PCクラスターコンソーシアムWS
＠理研CCS,　2026/07/02

生成AIを用いた少数量子多体計算コードの開発

筑波大学 計算科学研究センター
清水 則孝

青山茂義(高エネ研), 角田佑介(筑波大), 額田彰(筑波大)

Supported by 科研費：学術変革A (25H01268)
「精密数値計算が切り拓く宇宙の量子物質科学」
and 筑波大CCS 学際共同利用

Introduction

- 普段は、核子(陽子と中性子)を構成要素とする原子核の配位混合計算をやっています。
- 今回は、^{4}He原子を構成要素とする少数多体系（クラスター）の計算コードの開発です。
- 筑波大学計算科学研究センターに所属しており、原子核の研究や、スパコン・富岳NEXTに向けたGPUコードの開発をしています。

Sirius @ CCS Tsukuba
AMD Instinct MI300A

Miyabi-G Nvidia GH200
Miyabi-C Intel Xeon @ JCAHPC

Pegasus @ CCS Tsukuba
Nvidia H100

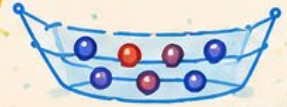
核子多体系をはじめとする少数量子多体系の計算

対象の例

- 原子核（軽い核から中重核まで）
  例）^{4}He, ^{16}O, ^{40}Ca, ^{48}Ca …

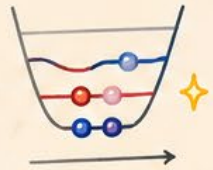
核子（陽子・中性子）の相関が鍵！

- 冷却原子の少数系
  例）^{6}Li, ^{40}K などの少数原子系
  （ユニタリー極限、ボソン・フェルミ混合系）

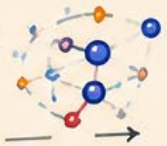


主なアプローチ

- **ガウス展開法（GEM）**
  基底関数をガウスの線形結合で展開
  高精度に相関を記述！
- **確率論的変分法（SVM）**
  モンテカルロで波動関数を最適化
  高次元・強相関系に強い！
- 量子モンテカルロ法（GFMC, AFDMC など）
- 変分法（VMC）
- 無限時間発展法（ITE）
- 有効場理論に基づく手法　など



核子間の強い相関を高精度に扱う！



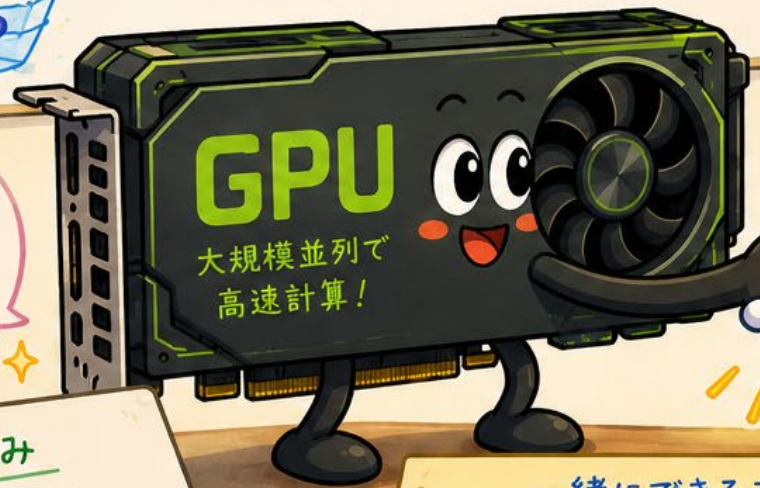
確率的にサンプリングして最適化！

```python
# many_body_nuclear.py
# Variational Monte Carlo for nuclei
import numpy as np
from qmcmodule import Hamiltonian, Sampler

H = Hamiltonian(nuclear_force='local_N2LO')
psi = TrialWaveFunction(params)
sampler = Sampler(psi, n_walkers=10000)

E, dE = sampler.optimize(H)
print(f"Ground state energy: {E:.6f} +/- {dE:.6f} MeV")

# Few-body cold atoms (unitary Fermi gas))
result = few_body_solver(
    interaction='unitary',
    N=6, mass_ratio=6.64)
print(result)
```



