「不老」における分子軌道計算プログラムABINIT-MPの高速化手法の検討

満田 晴紀, 星野 哲也（名大）, 望月 祐志（立教大）,
板倉 耕太（FOCUS）, 片桐 孝洋（名大）, 大島 聡史（九大）
永井 亨, 河合 直聡（名大）

分子軌道計算プログラム ABINIT-MP

- **フラグメント分子軌道法(FMO)を用いた計算シミュレーションソフトウェア**
- **専用GUIによる可視化が容易**
- **Fortranベース、MPIまたはOpenMPによる並列化**
- **HPCI拠点で利用可能**
  - ✓A64FX：富岳(理研), 不老(名大), Wisteria(東大)
  - ✓TSUBASA：SQUID(阪大), AOBA-A(東北大)
  - ✓Intel Xeon系：ITO(九大), TSUBAME(東工大), Grand Chariot(北大)など
- **GPUへの移植も進んでいる**
- **性能解析のためのミニアプリが提供されている**
  - ✓今回はこのミニアプリを用いる
- **性能分析や最適化はまだ不十分**

図 4 PDB-ID=1KENの水和モデル

出典:計算工学ナビ　ABINIT-MP Open シリーズ (Ver.2 Rev.4) http://www.cenav.org/?s=ABINIT-MP

研究目的

1. **A64FXにおけるABINIT-MPミニアプリの性能評価**
   - ✓A64FXは省電力に重きを置いたプロセッサであり、省電力が重要な今後のプロセッサもA64FXに似た性能特性を持つと考えられるため、A64FXにおける性能評価は重要である
   - ✓ABINIT-MPミニアプリではFMO計算に4つの手法を提供しているが、特にA64FXにおいてそれぞれの比較評価は十分ではない

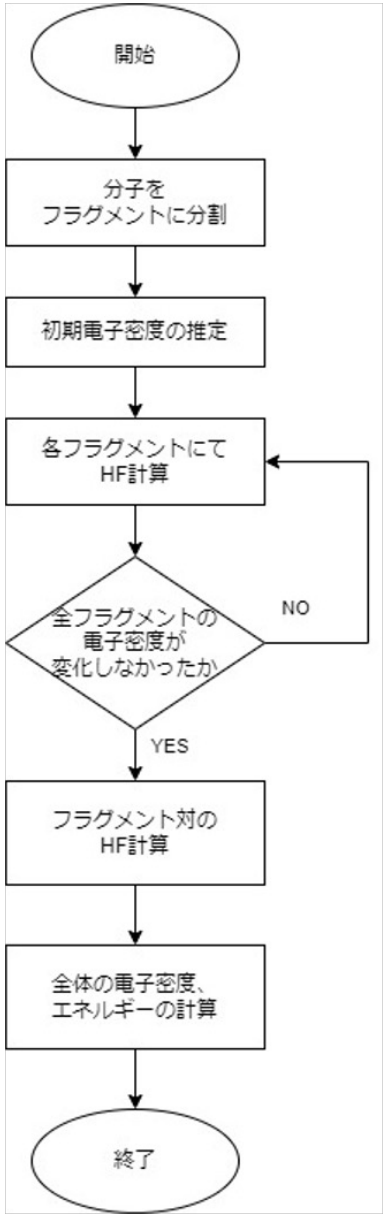
   → 本発表ではA64FXにおいてプロファイラを用いて4つの計算手法の性能分析を行う
2. **ベクトル化を念頭においたコード最適化**
   - ✓分子軌道計算における軌道の組み合わせで81個の個別ルーチンが用意されており、ループ中でselect文により呼び分けられるため分岐コストが大きい
   - ✓個別ルーチン内のベクトル化対象ループ長が極端に小さい
   - ✓ベクトル化対象ループ内のカットオフのための分岐がベクトル化を妨げている

   → 本発表では分岐コストを減らすための実行順序ソート手法、ループ長の確保手法、最内ループ内分岐の削減手法について提案・評価
3. **自動チューニング(AT)適用に関する検討**
   - ✓例えばベクトル化を促進するためのループブロッキング幅はプロセッサにより最適値が異なりうる

