Catalogação na publicação
Seção de Catalogação e Classificação

F475p Figueirêdo, Ítalo Nicácio Dos Santos Gomes de.
Paralelização do algoritmo MOZYME no MOPAC / Ítalo Nicácio Dos Santos Gomes de Figueirêdo. - João Pessoa, 2024.
19 f. : il.

Orientação: Gerd Rocha.
Coorientação: Julio Maia.
TCC (Graduação) - UFPB/Informática.

1. Paralelização. 2. MOZYME. 3. MOPAC. 4. OpenMP. I. Rocha, Gerd. II. Maia, Julio. III. Título.

UFPB/CI CDU 004.421

Elaborado por Michelle de Kássia Fonseca Barbosa - CRB-738

Paralelização do algoritmo MOZYME no MOPAC

Ítalo Nicácio dos Santos Gomes de Figueirêdo[1], **Julio Daniel de Carvalho Maia**[1,2] e **Gerd Bruno da Rocha**[3]

[1]Centro de Informática – Universidade Federal da Paraíba (UFPB), João Pessoa – PB – Brasil
[2]Advanced Micro Devices, Inc.
[3]Departamento de Química – Universidade Federal da Paraíba (UFPB), João Pessoa – PB – Brasil

italonicacio@gmail.com, jmaia@amd.com, gbr@academico.ufpb.br

14 de novembro de 2024

Resumo

Este artigo explora a paralelização do algoritmo MOZYME, uma técnica de escalonamento linear implementada no software de química quântica MOPAC. A paralelização visa permitir cálculos em sistemas moleculares maiores, como biomoléculas. O foco deste trabalho está na paralelização de duas sub-rotinas cruciais do MOZYME: density_for_MOZYME e diagg1. A ferramenta para paralelização escolhida foi o OpenMP, que permite a distribuição da carga de trabalho entre múltiplos núcleos de processamento. O Intel® VTune™ Profiler foi utilizado para analisar o desempenho, identificando as sub-rotinas density_for_MOZYME e diagg1 como as que mais demandam recursos computacionais. Para avaliar a eficácia da paralelização, foi realizado um benchmark utilizando um processador AMD Ryzen Threadripper PRO 5995WX com 64 núcleos físicos e 128 threads. Os testes abrangeram diversos sistemas moleculares, variando em tamanho e complexidade, com diferentes raios de corte para interações atômicas. Os resultados demonstram um speedup de até 3x no tempo total do MOPAC em um cálculo MOZYME. As sub-rotinas density e diagg1 alcançaram speedups de até 30x e 6,7x, respectivamente.

Abstract

This article explores the parallelization of the MOZYME algorithm, a linear scaling technique implemented in the MOPAC quantum chemistry software. Parallelization aims to enable calculations on larger molecular systems, such as biomolecules. The focus of this work is on the parallelization of two crucial MOZYME subroutines: density_for_MOZYME and diagg1. The parallelization tool chosen was OpenMP, which allows the workload to be distributed among multiple processing cores. The Intel® VTune™ Profiler was used to analyze performance, identifying the density_for_MOZYME and diagg1 subroutines as those that demand the most computing resources. To evaluate the effectiveness of parallelization, a benchmark was carried out using an AMD Ryzen Threadripper PRO 5995WX processor with 64 physical cores and 128 threads. The tests covered several molecular systems, varying in size and complexity, with different cut-off radii for atomic interactions. The results show a speedup of up to 3x in total MOPAC time in a MOZYME calculation. The density and diagg1 subroutines achieved speedups of up to 30x and 6.7x, respectively.

1 Introdução

O progresso na aplicação dos métodos de química quântica em sistemas moleculares com muitos átomos (ex: biomoléculas) tem uma forte dependência com o poder computacional disponível e, de fato, essa é uma das preocupações que está presente com mais força nas mentes de muitos químicos

quânticos e computacionais [1]. A busca por soluções para essa barreira impulsiona o desenvolvimento de computadores mais rápidos, processadores com múltiplos núcleos e algoritmos mais eficientes.

Uma das abordagens promissoras para contornar essa limitação são os métodos semiempíricos de química quântica, que se destacam por reduzir o tempo computacional, tornando-se ferramentas valiosas para o estudo de moléculas de interesse biológico. A criação de códigos eficientes que aproveitam ao máximo os recursos computacionais de `clusters` com múltiplas `CPUs` e `GPUs` também desponta como uma estratégia crucial para a análise quântica de sistemas de grande escala.

O programa **MOPAC** [2] (Molecular Orbital PACkage), mantido pelo grupo **openMOPAC**, exemplifica essa busca por maior eficiência computacional. O MOPAC, que teve seu código aberto em setembro de 2021, permite contribuições da comunidade científica, impulsionando o desenvolvimento de algoritmos de escalonamento linear, como o **MOZYME** [3], que possibilitam a análise de sistemas com milhares de átomos. O algoritmo MOZYME, por exemplo, utiliza um formato de matriz esparsa chamado matriz simétrica de blocos variáveis (*Symmetric Variable Block Matrix*, **SVBR**) para otimizar o uso de memória e tempo de processamento.

O grupo de pesquisa do Laboratório de Química Quântica Computacional (LQQC) tem se dedicado a acelerar cálculos de química quântica, especialmente para sistemas complexos como biomoléculas. Um exemplo é a pesquisa de Maia, J.D.C e colaboradores [4] que demonstra o potencial da computação paralela para tornar os cálculos de química quântica mais eficientes. Nesse trabalho, os autores apresentaram o emprego de bibliotecas otimizadas para álgebra linear como LAPACK, BLAS, MAGMA e CUBLAS podem acelerar as subrotinas do MOPAC, como pseudodiagonalização e montagem da matriz de densidade. Além disso, o trabalho de Maia, J.D.C e colaboradores [5] propôs uma nova abordagem ao algoritmo MOZYME, nessa nova abordagem é implementando o algoritmo SP2, que substitui a pseudodiagonalização da matriz de Fock e a montagem da matriz de densidade, que são um dos principais gargalos em cálculos SCF, por um procedimento conhecido como purificação da matriz densidade.

A pesquisa de Fukushima e colaboradores [6] evidenciou a capacidade do uso de cálculos quânticos semiempíricos, acelerados pelo algoritmo MOZYME, em fornecer compreensões cruciais sobre a reatividade enzimática, expandindo o conhecimento sobre a relação entre estrutura eletrônica e função biológica em macromoléculas. O estudo também demonstrou a influência da hidratação na estrutura eletrônica das enzimas e na capacidade preditiva do método, comprovando a importância de considerar o ambiente aquoso para uma análise precisa da função enzimática.

Neste trabalho, será apresentado a paralelização de duas sub-rotinas do algoritmo **MOZYME**, a `density_for_MOZYME` e a `diagg1`. A paralelização dessas sub-rotinas resultou em ganhos de desempenho significativos. A análise dos resultados mostrou que o **MOPAC** consegue obter um *speedup* total de **3x** no tempo de execução, evidenciando a eficácia das otimizações realizadas. Individualmente, a `density_for_MOZYME` alcançou um *speedup* de até **30x**, enquanto a sub-rotina `diagg1` obteve um *speedup* de **6,7x**. Esses resultados demonstram que a continuidade na otimização e paralelização do MOZYME promete torná-lo ainda mais eficiente.

2 Fundamentos Teóricos e Trabalhos Relacionados

2.1 Método de Hartree–Fock

O método de **Hartree-Fock** [7] é um método **ab initio** aproximativo que calcula a função de onda e a energia de sistemas de muitos corpos em estado estacionário. Este método é fundamental em química quântica e física computacional, pois permite a obtenção de soluções aproximadas para a equação de Schrödinger em sistemas com muitos elétrons. No método **Hartree-Fock**, a função de onda do sistema é representada por um determinante de Slater, que garante a antissimétrica necessária para obedecer ao princípio de exclusão de Pauli.