いい計算をしてくれてありがとう！僕はアルゴリズムとコードでサポートするよ！

複雑な相関を解き明かすのが面白いんだ！

GPUの強み

- ☑ 大規模並列計算
- ☑ 高いメモリ帯域
- ☑ 多数のサンプルを高速生成
- ☑ 行列・テンソル演算に強い！

一緒にできること

- ▷ 原子核の構造・反応の高精度予測
- ▷ 冷却原子少数系の普遍性解明
- ▷ 新しい相互作用の探索
- ▷ 量子シミュレーションの発展

物質の根源に迫ろう！

AIコーディングの強み

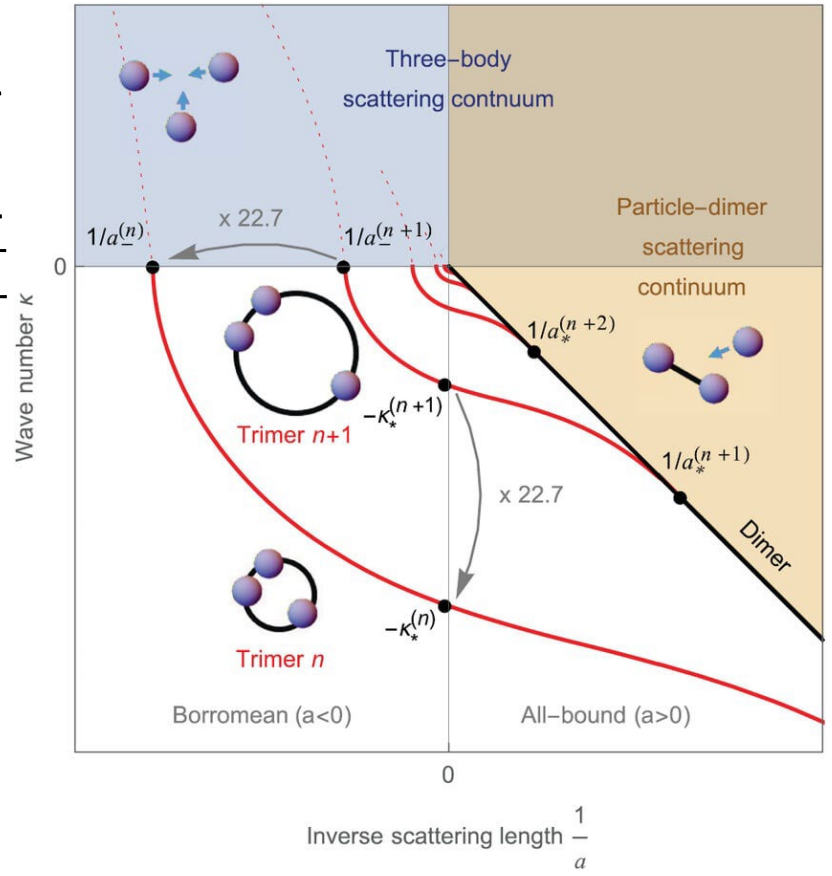
- ☑ アルゴリズム設計
- ☑ 高速なコード実装（C++/CUDAなど）
- ☑ バグ検出と最適化
- ☑ 可視化・解析を自動化

ヘリウム4原子の少数クラスター系

- スピンレスボゾン系(波動関数は交換対称)

- ファンデルワールス力によって、浅く束縛して少数系を作る。原子内の電子のパウリ排他律による強い斥力芯があり、高精度の量子多体計算が必要。

- Trimer(3体系)の励起状態：エフィモフ状態
  - N体系では？



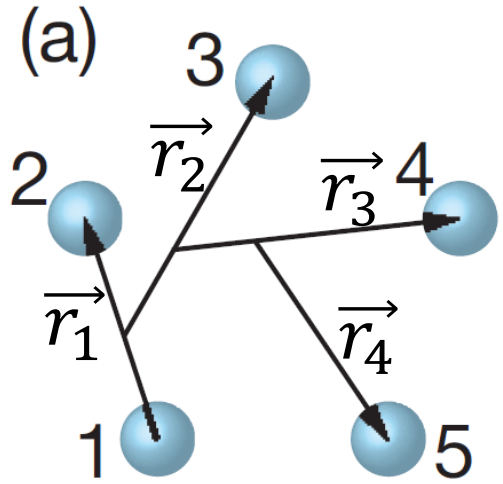
P. Naidon and S. Endo, "Efimov Physics: a review"