   → **本発表ではATの適用可能性を検討。実際のAT適用は今後の課題**

ABINIT-MPのFMO計算手法

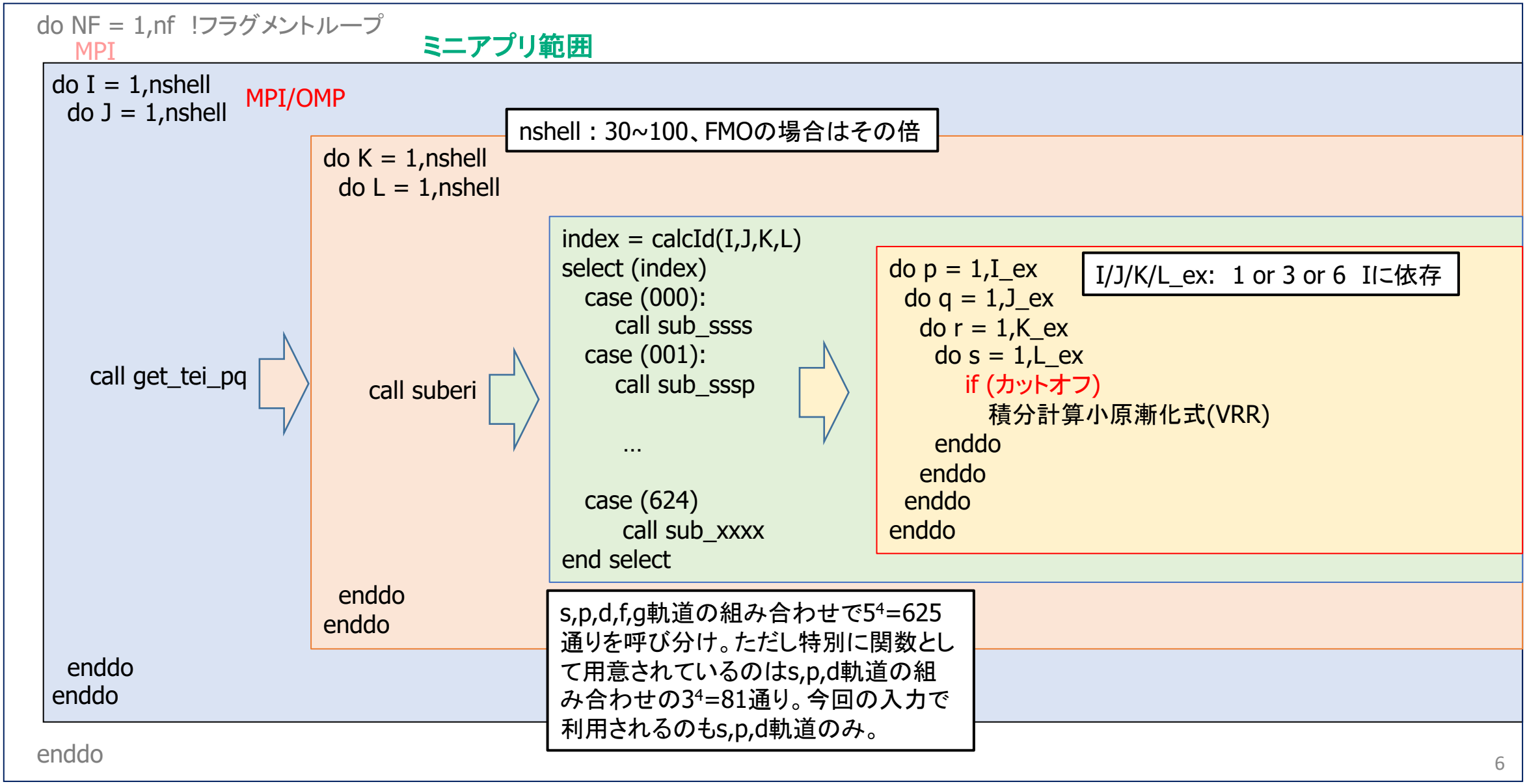
- **フラグメント分子軌道(FMO)法**
  - ✓生体高分子のような巨大分子について非経験的分子軌道計算を行うことができる
  - ✓以下の手順をとる(右図参照)
    1. 分子間のハミルトニアンから電子分を取り出す
    2. 電子に関するシュレディンガー方程式を立てる
    3. シュレディンガー方程式をHatree-Fock(HF)近似で解いて系の電子エネルギーを求める
- **ABINIT-MPにおけるFMO計算**
  - ✓HF計算において最も重い二電子積分生成処理をObaraらの垂直漸化式関係(VRR)法により行う
    - ➢VRR法は軌道角運動量のシフトアップによってs型(角運動量0), p型(同1), d型(同2)の軌道に関する積分を生成するアルゴリズム
  - ✓s/p/d型関数の組み合わせによって3^4=81種類のルーチンを生成、select文によって呼び分ける
    - ➢s型が4つの[ss-ss]の計算が基準
    - ➢ベクトル化等の最適化は81種類のルーチン全て書き換える必要がある



ABINIT-MPミニアプリの計算手法

- **ABINIT-MPの評価のため2電子積分生成とその積分を使ったHF計算のみ抜き出したもの**
  - ✓ABINIT-MPでは2電子積分の生成のみで実行時間の60%程度を占める
- **以下の4手法を評価できる**
  - ✓Obara
    - ABINIT-MP本体同様、ObaraらのVRR法を用いている
    - 81種類のサブルーチンを呼び分けて2電子積分生成及びHF計算を行う
  - ✓Obara-vector
    - NEC TSBASAを念頭におき、Obaraをベースに変更を加えたもの
    - 多重ループを一重化して最内ループを長くしている
    - 81種類のサブルーチンを呼び分けるのは同様
  - ✓Obara-general
    - 81種類のサブルーチンを汎用的な一つのサブルーチンとしてまとめたもの
  - ✓Hgp-general
    - Head-Gordonらの水平漸化式関係(HRR)法を用いたもの
    - 角運動量和が同じ組み合わせに対して一括生成を行うため、基底関数の短縮長が長い場合にはVRR法に比べて演算数の低減が期待できる

