z^l \exp(-\alpha r^2)$$

2電子積分の生成について#2

- 軌道タイプの組み合わせに応じて個別のルーチンに
- 高い軌道角運動量のルーチンの最深部ループは長い
- 微分も同様に組み合わせ毎 （_gradが付く）

```
sub_dddd.F90
sub_dddf.F90
sub_dddp.F90
sub_ddds.F90
sub_ddfd.F90
sub_ddff.F90
sub_ddfp.F90
sub_ddfs.F90
sub_ddpd.F90
sub_ddpf.F90
sub_ddpp.F90
sub_ddps.F90
sub_ddsd.F90
sub_ddsf.F90
sub_ddsp.F90
sub_ddss.F90
sub_dfdd.F90
sub_dfdf.F90
sub_dfdp.F90
sub_dfds.F90
sub_dffd.F90
sub_dfff.F90
sub_dffp.F90
sub_dffs.F90
sub_dfpd.F90
sub_dfpf.F90
sub_dfpp.F90
sub_dfps.F90
sub_dfsd.F90
sub_dfsf.F90
sub_dfsp.F90
sub_dfss.F90
sub_dpdd.F90
sub_dpdf.F90
sub_dpdp.F90
sub_dpds.F90
sub_dpfd.F90
sub_dpff.F90
sub_dpfp.F90
sub_dpfs.F90
sub_dppd.F90
sub_dppf.F90
sub_dppp.F90
sub_dpps.F90
sub_dpsd.F90
sub_dpsf.F90
sub_dpsp.F90
sub_dpss.F90
sub_dsdd.F90
sub_dsdf.F90
sub_dsdp.F90
sub_dsds.F90
sub_dsfd.F90
sub_dsff.F90
sub_dsfp.F90
sub_dsfs.F90
sub_dspd.F90
sub_dspf.F90
sub_dspp.F90
sub_dsps.F90
sub_dssd.F90
sub_dssf.F90
sub_dssp.F90
sub_dsss.F90
sub_fddd.F90
sub_fddf.F90
sub_fddp.F90
sub_fdds.F90
sub_fdfd.F90
sub_fdff.F90
sub_fdfp.F90
sub_fdfs.F90
sub_fdpd.F90
sub_fdpf.F90
sub_fdpp.F90
sub_fdps.F90
sub_fdsd.F90
sub_fdsf.F90
sub_fdsp.F90
sub_fdss.F90
sub_ffdd.F90
sub_ffdf.F90
sub_ffdp.F90
sub_ffds.F90
sub_fffd.F90
sub_ffff.F90
sub_fffp.F90
sub_fffs.F90
sub_ffpd.F90
sub_ffpf.F90
sub_ffpp.F90
sub_ffps.F90
sub_ffsd.F90
sub_ffsf.F90
sub_ffsp.F90
sub_ffss.F90
sub_fpdd.F90
sub_fpdf.F90
sub_fpdp.F90
sub_fpds.F90
sub_fpfd.F90
sub_fpff.F90
sub_fpfp.F90
sub_fpfs.F90
sub_fppd.F90
sub_fppf.F90
sub_fppp.F90
sub_fpps.F90
sub_fpsd.F90
sub_fpsf.F90
sub_fpsp.F90
sub_fpss.F90
sub_fsdd.F90
sub_fsdf.F90
sub_fsdp.F90
sub_fsds.F90
sub_fsfd.F90
sub_fsff.F90
sub_fsfp.F90
sub_fsfs.F90
sub_fspd.F90
sub_fspf.F90
sub_fspp.F90
sub_fsps.F90
sub_fssd.F90
sub_fssf.F90
sub_fssp.F90
sub_fsss.F90
sub_pddd.F90
sub_pddf.F90
sub_pddp.F90
sub_pdds.F90
sub_pdfd.F90
sub_pdff.F90
sub_pdfp.F90
sub_pdfs.F90
sub_pdpd.F90
sub_pdpf.F90
sub_pdpp.F90
sub_pdps.F90
sub_pdsd.F90
sub_pdsf.F90
sub_pdsp.F90
sub_pdss.F90
sub_pfdd.F90
sub_pfdf.F90
sub_pfdp.F90
sub_pfds.F90
sub_pffd.F90
sub_pfff.F90
sub_pffp.F90
sub_pffs.F90
sub_pfpd.F90
sub_pfpf.F90
sub_pfpp.F90
sub_pfps.F90
sub_pfsd.F90
sub_pfsf.F90
sub_pfsp.F90
sub_pfss.F90
sub_ppdd.F90
sub_ppdf.F90
sub_ppdp.F90
sub_ppds.F90
sub_ppfd.F90
sub_ppff.F90
sub_ppfp.F90
sub_ppfs.F90
sub_pppd.F90
sub_pppf.F90
sub_pppp.F90
sub_ppps.F90
sub_ppsd.F90
sub_ppsf.F90
sub_ppsp.F90
sub_ppss.F90
sub_psdd.F90
sub_psdf.F90
sub_psdp.F90
sub_psds.F90
sub_psfd.F90
sub_psff.F90
sub_psfp.F90
sub_psfs.F90
sub_pspd.F90
sub_pspf.F90
sub_pspp.F90
sub_psps.F90
sub_pssd.F90
sub_pssf.F90
sub_pssp.F90
sub_psss.F90
sub_sddd.F90
sub_sddf.F90
sub_sddp.F90
sub_sdds.F90
sub_sdfd.F90
sub_sdff.F90
sub_sdfp.F90
sub_sdfs.F90
sub_sdpd.F90
sub_sdpf.F90
sub_sdpp.F90
sub_sdps.F90
sub_sdsd.F90
sub_sdsf.F90
sub_sdsp.F90
sub_sdss.F90
sub_sfdd.F90
sub_sfdf.F90
sub_sfdp.F90
sub_sfds.F90
sub_sffd.F90
sub_sfff.F90
sub_sffp.F90
sub_sffs.F90
sub_sfpd.F90
sub_sfpf.F90
sub_sfpp.F90
sub_sfps.F90
sub_sfsd.F90
sub_sfsf.F90
sub_sfsp.F90
sub_sfss.F90
sub_spdd.F90
sub_spdf.F90
sub_spdp.F90
sub_spds.F90
sub_spfd.F90
sub_spff.F90
sub_spfp.F90
sub_spfs.F90
sub_sppd.F90
sub_sppf.F90
sub_sppp.F90
sub_spps.F90
sub_spsd.F90
sub_spsf.F90
sub_spsp.F90
sub_spss.F90
sub_ssdd.F90
sub_ssdf.F90
sub_ssdp.F90
sub_ssds.F90
sub_ssfd.F90
sub_ssff.F90
sub_ssfp.F90
sub_ssfs.F90
sub_sspd.F90
sub_sspf.F90
sub_sspp.F90
sub_ssps.F90
sub_sssd.F90
sub_sssf.F90
sub_sssp.F90
sub_ssss.F90
sub_xxxx.F90
suberi.F90
swz_init.F90
```