Stochastic Variational Method (SVM)による少数多体計算

- ヤコビ座標で表現された相関ガウス基底の重ね合わせで波動関数を表現する。

$$\Psi_{N_b}(\mathbf{x}) = \sum_{\mathbf{k=1}}^{\mathbf{N_b}} \mathbf{c_k} \mathcal{S} \phi_{\mathbf{k}}(\mathbf{x})$$

$$\phi_A(\mathbf{x}) = \exp\left(-\frac{\mathbf{1}}{\mathbf{2}}\mathbf{x^T A x}\right) = \exp\left(-\frac{\mathbf{1}}{\mathbf{2}}\sum_{\mathbf{i,j=1}}^{\mathbf{N_j}} \mathbf{A_{ij}(r_i, r_j)}\right)$$



- $A_{ij}^{(k)}$が変分パラメータ。5体系では4x4行列。
- 基底数(N_b)を一つずつ4800まで増やしていく。
- 基底を増やすごとに、乱数で多数($N_t = 1200$)の基底の候補を生成して、ハミルトニアン行列を対角化、エネルギー期待値が下がるものを次の基底として選ぶ。（候補ごとに並列計算可能）

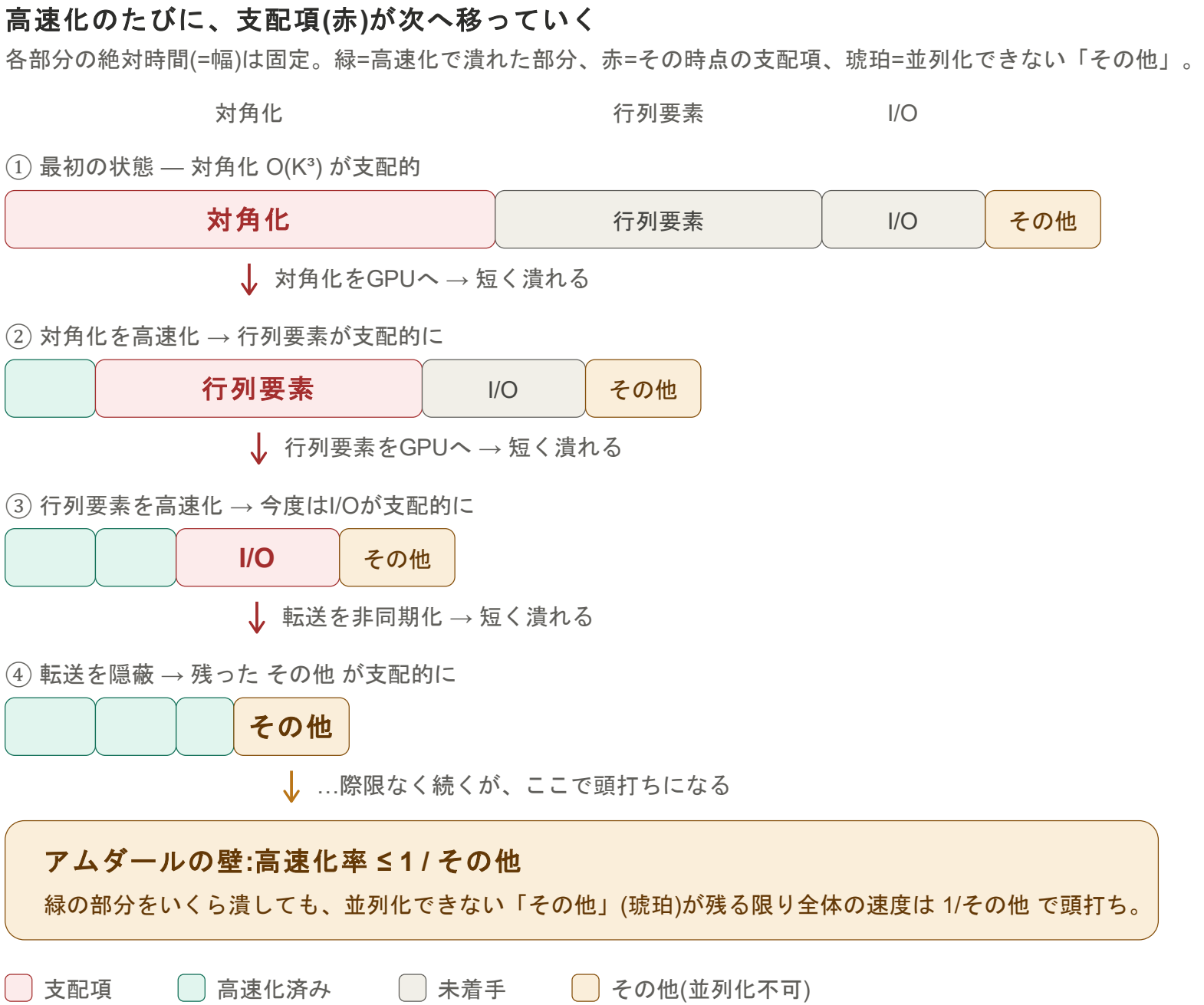
GPU対応は「もぐらたたき」

人間がコーディングしていると、どこかで手間と高速化のトレードオフを考えて途中でうちきり。

AIを用いた高速化で限界まで高速化することが可能。

チューニング・GPU対応

- ルールが明確、performanceも明確に定義できる。
- AIコーディングがもっとも有用な分野のひとつ



（私の）AIコーディング手順

- 自然言語（日本語）で指示。定式化から確認していき、メモを作る。
- メモに基づいて小さく・正しく動くコードを作っていく。
- 定式化＋コーディングを繰り返して機能をふやしていく。先行研究との比較や、物理を考えて正しいかを確認するのが人間の主な仕事。