O procedimento iterativo do método Hartree-Fock envolve a construção de uma matriz de Fock, que é um operador Hamiltoniano efetivo de uma partícula, e a resolução de um problema de autovalores generalizado. A matriz de Fock é diagonalizada para obter os coeficientes dos orbitais moleculares, que são então usados para calcular a matriz de densidade. Este processo é repetido até que a convergência na energia total seja alcançada, resultando em uma descrição autoconsistente do sistema eletrônico.

Além disso, o método Hartree-Fock serve como base para métodos mais avançados, como a Teoria do Funcional da Densidade (DFT) e métodos de correlação eletrônica pós-Hartree-Fock, que incluem correções para os efeitos de correlação eletrônica que o método Hartree-Fock não captura.

O método de campo autoconsistente (*Self-Consistent Field*, SCF) é um procedimento iterativo utilizado para resolver as equações de Hartree-Fock. O objetivo é encontrar os valores dos coeficientes C

que minimizam a energia total do sistema.

$$FC = SCE \tag{1}$$

Na Equação 1, a matriz F representa o operador Hamiltoniano de 1 elétron, ou matriz de Fock. C é a matriz de autovetores que contem os coeficientes dos orbitais moleculares, S é a matriz de sobreposição com informações sobre a sobreposição entre orbitais atômicos e E é a matriz que representa a energia dos orbitais moleculares.

Como C aparece em ambos os lados da equação, é necessário resolver as equações iterativamente até que os valores de C em ambos os lados sejam iguais e a energia seja a mais baixa possível. Este processo é chamado de método de campo autoconsistente (SCF).

A Figura 1 representa um diagrama de fluxo que ilustra os passos do método de SCF:

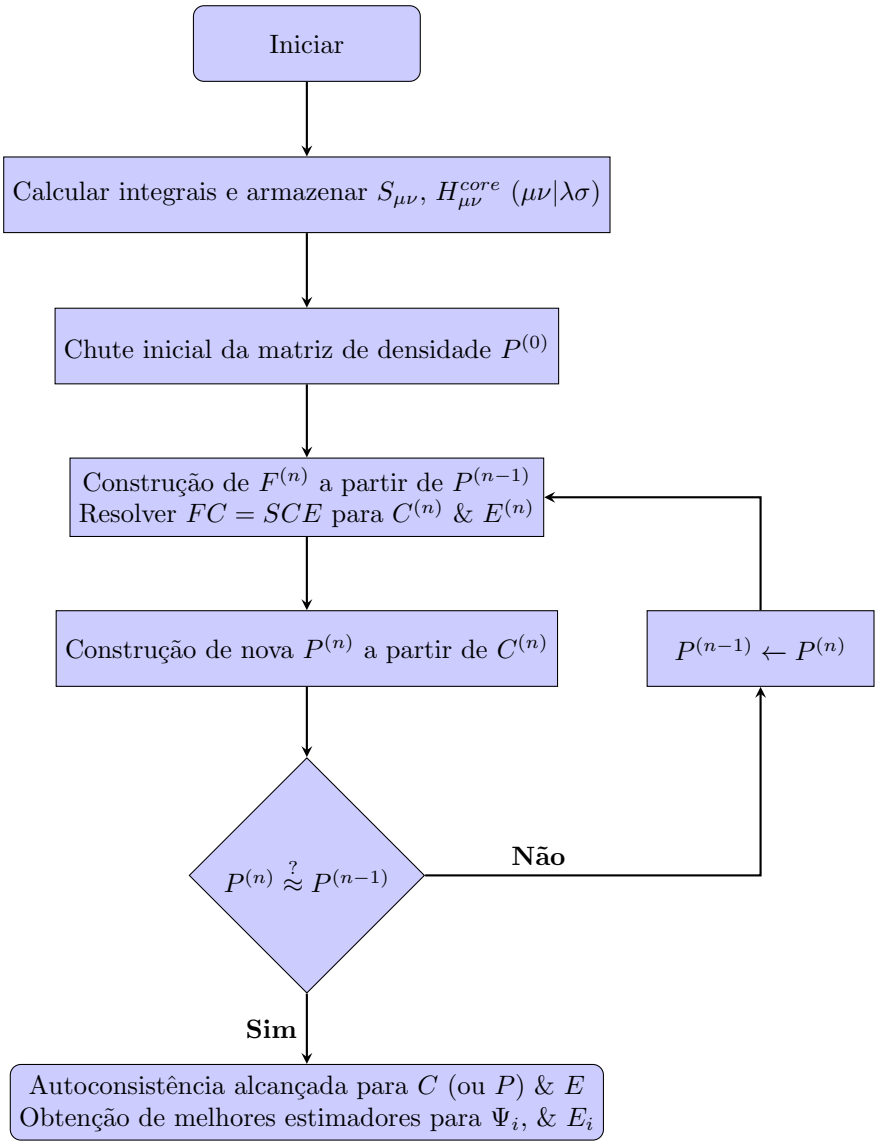

Figura 1: Fluxograma do método SCF
Fonte: Autoria Própria.

A H^{Core} é a matriz do Hamiltoniano de caroço, μ e ν são índices que são utilizados para identificar os orbitais atômicos individuais que compõem o sistema. A notação $(\mu\nu|\lambda\sigma)$ representa uma integral que descreve a repulsão eletrostática entre dois elétrons localizados em regiões descritas pelos produtos de orbitais atômicos $\mu\nu$ e $\lambda\sigma$.

A matriz P é conhecida como matriz de densidade, é a representação da distribuição dos elétrons, e ela é construída a partir da matriz C. A função de onda que é representada pelo Ψ descreve o estado quântico do sistema, incluindo posição e o momento dos elétrons.

2.2 Algoritmo MOZYME

Quando se trata de sistemas moleculares grandes, como **proteínas** e **polímeros**, o uso de métodos convencionais de **SCF** se tornam impraticáveis, devido ao tempo de calculo. Então o algoritmo **MOZYME** foi desenvolvido para resolver esse problema de tempo. Ele tem como característica o escalonamento linear de tempo em relação a quantidade de átomos. Conforme descrito por Maia, J.D.C e colaboradores [5] temos que as principais características do MOZYME são as seguintes:

1. Usar orbitais moleculares localizados (LMOs) que correspondem a estruturas eletrônicas de diagramas de Lewis na construção da primeira matriz **C**.
2. Empregar raios de corte átomo-átomo definidos pelo usuário para diminuir o número de cálculos de integrais de repulsão de 2 elétrons.
3. Pseudo-diagonalizar **F** durante cálculos SCF [8], em vez de realizar diagonalizações regulares.