相関エネルギー補正のMP2のアルゴリズム2種

```
Loop over i-batch [parallelizable when needed]
  Loop over σ [to be parallelized for worker processes]
    Loop over λ
      Generate (μν,λ[σ]) list [canonical relation for μν]
      Do 1/4 transformation of μ→i [screening & DAXPY]
      Do 2/4 transformation of ν→a [DDOT]
      Do 3/4 transformation of λ→j [screening & DAXPY]
    End of loop over λ
    Do 4/4 transformation for σ→b [screening & DAXPY]
  End of loop over σ [“all-reduce” must be done for (ia/jb)]
  Calculate partial MP2 energy
End of loop over i-batch
```

```
Loop over ij-batch ! size depending on available memory
  Loop over σ ! to be parallelized
    Loop over λ
      Preparing (μν|λσ) ! for canonical μν-pair
      Forming (iν|λσ) ! DGEMM, fixed λσ, running over μ
      Forming (ia|λσ) ! DGEMM, fixed λσ, running over ν
      Forming (ia|jσ) ! DGEMM, fixed σ, direct-product for fixed λ
    End of loop over λ
    Forming (ia|jb) ! DGEMM, direct-product for fixed σ
  End of loop over σ ! all-reduce operation as barrier
  Calculate partial MP2 energy with respect to ij-batch
End of loop over ij-batch
```

MP2相関エネルギー補正

$$E_{\mathrm{MP2}} = \sum_{ijab} \frac{(ia|jb)\left[2(ia|jb)-(ib|ja)\right]}{\varepsilon_i+\varepsilon_j-\varepsilon_a-\varepsilon_b}$$

添字の変換：4N⁵コスト

$$(ia|jb) = \sum_{\sigma} C_{\sigma b}\left(\sum_{\lambda} C_{\lambda j}\left(\sum_{\nu} C_{\nu a}\left(\sum_{\mu} C_{\mu i}(\mu\nu|\lambda\sigma)\right)\right)\right)$$

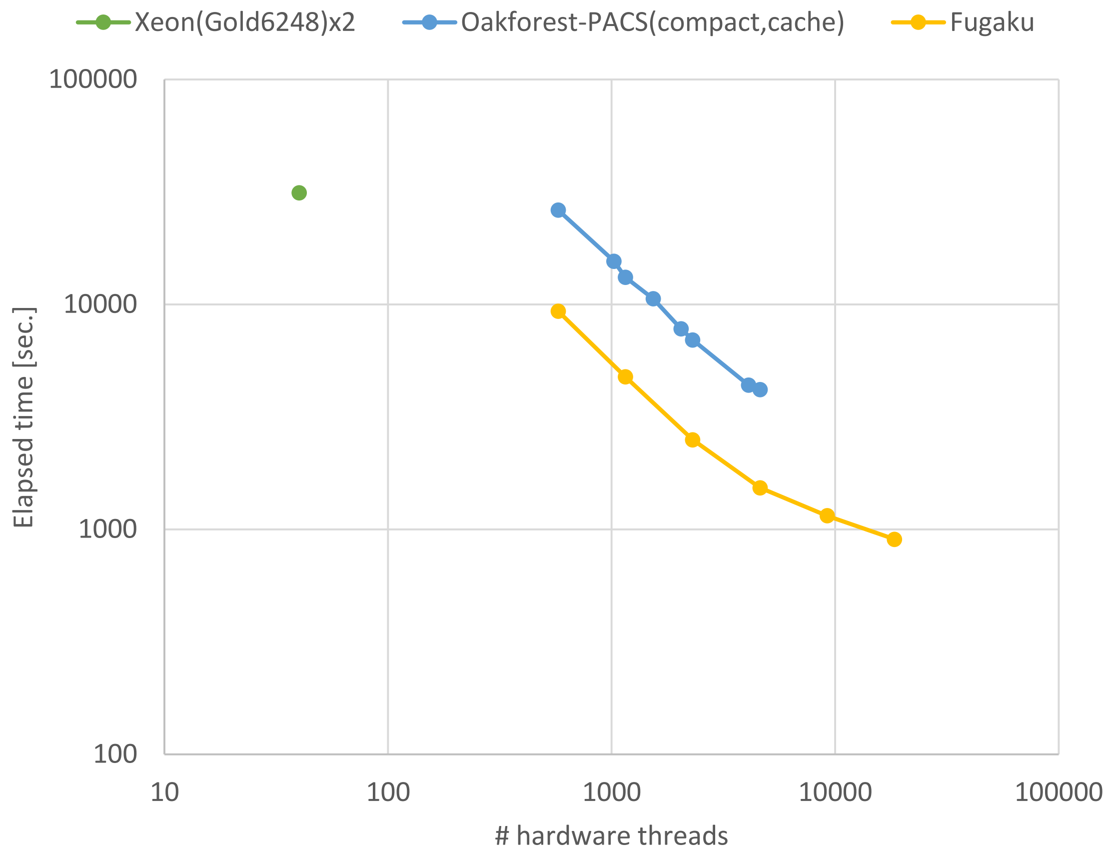
【第一版】
- DAXPYとDDOTを使い、閾値判断を優先
- 実効的演算数を下げる方針、15年程前のチップでは有効

【第二版】
- DGEMMの高性能に期待
- 2段目と4段目をDGEMM処理がデフォルト
- 最近のスパコンでは4段ともDGEMMが最も速い

下記のタイミングはVer. 1 Rev. 22を使った2020年9月当時の測定結果.

FMO-MP2/6-31G*ジョブのスケーリング

Ver. 1 Rev. 22を使用



Xeon; 40 cores
OFP; up to 4608 cores
Fugaku; up to 18432 cores

September 2020

- PDB ID: 6LU7 = SARS-CoV-2 Mpro + N3 ligand の系
- MP2の積分変換は全てDGEMMで実行
- Dimer-ESのCMM近似は (>5のリージョンで使用)
- 「富岳」はOakforest-PACSよりも2.8倍ほど速い

改良の第一弾：Ver. 2 Rev. 4（jh210036-NAH）

Ver. 1系からVer. 2系へ移行の基本方針

◇Ver. 1の最終リリース版はRev. 22 （2020年6月）

f関数のサポート（直交10成分のみ）
MFMOによる指定領域のみでの相関計算が可能
加速されたPIEDA計算（Rev. 20 → Rev. 22の理由：「富岳」のコロナPJで改良）

