大阪大学
THE UNIVERSITY OF OSAKA

D3 CENTER
DIGITAL DESIGN, DATABILITY, AND DECISION INTELLIGENCE

スーパーコンピュータ
「不老・弐」へ向けたユーザ集会
2026.9.8@名古屋大学

GPU時代に向けた量子化学プログラムABINIT-MPの性能最適化

大阪大学D3センター　坂倉　耕太

本日の内容

1．mdxIIのご紹介

2．ABINIT-MPのGPU化

3．AIコーディング

mdxIIのご紹介

D3センター

大阪大学吹田キャンパス

OCTOPUS
Osaka university Compute & sTOrage Platform
Urging open Science

SQUID
Supercomputer for Quest to Unsolved Interdisciplinary Datascience

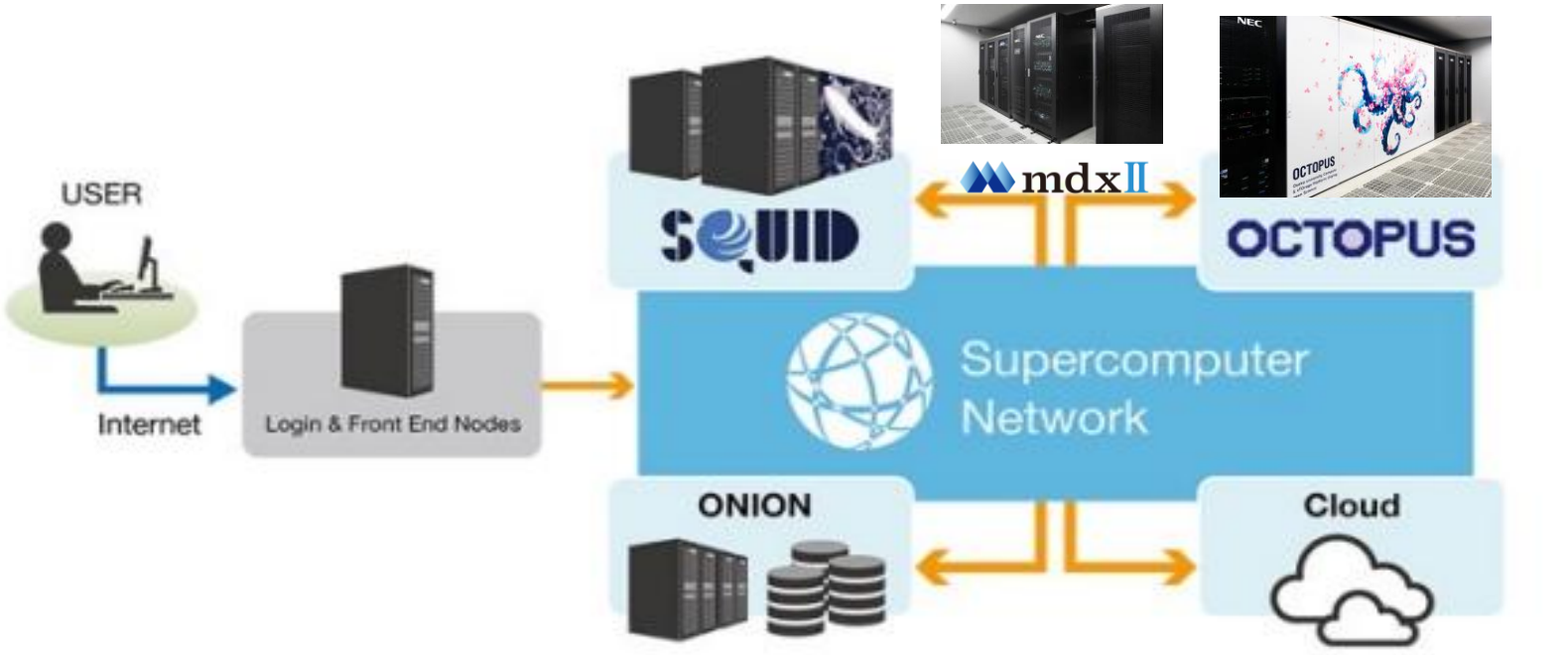
ONION

1 全国の研究者が利用可能

2 多様な計算ニーズへの対応

3 ペタフロップス級大規模計算能力

4 安定した動作環境の提供



計算基盤
- SQUID 16.594PFLOPS
- OCTOPUS後継機　2.293PFLOPS(2025.09稼働予定)

データ集約基盤　ONION
- オブジェクトストレージ　HyperStore(0.5PB)
- SQUID並列ファイルシステム　ExaScaler(21PB)