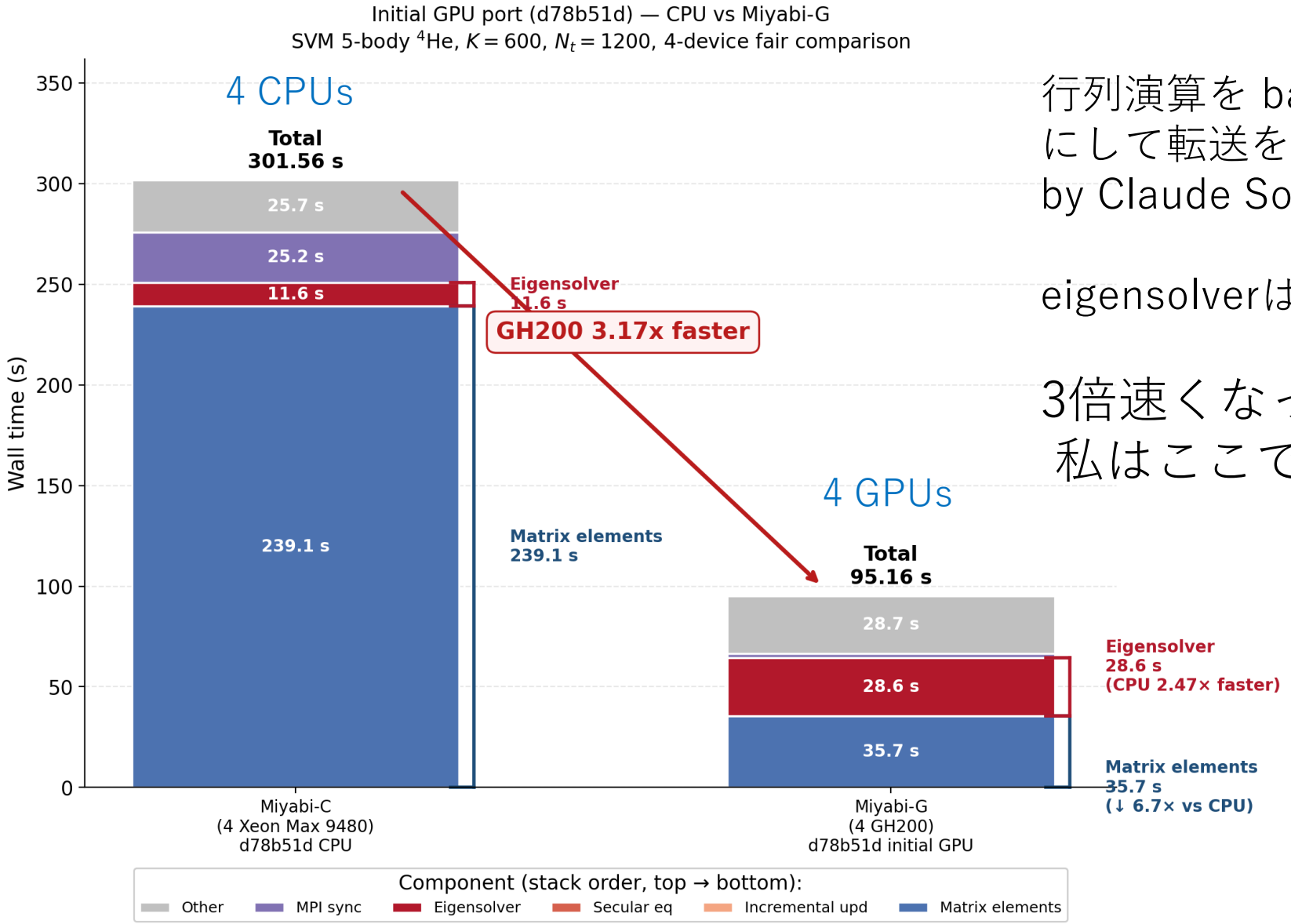
- 一通り完成の後、チューニング作業、GPUへの移植。CPUの計算結果と変わらないか確認すればよいので、かなり自動化できる。
  - AIが常に計算結果を確認して、必要ならデバッグしてくれる。
  - AIのチューニング事前性能予測はすごくもっともらしいが、信用してはならない。検討・実際に実装して計測。
  - 人間は、壁打ちして計算手法のアイデアをだす。

AIコーディング (Claude CLI)

- 利点
  - コード開発が高速。try and error を多数試せる。
  - 効果が無くてもダメージが小さいので、大がかりな改変を必要とする最適化を気軽に試せる。
  - Libraryのインターフェースや細かい言語仕様を「よきにはからってくれる」
  - こちらが知らない計算手法も提案してくれる

- 注意点
  - コンテキストウィンドウの制限：一定期間作業していると、「忘れる」
  - 時々、間違ったことを定量的にもっともらしく言う。
  - ライブラリを使うより、自前で実装したがる
  - すごいスピードでどんどん複雑なコードができあがっていく