◇Ver. 1からVer. 2への移行 （jh210036-NAHでの活動）

「富岳」の時代を意識
統計的な相互作用解析の重視 ⇔ Ver. 1 Rev. 22比で数倍の高速化
水和タンパク質モデルの扱い ⇔ 数万フラグメントへの対応（従前は5千程）
機械学習/統計評価/データ科学的な解析が主流に
BioStation Viewerのためのファイル書出しを無効化 （N_f^2配列の保持が困難）
ログファイルの出力量をオプション指定で大幅に削減可能

◇最新のリリース版 - Ver. 2 Rev. 4 （2021年9月）

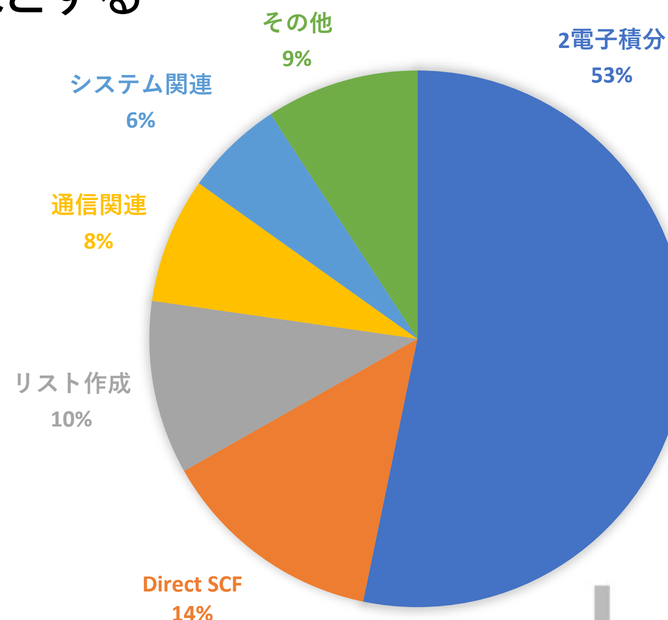
A64FXでの高速化 ⇔ FMO-MP2ジョブで1.3～1.5倍の加速
超大規模系への対応 ⇔ 1.1万フラグメントの水和タンパク質モデルが計算可
機械学習のためのデータダンプ機能
動的分極率の算定

A64FXでのコスト分析 (Ver. 1 Rev. 22)

・Ala_9GlyのFMO-MP2/6-31G*のテストジョブ
・12スレッド8プロセス (2ノード実行:FX1000)

- プログラム全体のコスト分布
  - 基本プロファイラによるプロセス0番、スレッド0番のコスト分布
  - 2電子積分処理が全体の約半分を占める
    ただし、81種の処理の総和であるため、
    1種あたりのコストは1%前後と非常に小さい
  - 通信に関連したコストは8%程度と小さい
  - 性能改善に向けたソース分析は以下を対象とする
    - 2電子積分
    - Direct SCF
    - リスト作成

2電子積分:81種のサブルーチン(sub_*)のコスト総和
Direct SCF:サブルーチンdirect_scf_gmatのコスト
リスト作成:3種のサブルーチン(get_tei_rs_fix, get_tei_pq_fix, get_ixijcs_to_proc_pqfix)のコスト総和
通信関連:通信に関連した処理(putofu_*, opal_*, mca_*)のコスト総和
システム関連:ライブラリやOSなどに関連した処理のコスト総和
その他:上記以外の処理の総和



A64FX向けの高速化

■2電子積分の生成

- {SSSS, PSSS, SPSS, SSPS, SSSP, PPSS, PSPS, PSSP, SPPS, SPSP, SSPP, DSSS, SDSS, SSDS, SSSD}の積分タイプを（手動で）SIMD化
- OCL指示詞の追加
- コンパイラオプション：-O3 -Knosimd -Kocl
- 改良の効果は20～30%程度 （幾つかの系でテスト：jh210036-NAH課題で実施）
- 最新リリースのVer. 2 Rev. 4に反映 （「GUI用のみ配列」の削除も：大規模系対応）

■Fock行列の構築

- 添字同値性を(1/2)n (n=1,2,3)で反映
- 加速は30%程度 （Ver. 2 Rev. 4では未）

```
do p=ixi1,ixi2
  do q=ixj1,ixj2
    do r=ixk1,ixk2
      do s=ixl1,ixl2
        ix=ix+1
        val = sint(ix)
        if((abs(val) <= tv)) cycle
        fock(q,p)=fock(q,p)+dc(s,r)*val*2.d0! クーロン項
        fock(s,r)=fock(s,r)+dc(q,p)*val*2.d0
        fock(r,p)=fock(r,p)-dc(s,q)*val*0.5d0! 交換項
        fock(s,p)=fock(s,p)-dc(r,q)*val*0.5d0
        fock(r,q)=fock(r,q)-dc(s,p)*val*0.5d0
        fock(s,q)=fock(s,q)-dc(r,p)*val*0.5d0
      end do
    end do
  end do
end do
```

■その他の改良（進行/検討中）

- 「無駄なプリント出力」の抑制オプション（logファイルが1千万行になることも…）
- モノマーSCC反復の外挿法の改良
- 対角化の改良は正準直交化と同時に

最新の作業版（2022年8月）ではVer. 1 Rev. 22比で速度1.7倍を達成している

SIMD化した積分ルーチンの例（sssp）

```
subroutine sub_sssp(zetam,pm,dkabm,etam,qm,dkcdm,  &
     ma,mb,mc,md,ngij,ngkl,a,b,c,d,sint,tv)
  !
  !        Nov.05,'02
  !                T.NAKANO & Y.  ABE
  !
  use constant
  use auxiliary_integral_table
  use integral_parameter
  implicit none
  real(8),intent(in)::zetam(*),pm(3,*),dkabm(*),  &
       etam(*),qm(3,*),dkcdm(*)
  integer,intent(in)::ma,mb,mc,md,ngij,ngkl
  real(8),intent(in)::a(3),b(3),c(3),d(3),tv
  real(8),intent(out)::sint(*)
  !-----------------------------------------------
  integer npq,nrs,ix
  real(8) p(3),q(3),qd(3),pq(3),wq(3),f(0:max_m),  &
       dkab,zeta,dkcd,eta,ze,rz,re,rho,a0,tt
  integer ts,i,j,k,l,m
  real(8) delta,t_inv
  real(8) ssss(0:1),f0,f1,qd1,qd2,qd3,wq1,wq2,wq3

  sint(1:3) = 0.0_8
                           !ocl eval
                           !ocl fp_relaxed
                           !ocl fp_contract
                           !ocl noswp
                           !ocl eval_concurrent
                           !ocl SIMD
```

```
  do npq=1,ngij
    if (abs(dkabm(npq)) > tv) then
      do nrs=1,ngkl
        if (abs(dkabm(npq)*dkcdm(nrs)) > tv) then
          ze  = 1.0_8/(zetam(npq)+etam(nrs))
          a0  = dkabm(npq)*dkcdm(nrs)*sqrt(ze)
          rz  = etam(nrs)*ze
          re  = zetam(npq)*ze
          rho = zetam(npq)*rz

          do i=1,3
!           qd(i)  = qm(i,nrs)-d(i)
            pq(i)  = qm(i,nrs)-pm(i,npq)
!           wq(i)  =-re*pq(i)
          end do

            qd1 = qm(1,nrs)-d(1)
            qd2 = qm(2,nrs)-d(2)
            qd3 = qm(3,nrs)-d(3)
            wq1 =-re*pq(1)
            wq2 =-re*pq(2)
            wq3 =-re*pq(3)
```