クラウドバースティング資源
- Micrsoft Azure
- Oracle Cloud Infrastructure

mdx II とは　＜学術向けIaaSクラウド>

OpenStack 基盤の**学術向け IaaS 型クラウド**。プロジェクトごとに独立した仮想環境を、必要な期間だけ低コストで利用

2,534
TFLOPS 総演算性能

60
H200 GPU

1.1
PB NVMe Lustre

1 **CPU ノード 60台**
Xeon Platinum 8480+ (Sapphire Rapids) ×2 ／ 512 GB

2 **GPU ノード 15台**
NVIDIA H200 SXM ×4 ／ 1 TB ／ 140 TFLOPS/ノード

3 **共同運営（9大学・2研究機関）**
初代 mdx は東京大学、mdx II は大阪大学 D3 センターに設置。
民間企業も利用可

パック単位で仮想マシンを構築

CPUパック / GPUパックを申請数に応じて組み合わせ、用途に応じたスペックの VM を構成（1 仮想コア = 0.5 物理コア）

CPUパック

1パックあたり

13,280 円/月～

CPU	**1 仮想コア**
メモリ	**2 GB**
最大 / VM	**224 パック**

例）100パック → 100仮想コア / 200 GB

GPUパック

1パックあたり

198,000 円/月～

GPU	**H200 ×1**
CPU / メモリ	**30 仮想コア / 246 GB**
最大 / VM	**4 パック（4 GPU）**

例）2パック → 2 GPU / 60仮想コア / 492 GB

ストレージ **900** 円/月 **1T**

「必要なときに必要な分だけ」

HPC 用途からデータプラットフォームのホスティングまで、独立した VM 環境で柔軟に運用

ABINIT-MPのGPU化

FMO法プログラムABINIT-MP

FMO法（フラグメント分子軌道法）

数万原子規模の巨大分子系にも対応

フラグメント間相互作用エネルギーIFIE/PIEDA

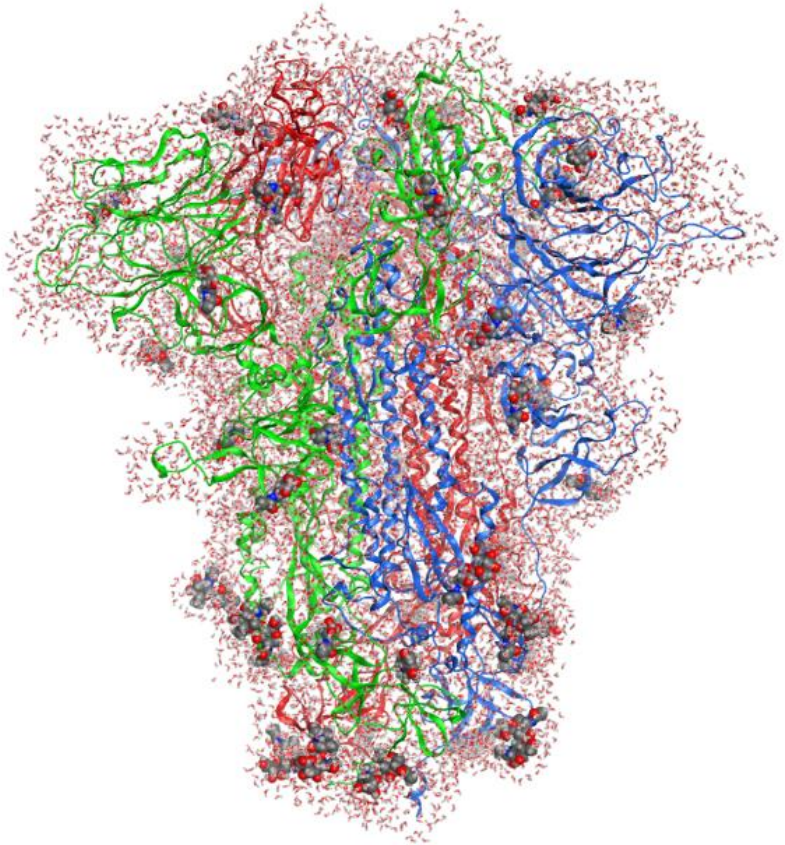
材料系、創薬分野に有益な情報が得られる
MP2法が必須（電子相関、分散力）

プログラム特性

・2重の収束計算処理（SCF,SCC）
・計算コスト分布は多様（様々な処理あり）
・数十万ライン数の巨大コード

実装環境

言語：Fortran
並列：MPI,MPI/OpenMP
動作実績：X86系,FX(富士通、富岳/不老),SX-AT（NEC,SQUID/AOBA）,Windows