Abaixo temos a Figura 2 que representa o funcionamento do algoritmo MOZYME:

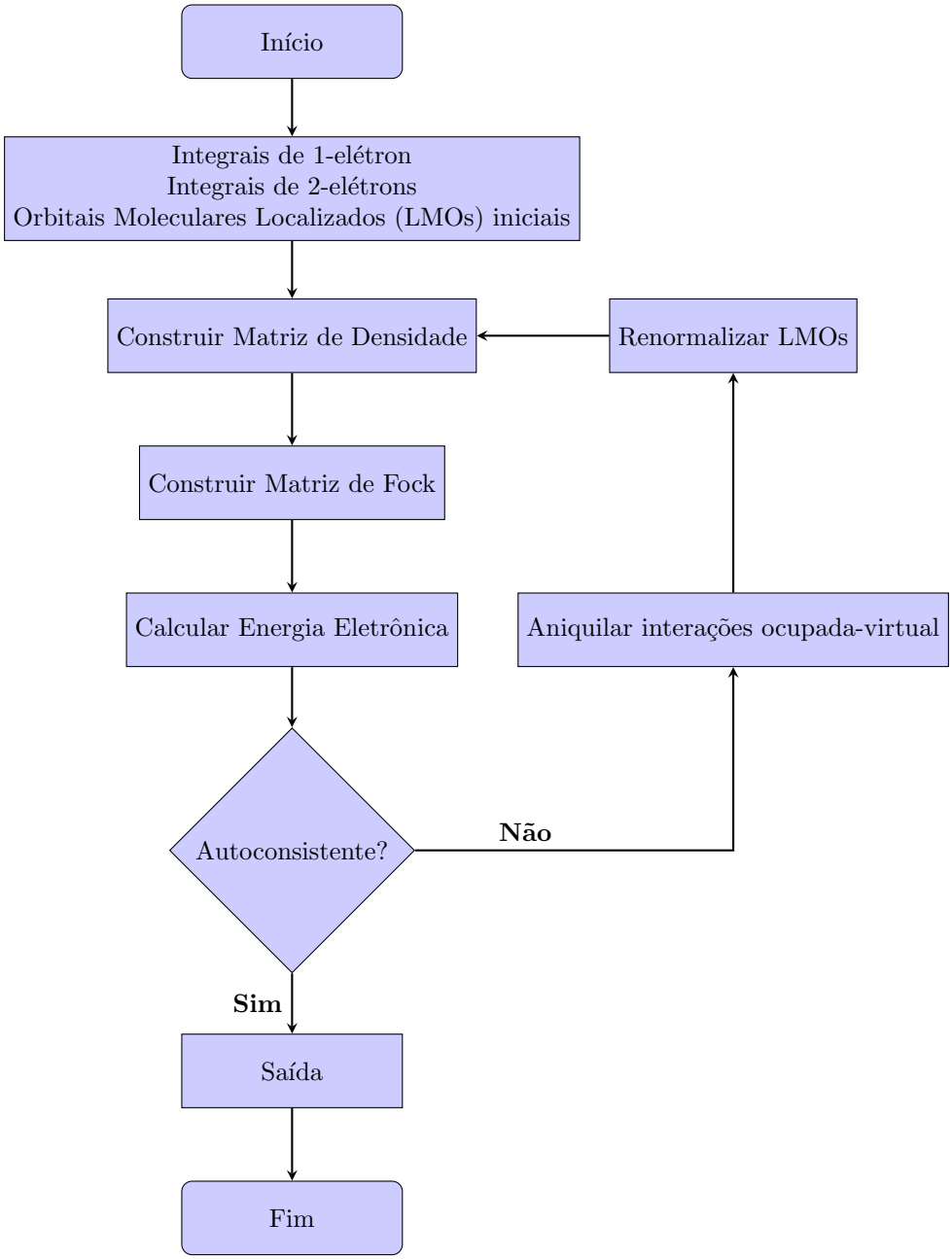

Figura 2: Fluxograma do método MOZYME
Fonte: Autoria Própria.

2.3 OpenMP

O **OpenMP**[9](do inglês *Open Multi-Processing*) é uma interface de programação para facilitar a implementação de paralelismo em softwares. No Fortran o OpenMP é utilizado a partir de diretivas de compilação, sendo possível identificá-las com o comando inicial `$OMP`.

Com o OpenMP é possível criar uma região paralela, que pode ser reconhecida pela diretiva `$OMP PARALLEL`. Nessa região paralela é feita a manipulação das **threads** do computador. Idealmente sempre que possível utilize o número de *threads* igual ao número de *cores* da maquina.

As variáveis que se encontram dentro região paralela podem ser separadas em **compartilhadas** e **privadas**. Por padrão o OpenMP define que todas as variáveis são do tipo compartilhada, ou seja, todas as **threads** podem acessar e modificar essa variáveis, enquanto que para **variáveis privadas** cada **thread** vai ter uma copia dessa variável em seu próprio escopo, podendo modificar elas sem interferir em outras **threads**.

É importante destacar que variáveis compartilhadas abrem margem para acontecer problemas de paralelismo, sendo um deles condições de corrida. Para contornar isso é possível definir quais variáveis podem ser **privadas** ou **compartilhadas** a partir das diretivas `$OMP PRIVATE` e `$OMP SHARED` respectivamente.

O OpenMP também fornece uma forma de que todas as variáveis precisam ser definidas dentro da região paralela. Para isso é necessário utilizar o `DEFAULT(NONE)`, ele é adicionado ao final da diretiva inicial da região paralela, ficando da seguinte forma `$OMP PARALLEL DEFAULT(NONE)`.

Para evitar problemas decorrente do paralelismo, é possível utilizar diretivas como `$OMP ATOMIC`, utilizada para operações simples, e `$OMP CRITICAL` que deve ser utilizado em casos mais complexos. Essas diretivas sincronizam serialmente as **threads** durante sua operação, por exemplo, durante a escrita de um valor em uma determinada região da memoria, é possível utilizar a diretiva `$OMP ATOMIC` para serializar a escrita assim impedindo que aconteça uma condição de corrida.

2.4 Ferramentas de Analise de Desempenho em Softwares

Análise de desempenho é uma técnica utilizada para mapear e entender o tempo gasto em funções específicas e o uso de paralelismo em um software. Essa análise é essencial para identificar gargalos de desempenho, otimizar o código e melhorar a eficiência geral do software. Abaixo, listamos alguns programas amplamente utilizados para realizar essas medições:

1. Intel® VTune™ Profiler [10]: Ferramenta avançada de análise de desempenho que oferece insights detalhados sobre o desempenho do software, incluindo análise de paralelismo e uso de hardware.

2. AMD μProf [11]: Ferramenta de análise de desempenho que permite a análise detalhada do desempenho de aplicativos em processadores AMD.

3. Tracy Profiler [12]: Ferramenta de análise de desempenho em tempo real que oferece uma visão detalhada do desempenho do software, permitindo a identificação rápida de gargalos.

4. NVIDIA Nsight Compute [13]: Ferramenta de análise de desempenho para GPUs NVIDIA que fornece uma análise detalhada do desempenho de kernels CUDA.

Essas ferramentas ajudam os desenvolvedores a identificar funções que consomem muito tempo, detectar problemas de paralelismo e otimizar o uso de recursos do sistema, resultando em um software mais eficiente e de melhor desempenho.

3 Metodologia

Para iniciar a paralelização do **MOZYME**, utilizamos uma das ferramenta de análise de desempenho apresentadas que foi a *Intel® VTune™ Profiler*. Com o VTune, foi possível identificar as sub-rotinas do MOZYME que demandam mais recursos computacional.

3.1 Análise de Desempenho do Algoritmo MOZYME

Para a análise de desempenho do **MOZYME** (Figura 3) utilizamos um sistema de moléculas de água, totalizando 8.211 átomos, que levou **234 segundos** para ser finalizado. Já para configuração do **MOPAC**, utilizamos as seguintes palavras-chaves "*PM7 1SCF XYZ MOZYME PL T=1D CUTOFF=9.0 LET*".