以下、次頁

SIMD化した積分ルーチンの例（続き）

```
        tt = (pq(1)*pq(1)+pq(2)*pq(2)+pq(3)*pq(3))*rho

        if (tt <= 38.0_8) then  ! Tf = 2*m+36 (for m=1)
          ts    = 0.5_8+tt*fmt_inv_step_size
          delta = ts*fmt_step_size-tt

!         f(0)  = ((fmt_table(3,ts)*inv6*delta  &
!                +  fmt_table(2,ts)*inv2)*delta &
!                +  fmt_table(1,ts))*delta      &
!                +  fmt_table(0,ts)
!         f(1)  = ((fmt_table(4,ts)*inv6*delta  &
!                +  fmt_table(3,ts)*inv2)*delta &
!                +  fmt_table(2,ts))*delta      &
!                +  fmt_table(1,ts)
          f0    = ((fmt_table(3,ts)*inv6*delta  &
                 +  fmt_table(2,ts)*inv2)*delta &
                 +  fmt_table(1,ts))*delta      &
                 +  fmt_table(0,ts)
          f1    = ((fmt_table(4,ts)*inv6*delta  &
                 +  fmt_table(3,ts)*inv2)*delta &
                 +  fmt_table(2,ts))*delta      &
                 +  fmt_table(1,ts)
        else
          t_inv = inv2/tt
!         f(0)  = sqrt(pi_over2*t_inv)
!         f(1)  = t_inv*f(0)
          f0    = sqrt(pi_over2*t_inv)
          f1    = t_inv*f0
        end if
```

```
!--------------------------------------------
!  ERI code generator Ver.20020228
!  2002/02/28
!  T. Nakano
!
!   (sssp)
!
!        ssss(0:1)=f(0:1)*a0
         ssss(0)=f0*a0
         ssss(1)=f1*a0
!        do l=1, 3
!          sint(l) = sint(l)+qd(l)*ssss(0)+wq(l)*ssss(1)
!        end do
         sint(1) = sint(1)+qd1*ssss(0)+wq1*ssss(1)
         sint(2) = sint(2)+qd2*ssss(0)+wq2*ssss(1)
         sint(3) = sint(3)+qd3*ssss(0)+wq3*ssss(1)
!--------------------------------------------
       end if
     end do
   end if
 end do
end subroutine sub_sssp
```

Ver. 2 Rev. 4での速度向上の例#1

HIV-1 protease / FMO-MP2/6-31G* / Benchmark 100 nodes @ Fugaku

Ver. 1 Rev. 22

```
====================
   ## TIME PROFILE
====================

       Elapsed time: Monomer SCF      =          452.4 seconds
       Elapsed time: Monomer MP2      =           17.4 seconds
       Elapsed time: Monomer (Total)  =          472.7 seconds
       Elapsed time: Dimer ES         =           99.7 seconds
       Elapsed time: Dimer SCF        =          278.6 seconds
       Elapsed time: Dimer MP2        =          269.1 seconds
       Elapsed time: Dimer (Total)    =          695.3 seconds
       Elapsed time: FMO (Total)      =         1168.0 seconds

## Time profile

Number of cores (total)    =    200
Number of cores (fragment) =      1

THREADS (FRAGMENT)         =     24


Total time =            1172.8 seconds
```

Ver. 2 Rev. 4

```
====================
   ## TIME PROFILE
====================

       Elapsed time: Monomer SCF      =          354.7 seconds
       Elapsed time: Monomer MP2      =           16.0 seconds
       Elapsed time: Monomer (Total)  =          373.4 seconds
       Elapsed time: Dimer ES         =          109.5 seconds
       Elapsed time: Dimer SCF        =          221.7 seconds
       Elapsed time: Dimer MP2        =          242.4 seconds
       Elapsed time: Dimer (Total)    =          673.4 seconds
       Elapsed time: FMO (Total)      =         1046.8 seconds

## Time profile

Number of cores (total)    =    200
Number of cores (fragment) =      1

THREADS (FRAGMENT)         =     24


Total time =            1050.2 seconds
```

- Ver. 2 Rev. 4はA64FX向け積分SIMD化、「不要配列」の整理などを反映済み
- より大型の系ではMP2ジョブで2-5割程度の速度向上
- cc-pVDZの方が短縮長が長いために加速効果が出やすい （他系でも評価）

Ver. 2 Rev. 4での速度向上の例#2

6VXX / FMO-MP2/6-31G* / Benchmark 8 racks @ Fugaku

Ver. 1 Rev. 22

```
=====================
   ## TIME PROFILE
=====================

      Elapsed time: Monomer SCF     =         2028.7 seconds
      Elapsed time: Monomer MP2     =           15.0 seconds
      Elapsed time: Monomer (Total) =         2068.6 seconds
      Elapsed time: Dimer ES        =          353.9 seconds
      Elapsed time: Dimer SCF       =          362.4 seconds
      Elapsed time: Dimer MP2       =          302.6 seconds
      Elapsed time: Dimer (Total)   =         1603.4 seconds
      Elapsed time: FMO (Total)     =         3672.1 seconds

## Time profile

Number of cores (total)    =    3072
Number of cores (fragment) =       1

THREADS (FRAGMENT)         =      48


Total time =           3759.3 seconds
```

Ver. 2 Rev. 4

```
=====================
   ## TIME PROFILE
=====================

      Elapsed time: Monomer SCF     =         1801.6 seconds
      Elapsed time: Monomer MP2     =           14.2 seconds
      Elapsed time: Monomer (Total) =         1839.1 seconds
      Elapsed time: Dimer ES        =          314.2 seconds
      Elapsed time: Dimer SCF       =          335.7 seconds
      Elapsed time: Dimer MP2       =          294.6 seconds
      Elapsed time: Dimer (Total)   =         1188.5 seconds
      Elapsed time: FMO (Total)     =         3027.7 seconds

## Time profile

Number of cores (total)    =    3072
Number of cores (fragment) =       1

THREADS (FRAGMENT)         =      48


Total time =           3090.8 seconds
```

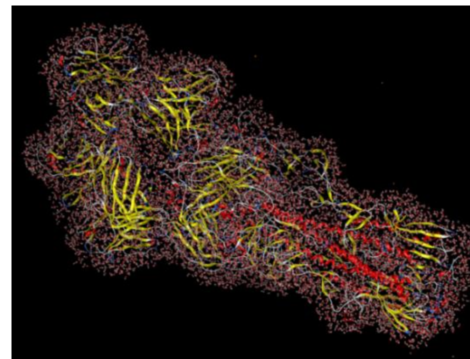
- Ver. 2 Rev. 4はA64FX向け積分SIMD化、「不要配列」の整理などを反映済み
- 対Ver. 1 Rev.22で1.2倍の加速 （cc-pVDZ; 8769.9秒→6356.6秒で1.4倍）

Ref.; 望月ら, *J. Comp. Chem. Jpn.* **20** (2021) 132.