Covid-19のスパイクタンパク質の水和モデル

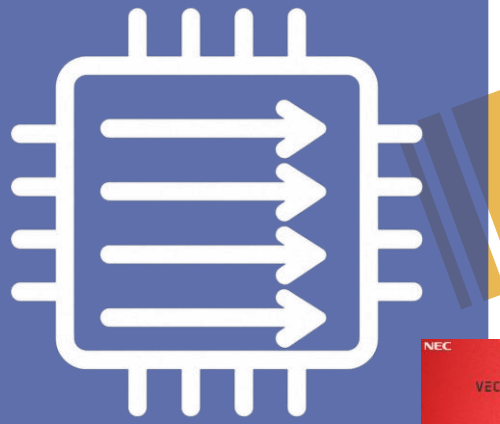
スパコンアーキテクチャの遷移、目標

HPC環境の複合型（Heterogeneous）へ急速転換

～2000年

地球シミュレータ

ベクトル型



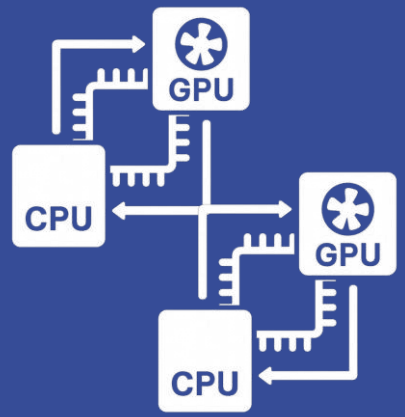
富岳,squid

PCクラスタ型

AMD intel FUJITSU A64FX

GPU複合型

不老弐(type2)
富岳NEXT,Miyabi



2020年～

intel AMD NVIDIA

次世代HPCを最大限活用の実現を目指す
アーキテクチャに対応したプログラミング、アルゴリズム開発が必須

GPU対応設計 -ABINIT-MPのGPU化指針-

富岳NEXT対応

- 富岳NEXTを見据えた設計と最適化

汎用（移植）性

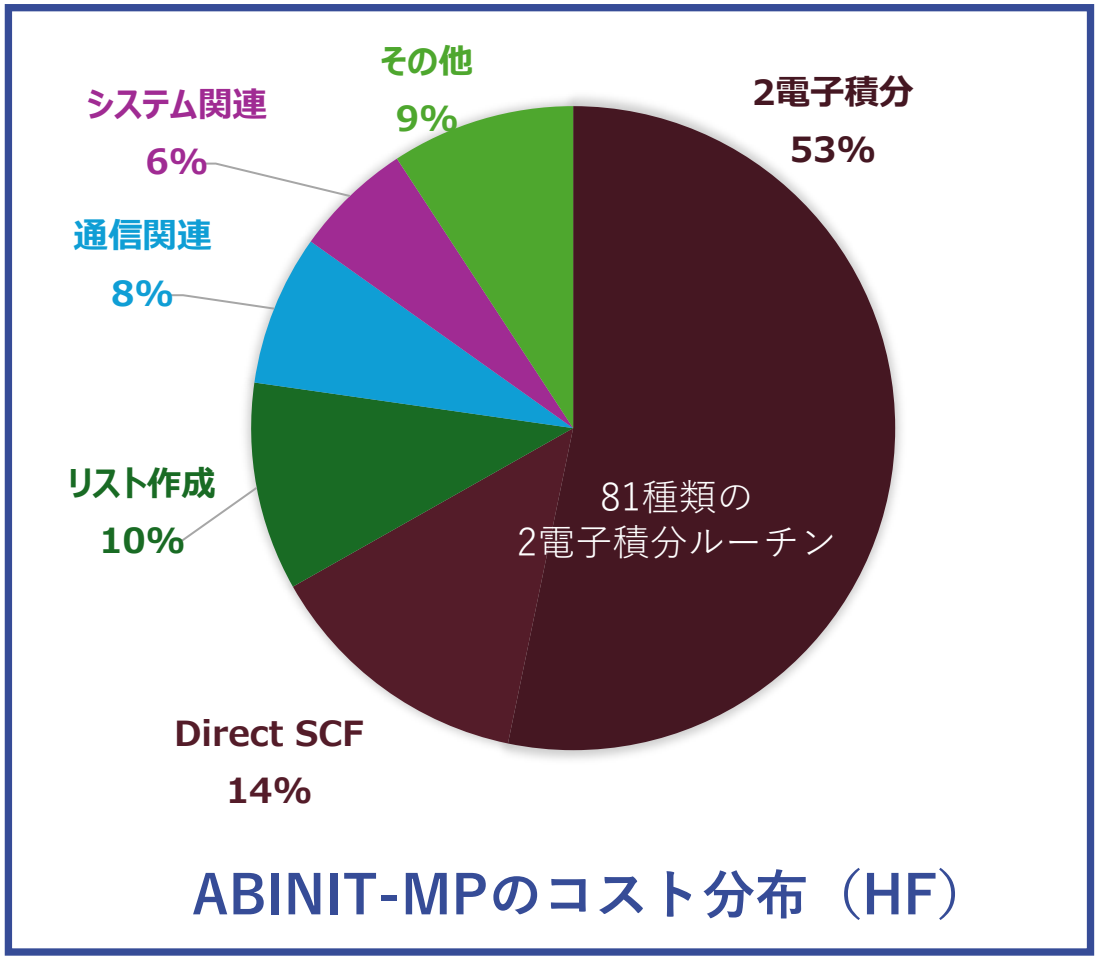
MPI,OpenMP,SIMD化,CPU->GPU転送対応

~~CUDA~~,OpenACC,~~OpenMP(offload)~~,~~HIP,SYCL~~

可読性・開発スピード

Fortran,~~C/C++~~

ABINIT-MPの特徴（コスト部分）

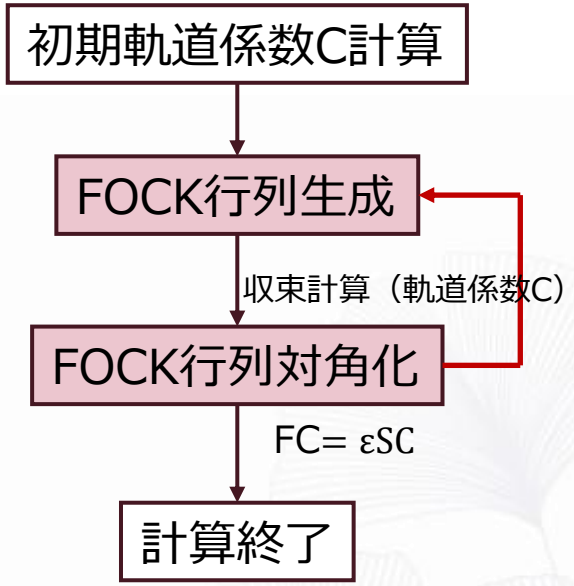


ABINIT-MPのコスト分布（HF）

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

2電子積分

$$(\mu\nu|\lambda\sigma) = \int d\boldsymbol{r_1} \int d\boldsymbol{r_2} \phi_\mu(\boldsymbol{r_1}) \phi_\nu(r_1) \frac{1}{r_{12}} \phi_\lambda(\boldsymbol{r_2}) \phi_\sigma(\boldsymbol{r_2})$$



