卒 業 論 文

広島大学情報科学部

諸隈颯

令和７年２月

卒 業 論 文

GPUを用いたSCF収束アルゴリズムの高速化

指導教官　中野浩嗣 教授

広島大学
情報科学部情報科学科

B210245 諸隈颯

提出年月: 令和 7 年 2 月

概 要

量子化学計算は，量子力学に基づき，実際に実験を行うことなくコンピュータによる計算のみで分子の特性や電子の軌道を求めることであり，化学や創薬など様々な分野で応用されている．量子化学計算では，シュレディンガー方程式を解くことで分子の特性や電子の軌道を求めるが，一般的に解くことが困難とされているため，様々な近似を用いることによって，ローターン方程式のような数値的に解くことのできる形に変形して解く．ローターン方程式を解には収束するまで繰り返し計算を行う必要があるが，ループ回数を削減させるために DIIS などの収束アルゴリズムを用いる．ローターン方程式の高速な収束には DIIS 計算の高速化が必要である．

本研究では，並列処理を得意とする GPU（Graphics Processing Unit）を活用し，DIIS 計算の高速化を実現した．同様の DIIS 計算を行う CPU 実装と比較して最大で 1.70849 倍の高速化を達成した．

目 次

図 目 次

第1章　はじめに

1.1　研究の背景

量子化学計算とは，量子力学に基づき，実際に実験を行うことなくコンピュータによる計算のみで分子の特性や電子の軌道を求めることであり，化学や創薬など様々な分野で応用されている．

分子の特性や分子軌道を近似的に求める手法の１つとして，ハートリー・フォック法がある．ハートリー・フォック法では，対応するハートリー・フォック方程式を解く必要があるが，この方程式は微積分方程式であるため，直接代数的に解くことができない．そこで，これを代数的に取り扱える形に変形したものがローターン方程式である．

ローターン方程式はエネルギーが収束するまで繰り返し計算を行う必要があるが，分子のサイズが大きくなるほど繰り返し回数は増加する．繰り返し回数を削減するアルゴリズムの１つに DIIS(Direct Inversion in the Iterative Subspace) がある．

GPU（Graphics Processing Unit）は複数のコアを用いた大量の並列処理を行うことができるハードウェアであり，画像処理に優れている．この高い並列処理性能を画像処理以外の計算に用いる分野である GPGPU（General-Purpose computing on Graphics Processing Units）が近年注目されており，量子化学計算の分野でも用いられている．

DIIS は行列演算を伴うため，行列サイズの増大に伴い計算時間が増加する．そこで，本論文では，ローターン方程式の収束アルゴリズムの１つである DIIS に対し，GPU を活用した効率的な並列計算アルゴリズムを設計することを目的とする．また，DIIS を GPU 上で実装し，CPU 上で逐次処理するアルゴリズムと，分子サイズや基底関数の異なる条件下で比較を行う．

1.2　本論文の概要

本論文では，第２章で GPU と GPU を扱うための CUDA プログラミングについて説明する．第３章では量子化学計算の概要について説明する．第４章では，ローターン方程式の収束アルゴリズムの１つである DIIS について説明する．第５章では本論文の提案手法を説明し，第６章では実験結果および考察を行う．最後に第７章で，まとめと今後の課題を説明する．

第2章　GPU・CUDA

本章では，本研究で使用したGPUおよびGPU向けの開発環境であるCUDAについて説明する．[1]

2.1　GPU

GPU(Graphics Processing Unit)とは，画像処理を目的として開発されたデバイスである．GPUには大量のコアが搭載されており，大量のスレッドを並列処理することが可能である．

2.1.1　GPUアーキテクチャの構造

GPUアーキテクチャの構造を図2.1に示す．GPUには，SM（Streaming Multiprocessor）と呼ばれる演算器と，そのデータを格納するグローバルメモリから構成される．GPUにはSMが複数搭載されており，それぞれが独立に動作することで並列計算を行っている．

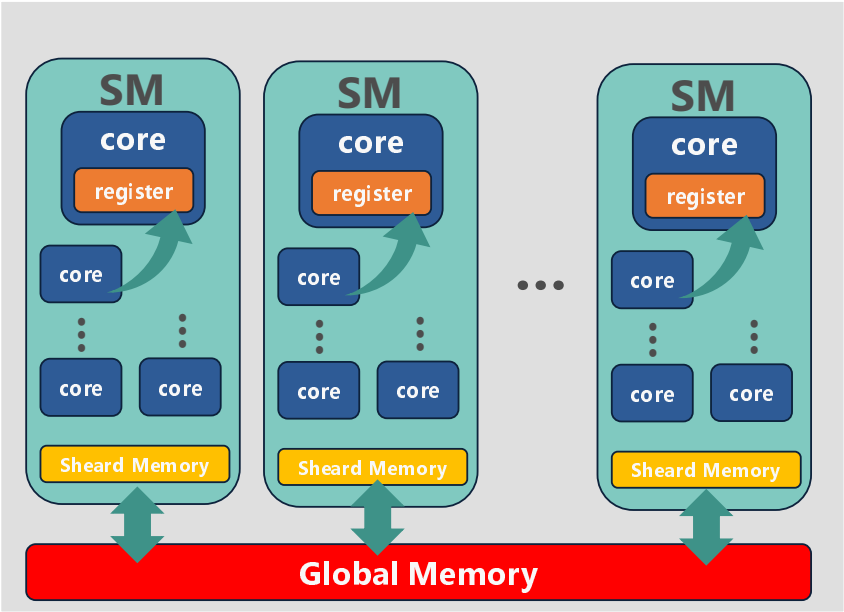

図 2.1: GPUアーキテクチャ

2.1.2 SM(Streaming Multiprocessor)

SM はGPU に複数個搭載されている計算ユニットであり，多数の core と Shared Memory から構成される．core は演算を行う最小単位であり，各コアが命令を実行する．また，各コアは Register と呼ばれるメモリを持つ．

2.1.3 Global Memory

Global Memory はGPU 全体で共有されるメモリである．GPU 内で最も容量の大きいメモリでありGPU 内の全てのコアからアクセス可能である．しかし，アクセス速度はShared Memory や Register と比べて遅い．CPU と GPU ではメモリが分かれており，CPU-GPU 間でのデータのやり取りには Global Memory が使用される．

2.2 CUDA

CUDA(Compute Unified Device Architecture) はNVIDIA が提供する GPU 向けの統合開発環境である．主に C/C++言語をベースに拡張したプログラミング言語を提供している．

2.2.1 プログラミングモデル

CUDA プログラムは，CPU 側で実行される部分と GPU 側で実行されるカーネル（kernel）に分かれる。カーネルは，GPU 上で実行される関数であり，並列処理により高速な計算が可能となる．また，GPU と CPU は異なるメモリ空間を使用するため，GPU で処理を行う際にはCPU から GPU へデータを転送する必要がある．(図 2.2 参照).