Obaraのプログラム構造



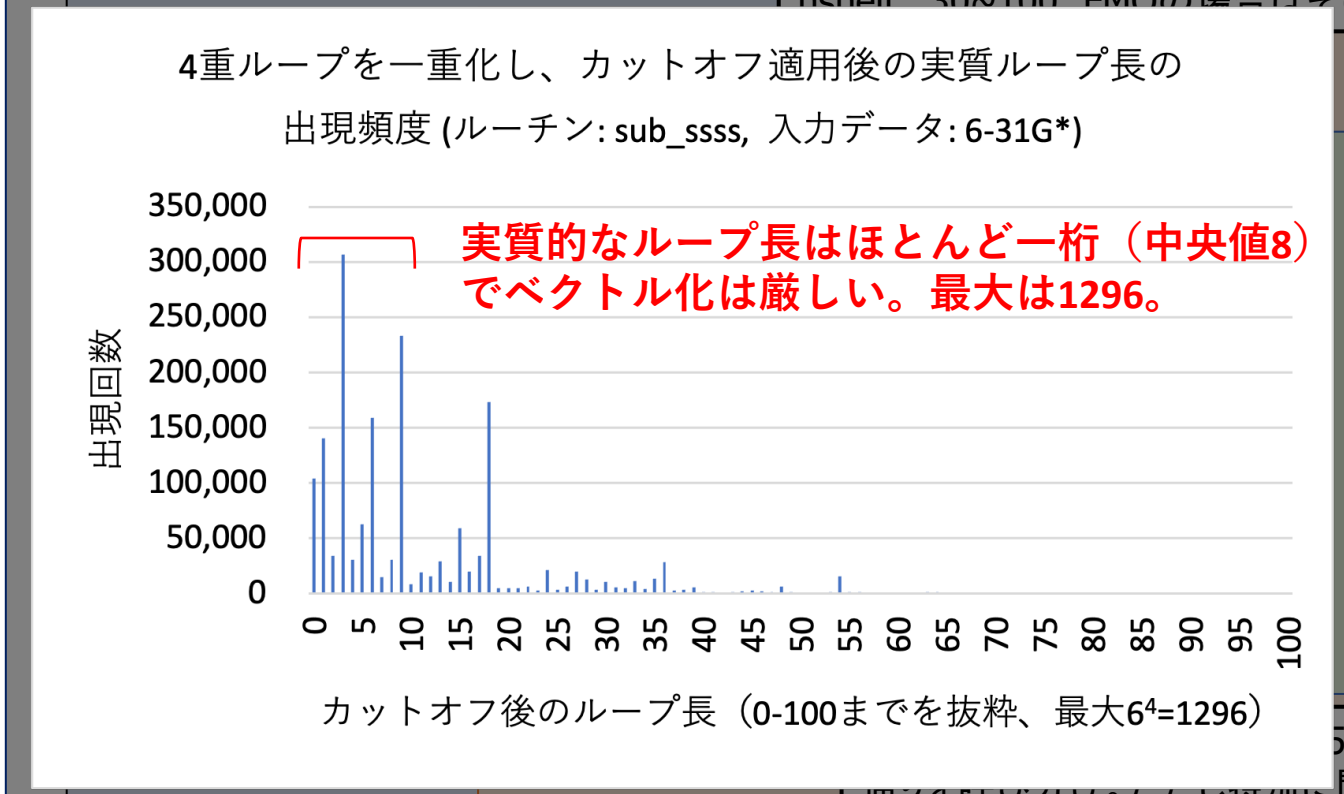
Obaraのプログラム構造

do NF = 1,nf !フラグメントループ
MPI

ミニアプリ範囲

do I = 1,nshell
do J = 1,nshell
MPI/OMP

nshell：30~100 FMOの場合はその倍




```
do p = 1,I_ex
  do q = 1,J_ex
    do r = 1,K_ex
      do s = 1,L_ex
        if (カットオフ)
          積分計算小原漸化式(VRR)
      enddo
    enddo
  enddo
enddo
```