大部分のコストは2電子積分計算とMP2積分変換部分

MP2計算のGPU化

2電子積分（MOベース）

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

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

2電子積分（AOベース）

- AO->MO変換は、4段階の行列積でコード化
- 計算コストはn^5のコスト
- 2電子積分(n^4)と合わせて計算の最コスト部分

AO->MO変換

Step 1： σ について縮約

$$(\mu\nu|\lambda b) = \sum_{\sigma} (\mu\nu|\lambda\sigma)\, C_{\sigma b}$$

Step 2： λ について縮約

$$(\mu\nu|jb) = \sum_{\lambda} (\mu\nu|\lambda b)\, C_{\lambda j}$$

Step 3： ν について縮約

$$(\mu a|jb) = \sum_{\nu} (\mu\nu|jb)\, C_{\nu a}$$

Step 4： μ について縮約

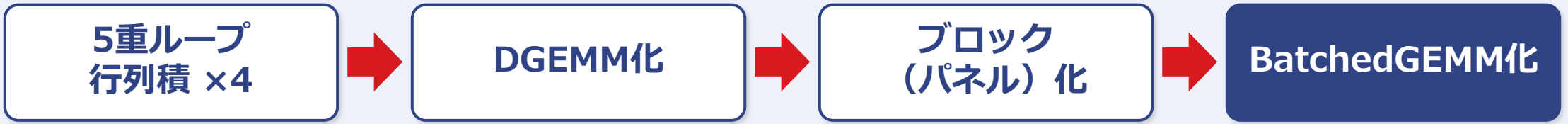
$$(ia|jb) = \sum_{\mu} (\mu a|jb)\, C_{\mu i}$$

GPU化の課題、開発手法

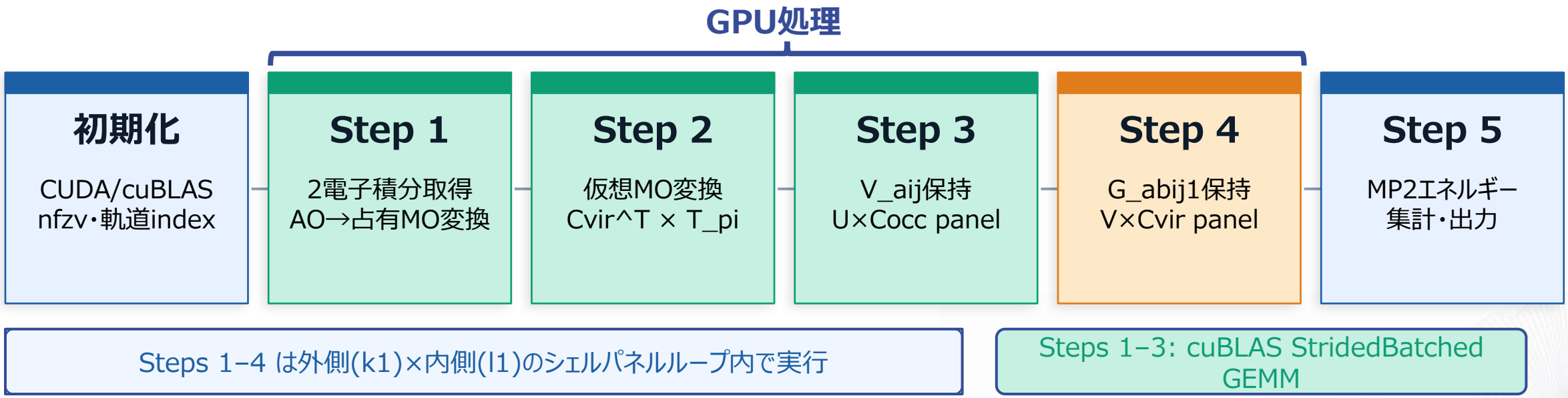
- **MPI/OpenMP併用**
  - GPU化が難しい処理はMPI/OpenMPで処理
  - MPS, MIGでGPU利用率を向上
- **計算粒度が小さく、インバランスが大きい**
  - 演算特性で21種類に分類
  - 演算量を見積もり降順ソート
  - 非同期処理
- **必要メモリは n^4（全保持は非現実的）＃MP2**
  - ブロック（パネル）化
  - cuBLAS StridedBatchedGEMM
- **フラグメント単位の処理（CPU–GPU転送が密）**
  - データ転送の最適化

- **複数プロセス → 1 GPU 共有**
- **小規模処理をまとめてGPUで処理**
- **中間テンソル・GPUメモリ量を削減**
- **cuBLAS の TensorCore で高速化**
- **SCC / SCF 収束計算下での最適化**

MP2 AO→MO 変換の実装の変遷



GPU化（MP2）　処理フロー



主な行列変換

- BUF_PQ (nbo×nbo×batch)　×　Cocc　→　T_pi (nbo×ndoc×IBR×IBS)
- Cvir^T (nvac×nbo)　　×　T_pi　→　U_ai (nvac×ndoc×IBR×IBS)
- U3 (nvac·ndoc × IBS)　×　B3　　→　V3　　→　V_aij 累積