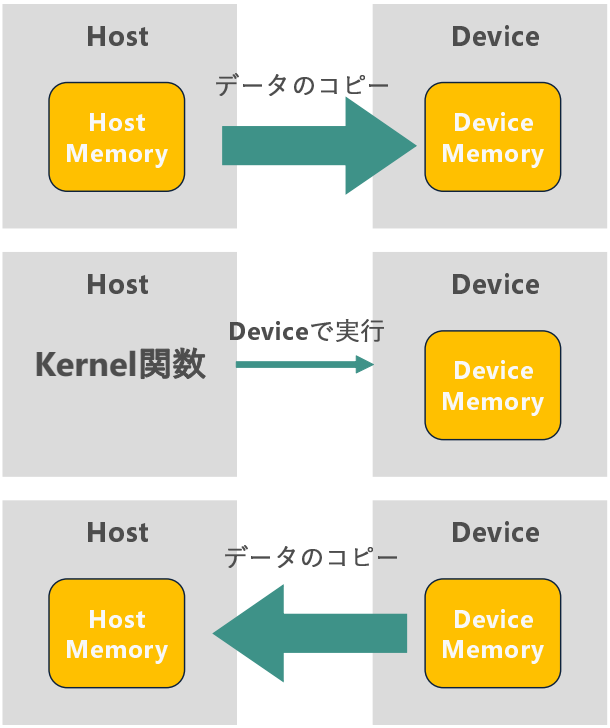

図 2.2: GPU 処理の流れ

2.2.2 階層的スレッド構造

図 2.3 のように CUDA は無数のスレッドを Grid, Block, Thread の階層的な構成で管理している．Grid>Block>Thread の順で大きさが決まっており，最小単位の Thread1 個が 1 個の core と対応している．Kernel の実行時に Grid, Block のサイズを指定することができる．全ての Thread は独立しており，固有の ID が与えられている．Kernel では，ID を指定して 1 つのスレッドが動作する内容を記述することで，指定されたすべての Thread が並列に命令を実行する．

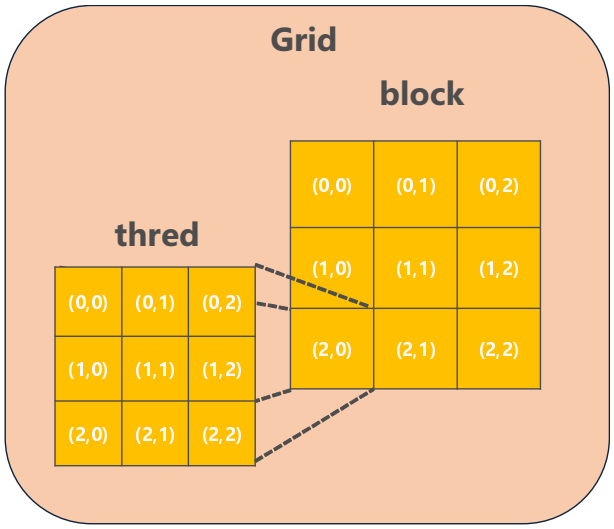

図 2.3: スレッド構造

2.2.3 CUDA メモリアーキテクチャ

節 2.2.2 で説明した Grid, Block, Thread のメモリ構造を，図 2.4 に示す．図が示すように Block, Thread は GPU の SM, core に対応している．そのため，Thrad ごとに Register が，Block ごとに Sheard Memory が搭載されている．そもため Sheard Memory のデータは同じ Block 内の Thread でのみ共有可能である．また，Grid は 1 つのカーネルを実行するときに指定した Block の集合である．Grid 内のすべての Thread は Global Memory にアクセスすることができる．

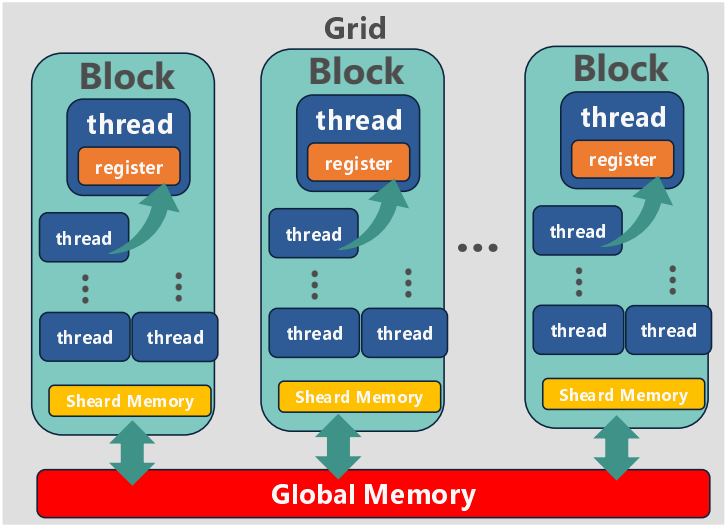

図 2.4: メモリ構造

第3章　量子化学計算

3.1　量子化学計算の概要

量子化学計算とは，量子力学の原理に基づいて分子や原子の特性や電子の軌道などを，実験をすることなく計算で解析する学問である．量子化学計算は，製造や創薬など様々な分野で応用されている．[2]

量子化学計算では，シュレディンガー方程式を解くことで波動関数やエネルギーを求めてその値から原子や分子の性質を調べる．しかし，シュレディンガー方程式が厳密に解くことができる場合は非常に限られている．したがってほとんどの場合で，様々な近似手法を用いて計算をすることが一般的である．

3.2　シュレディンガー方程式

シュレディンガー方程式は，量子力学における基礎方程式であり，量子力学において粒子の状態を記述する方程式である．[3] シュレディンガー方程式には，時間に依存するものと依存しないものの２つに分けられる．時間に依存しないシュレディンガー方程式を式3.1 に示す.

$$\hat{H}\Psi = E\Psi \tag{3.1}$$

Ψ は波動関数と呼ばれる粒子の状態を定める関数であり，E はその状態に対応するエネルギーである．$\hat{H}$ はハミルトニアンと呼ばれる演算子である．時間に依存しないシュレディンガー方程式は，ある波動関数 Ψ に演算子 $\hat{H}$ を施した場合に，その結果が元の波動関数に定数 E をかけたものと等しいという意味である．このような形の方程式を固有値方程式という.

ここで，1 電子におけるハミルトニアンを式??に，多粒子系におけるハミルトニアンを式??に示す.

$$\hat{H} = -\frac{1}{2}\nabla^2 + V(\boldsymbol{r}) \tag{3.2}$$

右辺の第１項は電子の運動エネルギーであり，∇^2 は x, y, z の３方向で２回微分をするという演算子である．また，$V(\boldsymbol{r})$ は電子のポテンシャルエネルギーである.

次に多粒子系のハミルトニアンを式 3.3 に示す.

$$\hat{H} = \sum_{i=1}^{N}\left(-\frac{1}{2}\nabla_i^2\right) + \sum_{A=1}^{M}\left(-\frac{1}{2}\nabla_A^2\right) - \sum_{i=1}^{N}\sum_{A=1}^{M}\left(\frac{Z_A}{r_{iA}}\right) + \sum_{i=1}^{N}\sum_{j>i}^{N}\left(\frac{1}{r_{ij}}\right) + \sum_{A=1}^{N}\sum_{B>A}^{N}\left(\frac{Z_A Z_B}{R_{AB}}\right) \tag{3.3}$$

右辺の第１項，第２項は運動エネルギー，第３項，第４項，第５項はCoulomb力となっている．Coulomb力は多数の原子核・電子との相互作用となっているため，多体問題となり解析的に解くことができない．そのため様々な近似手法を用いることでシュレディンガー方程式の近似解を求める．

3.3 ハートリー・フォック法

ここでは，シュレディンガー方程式に様々な近似法を用いて，ハートリー方程式を導出する方法を説明する．

3.3.1 Born-Oppenheimer 近似

原子核の質量は電子に比べると非常に大きく，最も軽い原子核でも電子の1836倍の質量をもつ．そのため，原子核の運動は電子よりもずっと遅い．そこで，原子核は固定しているとみなし，電子の運動のみを考える．これがBO(Born-Oppenheimer)近似である．BO近似を用いたハミルトニアンを，式3.4に示す．

$$\hat{H} = \sum_{i=1}^{N}\left(-\frac{1}{2}\nabla_i^2\right) - \sum_{i=1}^{N}\sum_{A=1}^{M}\left(\frac{Z_A}{r_{iA}}\right) + \sum_{i=1}^{N}\sum_{j>i}^{N}\left(\frac{1}{r_{ij}}\right) \tag{3.4}$$