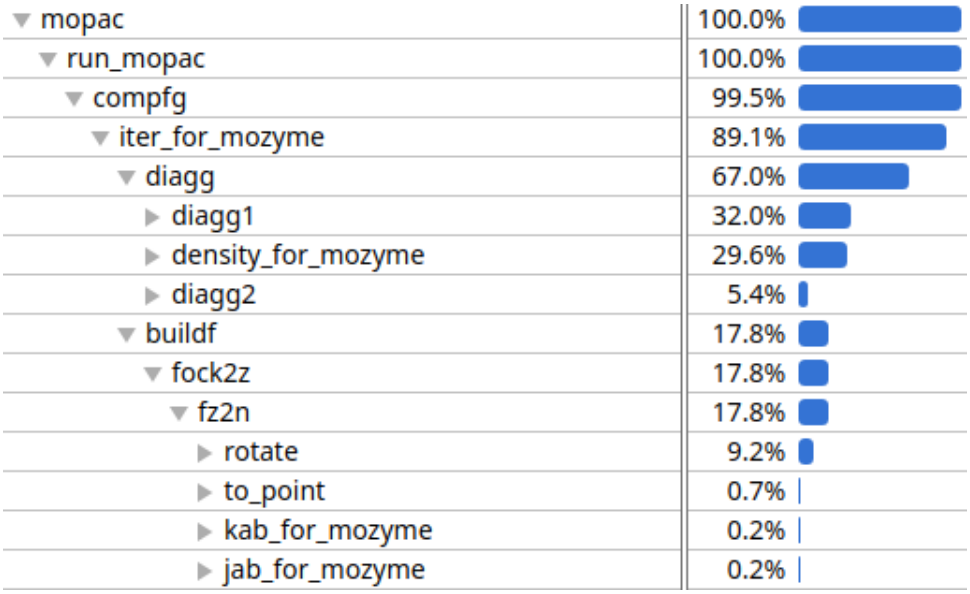

Figura 3: Resultados da avaliação de desempenho do MOPAC com VTune.
Fonte: Autoria Própria.

O executável do **MOPAC** utilizado para a análise de desempenho, foi compilado utilizando o compilador fortran do projeto de software livre GNU, o **gfortran** [14]. Os parâmetros de compilação utilizados foram dois ***-g*** e ***-O2***, esses parâmetros foram escolhidos com base nas recomendações para o uso do VTune.

O parâmetro ***-g*** foi utilizada para definir níveis de depuração, ou seja, o compilador realiza algumas marcações no binário para que possa ser utilizado por algum software de depuração. O segundo parâmetro citado, ***-O2***, foi usado para definir nível de otimização.

O hardware utilizado para a análise de desempenho foi uma máquina contendo um processador **Intel Core i7-9750h**, com clock base de **2.6 GHz** e memória de **16GB DDR4-2666**.

Como pode ser observado na Figura 3, durante cada iteração do algoritmo **SCF** (sendo identificado pela sub-rotina `iter_for_mozyme`) as sub-rotinas que mais utilizam recursos computacionais são a `diagg` e a `buildf`, mas é notável que a sub-rotina `diagg` é responsável por **67%** do tempo de execução do programa.

A sub-rotina `diagg` é composta pela `diagg1`, `diagg2` e `density_for_mozyme` (identificaremos ela apenas por `density`). Sendo `diagg1` e `density` as sub-rotinas alvos para paralelismo.

3.2 Paralelização de Sub-Rotinas

A paralelização das sub-rotinas `density` e `diagg1` foi realizada utilizando **OpenMP** para distribuir o trabalho entre múltiplas threads da `CPU`. A reestruturação do código foi necessária para lidar com a variável `lijbo`, que determina se a matriz `nijbo` deve ser usada diretamente ou se a função `ijbo` deve ser chamada para obter os índices. Isso garante que o código possa lidar com diferentes cenários de forma eficiente, utilizando a abordagem mais adequada com base no valor de `lijbo`.

- **lijbo**: Variável lógica que determina se a matriz `nijbo` deve ser usada diretamente ou se a função `ijbo` deve ser chamada para obter os índices.
- **nijbo**: Matriz que armazena os índices pré-calculados para interações entre átomos.
- **ijbo**: Função que calcula os índices para interações entre átomos quando `lijbo` é falso.

Os pseudocódigos abaixo, demonstram a reestruturação que foi feita durante o desenvolvimento. Nesse exemplo usaremos o pseudocódigo da `density` para exemplificar. A Figura 4 apresenta o código antes da reestruturação e a Figura 5 apresenta o código após a reestruturação.

```
Início da subrotina density
    ...
    Para cada i de 1 até Número de Orbitais Moleculares Ocupados faça
        Se lijbo então
            Calcular densidade usando nijbo
        Senão
            Calcular densidade usando ijbo
        Fim Se
    Fim Para
    ...
Fim da subrotina density
```

Figura 4: Pseudocódigo da sub-rotina density antes da reestruturação do código

```
Início da subrotina density
    ...
    Se lijbo então
        Para cada i de 1 até Número de Orbitais Moleculares Ocupados faça
            Calcular densidade usando nijbo
        Fim Para
    Senão
        Para cada i de 1 até Número de Orbitais Moleculares Ocupados faça
            Calcular densidade usando ijbo
        Fim Para
    Fim Se
    ...
Fim da subrotina density
```

Figura 5: Pseudocódigo da reestruturação da sub-rotina density.

3.2.1 Sub-rotina density

Na sub-rotina `density`, a paralelização foi aplicada nas seções de código que calculam a matriz de densidade `p`. A sub-rotina é responsável por calcular a matriz `p` que representa a distribuição eletrônica no sistema molecular. Ela é calculada a partir da matriz de autovetores comprimidos e das informações sobre a ocupação dos orbitais moleculares. A matriz de densidade é essencial para determinar várias propriedades eletrônicas e energéticas do sistema. A corretude da implementação avaliamos pelo número de iterações e pelo calor de formação, esses dois dados podem ser checados no arquivo de registro com extensão `.out` que o MOPAC gera quando inicia o cálculo.

Abaixo temos a descrição das variáveis que são utilizadas e o pseudocódigo da `density`.

- `p` - Matriz de densidade
- `mode` - Modo de operação
- `nclose_loc` - Número de orbitais moleculares ocupados
- `partp` - Matriz de densidade parcial
- `sum` - Soma dos elementos da matriz de densidade
- `spinfa` - Fator de spinS
- `ncocc` - Número de coeficientes de orbitais ocupados
- `nncf` - Endereço inicial dos números de átomos nos LMO
- `ncf` - Número de átomos envolvidos no LMO
- `icocc` - Números dos átomos nos LMO
- `iorbs` - Número de orbitais em cada átomo
- `cocc` - Coeficientes dos orbitais atômicos dos LMO
- `nijbo` - Endereço inicial dos elementos da matriz de densidade

Descrição dos valores que mode pode assumir.