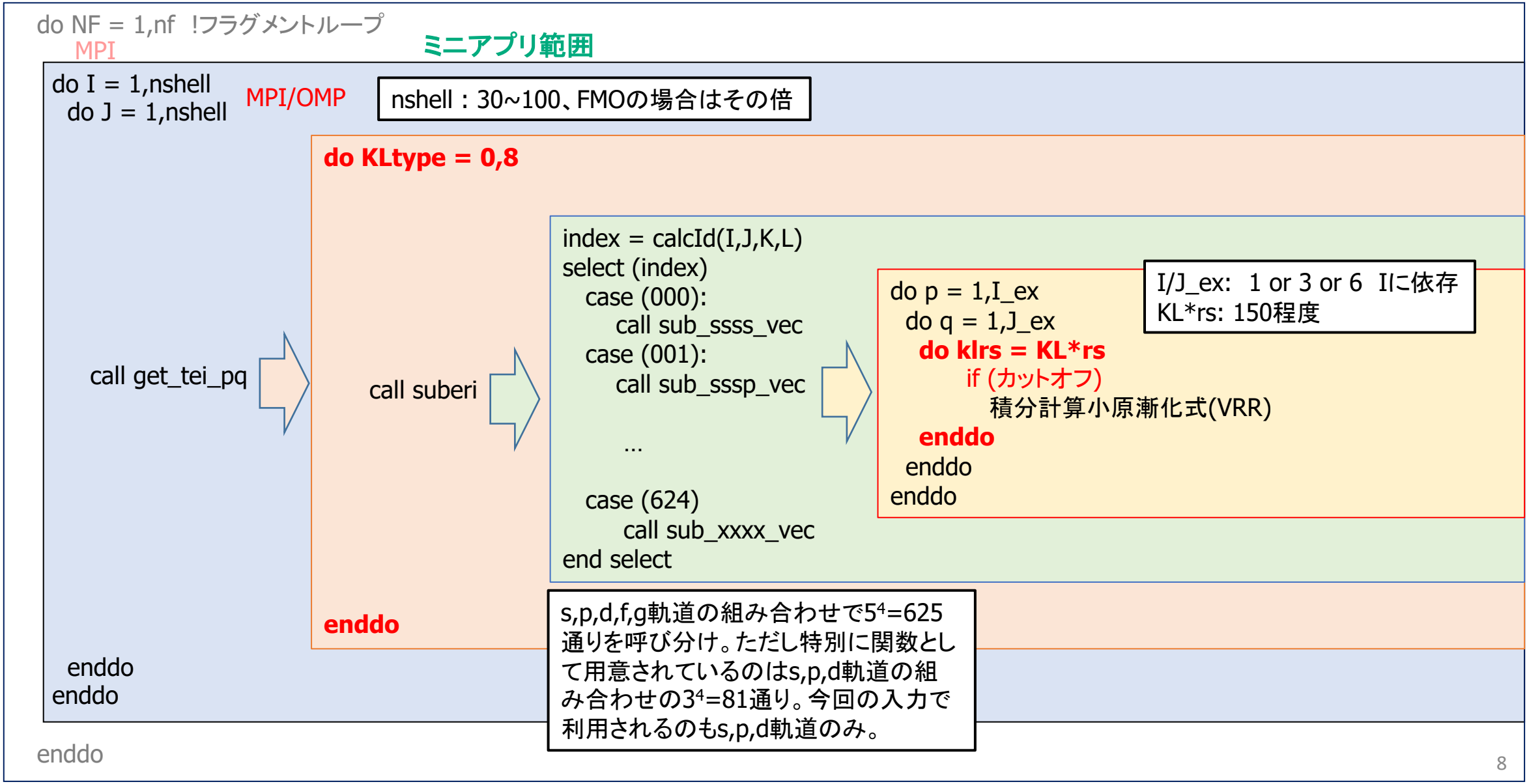
I/J/K/L_ex: 1 or 3 or 6 Iに依存

5^4=625
通りを呼び分け。ただし特別に関数として用意されているのはs,p,d軌道の組み合わせの3^4=81通り。今回の入力で利用されるのもs,p,d軌道のみ。

enddo
enddo

enddo

Obara-vectorのプログラム構造



Obara-vectorのプログラム構造

do NF = 1,nf !フラグメントループ

MPI

ミニアプリ範囲

ルーチン sub_ddddに対する富士通コンパイラ最適化出力

```
<<< Loop-information Start >>>
<<<  [OPTIMIZATION]
<<<    PREFETCH(HARD) Expected by compiler :
<<<      etam, dkcdm, qm, c, d, xxxx
<<<    PREFETCH(SOFT) : 2570
<<<     SEQUENTIAL : 2570
<<<      xxxx: 2570
<<<    SPILLS :
<<<      GENERAL   : SPILL 1118  FILL 2406
<<<      SIMD&FP   : SPILL 14392  FILL 28443
<<<      SCALABLE  : SPILL 0  FILL 0
<<<      PREDICATE : SPILL 0  FILL 0
<<< Loop-information  End >>>
```

```
do p = 1,I_ex
  do q = 1,J_ex
    do klrs = KL*rs
        if (カットオフ)
          積分計算小原漸化式(VRR)
    enddo
  enddo
enddo
```