BO近似によって導出された，電子状態に対応するハミルトニアンを，電子ハミルトニアンという．

3.3.2 Slater 行列式

多電子における波動関数はそのまま用いて解くのは難しいため，１電子波動関数の積で近似する．この近似をハートリー近似という

$$\Psi(\boldsymbol{r_1}, \boldsymbol{r_2}, \cdots, \boldsymbol{r_n}) = \psi_1(\boldsymbol{r_1})\psi_2(\boldsymbol{r_2})\cdots\psi_n(\boldsymbol{r_n}) \tag{3.5}$$

しかし．この近似には，電子の位置を交換すると符号は反転するという反対称性を満たさない．

この性質を満たすためにSlater行列式を導入する．Slater行列式を式3.6に示す．

$$\Psi(\boldsymbol{r_1}, \boldsymbol{r_2}, \cdots, \boldsymbol{r_n}) = \frac{1}{\sqrt{n!}} \begin{vmatrix} \psi_1(\boldsymbol{r_1}) & \psi_1(\boldsymbol{r_2}) & \cdots & \psi_1(\boldsymbol{r_n}) \\ \psi_2(\boldsymbol{r_1}) & \psi_2(\boldsymbol{r_2}) & \cdots & \psi_2(\boldsymbol{r_n}) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_n(\boldsymbol{r_1}) & \psi_2(\boldsymbol{r_2}) & \cdots & \psi_n(\boldsymbol{r_n}) \end{vmatrix} \tag{3.6}$$

行列式の性質より Slater 行列式の行や列を交換すると，行列式全体の符号は反転するため，反対称性を満たす．

3.3.3 変分法

シュレディンガー方程式の代表的な近似計算方法の 1 つに変分法がある．本節では，変分法の手順を説明する．

まず，試行関数 $\psi(\boldsymbol{r})$ を用いて，ハミルトニアンの期待値を求める．期待値を求める式を式 3.7 に示す．

$$E[\psi(\boldsymbol{r})] = \frac{\langle\psi|\hat{H}|\psi\rangle}{\langle\psi|\psi\rangle} \tag{3.7}$$

ここで得られた期待値 $E[\psi(\boldsymbol{r})]$ をエネルギー期待値という．この期待値 E がなるべく最小になるような波動関数 $\tilde{\psi}(\boldsymbol{r})$ を求める．具体的には E を偏微分した値が 0 になるような $\tilde{\psi}(\boldsymbol{r})$(停留値) を求める．

このようにして求めた波動関数 $\tilde{\psi}(\boldsymbol{r})$ を用いたエネルギー期待値 $E[\tilde{\psi}(\boldsymbol{r})]$ は，真の基底状態のエネルギー E_0 より低くならない．このような性質を変分原理という．この原理によって，変分法で求めたエネルギー期待値が厳密解に近いことを示している．

変分法は後述するハートリー・フォック法を含む，様々な手法に用いられている．

3.3.4 ハートリー・フォック方程式

ハートリー・フォック法ではハートリー・フォック方程式を用いてシュレディンガー方程式を解く手法である．この節では，前節で説明した変分法を用いてハートリー・フォック方程式を導出する．

式 3.6 の通り，ハートリー・フォック法では多電子波動関数を Slater 行列式で近似する．次に，BO 近似したハミルトニアン (式 3.4) の 1 電子の項をまとめる．

$$
\begin{aligned}
\hat{H} &= \sum_{i=1}^{N}\left(-\frac{1}{2}\nabla_i^2\right) - \sum_{i=1}^{N}\sum_{A=1}^{M}\left(\frac{Z_A}{r_{iA}}\right) + \sum_{i=1}^{N}\sum_{j>i}^{N}\left(\frac{1}{r_{iA}}\right) \\
&= \sum_{i=1}^{N}\left(-\frac{1}{2}\nabla_i^2 - \sum_{A=1}^{M}\frac{Z_A}{r_{iA}}\right) + \sum_{i=1}^{N}\sum_{j>i}^{N}\left(\frac{1}{r_{ij}}\right) \\
&= \sum_{i=1}^{N}(h_1) + \sum_{i=1}^{N}\sum_{j>i}^{N}\left(\frac{1}{r_{ij}}\right)
\end{aligned} \tag{3.8}
$$

Slater 行列式 (式 3.6) を展開したものを用いて，エネルギー期待値 (式 3.7) を計算したものを式 3.9 に示す.

$$
\begin{aligned}
E &= \sum_{i}^{n} h_{ij} + \frac{1}{2}\sum_{i,j}^{n}(J_{ij} - K_{ij}) \\
h_{ij} &= \int \phi_i(x_1) h_i \phi_i(x_1) dx_1 \\
J_{ij} &= \iint \phi_i^*(x_1)\phi_j^*(x_2)\frac{1}{r_{ij}}\phi_i(x_1)\phi_j(x_2)dx_1dx_2 \\
K_{ij} &= \iint \phi_i^*(x_1)\phi_j^*(x_2)\frac{1}{r_{ij}}\phi_j(x_1)\phi_i(x_2)dx_1dx_2
\end{aligned} \tag{3.9}
$$

J_{ij} をクーロン積分，K_{ij} を交換積分という.

ここでの波動関数 Ψ は規格直交系である必要がある．規格直行系の条件を式 3.10 に示す.

$$
\int_{\infty}^{\infty} \Psi_i^* \Psi_j^* = \delta_{ij} \begin{cases} 1 & (i = j) \\ 0 & (i \neq j) \end{cases} \tag{3.10}
$$

δ_{ij} はクロネッカーのデルタで，同じ波動関数の内積は 1 となり，異なる波動関数の内積は 0 となる.

変分法より，条件（式 3.10）のもとで，エネルギー期待値 (式 3.9) を最小化 $\tilde{\Psi}$ する．制約条件が与えられた最小化問題はラグランジュの未定乗数法を用いて解くことができる．ラグランジュの未定乗数法を用いて式変形すると，ハートリー・フォック方程式を得る．ハートリー・フォック方程式を式 3.11 に示す.

$$
\begin{aligned}
\hat{F}\psi_k(x_1) &= \epsilon_k\psi(x_1) \quad (k = 1, 2, \cdots, n) \\
\hat{F}\psi_k(x_1) &= \left(h_1 + \sum_{j\neq k}^{n}(\hat{J}_j - \hat{K}_j)\right)\psi_k(x_1) \\
\hat{J}\psi_k(x_1) &= \left(\int \frac{\psi_j^*(x_2)\phi_j(x_2)}{r_12}dx_2\right)\psi_k(x_1) \\
\hat{K}\psi_k(x_1) &= \left(\int \frac{\psi_j^*(x_2)\psi_k(x_2)}{r_12}dx_2\right)\psi_j(x_1)
\end{aligned}
\tag{3.11}
$$

ここで，$\hat{F}\psi_k(x_1)$ は Fock 演算子，$\hat{J}\psi_k(x_1)$ は Coulomb 演算子，$\hat{K}\psi_k(x_1)$ は交換演算子である．

3.4 Roothaan-Hall 方程式

節 3.3 で紹介したハートリー・フォック方程式は，偏微分・積分方程式なので代数的に解くことはできない．本節では，ハートリー・フォック方程式を代数的に解くことができる形に変形した Roothaan-Hall 方程式について説明する．