超大規模系への対応例

従前の2倍の系が計算可能に

- インフルHA+Fab抗体×2（PDB id: 1KEN）の水和モデル
- フラグメント総数は11307、水と対イオンを含む
- 「不老」の1ラック、FMO-MP2/cc-pVDZは9.2時間で完走
- 「富岳」の8ラック、FMO-MP3/cc-pVDZは6.7時間で完走
- FMO-MP2ではモノマーSCCが半分弱のコスト ⇒ 要対応
- ダイマー部分で「謎の時間」が顕在化 ⇒ 要対応
- 水クラスターでは2万フラグメントのMP2ジョブも完走確認



```
=====================
  ## TIME PROFILE
=====================

     Elapsed time: Monomer SCF     =        14546.6 seconds
     Elapsed time: Monomer MP2     =           32.5 seconds
     Elapsed time: Monomer (Total) =        14741.5 seconds
     Elapsed time: Dimer ES        =         4021.8 seconds
     Elapsed time: Dimer SCF       =         7215.9 seconds
     Elapsed time: Dimer MP2       =         2492.4 seconds
     Elapsed time: Dimer (Total)   =        18240.6 seconds
     Elapsed time: FMO (Total)     =        32982.1 seconds

## Time profile

Number of cores (total)    =    384
Number of cores (fragment) =      1

THREADS (FRAGMENT)         =     48

Total time =        33120.9 seconds
```

「不老」-2021年6月時点

```
=====================
  ## TIME PROFILE
=====================

     Elapsed time: Monomer SCF     =         7114.0 seconds
     Elapsed time: Monomer MP3     =          343.1 seconds
     Elapsed time: Monomer (Total) =         7532.4 seconds
     Elapsed time: Dimer ES        =          534.8 seconds
     Elapsed time: Dimer SCF       =          891.4 seconds
     Elapsed time: Dimer MP3       =         4265.7 seconds
     Elapsed time: Dimer (Total)   =        16306.3 seconds
     Elapsed time: FMO (Total)     =        23838.7 seconds

## Time profile

Number of cores (total)    =   3072
Number of cores (fragment) =      1

THREADS (FRAGMENT)         =     48

Total time =        24203.2 seconds
```

「富岳」-2021年9月時点

HPCI拠点でのライブラリ整備状況 （2021年12月）

赤:A64FX, 緑:SX-Aurora TSUBASA, 紫:Xeon

■登録サイトと版

- 北大「Grand Chariot」： Ver. 1 Rev. 22 & Ver. 2 Rev. 4
- 東北大「AOBA-A」： Ver. 1 Rev. 22 (Vectorized version)
- JCAHPC「Oakforest-PACS」： Ver. 1 Rev. 22 (system decommissioned)
- 東大「Wisteria / Odyssey & Aquarius」： Ver. 1 Rev. 22 & Ver. 2 Rev. 4
- 東工大「TSUBAME3.0」： Ver. 1 Rev. 22 & Ver. 2 Rev. 4
- 海洋機構「Earth Simulator 4」： Ver. 1 Rev. 22 (Vectorized version)
- 分子研「RCCS」： Ver. 1 Rev. 22 & Ver. 2 Rev. 4
- 名大「不老 Type I」： Ver. 1 Rev. 22 & Ver. 2 Rev. 4
- 阪大「SQUID」： Ver. 1 Rev. 22 (Vectorized version)
- R-CCS「富岳」： Ver. 1 Rev. 22 & Ver. 2 Rev. 4
- 計算科学振興財団「FOCUSスパコン」： Ver. 1 Rev. 22 & Ver. 2 Rev. 4
- 九大「ITO Subsystem-A」： Ver. 1 Rev. 22 & Ver. 2 Rev. 4

- Ver. 1 Rev. 22はBioStation Viewerとの関係で当面は併存
- ベクトル化Ver. 2 Rev. 4（準備中）も順次追加登録の予定
- 2022年末にはVer. 2 Rev. 4からRev. 8に更新の予定
- HPCI拠点のスパコン機種変更に応じた対応も適宜行う方針

Ver. 2 Rev. 8へ向けての改良（jh210036-NAH&jh220010）

【坂倉&中野】

Ver. 2 Rev. 4の後の追加改善 (Rev. 8向け)

- 2021/12/9段階
  - 積分生成でSIMD化に加えてループ分割を実施
    (SSSS, PSSS, SPSS, SSPS, SSSP, PPSS, PSPS, PSSP,
    SPPS, SPSP, SSPP, DSSS, SDSS, SSDS, SSSD)
    ⇒ 積分タイプにも拠るが20%～50%の加速
  - HF計算のFock行列構築からif文の除去
  - 「不老」でのピーク性能比はMP2で2.4～2.8% (MP4だと9.5～10%)

- 2022/7/13段階
  - モノマーSCCのアンダーソン外挿をFock行列から密度行列のベースへ
    ⇒ 系と基底関数にも拠るが反復数を1,2割削減 (逆に増えることも)

- その他(試行中)
  - IFIEでの「不要プリント」の抑制 (プリント時間が無視できない)
  - モノマーSCCのHF部分の簡易積分バッファリング
  - 通信量削減のためにダイマーHFの収束判定をスキップ
  - 積分アルゴリズムの変更 (HRR方式、SX向けベクトル化版の試行)