I/J_ex: 1 or 3 or 6 Iに依存
KL*rs: 150程度

最内ループ長を確保するためにさらに内側の細かいループを全てunrollingしており、その結果として大量の変数、ループボディ1万行超。富士通コンパイラの出力では、レジスタスピル14,392となってベクトル化されない。

end select

enddo

み合わせの3^4=81通り。今回の入力で利用されるのもs,p,d軌道のみ。

enddo

ミニアプリの性能評価

A64FX及びIntel Xeon (CascadeLake)における実行時間(秒)
5回実行した際の中央値

	データセット	6-31G	6-31G*	cc-pVDZ
A64FX (2.2GHz) OpenMP 48コア使用 3,379.2 GFLOPS 富士通コンパイラ4.9.0 –Kopenmp,fast,nosimd,ocl –Nlst=t	Obara	2.1	5.4	12.6
	Obara-vector	5.9	15.8	45.1
	Obara-general	185.8	346.4	638.1
	Hgp-general	361.8	656.9	1208.0
Intel Xeon Platinum 8280 (CascadeLake) OpenMP 1ソケット28コア使用 2,419.2 GFLOPS Intel compiler 2021.7.1 –O3 –fopenmp	Obara	1.2	3.1	7.6
	Obara-vector	3.2	11.6	37.0
	Obara-general	5.1	13.9	32.9
	Hgp-general	3.9	11.7	27.2

- **いずれもアダマンタン（$C_{10}H_{16}$）分子に対する計算**
- **基底関数が6-31G, 6-31G*, cc-pVDZで異なる**
  - ✓ 6-31Gではd関数がないので、呼び出されるサブルーチンは2^4=16パターンのみ
  - ✓ 6-31G*, cc-pVDZでは3^4=81通りのルーチンが呼び出される
  - ✓ cc-pVDZは水分子にもp型の分極関数を加えたもので、6-31G系より短縮長が長く、カットオフが多い
- **A64FXが全般的に遅い**
  - ✓ Obara, Obara-vectorを比較すると、1.22-1.85倍ほど遅い
  - ✓ いずれもベクトル化率が低い
  - ✓ *-generalが特に遅いのは、critical指示文を使っているため
    - ➢ 最適化の進度がObara, Obara-vectorより遅れていることによるものと思われる

提案①：ベクトル化を念頭においた実行順ソート

do NF = 1,nf !フラグメントループ

MPI

ミニアプリ範囲

```
!-- ループ前にあらかじめ実行順をソート

!$omp parallel
do index = 0, 624


 select (index)
   case (000):
     call sub_ssss
   case (001):
     call sub_sssp

       …

   case (624)
      call sub_xxxx
 end select



enddo
!$omp end parallel
```

```
!$omp do
do IJKL = 1, num_xxxx
  I = index_to_i(IJKL)
  J = index_to_j(IJKL)
  K = index_to_k(IJKL)
  L = index_to_l(IJKL)
  do p = 1,I_ex
    do q = 1,J_ex
      do r = 1,K_ex
        do s = 1,L_ex
          if (カットオフ)
             積分計算小原漸化式(VRR)
        enddo
      enddo
    enddo
  enddo
enddo
!$omp end do nowait
```

Select文のキーとなるindex (s, p, dの組み合わせによって決まる数値）をあらかじめ参照し、その際のループインデックスを覚えておく。

外側の4重ループを一重化。$(30\text{-}100)^4$ なのでそれなりのサイズのループをselectによる分岐後に実行でき、ベクトル化を行いやすくなった。
発想としてはObara-vectorに似ているが、このループを最内ループ化はしない点で異なる。

Sedコマンドによって元ソースから切り貼りを行い、81個の関数を生成している

ループの計算コストは一定ではなく負荷不均衡が生じるため、nowaitにより同期コストを減らす。それぞれのサブルーチンは独立実行可。

enddo

提案①のメリット・デメリット

- **メリット**
  - ✓分岐・関数呼び出しが減る
  - ✓分岐後のループ長が長く、ベクトル化などの最適化を適用しやすい
    - ➢これが今回の実装を行なった主目的
    - ➢GPUなどにおいても有効と考えられる
- **デメリット**
  - ✓ソートのオーバーヘッド
    - ➢ただし、電子状態が安定するまでの繰り返し計算の間、ソートは一回のみで良い
  - ✓冗長計算の増加
    - ➢外側の4重ループを一重化した影響
  - ✓キャッシュ効率の低下
  - ✓メモリアクセス効率の低下
    - ➢データの格納順を考えることで影響を減らせるが、今後の課題

提案①実装の性能評価

A64FX及びにおける実行時間(秒)

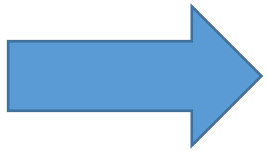
	データセット	6-31G	6-31G*	cc-pVDZ
A64FX (2.2GHz) OpenMP 48コア使用 3,379.2 GFLOPS 富士通コンパイラ4.9.0 –Kopenmp,fast,nosimd,ocl –Nlst=t	Obara	2.1	5.4	12.6
	Obara-vector	5.9	15.8	45.1
	提案手法(括弧内はソートの内時間)	2.5 (0.2)	5.8 (0.3)	17.1 (0.6)