3.4.1 基底関数

分子中の電子の波動関数を取り扱う方法の一つに，分子軌道法がある．分子軌道法では，分子中での電子の波動関数は，分子全体に広がっている軌道であると考える．このような軌道は分子軌道と呼ばれ，LCAO-MO 法（Liner Combination of Atomic Orbital - Molecular Orbital）により原子軌道の線形結合であらわされる．

量子科学計算では，LCAO-MO 法の考え方を拡張させ，原子軌道に限らず様々な既知の関数を用いて近似する方法が用いられている．式 3.12 に近似した式を示す．

$$
\psi(\boldsymbol{r}) = \sum_{\mu}^{N} c_\mu\chi_\mu(\boldsymbol{r}) = c_1\chi_1(\boldsymbol{r}) + c_2\chi_2(\boldsymbol{r}) + \cdots + c_N\chi_N \tag{3.12}
$$

ここで，既知の関数 χ を基底関数という．基底関数には一般的に CGTO(Contracted Gauss Type Orbital) が用いられる．CGTO は PGTO(Primitive Gauss Type Orbital) と呼ばれる原子ガウス型関数の線形結合である．

$$
\psi(\boldsymbol{r}) = \sum_{\mu}^{N} c_\mu\chi_\mu(\boldsymbol{r}) = c_1\chi_1(\boldsymbol{r}) + c_2\chi_2(\boldsymbol{r}) + \cdots + c_N\chi_N \tag{3.13}
$$

基底関数には，PGTO の組み合わせ方によってさまざまな種類がある．対象の分子によって基底関数を使い分けることができる．

3.4.2 Roothaan-Hall 方程式

ハートリー・フォック方程式は微積分方程式のため，代数的に解くことはできない．そこで，節 3.4.1 のように波動関数 ϕ を基底関数の線形結合で近似する．

$$\begin{aligned} \hat{F}\psi_k(x_1) &= \epsilon_k\psi(x_1) \quad (k = 1, 2, \cdots, n) \\ \psi_k(x_1) &= \sum_{\mu}^{N} c_{\mu k}\chi_\mu(x_i) \end{aligned} \tag{3.14}$$

これを式変形することで代数方程式に変換できる．この代数方程式を Roothaan-Hall 方程式という．Roothaan-Hall 方程式を行列で表記した式を，式 3.15 に示す．

$$\begin{aligned} \boldsymbol{FC} &= \boldsymbol{SC\epsilon} \\ F_{ij} &= \int \chi_i^*(x)\hat{F}\chi_j^*(x)dx \\ S_{ij} &= \int \chi_i^*(x)\chi_j(x)dx \end{aligned} \tag{3.15}$$

ここで，$\boldsymbol{F}$ をフォック行列，$\boldsymbol{S}$ を重なり行列という．

Roothaan-Hall 方程式は一般固有方程式であるため，固有値問題に変形して解く必要がある．次節では，自己無頓着場における Roothaan-Hall 方程式の解き方である．SCF（Self-consistent Field）計算について説明する．

3.5 SCF（Self-consistent Field）

本節では，自己無頓着場（Self-consistent Field）における Roothan-Hall 方程式の解法について説明する．

3.5.1 対象分子の設定

対象となる分子において，計算に必要な情報を用意する．具体的には，原子核 A の電荷 Z_A，座標 $\boldsymbol{R}_A$, 基底関数 χ_μ, 電子数 N を用意する．また，原子核の情報を用いて原子核間の反発エネルギー V_{nuc} を計算する．

$$V_{nuc} = \sum_{A=1}^{M}\sum_{B>A}^{M}\frac{Z_A Z_B}{R_{AB}} \tag{3.16}$$

ここで，M は対象分子の原子核の総数である．

3.5.2 分子積分計算

SCF 計算に必要な分子積分を事前に計算する．分子積分の式を以下に示す.

$$S_{ij} = \int \chi_i^*(x)\chi_j(x)dx \tag{3.17}$$

$$T_{ij} = \int \chi_i^*(x)\left(-\frac{1}{2}\nabla\right)\chi_j(x)dx \tag{3.18}$$

$$V_{ij} = \int \chi_i^*(x)\left(-\frac{Z}{r}\right)\chi_j(x)dx \tag{3.19}$$

$$Q_{ijkl} = \iint \chi_i^*(x_1)\chi_j^*(x_2)\frac{1}{r_{12}}\chi_k^*(x_1)\chi_l^*(x_2)dx_1dx_2 \tag{3.20}$$

ここで，$\boldsymbol{S}$ は重なり積分，$\boldsymbol{T}$ は運動エネルギー積分，$\boldsymbol{V}$ は核引力積分，$\boldsymbol{Q}$ は電子反発積分という.

また，運動エネルギーと各引力積分の和であるコアハミルトニアン $\boldsymbol{H}$ も計算する．[4]

$$\boldsymbol{H} = \boldsymbol{T} + \boldsymbol{V} \tag{3.21}$$

式 3.22 が示すように，CGTO を基底関数とする分子積分は，PGTO の分子積分の和で求めることができる．分子積分を解析的に解くことはできないため，様々な数値的に解くためのアルゴリズムが提案されている．重なり積分や 1 電子積分 (運動エネルギー積分，核引力積分) は MDMcMurchie-Davidson）法や OS（Obara-Saika）法など，電子反発積分には Head-Gordon-Pople algorithm などのアルゴリズムが存在する．[5][6][7]

$$
\begin{aligned}
S_{ij} &= \sum_{a=1}^{i}\sum_{b=1}^{j} D_a D_b [a|b] \\
[a|b] &= \int \chi_a^*(\boldsymbol{r})\chi_b(\boldsymbol{r})d\boldsymbol{r} \\
T_{ij} &= \sum_{a=1}^{i}\sum_{b=1}^{j} D_a D_b k_{ab} \\
k_{ab} &= \int \chi_a^*(\boldsymbol{r})\left(-\frac{1}{2}\nabla\right)\chi_b(\boldsymbol{r})d\boldsymbol{r} \\
V_{ij} &= \sum_{c=1}^{N_c}\sum_{a=1}^{i}\sum_{b=1}^{j} D_a D_b v_{ab}(c) \\
v_{ab}(c) &= \int \chi_a^*(\boldsymbol{r})\left(-\frac{Z_c}{r_c}\right)\chi_b(\boldsymbol{r})d\boldsymbol{r} \\
Q_{ijkl} &= \sum_{a=1}^{i}\sum_{b=1}^{j}\sum_{c=1}^{k}\sum_{d=1}^{l} D_a D_b D_c D_d [ab|cd] \\
[ab|cd] &= \iint \chi_a^*(\boldsymbol{r_1})\chi_b^*(\boldsymbol{r_2})\frac{1}{\boldsymbol{r}_{12}}\chi_c^*(\boldsymbol{r_1})\chi_d^*(\boldsymbol{r_2})d\boldsymbol{r_1}d\boldsymbol{r_2}
\end{aligned}
\tag{3.22}
$$

3.5.3 変換行列 X

Roothaan-Hall 方程式は一般固有値問題のため直接解くことができない．そのため一般固有値問題を通常の固有値問題に変換する必要がある．通常の固有値問題に変換するためには，重なり行列 S の変換行列 X を用いて直行化する必要がある．

1. 重なり行列 S の対角化重なり行列 S の対角化を行う．

$$
\boldsymbol{U}^\dagger \boldsymbol{S}\boldsymbol{U} = \boldsymbol{s} \tag{3.23}
$$

ここで．$\boldsymbol{s}$ は $\boldsymbol{S}$ の固有値を要素に持つ対角行列．$\boldsymbol{U}$ は $\boldsymbol{S}$ の固有ベクトルである．