GPU化（MP2）コードの技術的特徴

技術的特徴（MP2）

パネル化 Tiling 設計

GPUメモリを節約しつつ batched GEMM にまとめる

cuBLAS StridedBatched

NVIDIA最適化カーネルを適切に活用

対称性の利用

ij_pair = j(j-1)/2+i でメモリ・演算量を半減

積分スクリーニング

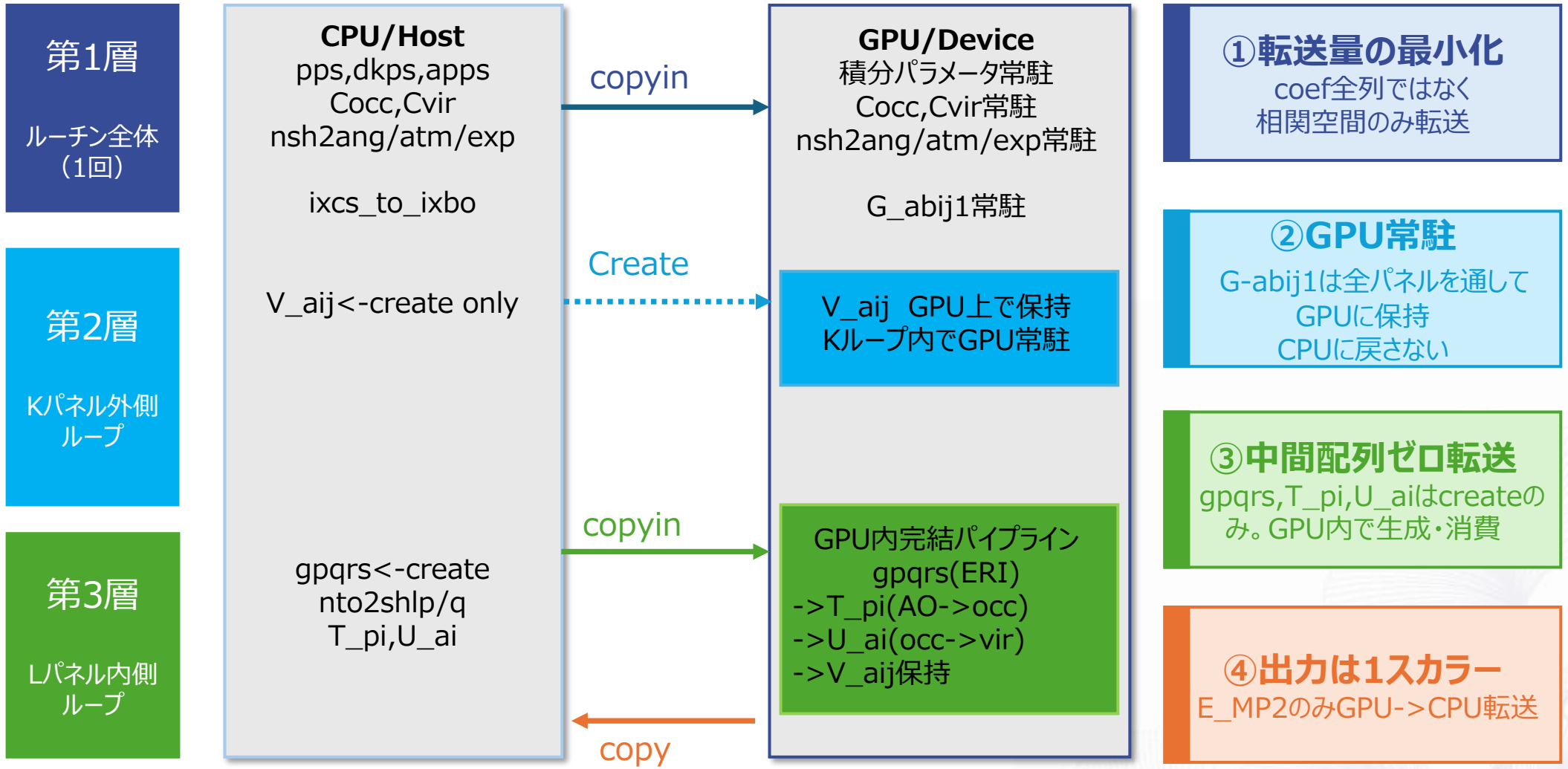
シェルペアインデックスによるスクリーニング

演算量重みによる再配置

ループ長積数による降順ソート

```
subroutine mp2_drive(nbo, nmo, nfzc, ndoc, coef, enemo, ixncs, ...) ! ERI shell-pair index tables included
   ! --- Frozen virtual screening (dynamic, threshold-based) ---
   if (chk_frozen_virtual) nfzv = count(enemo > thrfzv)
   Cocc(nbo,ndoc) = coef(:, nfzc+1 : nfzc+ndoc)  ! correlated occupied MOs
   Cvir(nbo,nvac) = coef(:, nfzc+ndoc+1 : nmo-nfzv) ! active virtual MOs
   ! --- GPU memory: G_abij1 persists across all panels ---
   !$acc enter data create(G_abij1(nvac, nvac, ndoc*(ndoc+1)/2)) ! ij symmetry
   do k1 = 1, ixncs, kb  ! outer shell panel (r-index)
      do l1 = 1, ixncs, lb ! inner shell panel (s-index, screened by swz)
         suberi_mp2_gpu2(gpqrs, ...) ! compute (pq|rs) on GPU; packed AO integrals
         ! Step 1: packed→dense + AO→occ MO [cuBLAS StridedBatched GEMM]
         BUF_PQ(nbo,nbo,IBR*IBS) = unpack(gpqrs) ! exploit (pq|rs)=(qp|rs)
         T_pi(nbo,ndoc,IBR,IBS) = DGEMM_batched(BUF_PQ, Cocc) ! stride=0 for Cocc
         ! Step 2: occ→vir MO [cuBLAS StridedBatched GEMM, Cvir^T]
         U_ai(nvac,ndoc,IBR,IBS) = DGEMM_batched(Cvir^T, T_pi)
         ! Step 3: contract s-panel into V_aij [cuBLAS, B3 zero-screened]
         V_aij(nvac,ndoc,ndoc,IBR) += DGEMM_batched(reshape(U_ai), screen(Cocc))
      end do ! l1
      ! Step 4: contract r-panel into G_abij1 [OpenACC parallel, i≤j]
      !$acc parallel  G_abij1(b,a,ij) += dot(V_aij(a,i,j,:), Cvir(:,b))  ! atomic
   end do ! k1
   ! Step 5: MP2 energy summation [OpenACC parallel, i≤j symmetry]
   !$acc parallel  E_MP2 += (2T-K)·T / (ε_i+ε_j-ε_a-ε_b)  ! T=direct, K=exch
end subroutine
```