- `mode == 0`:
  - Inicializa a matriz de densidade `p` com zeros.
  - Utilizado para uma nova e completa cálculo da matriz de densidade.
- `mode == -1`:
  - Remove a densidade devida aos antigos LMOs usados no SCF.
  - Utilizado quando a matriz de densidade completa está em `partp` e é necessário remover a densidade dos antigos LMOs.
- `mode == 1`:
  - Constrói a matriz de densidade parcial a partir de `partp` e adiciona à matriz de densidade `p`.
  - Utilizado quando a matriz de densidade parcial está em `partp` e é necessário completar a matriz de densidade `p`.
- Outros valores de `mode`:
  - Utiliza um fator de spin `spinfa` igual a 1.0.
  - Pode ser utilizado para outros cálculos específicos onde `mode` não é 0, -1 ou 1.

```
Início da subrotina density
    Se mode == 0 então
        Inicializar p com zeros
    Senão Se mode == -1 então
        p = -0,5 * partp
    Senão
        p = 0,5 * partp
    Fim Se

    Para cada i de 1 até nclose_loc faça
        loop = ncocc(i)
        ja = 0
        Para cada jj de nncf(i) + 1 até nncf(i) + ncf(i) faça
            j = icocc(jj)
            nj = iorbs(j)
            ka = loop
            Para cada kk de nncf(i) + 1 até nncf(i) + ncf(i) faça
                k = icocc(kk)
                Se j == k então
                    l = nijbo(j, k)
                    Para cada j1 de 1 até nj faça
                        sum = cocc(ja + j1 + loop)
                        Para cada k1 de 1 até j1 faça
                            k2 = ka + k1
                            l = l + 1
                            p(l) = p(l) + cocc(k2) * sum
                        Fim Para
                    Fim Para
                Senão Se j > k e nijbo(j, k) >= 0 então
                    l = nijbo(j, k)
                    Para cada j1 de 1 até nj faça
                        sum = cocc(ja + j1 + loop)
                        Para cada k1 de 1 até iorbs(k) faça
                            k2 = ka + k1
                            l = l + 1
                            p(l) = p(l) + cocc(k2) * sum
                        Fim Para
                    Fim Para
                Fim Se
                ka = ka + iorbs(k)
            Fim Para
            ja = ja + nj
        Fim Para
    Fim Para

    Se mode == 0 ou mode == 1 então
        spinfa = 2
    Senão Se mode == -1 então
        spinfa = -2
    Senão
        spinfa = 1
    Fim Se

    Se Abs(spinfa - 1) > 1e-10 então
        p = spinfa * p
    Fim Se

Fim da subrotina density
```

Figura 6: Pseudocódigo da sub-rotina density.

Para a paralelização da `density`, foram realizadas as seguintes ações:

- **Diretivas OpenMP**: As diretivas OpenMP foram utilizadas para paralelizar os laços de repetição que calculam a matriz de densidade. As diretivas `!$OMP PARALLEL` e `!$OMP DO` foram inseridas para

distribuir o trabalho entre múltiplos `threads` da `CPU`. Isso permite que diferentes partes do cálculo sejam executadas simultaneamente, aumentando a eficiência e reduzindo o tempo de execução.

- **Variáveis Privadas e Compartilhadas**: Durante a paralelização, foi necessário definir quais variáveis seriam privadas para cada `thread` e quais seriam compartilhadas entre os `threads`. As variáveis de loop e outras variáveis temporárias foram definidas como privadas para evitar condições de corrida. As variáveis que armazenam os dados de entrada e saída, como **p** e outras matrizes e vetores relevantes, foram definidas como compartilhadas para garantir que todos os threads trabalhassem com os mesmos dados.

- **Atomicidade**: Para garantir a correção dos cálculos, especialmente nas operações de atualização da matriz `p`, foi utilizo a diretiva `!$OMP ATOMIC`. Essa diretiva garante que as operações de atualização sejam atômicas, ou seja, que apenas um `thread` possa executar a operação de atualização por vez. Isso evita condições de corrida e garante que os resultados finais sejam corretos.

Essas modificações permitiram que a sub-rotina **density** se beneficiasse do paralelismo, resultando em uma execução mais rápida e eficiente do cálculo da matriz de densidade.

A Figura 7 apresenta o pseudocódigo da **density** com paralelização, algumas partes do código foram omitidas para manter o foco nas seções relevantes.

```
Início da subrotina density
    ...

    Inicio da região paralela
    Variáveis Compartilhadas (nclose_loc, ncocc, nncf, ncf, &
    icocc, nijbo, cocc, p, iorbs)
    Variáveis Privadas (i, j, ja, jj, nj, ka, &
    loop, kk, k, l, j1, sum, k1, k2)
    Distribuindo iterações do laço de forma uniforme entre as threads
    Para cada i de 1 até nclose_loc faça
        ...
                Se j == k então
                    l = nijbo(j, k)
                    Para cada j1 de 1 até nj faça
                        sum = cocc(ja + j1 + loop)
                        Para cada k1 de 1 até j1 faça
                            k2 = ka + k1
                            l = l + 1
                            Atualização atômica da matriz de densidade
                            p(l) = p(l) + cocc(k2) * sum
                        Fim Para
                    Fim Para
                Senão Se j > k e nijbo(j, k) >= 0 então
                    l = nijbo(j, k)
                    Para cada j1 de 1 até nj faça
                        sum = cocc(ja + j1 + loop)
                        Para cada k1 de 1 até iorbs(k) faça
                            k2 = ka + k1
                            l = l + 1
                            Atualização atômica da matriz de densidade
                            p(l) = p(l) + cocc(k2) * sum
                        Fim Para
                    Fim Para
                Fim Se
        ...
    Fim Para

    Fim da região paralela

    ...

    Se Abs(spinfa - 1) > 1e-10 então
        p = spinfa * p
    Fim Se

Fim da subrotina density
```

Figura 7: Pseudocódigo da paralelização da sub-rotina density.

3.2.2 Sub-rotina diagg1

A sub-rotina `diagg1` é responsável por calcular todos os elementos significativos da matriz que conecta orbitais moleculares **ocupados** e **virtuais**. Esses elementos são armazenados em uma matriz Fock sobre orbitais moleculares (`fmo`) e seus índices são armazenados em `ifmo`. A sub-rotina também calcula os níveis de energia dos LMOs ocupados e virtuais. Além disso, `diagg1` calcula e armazena os autovalores dos orbitais moleculares virtuais em `eigv` e os autovalores dos orbitais moleculares ocupados em `eigs`. Outros vetores importantes utilizados incluem `aocc`, que armazena as contribuições dos átomos nos LMOs ocupados, e `avir`, que armazena as contribuições dos átomos nos LMOs virtuais.

A `diagg1` contém dois laços de repetição principais: O primeiro calcula os níveis de energia dos orbitais moleculares virtuais (`eigv`) e avalia as interações entre orbitais ocupados e virtuais, utilizando vetores como `ws` (armazenamento temporário), `avir` (contribuições dos átomos nos LMOs virtuais), `aov` (contribuições dos átomos nos LMOs), `fmo` (matriz Fock sobre orbitais moleculares) e `ifmo` (índices da matriz Fock). A segunda função calcula os níveis de energia dos orbitais moleculares ocupados (`eigs`),

utilizando arrays como `aocc` (contribuições dos átomos nos LMOs ocupados) e `avir`. O código também inclui laços para calcular interseções átomo-átomo da matriz de Fock sobre orbitais atômicos (`fao`), utilizando vetores como `cocc` (coeficientes dos orbitais ocupados) e `p` (matriz de densidade) e somando os resultados intermediários em `sum` e `sum1`.

Inicialmente, o número de *threads* é configurado com base no número de orbitais virtuais (`nvir`). Se nvir for menor ou igual ao número máximo de *threads* disponíveis, apenas uma thread é utilizada. Caso contrário, o número máximo de *threads* é igual ao número de *cores*. Cada *thread* calcula uma fatia estática dos orbitais virtuais, garantindo uma distribuição equilibrada da carga de trabalho.

Matrizes locais são alocados para cada thread. Essas matrizes são utilizados para armazenar dados intermediários durante os cálculos. Dentro da região paralela, cada thread processa sua fatia dos orbitais virtuais, calculando os níveis de energia e as interações ocupadas-virtuais.

Após o cálculo paralelo, as *threads* são sincronizadas e os resultados são agregados. O índice `ijc` é atualizado para refletir o número total de elementos de matriz significativos. Essa abordagem garante que os cálculos sejam realizados de forma eficiente e precisa, aproveitando ao máximo os recursos de hardware disponíveis.