2. 変換行列 X の計算 1 で求めた $\boldsymbol{s}$, $\boldsymbol{U}$ を用いて変換行列 $\boldsymbol{X}$ を計算する．

$$
\boldsymbol{X} = U s^{-\frac{1}{2}} \tag{3.24}
$$

ここで求めた変換行列を用いて $\boldsymbol{C}$ と $\boldsymbol{F}$ を直行化すると，通常の固有値問題に変換できる．

$$
\begin{aligned}
\boldsymbol{FC} &= \boldsymbol{SC\epsilon} \\
\boldsymbol{X}^{\dagger}\boldsymbol{F}(\boldsymbol{X}\boldsymbol{X}^{-1})\boldsymbol{C} &= \boldsymbol{X}^{\dagger}\boldsymbol{S}(\boldsymbol{X}\boldsymbol{X}^{-1})\boldsymbol{C\epsilon} \quad (\boldsymbol{X}\boldsymbol{X}^{-1} = \boldsymbol{I}) \\
\boldsymbol{F'C'} &= \boldsymbol{C'\epsilon} \\
\boldsymbol{X}^{\dagger}\boldsymbol{SX} &= \boldsymbol{I} \\
\boldsymbol{C'} &= \boldsymbol{X}^{-1}\boldsymbol{C} \\
\boldsymbol{F'} &= \boldsymbol{X}^{\dagger}\boldsymbol{FX}
\end{aligned}
\tag{3.25}
$$

3.5.4 密度行列P

コアハミルトニアン（式 3.21）に変換行列（式 3.24）を用いて直行化する．

$$
\boldsymbol{H'} = \boldsymbol{X}^{\dagger}\boldsymbol{HX} \tag{3.26}
$$

$\boldsymbol{H'}$ を式 3.23 と同様に対角化を行い，固有ベクトル $\boldsymbol{C'}$ を計算する．
求めた $\boldsymbol{C'}$ の逆変換により係数行列 $\boldsymbol{C}$ を求める．

$$
\boldsymbol{C} = \boldsymbol{XC'} \tag{3.27}
$$

最後に係数行列 $\boldsymbol{C}$ を用いて密度行列 $\boldsymbol{P}$ を求める．

$$
P_{ij} = 2\sum_{k} C_{ik} * C_{jk} \tag{3.28}
$$

3.5.5 フォック行列の計算

式 3.20 で求めた電子反発積分と，式 3.21 のコアハミルトニアン，式 3.28 の密度行列を用いてフォック行列を計算する．

$$
F_{ij} = H_{ij} + \sum_{k,l}\left[Q_{ijkl} - \frac{1}{2}Q_{ikjl}\right]P_{kl} \tag{3.29}
$$

3.5.6 エネルギー計算

式 3.29 で求めたフォック行列を用いて RHF エネルギーを計算する．

$$
E_{RHF} = \frac{1}{2}\sum_{i,j} P_{ij}[H_{ij} + F_i j] \tag{3.30}
$$

求めたRHFエネルギーに各反発エネルギー（式3.16）を加えると，分子の全エネルギーが求まる．

$$E_{tot} = E_{RHF} + V_{nuc} \tag{3.31}$$

3.5.7 密度行列Pの更新

求めたフォック行列を用いて，節3.5.4と同様の方法で密度行列を計算する．具体的には，フォック行列$\boldsymbol{F}$を直行化した$\boldsymbol{F'}$に対して対角化を行い固有ベクトル$\boldsymbol{C'}$を求める．その後は節3.5.4と同様に計算して$\boldsymbol{P}$を求める．

$$\boldsymbol{F'} = \boldsymbol{X}^{\dagger}\boldsymbol{F}\boldsymbol{X} \tag{3.32}$$

3.5.8 収束判定

更新した密度行列の値を用いて．フォック行列を更新する．更新したフォック行列を用いてエネルギーの値と密度行列を更新する．このように節3.5.5, 3.5.6, 3.5.7を繰り返し計算する．

この繰り返し計算をエネルギーや密度行列の変化が閾値以下になるまで繰り返す．本研究ではエネルギーが閾値以下にになることを収束条件とする．

$$\Delta E \equiv |E_0^{RHF(n)} - E_0^{RHF(n-1)}| \leq \Delta E_{conv} \tag{3.33}$$

ここで，E_{conv}はエネルギーの閾値である．

収束した分子軌道が作るポテンシャルはそれ以上変化しない．このような状態を自己無頓着場（Self-consistent Field）という．収束条件を満たした場合，繰り返し計算を終了して，その時点でのエネルギーをSCFでの解とする．

第4章　既存手法

前章で説明した通り，SCF 計算では収束条件を満たすまで繰り返し計算を行う．しかし，分子のサイズがおおきくなったり，基底関数を変更したりすると，収束が安定せず，繰り返し回数が大幅に増加する場合がある．そのため，フォック行列の更新方法を改良することで収束を安定させ，繰り返し回数を削減させる．本章では，フォック行列を更新する際に使用する既存のアルゴリズムについて説明する．

4.1　Damping

Damping は現在のフォック行列を，前回のループで計算したフォック行列と按分することでフォック行列の急激な変化を防ぎ収束を安定させるという手法である．[8] フォック行列の更新式を式 4.1 に示す．

$$\boldsymbol{F} = (1-\alpha)\boldsymbol{F_{old}} + \alpha\boldsymbol{F_{new}} \tag{4.1}$$

ここで，F_{old} は前回のループで計算したフォック行列，F_{new} は現在のループでのフォック行列である．また，α は按分するためのパラメータである．

α は定数としてあらかじめ設定する方法と，ループごとに最適なパラメータを計算する方法がある．パラメータの求め方を式 4.2 に示す．

$$\begin{aligned} s &= Tr[\boldsymbol{F_{old}}(\boldsymbol{F_{new}} - \boldsymbol{P_{old}})] \\ c &= Tr[(\boldsymbol{F_{new}} - \boldsymbol{F_{old}})(\boldsymbol{P_{new}} - \boldsymbol{P_{old}})] \\ \alpha &= \begin{cases} 1 & (c \leq -\frac{s}{2}) \\ -\frac{s}{2c} & ohterwise \end{cases} \end{aligned} \tag{4.2}$$

ここで，$Tr[\ \ ]$ は行列のトレースである．このようにして求めたパラメータを用いて damping を行うことで，ループごとに最適なフォック行列の更新を行うことができる．

4.2　DIIS(Direct Inversion in tne Iterative Subspace)

DIIS（Direct Inversion in the Iterative Subspace）は Pulay が提案したフォック行列の更新方法である．[9] フォック行列を 1 回前の情報だけではなく，過去数回分のフォック行

列の情報を利用してフォック行列を更新することでSCFの収束を速くするという方法である．本節では，DIISの具体的な手法を説明する．

4.2.1 誤差行列

SCF計算の各ループで誤差行列$\boldsymbol{e}$を定義する．[10] i回目のループにおける誤差行列を式4.3に示す.

$$\boldsymbol{e_i} = \boldsymbol{F_i P_i S} - \boldsymbol{S P_i F_i} \tag{4.3}$$

4.2.2 DIIS法の導出

DIIS計算の目的は過去n回のフォック行列の線形結合の係数を最適化することである.

$$\boldsymbol{F} = \sum_{i=1}^{n} c_i \boldsymbol{F_i} \tag{4.4}$$

収束したときのフォック行列を$\boldsymbol{F_{true}}$とすると，

$$\begin{aligned} \boldsymbol{F} &= \sum_{i=1}^{n} c_i \boldsymbol{F_i} \\ &= \sum_{i=1}^{n} c_i (F_{ture} - \boldsymbol{e_i}) \\ &= F_{ture} \sum_{i=1}^{n} c_i - \sum_{i=1}^{n} c_i \boldsymbol{e_i} \end{aligned} \tag{4.5}$$

式4.5が示すように，収束したときのフォック行列になるべく近くなるようにフォック行列を更新するには，$\sum_{i=1}^{n} c_i = 1$の条件の下で，$||\sum_{i=1}^{n} c_i \boldsymbol{e_i}||^2$をなるべく小さくすればいい.