最初のGPU移植



行列演算を batched cuBLASライブラリにして転送を最小限にした最初の移植版
by Claude Sonnet 4.6

eigensolverはCPU実行

3倍速くなった！
私はここで満足していた、、、

ベンチマーク用小規模設定
SVM: $N_t = 1200, N_b = 600$

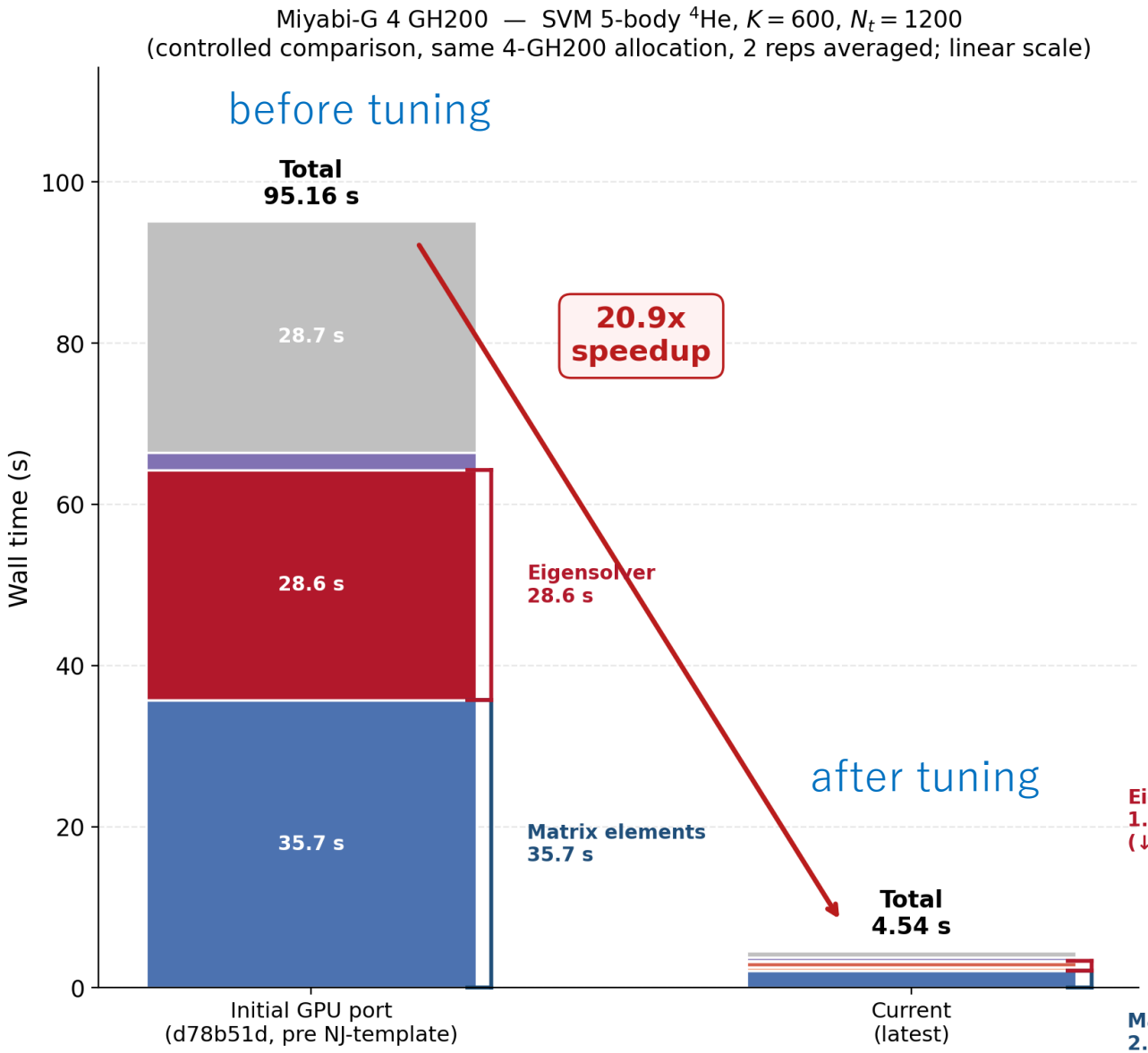
Xeon CPU x 4 @ Miyabi-C
GH200 x 4 @ Miyabi-g

GPU高速化

- CUDA(HIP)実装、cuBLAS, hipBLASの利用。
- 多数の候補のエネルギー期待値計算をチャンクしてまとめて計算することにより、計算密度の確保。
  - 行列ベクトル積(dgemv)→行列行列積(dgemm)化による高速化
- Coalesced accessに配慮したループ
- 一般化固有値問題の永年方程式解法の導入・GPU実装。
- 自前のカーネルを用意して、中間配列の消去
  - 配列のサイズはコンパイル時定数
  - Cholesky 分解＋逆行列
  - Sherman-Morrison update
  - アルゴリズムを register 常駐 + unroll + 1 kernel で動かす

自然言語で指示して、AIによるコード実装

Performance tuning at Miyabi-G (GH200) by Claude Opus



Eigensolver:
　永年方程式による解法の導入
Matrix elements:
　address coalescing
　独自kernelによるregister保持

　実行性能 12.3%(実測)