A variável *idiagg*, que originalmente congelava a esparsidade da matriz *fmo* após a sexta iteração e em todas as iterações pares subsequentes, teve essa restrição removida na versão paralela. Isso foi feito para evitar um bug que ocorria quando a esparsidade era congelada em iterações pares, permitindo que a matriz *fmo* cresça em todas as iterações e garantindo a ortogonalidade dos orbitais moleculares.

3.3 *Benchmark*

Para avaliar o desempenho da paralelização do **MOZYME** foi desenvolvido um *script* de *benchmarking* em Python. O *script* executa o **MOPAC** com vários arquivos de entrada e diferentes números de *threads*, medindo o tempo de execução (*wall clock time*) para cada configuração. Além disso, o MOPAC foi configurado para registrar o tempo de execução de cada sub-rotina, permitindo uma análise detalhada do desempenho de cada função individualmente.

3.3.1 Configuração do *Benchmark*

Os testes de *benchmark* foram realizados em uma máquina equipado com um processador AMD Ryzen Threadripper PRO 5995WX, contendo 64 núcleos físicos e 128 *threads*, e 128 GB de RAM. O *script* de benchmarking foi configurado para utilizar diferentes números de *threads*, variando de um único *thread* até o máximo de 128 *threads*, para avaliar a escalabilidade do **MOZYME**.

Os sistemas que foram utilizados nos testes incluem:

Sistema	**Descrição**	**Quantidade de Átomos**	**Raio de Corte (Angstroms)**
CH2O	Molécula de CH_2O	4	4
water_711_9	Sistema de águas	711	9
water_711_15	Sistema de águas	711	15
glu_ala_0192_9	Sistema de Glu-Ala 192	4994	9
glu_ala_0192_15	Sistema de Glu-Ala 192	4994	15
water_8211_9	Sistema de águas	8211	9
water_8211_15	Sistema de águas	8211	15
glu_ala_1280_9	Sistema de Glu-Ala 1280	33282	9
glu_ala_1280_15	Sistema de Glu-Ala 1280	33282	15

Tabela 1: Descrição dos sistemas com suas respectivas quantidades de átomos e raios de corte.

Todos os sistemas, exceto o CH_2O, podem ser encontrados no website ErgoSCF [15].

O *script* de *benchmarking* executa o **MOPAC** com cada um desses arquivos de entrada, variando o número de *threads* e medindo o tempo de execução para cada configuração. Os resultados são então analisados para determinar a eficiência da paralelização e a escalabilidade do **MOZYME**.

4 Resultados e discussões

Esta seção iremos apresentar os resultados obtidos após a realização de experimentos com diversas moléculas, serão apresentados tabelas e gráficos. Como apresentado na seção anterior, foi criado um *benchmark* para validar a nossa implementação paralela do MOZYME.

Abaixo temos a Tabela 2 que mostra os resultados do *benchmark* de tempo de execução para os sistemas apresentados na Tabela 1. A Tabela 2 apresenta os tempos de execução (em segundos) para cada sistema, utilizando 1, 2, 4, 8, 16, 32, 64 e 128 *threads*.

Sistema	1	2	4	8	16	32	64	128
CH2O	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,02
water_711_9	4,38	3,07	2,32	2,06	2,09	2,02	1,99	2,27
water_711_15	6,53	4,89	3,99	3,67	4,03	3,84	3,80	4,05
glu_ala_0192_9	98,97	67,04	50,57	43,72	40,31	41,05	37,05	37,18
glu_ala_0192_15	120,75	81,78	61,93	53,87	49,86	46,71	45,50	45,34
water_8211_9	272,40	167,21	120,79	99,14	90,50	82,43	79,40	80,10
water_8211_15	444,60	268,04	199,45	168,03	155,59	142,55	138,66	139,05
glu_ala_1280_9	2.575,33	1.950,00	1.682,43	1.473,62	1.269,86	1.152,77	1.092,61	1.125,76
glu_ala_1280_15	3.053,06	2.288,74	1.937,32	1.683,86	1.461,26	1.325,26	1.259,17	1.245,07

Tabela 2: *Benchmark* de Tempo de Execução (em segundos) por Número de *Threads*
Fonte: Autoria Própria.

Os resultados mostram uma clara melhoria no tempo de execução conforme o número de *threads* aumenta. Isso é especialmente evidente nos sistemas maiores, onde a paralelização tem um impacto significativo na redução do tempo de cálculo. Para a `glu_ala_1280_15` foi possível um *speedup* de **2,8x** quando comparado o uso de 1 *thread* com 128 *threads*.

Abaixo temos a Figura 8 que apresenta os resultados do sistema de águas com 8211 e a Figura 9 mostra os resultados para o sistema de Glu-Ala 1280. Cada figura apresentam uma comparação entre os raios de corte de 9 e 15 Angstroms.

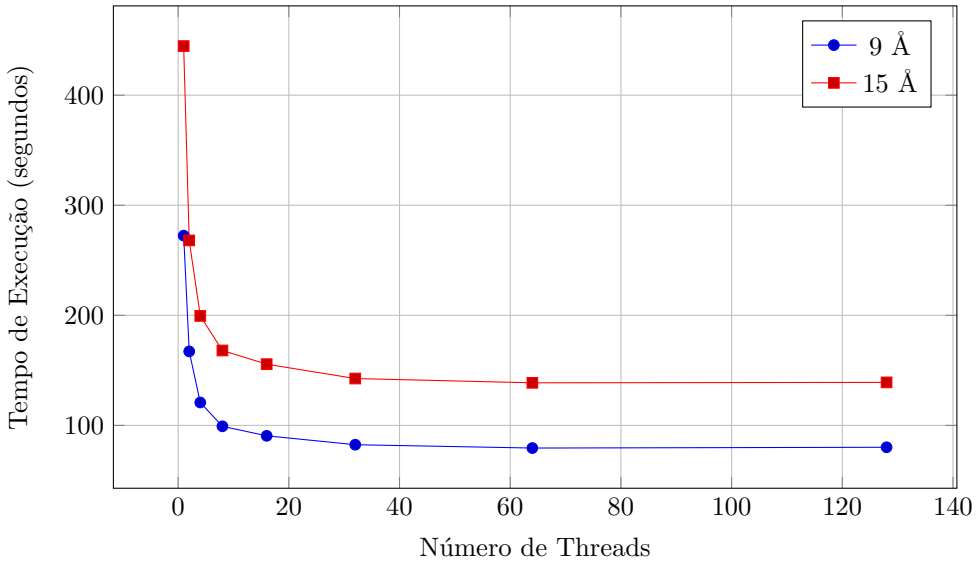

Figura 8: Gráfico de Tempo de Execução por Número de *Threads* para water_8211 com raio de corte 9 e 15

Fonte: Autoria Própria.