したがって，次の最小化問題を解くことに帰着する.

$$\begin{aligned} &minimize \quad ||\sum_{i=1}^{n} c_i \boldsymbol{e_i}||^2 \\ &subject \quad to \quad \sum_{i=1}^{n} c_i = 1 \end{aligned} \tag{4.6}$$

条件付きの最小化問題を解くには，ラグランジュの未定乗数法を用いる．ラグランジュの未定乗数法を用いた式を式4.7に示す.

$$\begin{bmatrix} \langle e_1|e_1\rangle & \langle e_1|e_2\rangle & \cdots & \langle e_1|e_n\rangle & -1 \\ \langle e_2|e_1\rangle & \langle e_2|e_2\rangle & \cdots & \langle e_1|e_n\rangle & -1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \langle e_n|e_1\rangle & \langle e_n|e_2\rangle & \cdots & \langle e_n|e_n\rangle & -1 \\ -1 & -1 & \cdots & -1 & 0 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ -1 \end{bmatrix} \tag{4.7}$$

この線型方程式を解くことで，F_{ture} になるべく近くなるような係数 c_i が得られる．これを式 4.4 に代入することでフォック行列を更新することができる．

4.2.3 DIIS の手順

本節では，DIIS の手順を説明する．

1. 誤差行列を計算する

   式 4.3 で誤差行列を計算する．

2. リストの更新事前に過去 n 回分のフォック行列と誤差行列を保存するリストをそれぞれ用意しておく．求めた誤差行列とフォック行列でリストを更新する．

$$\begin{aligned} \boldsymbol{F_0}, \boldsymbol{F_1}, \cdots, \boldsymbol{F_n} \\ \boldsymbol{e_0}, \boldsymbol{e_1}, \cdots, \boldsymbol{e_n} \end{aligned} \tag{4.8}$$

   値が新しいものから順に 0, 1, 2, ..., n とする．

3. 行列 B の作成線型方程式を作るために行列 B を作成する．

$$\boldsymbol{B} = \begin{bmatrix} \langle e_1|e_1\rangle & \langle e_1|e_2\rangle & \cdots & \langle e_1|e_n\rangle & -1 \\ \langle e_2|e_1\rangle & \langle e_2|e_2\rangle & \cdots & \langle e_1|e_n\rangle & -1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \langle e_n|e_1\rangle & \langle e_n|e_2\rangle & \cdots & \langle e_n|e_n\rangle & -1 \\ -1 & -1 & \cdots & -1 & 0 \end{bmatrix} \tag{4.9}$$

   ここで，$\langle e_i|e_j\rangle$ は内積であり，e_i と e_j の各要素の積の総和である．

$$\langle e_i|e_j\rangle = \sum_{\mu\nu} e_i(\mu,\nu) * e_j(\mu,\nu) \tag{4.10}$$

4. 線型方程式を解く式 4.7 を解いて，係数 c を求める．

5. フォック行列を更新する式 4.4 に求めた係数 c を代入することでフォック行列を更新する．

第5章　提案手法

この章では，本論文における提案手法について説明する．

5.1　提案手法の概要

本論文では，第４章で説明したDIISに対して，GPUを用いて並列に計算する手法を提案する．DIISは各ステップで行列を扱った演算を行っているため，各要素を並列に計算することにより，逐次計算に比べて高速に計算することが可能である．

本論文では，行列積や行列方程式の解法など基本的な行列演算には，Nvidia社が提供する行列演算を高速に計算するライブラリであるcublas，cusolverを使用した．

また本論文では，誤差行列の内積を求める際には要素ごとの積を並列に計算を行った．また，計算した積の総和を求める際には，Sheard Memoryを用いることでなるべく高速に総和を求めた．

5.2　誤差行列の計算

式4.3を計算する．

行列積の演算には，Nvidia CUDA上のBLASライブラリであるcuBLASのcublasDgemm()を用いた．cublasDgemm()はDouble型の行列積（式5.1）を計算する関数である．

$$\boldsymbol{C} = \alpha \boldsymbol{A}\boldsymbol{B} + \beta \boldsymbol{C} \tag{5.1}$$

行列積の演算は各要素が独立かつ同様の計算を行うため，並列計算することによって逐次計算するより計算の大幅な高速化を実現できる．

5.3　リストのデータ構造

DIISではフォック行列，誤差行列それぞれの過去n回分のリストを保持する必要がある．本論文では，あらかじめ行列n個分のメモリを確保しておき，各ループで計算した行列を格納していく．

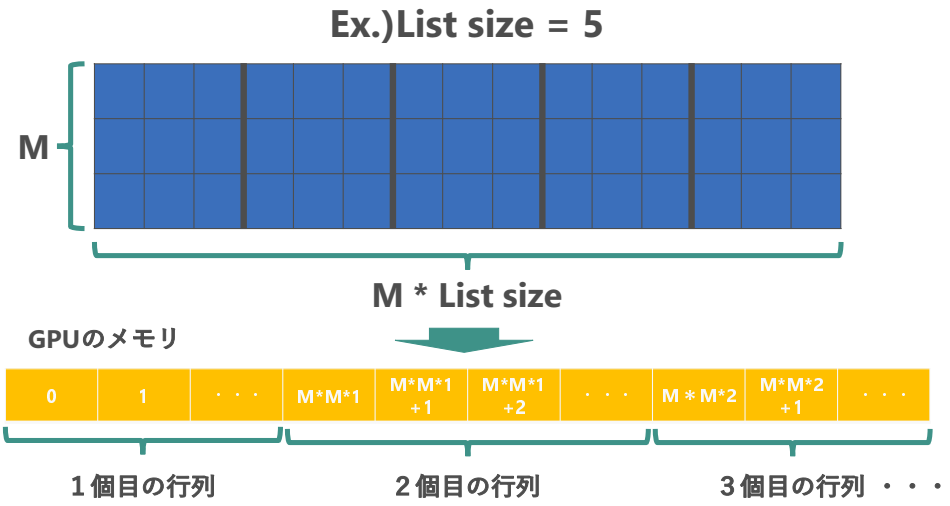

図 5.1: リストのデータ構造

図 5.1 のようにリストの先頭のインデックスを移動させることによってリストの更新を実装している．

5.4 内積計算

内積計算を行うためのアルゴリズムを 1 に示す．

Algorithm 1 Inner Product algorithm

```
for i ← 0 do
    for j ← 0 do
        A_ij ← e1_ij * e2_ij
    end for
end for
for i ← 0 do
    for j ← 0 do
        sum ← sum + A_ij
    end for
end for
```

Algorithm1 示すように，誤差行列の各要素の積を別の行列 A に代入した後，行列 A の各要素の総和を計算する．要素の積の計算はすべての要素が独立に計算しているため，並