- **いずれもアダマンタン（$C_{10}H_{16}$）分子に対する計算**
- **基底関数が6-31G, 6-31G*, cc-pVDZで異なる**
  - ✓ 6-31Gではd関数がないので、呼び出されるサブルーチンは2^4=16パターンのみ
  - ✓ 6-31G*, cc-pVDZでは3^4=81通りのルーチンが呼び出される
  - ✓ cc-pVDZは水分子にもp型の分極関数を加えたもので、6-31G系より短縮長が長く、カットオフが多い
- **提案手法はObaraと比較すると少し遅いが、Obara-vectorより速い**
  - ✓ ソートは1コアのみで実行している。並列化により高速化の見込みはある
  - ✓ 少しのオーバーヘッドでベクトル長を稼げたことが成果
    - ➢ ただしこの段階では全くベクトル化はされていない

提案②：カットオフ分岐の排除及びブロッキング

- **分岐排除基本方針**
  - ✓ 条件判定部と計算部にループを分割する
  - ✓ 実行されるループボディの回数を数え、計算部のループ長とする
  - ✓ 実行される時のループインデックスを記憶しておき、計算ループで参照する

```
do p = 1,I_ex
  do q = 1,J_ex
    do r = 1,K_ex
      do s = 1,L_ex
        if (カットオフ)
          積分計算小原漸化式(VRR)
      enddo
    enddo
  enddo
enddo
```



```
ix = 0
do p = 1,I_ex
  do q = 1,J_ex
    do r = 1,K_ex
      do s = 1,L_ex
        if (カットオフ) then
          ix + 1
          p_array(ix) = p
          q_array(ix) = q
          r_array(ix) = r
          s_array(ix) = s
        end if
      enddo
    enddo
  enddo
enddo

do i = 1, ix
   p = p_array(ix)
   q = q_array(ix)
   r = r_array(ix)
   s = s_array(ix)
   積分計算小原漸化式(VRR)
enddo
```

条件判定ループ

計算ループ

これで計算ループからは分岐がなくなり、SIMD化が促進されるはずだが、そのままではループ長が短すぎる（中央値8）ため、効果が薄い

提案②：カットオフ分岐の排除及びブロッキング

- **ブロッキングの方針**
  - ✓提案①で一重化したループに対してブロック化を行う
  - ✓ある程度まとめて条件判定を行う
    - ➢この際、内側４重ループが取りうる長さ（=6^4=1296)分のバッファを用意しておく必要がある
  - ✓計算ループの長さの中央値が8だったので、BLK=32あたりで決め打ち
    - ➢このパラメータは最適化の余地があるが、今後の課題

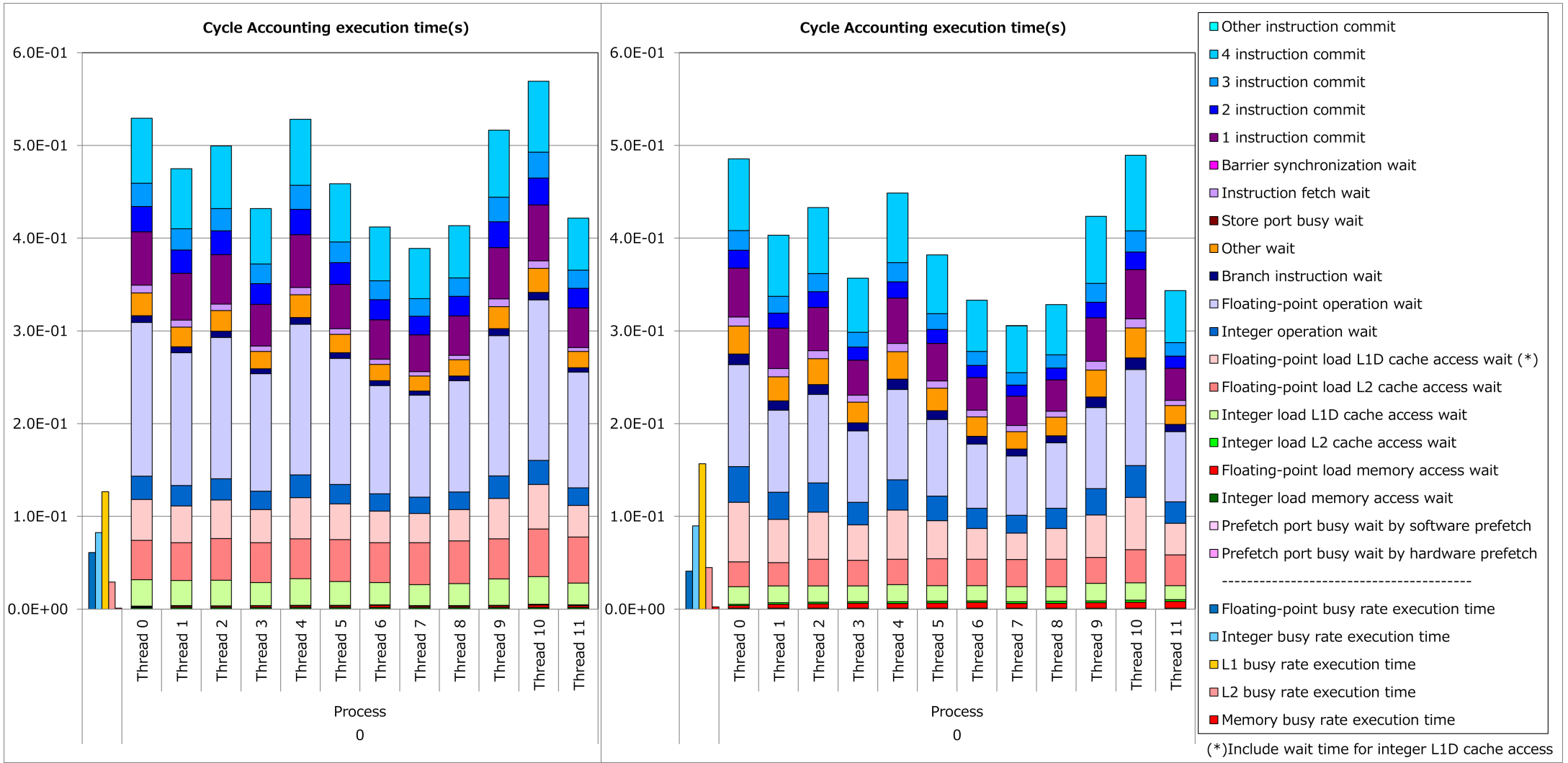
```
!$omp do
do IJKL = 1, num_xxxx, BLK
  ix_blk = 0
  do b = 0, BLK-1
    I = index_to_i(IJKL+b)
    J = index_to_j(IJKL+b)
    K = index_to_k(IJKL+b)
    L = index_to_l(IJKL+b)
    do p = 1,I_ex
      do q = 1,J_ex
        do r = 1,K_ex
          do s = 1,L_ex
            if (カットオフ)
            ix_blk = ix_blk + 1
          enddo
        enddo
      enddo
    enddo
  end do

  do i = 1, ix_blk
    !積分計算小原漸化式(VRR)
  end do

 end do
!$omp end do nowait
```