積分ルーチンのループ分割例

Sub_ssssの例

```
do npq=1,ngij
  if (abs(dkabm(npq))  > tv) then
   do nrs=1,ngkl
    if (abs(dkabm(npq)*dkcdm(nrs))  > tv) then
     ze  = 1.0_8/(zetam(npq)+etam(nrs))
     a0  = dkabm(npq)*dkcdm(nrs)*sqrt(ze)
     rz  = etam(nrs)*ze
     rho = zetam(npq)*rz
     tt  = ((qm(1,nrs)-pm(1,npq))*(qm(1,nrs)-pm(1,npq))  &
        +(qm(2,nrs)-pm(2,npq))*(qm(2,nrs)-pm(2,npq))  &
        +(qm(3,nrs)-pm(3,npq))*(qm(3,nrs)-pm(3,npq)))*rho
     if (tt <= 36.0_8) then ! Tf = 2*m+36 (for the case of m=0)
      ts    = 0.5_8+tt*fmt_inv_step_size
      delta = ts*fmt_step_size-tt
      ssss(0) = (((fmt_table(3,ts)*inv6*delta  &
          +fmt_table(2,ts)*inv2)*delta  &
          +fmt_table(1,ts))*delta  &
          +fmt_table(0,ts))*a0
     else
      ssss(0) = sqrt(pi_over4/tt)*a0
     end if
     sint(1) = sint(1)+ssss(0)
    end if
   end do
  end if
end do
```

```
do npq=1,ngij
  if (abs(dkabm(npq)) <= tv) cycle
  do nrs=1,ngkl
   if (abs(dkabm(npq)*dkcdm(nrs)) <= tv) cycle
   ix = ix + 1
   ze  = 1.0_8/(zetam(npq)+etam(nrs))
   xa0(ix)  = dkabm(npq)*dkcdm(nrs)*sqrt(ze)
   rz  = etam(nrs)*ze
   rho = zetam(npq)*rz
   xtt(ix)  = ((qm(1,nrs)-pm(1,npq))*(qm(1,nrs)-pm(1,npq)) &
      +(qm(2,nrs)-pm(2,npq))*(qm(2,nrs)-pm(2,npq)) &
      +(qm(3,nrs)-pm(3,npq))*(qm(3,nrs)-pm(3,npq)))*rho
  enddo
 enddo
 sint(1)      = 0.0_8
 do npqrs=1,ix
  tt = xtt(npqrs)
  if (tt <= 36.0_8) then ! Tf = 2*m+36 (for the case of m=0)
   ts    = 0.5_8+tt*fmt_inv_step_size
   delta = ts*fmt_step_size-tt
   ssss(0) = (((fmt_table(3,ts)*inv6*delta &
      +fmt_table(2,ts)*inv2)*delta &
      +fmt_table(1,ts))*delta &
      +fmt_table(0,ts))*xa0(npqrs)
  else
   ssss(0) = sqrt(pi_over4/tt)*xa0(npqrs)
  end if
  sint(1) = sint(1)+ssss(0)
 enddo
```

- 満田氏はSIMD化、ループ分割の試行、さらに詳細な性能評価を実施
- 積分ルーチンの今後のチューニングに関しての有効指針を提示

「A64FXを用いたフラグメント分子軌道計算プログラムの性能評価」満田ら; 情処学会2022春 - 学生奨励賞受賞 発表

改良ssppルーチン#1

```
subroutine sub_sspp(zetam,pm,dkabm,etam,qm,dkcdm, &
     ma,mb,mc,md,ngij,ngkl,a,b,c,d,sint,tv)
  !
  !        Nov.06,'02
  !                T.NAKANO & Y.  ABE
  !
  use constant
  use auxiliary_integral_table
  use integral_parameter
  implicit none
  real(8),intent(in)::zetam(*),pm(3,*),dkabm(*),  &
       etam(*),qm(3,*),dkcdm(*)
  integer,intent(in)::ma,mb,mc,md,ngij,ngkl
  real(8),intent(in)::a(3),b(3),c(3),d(3),tv
  real(8),intent(out)::sint(*)
  !------------------------------------------------
  integer npq,nrs,i,k,m,ix
  real(8)  :: pq(3),ze,rz,re,rho,tt

  integer ts
  real(8)  delta,t_inv,zssss
  real(8)::f0,f1,f2
  real(8)  ssss(0:2),ssps(1:3,0:1),sspp(1:3,1:3,0:0)

  real(8)  :: xqc(ngij*ngkl,3),xqd(ngij*ngkl,3),xwq(ngij*ngkl,3)
  real(8)  :: xtt(ngij*ngkl),xre(ngij*ngkl),xeta2(ngij*ngkl),xa0(ngij*ngkl)
  integer :: npqrs
```

改良ssppルーチン#2

```
ix = 0
 !ocl eval
 !ocl fp_relaxed
 !ocl fp_contract
 !ocl noswp
 !ocl eval_concurrent
 !ocl SIMD
 do npq=1,ngij
   if (abs(dkabm(npq)) <= tv) cycle
   do nrs=1,ngkl
     if (abs(dkabm(npq)*dkcdm(nrs)) <= tv) cycle
     ix = ix + 1

     ze    = 1.0_8/(zetam(npq)+etam(nrs))
     rz    = etam(nrs)*ze
     re    = zetam(npq)*ze
     rho   = zetam(npq)*rz
     do i=1,3
       pq(i)  = qm(i,nrs)-pm(i,npq)
       xqc(ix,i)  = qm(i,nrs)-c(i)
       xqd(ix,i)  = qm(i,nrs)-d(i)
       xwq(ix,i)  = -re*pq(i)
     end do

     xtt(ix)  = (pq(1)*pq(1)+pq(2)*pq(2)+pq(3)*pq(3))*rho
     xre(ix)  = re
     xa0(ix)    = dkabm(npq)*dkcdm(nrs)*sqrt(ze)
     xeta2(ix)  = 0.5_8/etam(nrs)

    enddo
  enddo
```