GPU化（MP2）　データ転送



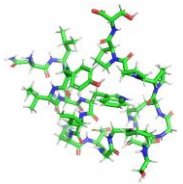
測定結果　TrpCage/6-31G/MP2@Wisteria-a

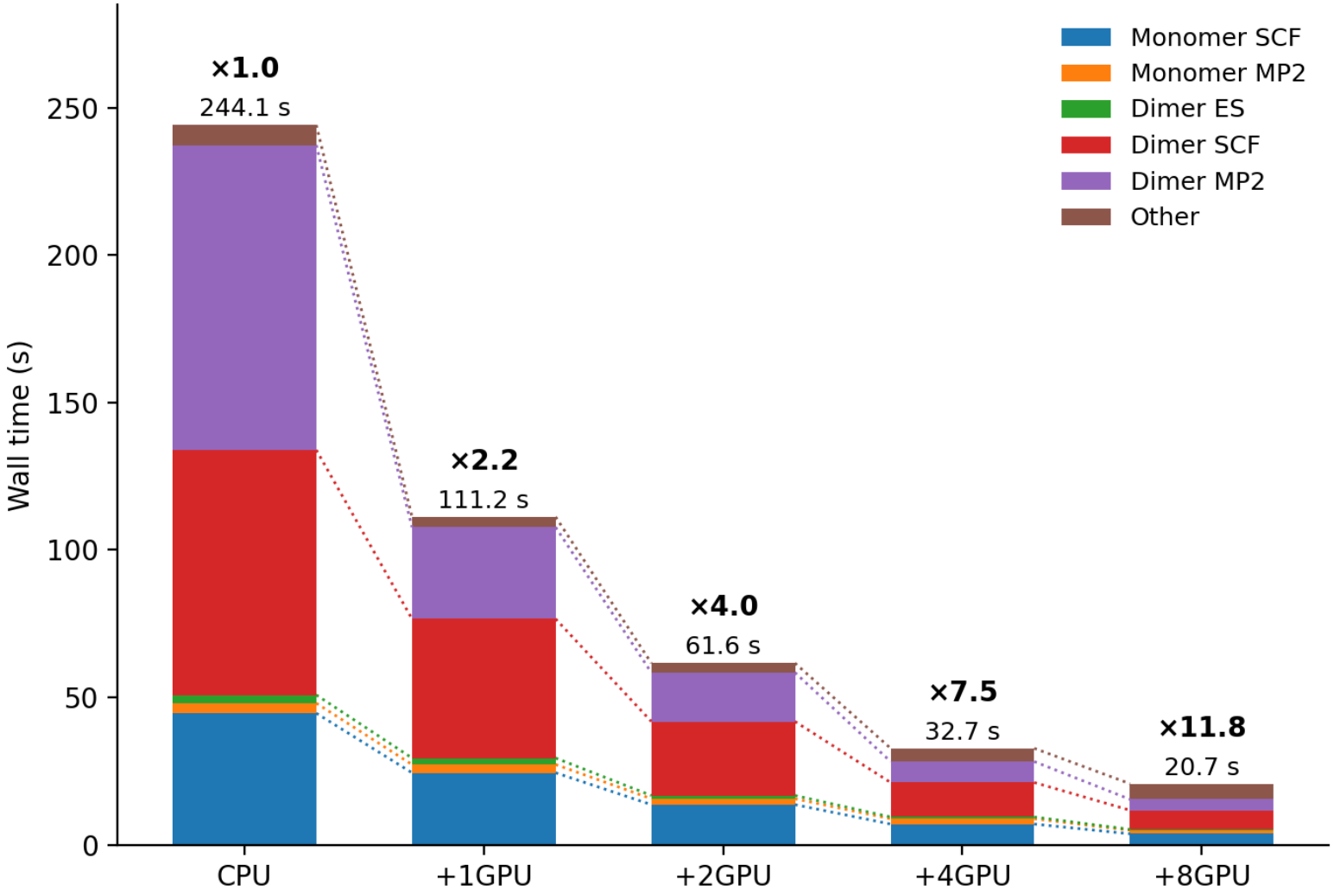
実行条件（MPI/OpenMP）
CPU:10MPI/7OpenMP

+1GPU:1MPI/8OpenMP
+2GPU:2MPI/8OpenMP
+4GPU:4MPI/8OpenMP
+8GPU:8MPI/8OpenMP

Wisteria BDEC-01

Wisteria-A(Aquarius)
CPU:Intel Xeon Platinum 8360Y 72core/node
GPU:NVIDIA A100　8基/node