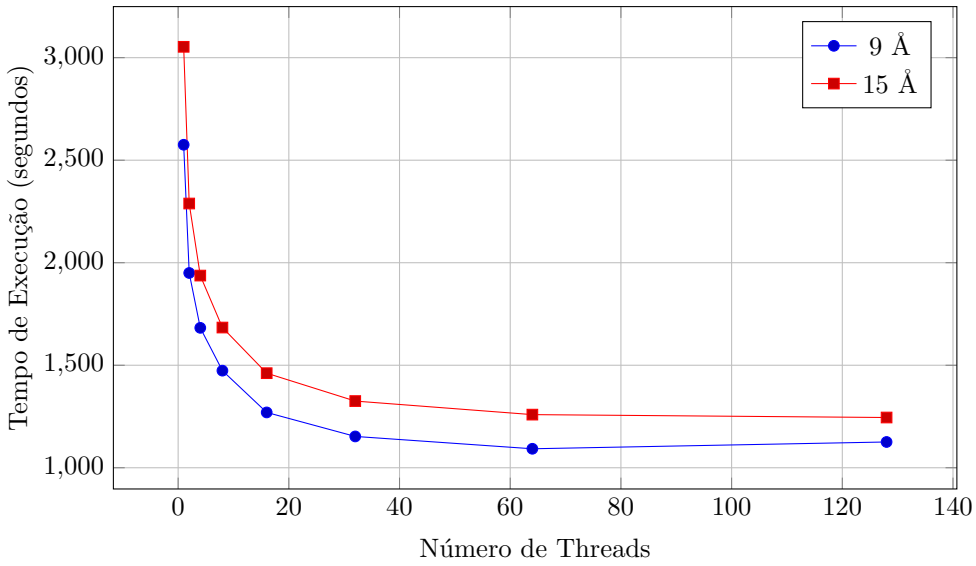

Figura 9: Gráfico de Tempo de Execução por Número de *Threads* para glu_ala_1280 com raio de corte 9 e 15

Fonte: Autoria Própria.

As figuras acima os ganhos do paralelismo no **MOZYME**. Quando não se á limitação de memoria pode ser possível até um aumento no raio de corte fazendo com que os cálculos sejam mais precisos.

É notável que a partir de 20 *threads* não se tem ganhos significativos nos dois sistemas moleculares. Abaixo temos a Figura 10 e 11 mostrando o *speedup* ao longo das *threads*.

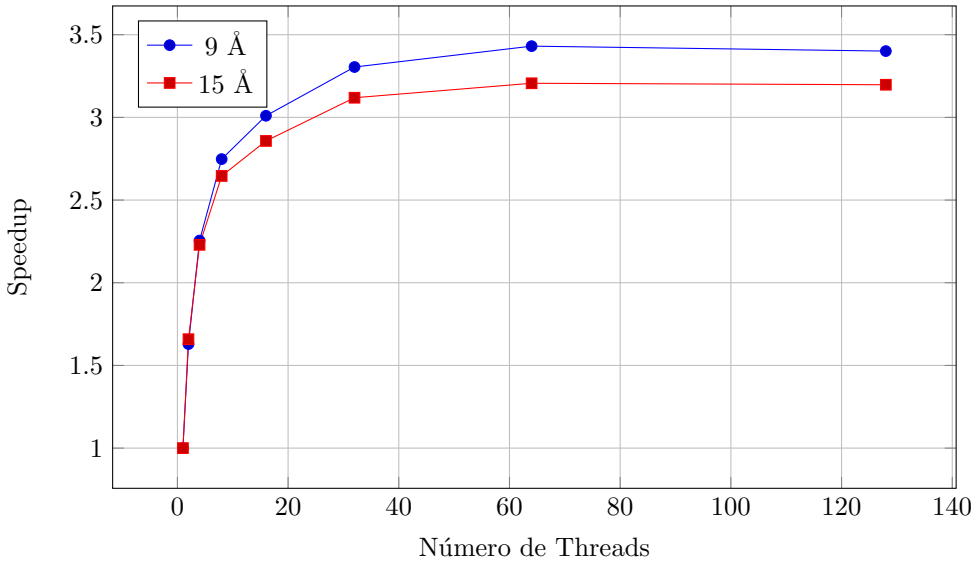

Figura 10: Gráfico de *speedup* para water_8211 com raios de corte 9 e 15
Fonte: Autoria Própria.

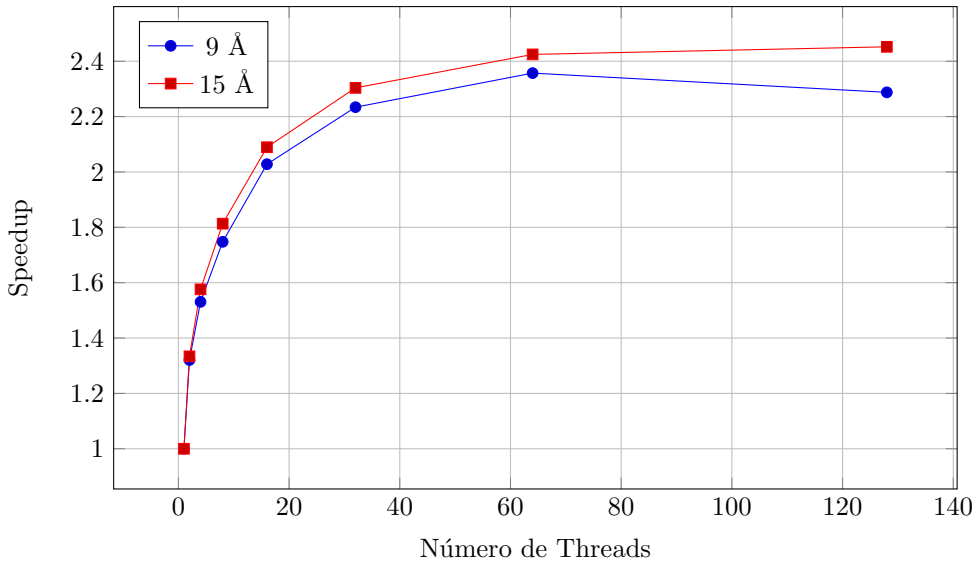

Figura 11: Gráfico de *speedup* para glu_ala_1280 com raios de corte 9 e 15
Fonte: Autoria Própria.

A Figura 11 apresenta um pico de *speedup*, para o raio de corte com 15 Angstroms, de **2,4x** com 128 *threads*. É esperado que o aumento do número de *threads* em algum momento pode levar a uma piora no tempo do calculo se comparado ao resultado anterior, isso pode ser observado com o raio de corte de 9 Angstroms, que com 128 *threads* o *speedup* é menor que com 64 *threads*.

Na Figura 10 temos que a paralelização do **MOZYME** obteve um *speedup* de **3,4** e **3,1**, para os raios de corte de 9 e 15 Angstroms respectivamente.

Em relação ao tempo de execução das sub-rotinas *density* e *diagg1*, temos que as Figuras 12 e 13 mostram a média do tempo de execução das iterações do *SCF* pelo número de *threads* e também o sistema molecular escolhido foi a Glu-Ala 1280 com raio de corte de 15 Angstroms.

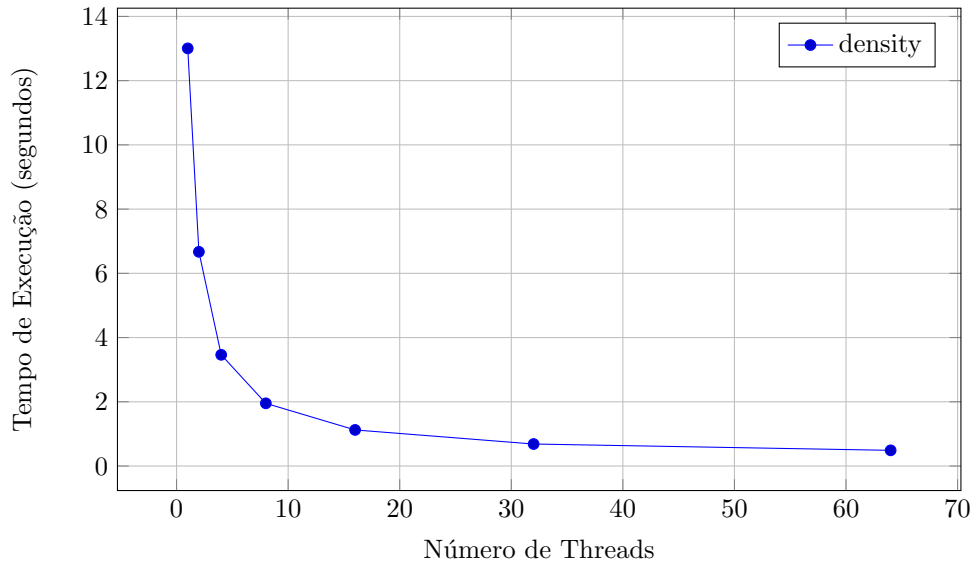

Figura 12: Gráfico da média do tempo de execução pelo número de *threads* para *density*
Fonte: Autoria Própria.