列に計算することができる．しかし，要素の合計の計算では各要素が同一のメモリである sum にアクセスするため，並列に計算を行うと正しい結果が得られない．

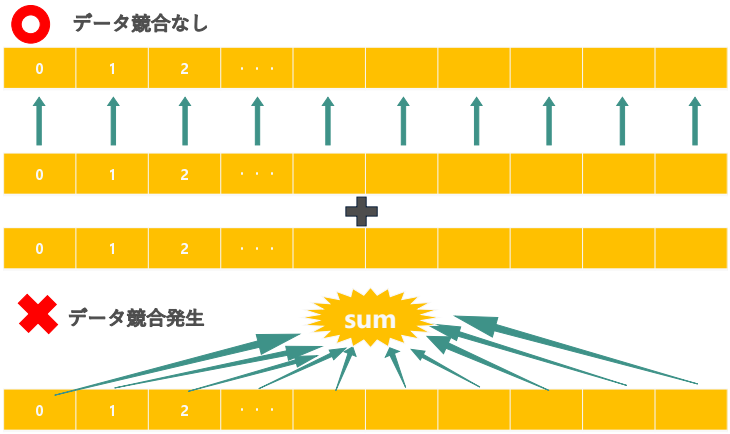

図 5.2: 合計を求める計算

GPU 上で要素の総和を求めるには，AtomicAdd を用いる．AtomicAdd は CUDA で提供されるアトミック演算の１種であり，複数のスレッドが同じメモリのアドレスに対して同時に加算を行う場合に，競合するのを防ぎ正しく演算を行うことができる関数である．具体的には，同じメモリに対して並列に計算を行うと競合が発生するため，加算を排他的に行うことによって競合の発生を防いでいる．

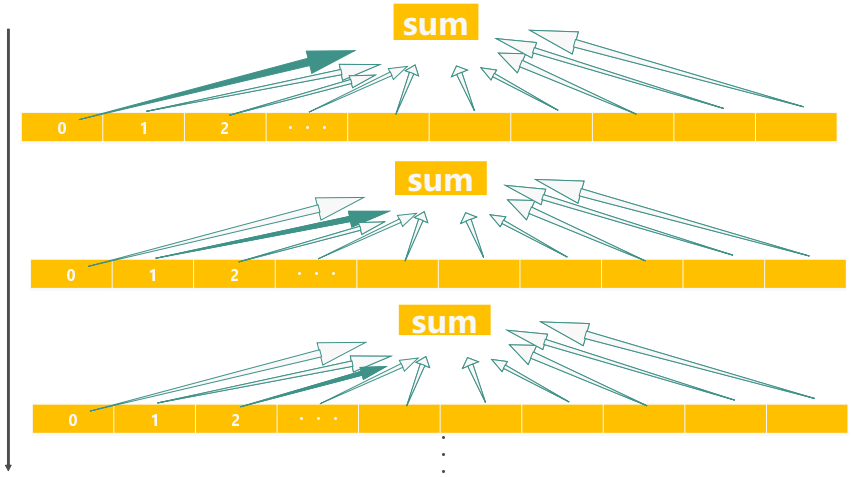

図 5.3: AtomicAdd を用いた計算

しかし，Atomic 演算は Global Memory にアクセスするため，節 2.1.3 で説明したようにデータのやり取りが遅い．そのため sheard Memory を用いた実装にすることでなるべく Global Memory にアクセスする回数を削減する．

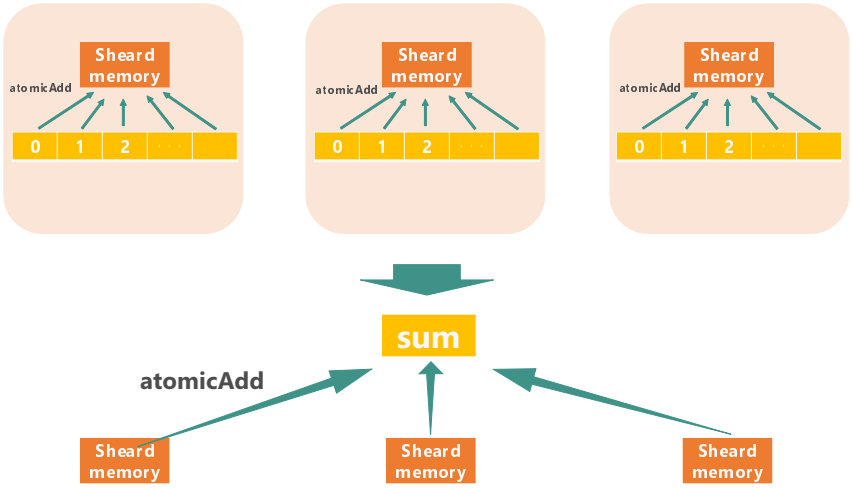

図 5.4: Sheard Memory を用いた AtomicAdd

5.5 行列 B の作成

内積計算のアルゴリズムを用いて行列 B を作成する．

$$\begin{aligned} \langle e_i|e_j\rangle &= \sum_{\mu\nu} e_i(\mu,\nu) * e_j(\mu,\nu) \\ &= \sum_{\mu\nu} e_j(\mu,\nu) * e_i(\mu,\nu) \\ &= \langle e_j|e_i\rangle \end{aligned} \tag{5.2}$$

式 5.2 より，$\langle e_i|e_j\rangle = \langle e_j|e_i\rangle$ となるので行列 B は対称行列である．対象行列のため，行列 B の下三角部分の内積は計算せずに上三角行列の結果を代入した．これによって計算回数が削減された．

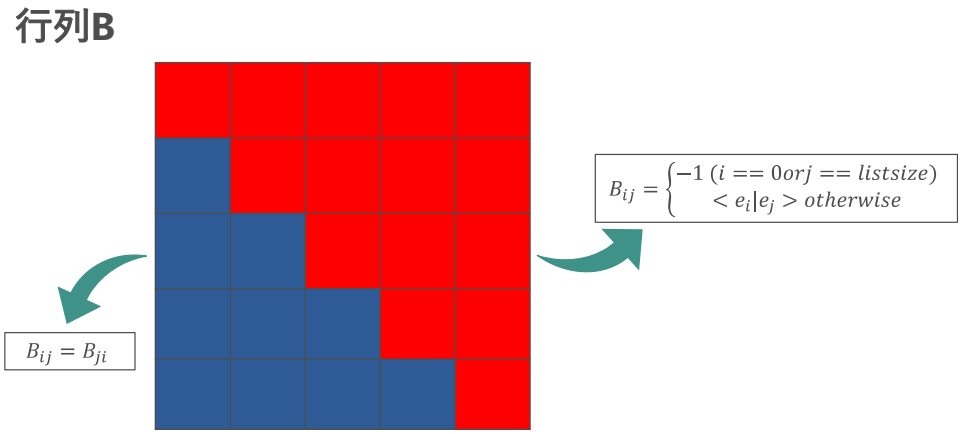

図 5.5: 行列 B の計算回数削減

5.6 線形方程式

線形方程式 4.7 を解く．線形方程式を解くのに cuSolver を用いた．cuSolver は NVIDIA の CUDA 上で動作する線形代数ライブラリである．まず，cuSolverDnDgetrf() という関数を用いて LU 分解を行う．その後，cusolverDnDgetrs() という関数で，LU 分解を用いて係数 c を計算する．

5.7 フォック行列の更新

求めた係数 c を用いて, 式 4.4 でフォック行列の更新を行う．フォック行列の更新を行うアルゴリズムを Algorithm2 に示す

Algorithm 2 Create New Fock algorithm

```
for i ← 0 to AO do
    for j ← 0 to AO do
        for k ← 0 to Csize do
            Fnew_ij ← Fnew_ij + Fk_ij * c_k
        end for
    end for
end for
```

Algorithm2 のとおり，行列の要素ごとに過去 n 回のフォック行列と求めた係数との積の総和をとる．各要素は並列に計算しているが，要素内の総和の計算は，for ループで逐次計算している．