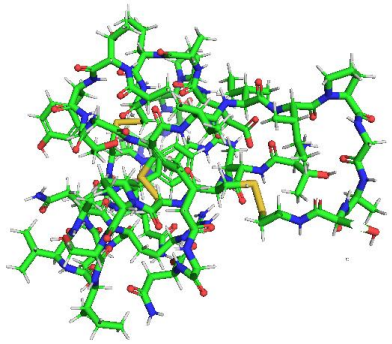


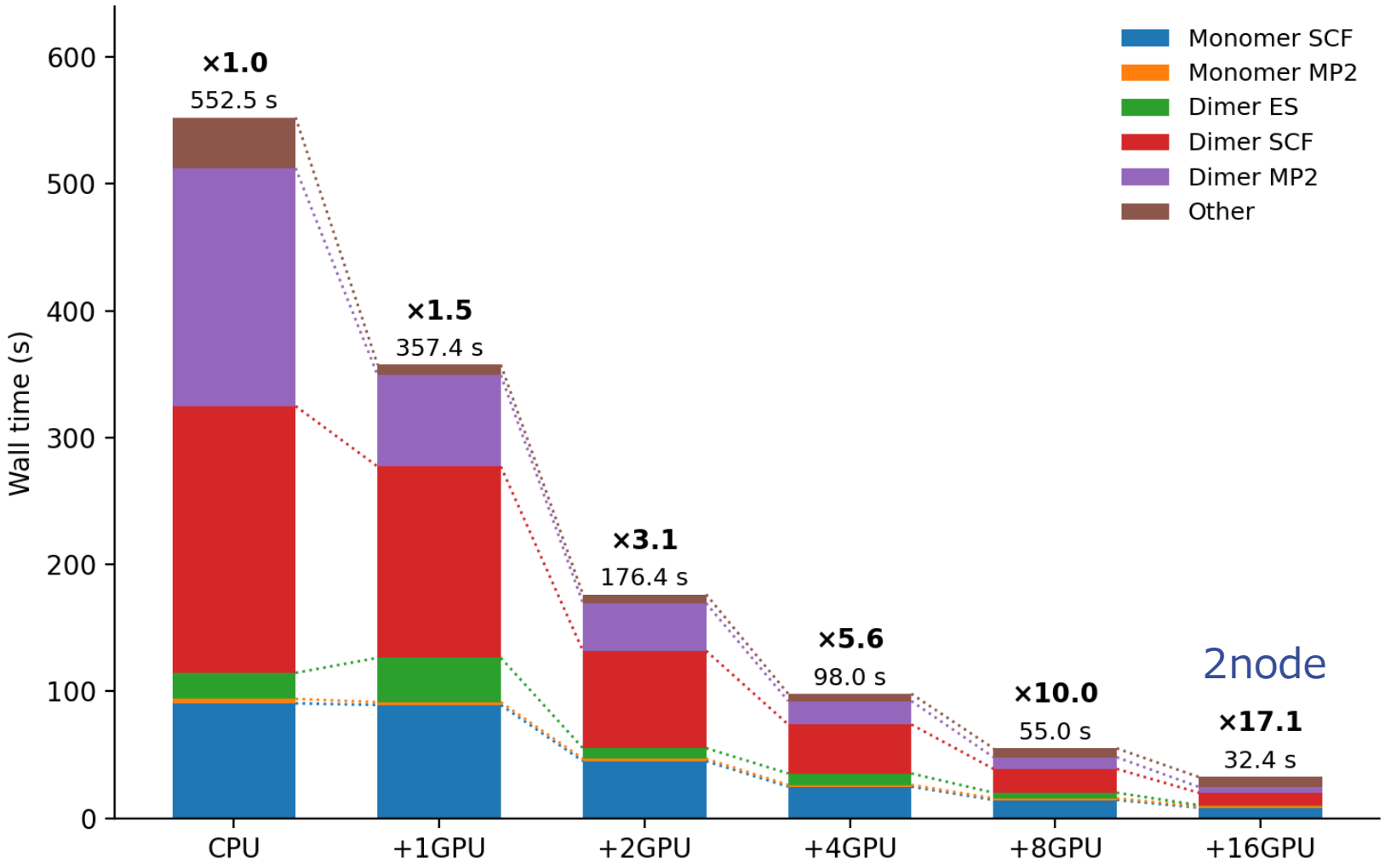
測定結果　Crambin/6-31G/MP2@Wisteria-a

実行条件（MPI/OpenMP）
CPU:10MPI/7OpenMP

+1GPU:1MPI/8OpenMP
+2GPU:2MPI/8OpenMP
+4GPU:4MPI/8OpenMP
+8GPU:8MPI/8OpenMP

+16GPU:8MPI/8OpenMP x 2node





AIコーディング

AI利用　ABINIT-MP 開発での活用範囲

コード生成だけでなく、最適化・デバッグ・ビルド・ジョブ投入まで開発サイクル全体で利用

1 **プログラム開発**
積分の d 軌道への拡張
MO 変換：単純ループ → batched GEMM

2 **最適化**
OpenMP 化、SIMD・ベクトル最適化
OpenACC 指示行のチューニング

3 **デバッグ**
正解値（Energy 等）を与えて自動デバッグ

4 **評価用ミニアプリの作成**
本体コードから該当機能だけを切り出し

5 **派生版のマージ**
分岐したブランチの統合・差分整理

6 **Make / ビルド環境の整備**
コンパイラ・最適化オプションの整理

7 **バッチジョブスクリプトの作成**
スケジューラ毎（SQUID / Genkai / Miyabi-G / 不老など）

「どうやって AI に自動でやらせるか」
人が書くのは、コードではなく "合否基準" と "手順"

プログラム開発（各積分ルーチンのd軌道拡張）

Dimer-ES integrals (Manual 6 / AI 15)