改良ssppルーチン#3

```
sint(1:9)=0.0_8
!ocl eval
!ocl fp_relaxed
!ocl fp_contract
!ocl noswp
!ocl eval_concurrent
!ocl SIMD
do npqrs = 1,ix
  tt = xtt(npqrs)
  if (tt <= 40.0_8) then ! Tf=2*m+36
    ts=0.5_8+tt*fmt_inv_step_size
    delta=ts*fmt_step_size-tt
    f0=((fmt_table(03,ts)*inv6*delta &
            +fmt_table(02,ts)*inv2)*delta &
            +fmt_table(01,ts))*delta &
            +fmt_table(00,ts)
    f1=((fmt_table(04,ts)*inv6*delta &
            +fmt_table(03,ts)*inv2)*delta &
            +fmt_table(02,ts))*delta &
            +fmt_table(01,ts)
    f2=((fmt_table(05,ts)*inv6*delta &
            +fmt_table(04,ts)*inv2)*delta &
            +fmt_table(03,ts))*delta &
            +fmt_table(02,ts)
  else
    t_inv=inv2/tt
    f0=sqrt(pi_over2*t_inv)
    f1=t_inv*f0
    f2=t_inv*3.0_8*f1
  end if
```

```
    ssss(0)=f0*xa0(npqrs)
    ssss(1)=f1*xa0(npqrs)
    ssss(2)=f2*xa0(npqrs)

    do m=0, 1
      ssps(1,m)=xqc(npqrs,1)*ssss(m)+xwq(npqrs,1)*ssss(m+1)
      ssps(2,m)=xqc(npqrs,2)*ssss(m)+xwq(npqrs,2)*ssss(m+1)
      ssps(3,m)=xqc(npqrs,3)*ssss(m)+xwq(npqrs,3)*ssss(m+1)
    end do

    do k=1, 3
      sspp(k,1,0)=xqd(npqrs,1)*ssps(k,0)+xwq(npqrs,1)*ssps(k,1)
      sspp(k,2,0)=xqd(npqrs,2)*ssps(k,0)+xwq(npqrs,2)*ssps(k,1)
      sspp(k,3,0)=xqd(npqrs,3)*ssps(k,0)+xwq(npqrs,3)*ssps(k,1)
    end do

    zssss=xeta2(npqrs)*(ssss(0)-xre(npqrs)*ssss(1))

    sspp(1,1,0)=sspp(1,1,0)+zssss
    sspp(2,2,0)=sspp(2,2,0)+zssss
    sspp(3,3,0)=sspp(3,3,0)+zssss

    sint(1)  = sint(1)+sspp(1,1,0)
    sint(2)  = sint(2)+sspp(1,2,0)
    sint(3)  = sint(3)+sspp(1,3,0)
    sint(4)  = sint(4)+sspp(2,1,0)
    sint(5)  = sint(5)+sspp(2,2,0)
    sint(6)  = sint(6)+sspp(2,3,0)
    sint(7)  = sint(7)+sspp(3,1,0)
    sint(8)  = sint(8)+sspp(3,2,0)
    sint(9)  = sint(9)+sspp(3,3,0)

  enddo

end subroutine sub_sspp
```

すべてのタイミングは2022年8月前半に測定（MP2は全段でDGEMMを指定）.

「不老」でのジョブ時間の比較#1

Ala9Gly - FMO-MP2 (MP2 - all DGEMM) / total job time / 10 fragments

12 threads - 8 process @ 2 nodes

	Ver. / Rev.	Date	Sec.	Min.	Acc.
6-31G*	V1 R22	2020/6/3	134.4	2.24	1.00
	V2 R4	2021/9/16	116.6	1.94	1.15
	V2 R4改	2021/12/9	96.4	1.61	1.39
	V2 R4改	2022/7/13	91.9	1.53	1.46
	V2 R4改	2022/8/9	76.6	1.28	1.75
cc-pVDZ	V1 R22	2020/6/3	303.6	5.06	1.00
	V2 R4	2021/9/16	240.4	4.01	1.26
	V2 R4改	2021/12/9	187.8	3.13	1.62
	V2 R4改	2022/7/13	189.5	3.16	1.60
	V2 R4改	2022/8/9	159.2	2.65	1.91

Chignolin - FMO-MP2 (MP2 - all DGEMM) / total job time / 10 fragments

12 threads - 8 process @ 2 nodes

	Ver. / Rev.	Date	Sec.	Min.	Acc.
6-31G*	V1 R22	2020/6/3	719.6	11.99	1.00
	V2 R4	2021/9/16	647.3	10.79	1.11
	V2 R4改	2021/12/9	547.8	9.13	1.31
	V2 R4改	2022/7/13	500.9	8.35	1.44
cc-pVDZ	V1 R22	2020/6/3	1738.8	28.98	1.00
	V2 R4	2021/9/16	1491.5	24.86	1.17
	V2 R4改	2021/12/9	1230.1	20.50	1.41
	V2 R4改	2022/7/13	1226.9	20.45	1.42

- SIMD化とループ分割の併用は有効
- 基底関数の短縮の長いcc-pVDZの方が加速は顕著
- Ala_9GlyではHFでの積分バッファリングを試行（最下段）
- 同じ10残基（フラグメント）でも効果が異なる

「不老」でのジョブ時間の比較#2

Trp-Cage - FMO-MP2 (MP2 - all DGEMM) / total job time / 20 fragments

24 threads - 20 process @ 10 nodes

	Ver. / Rev.	Date	Sec.	Min.	Acc.
6-31G*	V1 R22	2020/6/3	469.6	7.83	1.00
	V2 R4	2021/9/16	413.9	6.90	1.13
	V2 R4改	2021/12/9	344.2	5.74	1.36
	V2 R4改	2022/7/13	294.2	4.90	1.60
cc-pVDZ	V1 R22	2020/6/3	1059.9	17.67	1.00
	V2 R4	2021/9/16	876.1	14.60	1.21
	V2 R4改	2021/12/9	706.7	11.78	1.50
	V2 R4改	2022/7/13	622.0	10.37	1.70

Crambin - FMO-MP2 (MP2 - all DGEMM) / total job time / 43 fragments

24 threads - 43 process @ 22 nodes

	Ver. / Rev.	Date	Sec.	Min.	Acc.
6-31G*	V1 R22	2020/6/3	572.6	9.54	1.00
	V2 R4	2021/9/16	509.4	8.49	1.12
	V2 R4改	2021/12/9	444.5	7.41	1.29
	V2 R4改	2022/7/13	457.7	7.63	1.25
cc-pVDZ	V1 R22	2020/6/3	1507.6	25.13	1.00
	V2 R4	2021/9/16	1232.6	20.54	1.22
	V2 R4改	2021/12/9	1005.2	16.75	1.50
	V2 R4改	2022/7/13	973.0	16.22	1.55

- モノマーSCCのアンダーソン外挿（密度）は効く場合も多々ある（「必ず」ではない）

「不老」でのジョブ時間の比較#3

200～400残基のタンパク質が応用計算の主対象になっている

HIV-Protease - FMO-MP2 (MP2 - all DGEMM) / total job time / 203 fragments

24 threads - 204 process @ 102 nodes

	Ver. / Rev.	Date	Sec.	Min.	Acc.
6-31G*	V1 R22	2020/6/3	971.0	16.18	1.00
	V2 R4	2021/9/16	866.0	14.43	1.12
	V2 R4改	2021/12/9	777.8	12.96	1.25
	V2 R4改	2022/7/13	741.9	12.37	1.31
cc-pVDZ	V1 R22	2020/6/3	2101.4	35.02	1.00
	V2 R4	2021/9/16	1737.2	28.95	1.21
	V2 R4改	2021/12/9	1526.5	25.44	1.38
	V2 R4改	2022/7/13	1564.4	26.07	1.34

SARS-CoV-2 Mainprotease - FMO-MP2 (MP2 - all DGEMM) / total job time / 395 fragments

24 threads - 384 process @ 192 nodes

	Ver. / Rev.	Date	Sec.	Min.	Acc.
6-31G*	V1 R22	2020/6/3	996.8	16.61	1.00
	V2 R4	2021/9/16	874.2	14.57	1.14
	V2 R4改	2021/12/9	793.6	13.23	1.26
	V2 R4改	2022/7/13	741.6	12.36	1.34
cc-pVDZ	V1 R22	2020/6/3	2230.6	37.18	1.00
	V2 R4	2021/9/16	1882.0	31.37	1.19
	V2 R4改	2021/12/9	1633.3	27.22	1.37
	V2 R4改	2022/7/13	1475.6	24.59	1.51

- 全体としてcc-pVDZ基底の方が優位に加速が得られる場合が多い
- 「富岳」でも少しテストしているが、加速は（やや）大きめに出る

まとめ

Ver. 2 Rev. 8のリリースは2022年度内を一応予定、ただ高速化以外の機能追加が多数のために未確定.