さらに 20倍速くなった！

ベンチマーク用小規模設定
SVM: $N_t = 1200, N_b = 600$

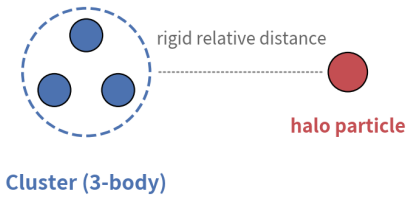
最大の落とし穴 — クラスタ間非対角行列要素

AI の「物理的に必須」主張を疑い、実機検証で解決

症状: 対角化で偽の固有値が、基底状態エネルギーより下に現れて計算が壊れる（Efimov 的 halo を含む試行波動関数で発生）

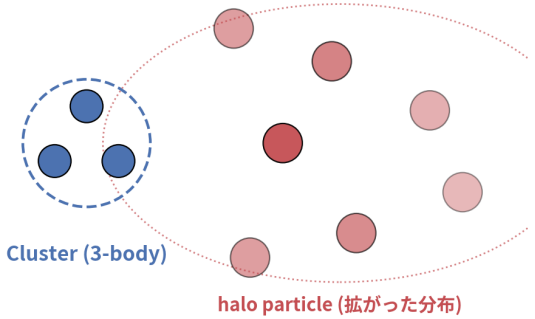
Without inter_cluster_rot
Cluster と halo 粒子の運動を独立に扱う (非対角 = 0)

halo radial と cluster 内部運動が無相関

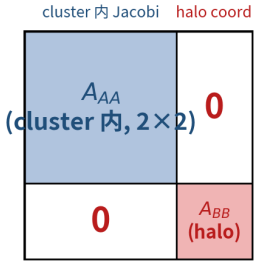


With inter_cluster_rot
halo 粒子の広がり ↔ cluster 内運動 を結合 (非対角 ≠ 0)

Efimov-like: halo の位置 ↔ cluster 内部運動 を相関

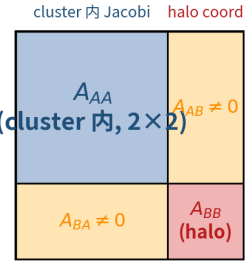


$\psi_k(\mathbf{x}) = \exp\left(-\frac{1}{2}\mathbf{x}^\top A_k \mathbf{x}\right)$ ： A は block diagonal



→ halo 自由度が独立に扱われ、偽固有値出現

$\psi_k(\mathbf{x}) = \exp\left(-\frac{1}{2}\mathbf{x}^\top A_k \mathbf{x}\right)$ ： A は full (cluster ↔ halo カップリング)



→ Efimov 的 halo を効率的に表現、偽固有値消失

AI の主張

- cluster ↔ halo の非対角 = 0 にすべき
- cluster の「特徴をキープ」に必須
- halo と結合すると cluster 性が崩れる

(強くこだわった)

- 直交性の検証を繰り返し提案

ユーザー判断 → 検証

- 非対角に有限値を入れろと指示
- 試行 → 偽固有値が消失
- 恒久対策: inter_cluster_rot 導入
  − (cluster ↔ halo 結合自由度, Efimov 的 halo を効率表現)

教訓: AI が「物理的に必須」と強く主張しても、最終判断はユーザーの直感 + 実機検証。確信度の高い出力ほど疑う規律が要る。

まとめ

- Stochastic Variational Methodによるヘリウム原子クラスターの少数量子多体計算コードを、AIコーディングによりfull scratchから開発した。
  - GPUに最適化、大きな効果を得た。ピーク比実行性能～12.3% @ GH200

- 物理学者がHPC用コードを書く必要がない時代が来ている
  - GPU移植・最適化作業は AI coding で負担が大きく軽減、高性能
    - 多数の試行錯誤・大きな改変も気軽に試せる。
    - CUDAを覚えなくても、高度な最適化が可能
    - ドメイン知識は重要
  - pythonの解析コードなどを短時間で移植・最適化できる

- AIが研究をする時代？
  - AIは作業が速く、 何でも知っているが時々知ったかぶりする共同研究者
  - AIによる研究遂行・論文執筆も現実的
  - 人間の役割は？