条件判定ループ

計算ループ

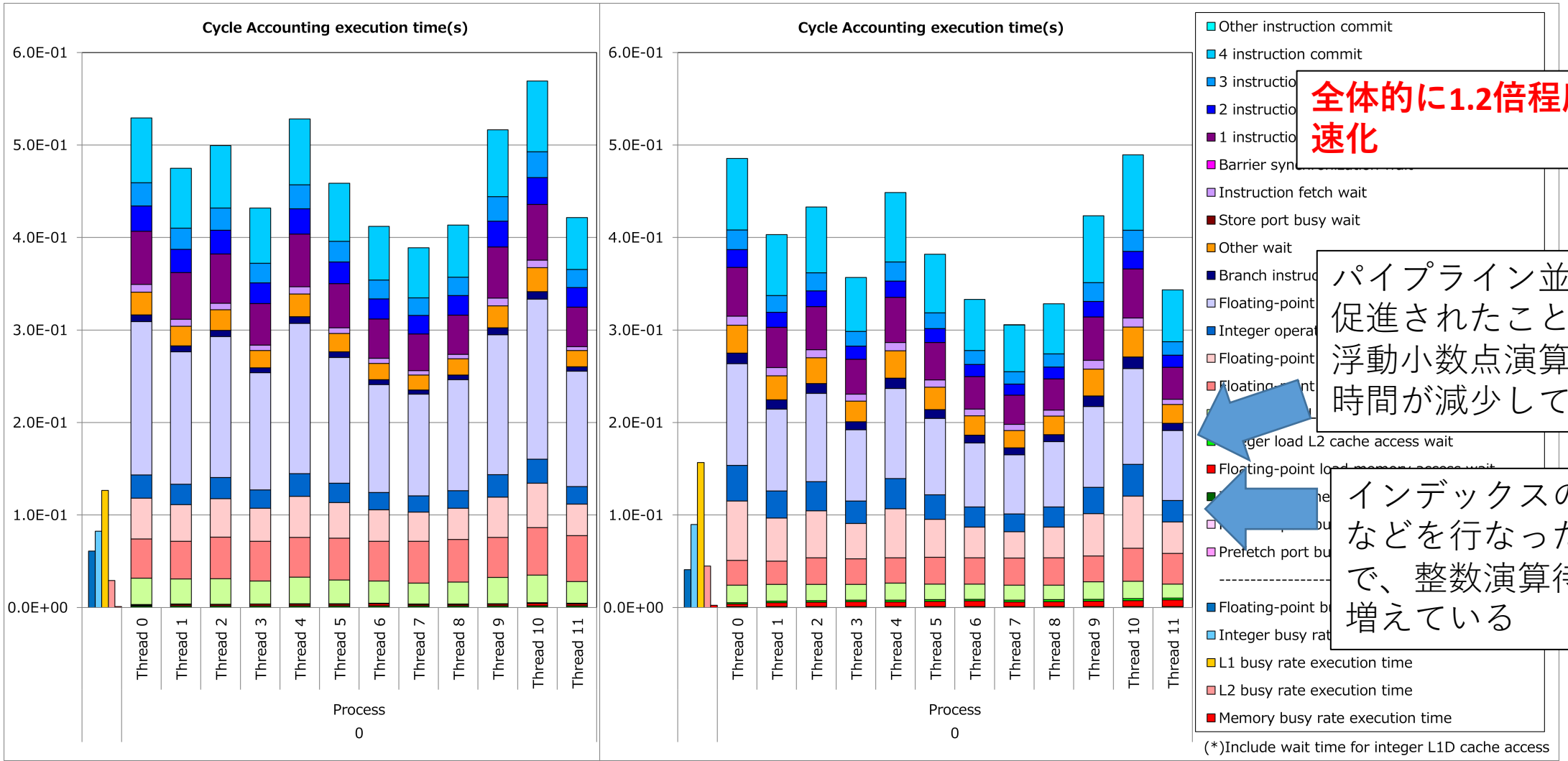
提案②実装の性能評価

- **今回はsub_ssssルーチンのみの適用（グラフもsub_ssssのみ）**
  - ✓81個の個別関数全てへの適用は今後の課題



提案②実装の性能評価

- **今回はsub_ssssルーチンのみの適用（グラフもsub_ssssのみ）**
  - ✓81個の個別関数全てへの適用は今後の課題



ATの適用に関する検討

- **今回行ったループのブロッキングパラメータはプロセッサによって最適な値が変わりうる**
- **81個の個別関数でも最適値が変わりうる**
- **入力データによって、カットオフが行われる割合に変化が出ることがわかっている**
  - ✓6-31G*基底ではカットオフされるのが68%程度に対し、cc-pVDZでは82%程度カットオフされ、実際に実行されるのは2割以下となる
- **これらの組み合わせを人手で検証するのは困難**

→ AT適用の効果があると考えられ、自動チューニング言語であるppOpen-ATによる自動チューニングを今後行う予定である

まとめ

- **分子軌道計算プログラムABINIT-MPに対し、性能分析を行った**
- **ループ構造が複雑であること、計算内容の異なる81種類のルーチンを呼び分けること、分岐後のループ長が短いこと、最内ループにカットオフ分岐があることから、ベクトル化が課題であることが判明した**
  - ✓ベクトル化向けのコードであるObara-vectorでも、実際にはほとんどベクトル化されていなかった
- **ベクトル化を念頭におき、実行順序のソート手法、ループ長の確保手法、ブロッキングによるベクトル化促進を行った**
  - ✓Sub_ssssルーチンに適用した結果、1.2倍程度の高速化を確認した

今後の課題

- 今回適用した実行順序の変更手法は、GPUなどのプロセッサでも効果的であると考えられ、GPU化が今後の課題である
- 最適化の適用はsub_ssssのみであり、他の80種のルーチンに対する適用も今後の課題である