第6章　実験

6.1　実験方法

水 (H_2O)，炭素分子 (C_2) ベンゼン (C_6H_6)，アゾベンゼン ($C_{12}H_{10}N_2$), に関して，DIIS を実装した SCF 計算を CPU・GPU を用いて行い，各ループに対する DIIS の計算時間を計測し，その平均時間を求めた．CPU での実装では，行列積，内積計算，線型方程式に対して逐次的に計算を行った．使用する基底は STO-6G, 6-31G を選択した．また，行列のサイズを n=5 の場合と，n=10 の場合でそれぞれ計測を行った．

6.2　実験環境

- CPU: Intel(R) Xeon(R) Gold 6338
  - クロック周波数：2.6[GHz]
- GPU: NVIDIA A6000
  - SM 数：84
  - Core 数：10752

6.3　実験結果

6.3.1　サイズ n=5 のとき

最初に，表 6.1 にリストサイズ n=5 の場合での，CPU と GPU を用いた DIIS の計算時間を示す．

実験結果より，今回用いたすべての条件において，提案手法は CPU 上で逐次処理する実装と比較して高速化されたことがわかる．特に軌道数が大きくなるほど高速化率は大きく，最大で 1.68572 倍の高速化率を達成している．

表 6.1: リストサイズ n=5 の場合

分子	基底関数系	軌道数 (AO)	既存手法 (CPU)[ms]	提案手法 (GPU)[ms]	高速化率
H_2O	STO-6G	7	2.28834	1.96403	1.16512
H_2O	6-31G	13	2.29494	1.97933	1.15945
C_2	6-31G	18	2.28704	2.26034	1.01181
C_6H_6	6-31G	66	8.56668	5.58607	1.53357
$C_{12}H_{10}N_2$	6-31G	146	75.82050	44.97800	1.68572

6.3.2 サイズ n=10 のとき

次に，表 6.2 にリストサイズ n=10 の場合での，CPU と GPU を用いた DIIS の計算時間を示す.

リストサイズ n=5 の時と比べて全体的に計算時間は増加している．実験で設定したすべての条件において，提案手法は CPU 上で逐次処理する実装と比較して高速で動作した．しかし，高速化率はサイズ n=10 の場合と比べて有意な差は見られなかった.

表 6.2: リストサイズ n=10 の場合

分子	基底関数系	軌道数 (AO)	既存手法 (CPU)[ms]	提案手法 (GPU)[ms]	高速化率
H_2O	STO-6G	7	2.28949	2.00351	1.14273
H_2O	6-31G	13	2.28252	1.99228	1.14568
C_2	6-31G	18	2.28604	2.26159	1.01081
C_6H_6	6-31G	66	8.55422	5.46237	1.56602
$C_{12}H_{10}N_2$	6-31G	146	77.64701	45.44750	1.70849

6.4 考察

今回の実験で設定したすべての条件において，提案手法はCPU上で逐次処理する実装と比較して高速で動作した．GPU側における高速化の要因として，DIISの処理には，行列演算を多く行っている．行列演算は行列の各要素ごとに同様の処理を行っているため，各要素を並列に処理することで，各要素を逐次的に処理した場合よりも高速に計算することができたからだと考える．また，分子のサイズが大きく，基底関数系をSTO-6Gから6-31Gに変更した場合で，より高い高速化率を達成することができた．要因としては，分子のサイズや基底関数系を変更することで波動関数の近似に用いるAO（原子軌道）の数が増加しているからだと推測される．DIISの誤差行列やフォック行列の1辺の大きさは，AOの数であるため，AOの数が増加するほど行列のサイズが大きくなり，GPUでの並列処理の利点が大きくなっている．

表 6.3: 軌道数の変化

分子	基底関数系	軌道数 (AO)	誤差行列の要素数	フォック行列の要素数	高速化率 (n=10)
H_2O	STO-6G	7	49	49	1.14273
H_2O	6-31G	13	169	169	1.14568
C_2	6-31G	18	324	324	1.01081
C_6H_6	6-31G	66	4,356	4,356	1.56602
$C_{12}H_{10}N_2$	6-31G	146	21,316	21,316	1.70849

しかし，リストサイズを大きくしても高速化率はほぼ同程度であった．リストサイズを大きくすると行列Bのサイズや内積計算の回数が増加するが，サイズは5しか増加しないため，その増加率はAOの数の増加率に比べて小さい．そのため，GPUによる利点が少なかったからだと考えられる．

表 6.4: リストサイズの変化

リストサイズ	行列 B の要素数	内積計算の回数	高速化率 (H_2O)
n = 5	36	15	1.14568
n = 10	121	55	1.14273

第7章　おわりに

7.1　まとめ

本研究では，SCF 計算の収束アルゴリズムの 1 つである DIIS の計算に関して，GPU を用いることで高速化を図った．結果として CPU による実装と比べて最大 1.70849 倍の高速化を達成し，GPU を用いた並列処理の有効性を示すことができた．

7.2　今後の課題

今回は SCF 計算の収束アルゴリズムに DIIS を用いたが，今日では，様々な DIIS の改良版が提案されている．例えば，フォック行列を近似するための係数を求めるために，エネルギーを用いる E-DIIS 法などがある．様々な DIIS の改良版に対しても GPU を用いた並列化を行っていきたい．

謝辞

本研究においては，ご多忙の中，熱心に指導をしてくださった中野浩嗣教授，伊藤靖朗准教授, 高藤大介助教に深く感謝致します．研究を進めるにあたり，とても貴重なご意見をいただきました．

また，研究しやすい環境を作ってくださった研究室の皆様にも，心より感謝しています．

参考文献

[1] NVIDIA. CUDA C++ Programming Giude Release 12.4, 2024.

[2] Yasuaki Ito, Satoki Tsuji, Haruto Fujii, Kanta Suzuki, Nobuya Yokogawa, Koji Nakano, and Akihiko Kasagi. Introduction to computational quantum chemistry for computer scientists. pp. 273–282, 2024.

[3] 中井浩巳. 手で解く量子化学 I. 丸善出版, 2022.

[4] Nobuya Yokogawa, Yasuaki Ito, Satoki Tsuji, Haruto Fujii, Kanta Suzuki, Koji Nakano, and Akihiko Kasagi. Parallel gpu computation of nuclear attraction integrals in quantum chemistry. pp. 163–169, 2024.

[5] Kanta Suzuki, Yasuaki Ito, Haruto Fujii, Nobuya Yokogawa, Satoki Tsuji, Koji Nakano, and Akihiko Kasagi. Gpu acceleration of head-gordon-pople algorithm. pp. 115–124, 2024.

[6] Haruto Fujii, Yasuaki Ito, Nobuya Yokogawa, Kanta Suzuki, Satoki Tsuji, Koji Nakano, and Akihiko Kasagi. A GPU implementation of McMurchie-Davidson algorithm for two-electron repulsion integral computation. In *The 15th International Conference on Parallel Processing and Applied Mathematics (to appear).*

[7] Satoki Tsuji, Yasuaki Ito, Haruto Fujii, Nobuya Yokogawa, Kanta Suzuki, Koji Nakano, and Akihiko Kasagi. Dynamic screening of two-electron repulsion integrals in gpu parallelization. pp. 211–217, 2024.

[8] Michael C. Zerner and Michael Hehenberger. A dynamical damping scheme for converging molecular scf calculations. *Chemical Physics Letters*, Vol. 62, No. 3, pp. 550–554, 1979.

[9] Péter Pulay. Convergence acceleration of iterative sequences. the case of scf iteration. *Chemical Physics Letters*, Vol. 73, No. 2, pp. 393–398, 1980.

[10] Péter. Pulay. Improved SCF Convergence Acceleration. *Computational Chemistry*, No. 14, pp. 556–560, Dec. 1982.