File	Shells	Impl.
gpu_des_ssss.F90	(ss|ss)	Manual
gpu_des_psss.F90	(ps|ss)	Manual
gpu_des_ppss.F90	(pp|ss)	Manual
gpu_des_psps.F90	(ps|ps)	Manual
gpu_des_ppps.F90	(pp|ps)	Manual
gpu_des_pppp.F90	(pp|pp)	Manual
gpu_des_dsss.F90	(ds|ss)	AI
gpu_des_dsps.F90	(ds|ps)	AI
gpu_des_dspp.F90	(ds|pp)	AI
gpu_des_dsds.F90	(ds|ds)	AI
gpu_des_dpss.F90	(dp|ss)	AI
gpu_des_dpps.F90	(dp|ps)	AI
gpu_des_dppp.F90	(dp|pp)	AI
gpu_des_dpds.F90	(dp|ds)	AI
gpu_des_dpdp.F90	(dp|dp)	AI
gpu_des_ddss.F90	(dd|ss)	AI
gpu_des_ddps.F90	(dd|ps)	AI
gpu_des_ddpp.F90	(dd|pp)	AI
gpu_des_ddds.F90	(dd|ds)	AI
gpu_des_dddp.F90	(dd|dp)	AI
gpu_des_dddd.F90	(dd|dd)	AI

Two-electron integrals (4-center) (Manual 6 / AI 16)

File	Shells	Impl.
gpu_ssss2.F90	(ss|ss)	Manual
gpu_psss2.F90	(ps|ss)	Manual
gpu_ppss2.F90	(pp|ss)	Manual
gpu_psps2.F90	(ps|ps)	Manual
gpu_ppps2.F90	(pp|ps)	Manual
gpu_pppp2.F90	(pp|pp)	Manual
gpu_dsss2.F90	(ds|ss)	AI
gpu_dsps2.F90	(ds|ps)	AI
gpu_dspp2.F90	(ds|pp)	AI
gpu_dsds2.F90	(ds|ds)	AI
gpu_dpss2.F90	(dp|ss)	AI
gpu_dpps2.F90	(dp|ps)	AI
gpu_dppp2.F90	(dp|pp)	AI
gpu_dpds2.F90	(dp|ds)	AI
gpu_dpdp2.F90	(dp|dp)	AI
gpu_ddss2.F90	(dd|ss)	AI
gpu_ddps2.F90	(dd|ps)	AI
gpu_ddpp2.F90	(dd|pp)	AI
gpu_ddds2.F90	(dd|ds)	AI
gpu_dddp2.F90	(dd|dp)	AI
gpu_dddd2.F90	(dd|dd)	AI
gpu_drest2.F90	d (rest)	AI

Environmental ESP integrals (3-center) (Manual 6 / AI 13)

File	Shells	Impl.
ifpr_ss_s_gpu.F90	ss / s	Manual
ifpr_ss_p_gpu.F90	ss / p	Manual
ifpr_ss_d_gpu.F90	ss / d	AI
ifpr_ps_s_gpu.F90	ps / s	Manual
ifpr_ps_p_gpu.F90	ps / p	Manual
ifpr_ps_d_gpu.F90	ps / d	AI
ifpr_pp_s_gpu.F90	pp / s	Manual
ifpr_pp_p_gpu.F90	pp / p	Manual
ifpr_pp_d_gpu.F90	pp / d	AI
ifpr_ds_s_gpu.F90	ds / s	AI
ifpr_ds_p_gpu.F90	ds / p	AI
ifpr_ds_d_gpu.F90	ds / d	AI
ifpr_dp_s_gpu.F90	dp / s	AI
ifpr_dp_p_gpu.F90	dp / p	AI
ifpr_dp_d_gpu.F90	dp / d	AI
ifpr_dd_s_gpu.F90	dd / s	AI
ifpr_dd_p_gpu.F90	dd / p	AI
ifpr_dd_d_gpu.F90	dd / d	AI
ifpr_xxdd_gpu.F90	xx / dd	AI

GPU化済s,p軌道ルーチン
＋
CPU用既存d軌道ルーチン

OpenAI Codex
Claude Code

GPU化d軌道ルーチン
（各数100-1000ライン）

年レベルの開発予定
→1日で完了

AIコーディング　プロンプトのイメージ

d軌道用2電子積分の GPU カーネルコード生成

GPU用のd軌道用の2電子積分を作ってください。

GPU用のs,p積分 gpu_ssss～gpu_pppp と、
CPU用のd積分を参考にすること。

実行方法、動作確認は ./data/run.sh で、
標準出力の Energy（Total）を確認すること。

一ファイル毎に開発 → 実行 → 動作確認し、
バージョン管理すること。

1 **参考コードだけで AI が解読**

アルゴリズム（OS/HGP）・**配列レイアウト・index 規約・対称性判定** — いずれも指示していない

2 **合否基準を Energy(Total) 一点に絞る**

生成 → 実行 → 差分確認 → 修正 の agentic loop を AI 単独で完走（人の介入なし）

3 **1 ファイル単位で開発・検証・commit**

CPU/GPU 版を #ifdef で共存、Git で差し戻し可能
都度作業レポートの作成

まとめ

- +GPUで10倍程度/nodeの加速を達成
  - ・大量の小規模計算処理に対して有効性を確認
  - ・実用レベルの実証として十分

- 公開版としてリリース予定（2027年度中）

- 実行条件の複雑化

「現在」ノード数、MPI数、OpenMP数
「今後」ノード数、MPI数、OpenMP数、GPU数（MPS,MIGの利用）

現行コードの改良では性能面で限界

RI積分(近似手法）、積分のインコア化、混合精度演算

機能を絞って、新規開発を目指す（FMO-X）

AI利用必須

人：アイデア
AI：実装
↓

AI：アイデア、実装
人：目的設定,意思決定

人がネック