- ブロッキングパラメータのチューニング等を81種類のルーチンや複数のプロセッサに対し適用することは人手は困難であり、今後ppOpen-ATを用いた自動最適化が課題である

提案①実装の性能評価

A64FX及びIntel Xeon (CascadeLake)における実行時間(秒)
5回実行した際の中央値

	データセット	6-31G	6-31G*	cc-pVDZ
A64FX (2.2GHz) OpenMP 48コア使用 3,379.2 GFLOPS 富士通コンパイラ4.9.0 –Kopenmp,fast,nosimd,ocl –Nlst=t	Obara	2.1	5.4	12.6
	Obara-vector	5.9	15.8	45.1
	提案手法(括弧内はソートの内時間)	2.5 (0.2)	5.8 (0.3)	17.1 (0.6)
Intel Xeon Platinum 8280 (CascadeLake) OpenMP 1ソケット28コア使用 2,419.2 GFLOPS Intel compiler 2021.7.1 –O3 –fopenmp	Obara	1.2	3.1	7.6
	Obara-vector	3.2	11.6	37.0
	提案手法(括弧内はソートの内時間)			

- **いずれもアダマンタン（$C_{10}H_{16}$）分子に対する計算**
- **基底関数が6-31G, 6-31G*, cc-pVDZで異なる**
  - ✓ 6-31Gではd関数がないので、呼び出されるサブルーチンは2^4=16パターンのみ
  - ✓ 6-31G*, cc-pVDZでは3^4=81通りのルーチンが呼び出される
  - ✓ cc-pVDZは水分子にもp型の分極関数を加えたもので、6-31G系より短縮長が長く、カットオフが多い
- **提案手法はObaraと比較すると少し遅いが、Obara-vectorより速い**
  - ✓ ソートは1コアのみで実行している。並列化により高速化の見込みはある
  - ✓ 少しのオーバーヘッドでベクトル長を稼げたことが成果
    - ➢ ただしこの段階では全くベクトル化はされていない