現状(2022年8月)のまとめ、今後の方向性

■高速化

・2電子積分のSIMD化とループ分割、Fock行列構築の改良、モノマーSCC加速

⇒ MP2ジョブでは(従前比で)1.5～1.7倍 （改造部分のみは1.8倍～2.2倍）

⇒ 22年度内に2倍を押さえたい （Ver. 2 Rev. 8のリリースに間に合わせる）

■大規模化

・可視化用の「不要配列」の削除

⇒ 1.1万フラグメントのMP3計算も可 （残基数2.2千の水和タンパク質）

■今後

・2電子積分のHRRアルゴリズムの検討

⇒ 検討&テスト中 （演算数は低減、ただループの最適化は今後）

・SX向けベクトル化積分の転用

⇒ 手動の修正は不可避 （ループ長は長い、ただ単なる移植では遅い）

・複数の積分計算ルーチンの使い分け

⇒ VRR/HRRなどの混成使用 （系や基底に拠る自動選択の可能性も）

・タンパク質の水和モデルでの水のクラスター化の前処理

⇒ 実効フラグメント数を削減 （3.3千残基のSタンパク質の計算で試行）

AlphaFold2 Ref.; J. Jumper et al., Nature 596 (2021) 583.

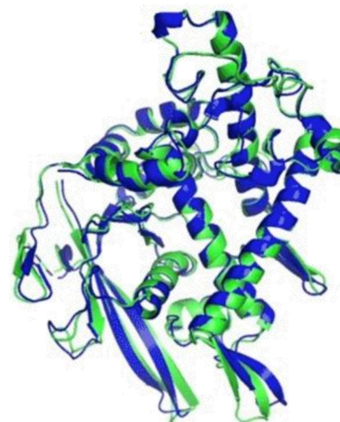
jh220010課題での別の話題；AlphaFold2

■特徴

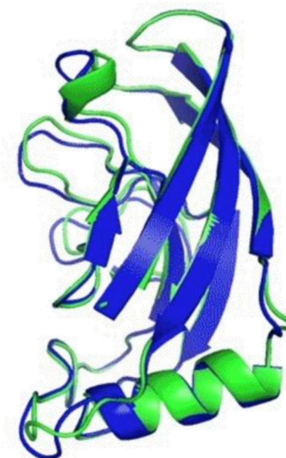
- タンパク質の立体構造を高精度かつ短時間に予測可能な公開ソフト
  → 2020年11月のCASP14コンテストで圧勝（9割近い正答）
- 深層学習技術を駆使
- アミノ酸1文字表記の配列のみが基本的な入力データ（FASTA形式）
- 小規模のサーバでも動作可能、PDB形式で出力
  → SSDやGPUがあるとベター

■計算処理のポイント

- 「類縁配列は類縁構造を持つ」と仮定
- 大規模データベース検索
  → 疎水性コアに注目、MSAの作成
  → 実はココが律速段階
- 深層学習の核はEvoformer処理
  → 物理化学的エネルギー計算無し



T1037 / 6vr4
90.7 GDT
(RNA polymerase domain)



T1049 / 6y4f
93.3 GDT
(adhesin tip)

● Experimental result
● Computational prediction

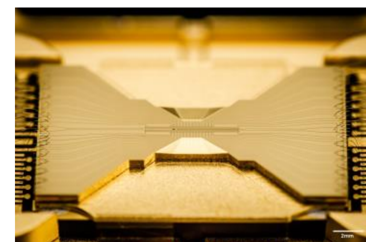
■「不老 Type II」での利用

- 大島先生、森脇先生（東大農）が21年秋に導入
- 新型コロナの変異株のモデリングに使用

jh220010課題での別の話題；cuQuantum

■量子コンピュータの概況

- 量子コンピュータ
  - → 方式：量子ゲートと量子アニーリング型
  - → デバイス：超電導、イオントラップ、中性原子、シリコン、光
  - → トレンド：NISQからFTQCへ、IBMは数千ビットの計画発表
- 量子シミュレータ
  - → ノイズの問題無し、多数存在、アルゴリズム開発に必携
  - → NVIDIAのcuQuantumはGPU利用で高速で大規模対応
- 量子化学計算は量子コンピュータの有力なアプリ
  - → 国内外で研究開発が盛ん
  - → VQE：量子-古典混成の代表的な手法
  - → PSA：　NISQからFTQCへの流れで実用化か
  - → テンソル分解や分割&統合系の手法も要注目



https://ionq.com/

■「不老 Type II」での試行

- cuQuantumのインストールと利用（NVIDIA森野氏、大島先生が整備）
  - → PSA系の量子化学シミュレーションで使用中（杉崎先生（大阪公大））
  - → 「9量子ビットでは高速化は3.5倍、19量子ビットでは32倍」

ABINIT-MPによるFMO計算のロードマップ

核内受容体(ER)
~300残基

インフルエンザHA
抗原抗体系**~1000残基**

インフルエンザ**HA**３量体
抗原抗体系**~2400残基**

結晶 - ペプチド複合系
~$(SiO_2)_{250}$-6残基-水和

粗視化**MD**用
パラメータ
~数万サンプル

新型コロナウイルス
抗原抗体系**~5300残基**

水和**DNA**
12塩基対＋2500wtr

mFruits

EGFR
チロシンキナーゼ

インフルエンザNA
タミフル**~400残基**

リガンド水和
10Å水和層

分子固体
~千個単位

粗視化→原子復元構造
~1万原子×サンプル数

計算 MP2 CIS/CIS(D) MP3 CD CCSD(T) FMO4 Dimer-ES CMM LRD

構造 PDB一点計算/モデル埋戻し FMO-MD MP2(p-opt) FMO-DPD MD生成多構造

解析 IFIE CAFI FILM BSSE FMO4-IFIE PIEDA SVD 統計/ML

電荷、溶媒効果 ESP/RESP NPA SCIFIE PB(SA) 大型液滴