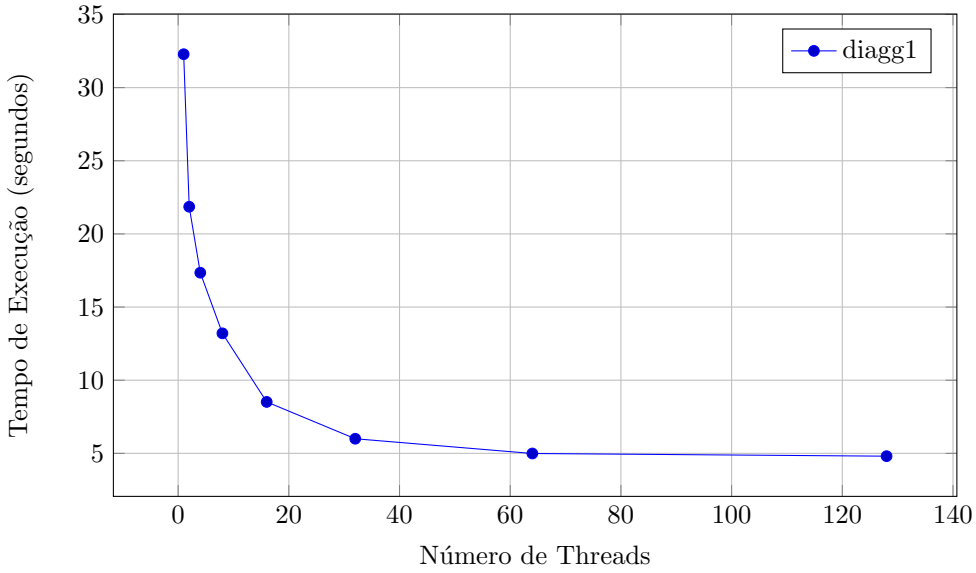

Figura 13: Gráfico da média do tempo de execução pelo número de *threads* para *diagg1*
Fonte: Autoria Própria.

Em relação ao *speedup* de cada um das duas sub-rotinas. Temos que a Figura 14 mostra o *speedup* referente a *diagg1* e a Figura 15 mostra o *speedup* referente a *density*.

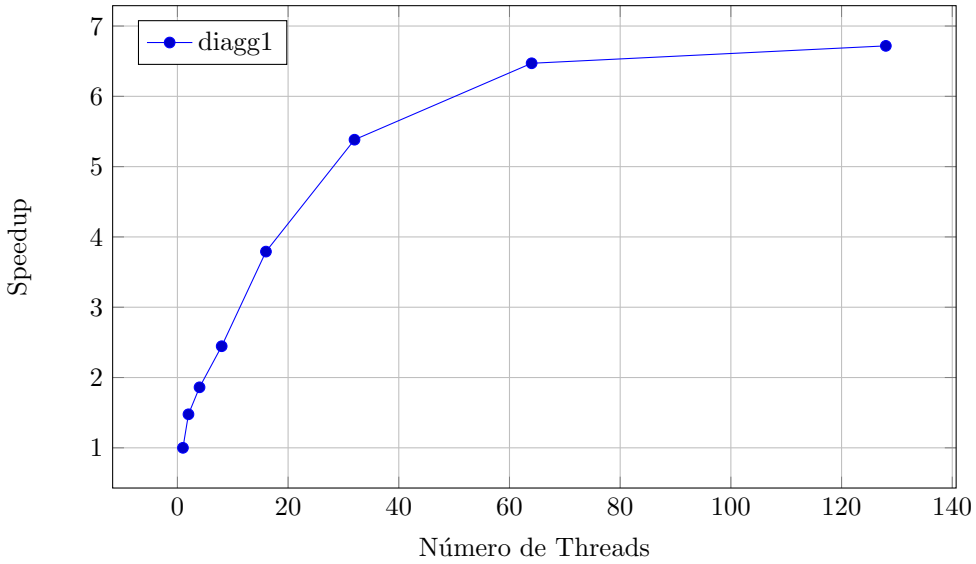

Figura 14: Gráfico de *speedup* pelo número de *threads* para *diagg1*
Fonte: Autoria Própria.

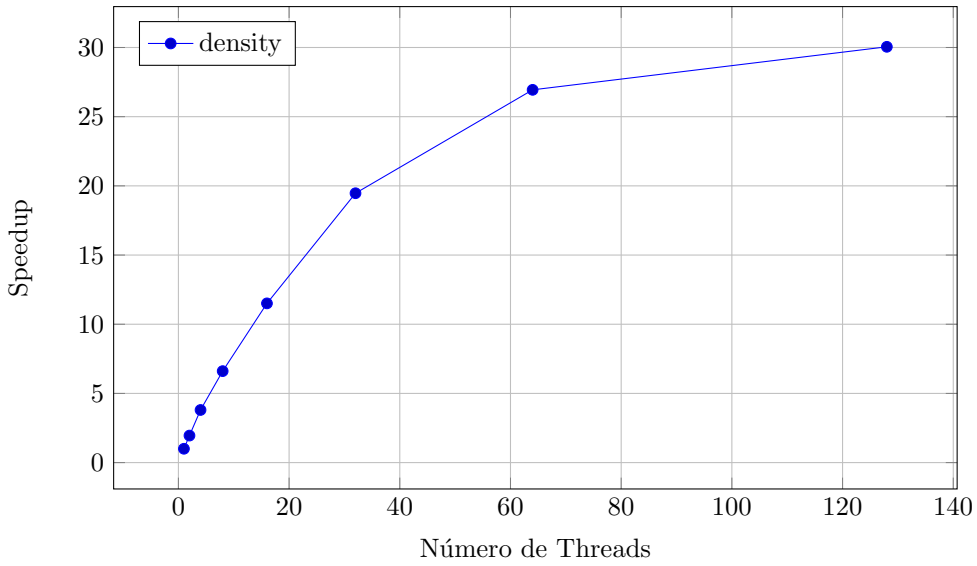

Figura 15: Gráfico de *speedup* pelo número de *threads* para *density*
Fonte: Autoria Própria.

A paralelização do **MOZYME** obtivemos um *speedup* para *diagg1* e *density* de **6,7x** e **30x**, respectivamente. O melhor *speedup* obtido para *diagg1* foi com o uso de 128 *threads* onde temos um tempo de `4,8` segundos e em comparação com 1 *thread* que foi `32,29` segundos e o melhor *speedup* para *density* foi obtido utilizando 128 *threads* com o tempo de `0,43` segundos e temos que com 1 *thread* o tempo foi de `13,0` segundos.

5 Conclusões

A paralelização das sub-rotinas *density* e *diagg1* do algoritmo **MOZYME** resultou na obtenção de um *speedup* total de 3x no tempo de execução do programa MOPAC, com a sub-rotina *density* alcançando um *speedup* de até 30x e a sub-rotina diagg1 obtendo um speedup de 6,7x. Esses resultados demonstram o potencial da otimização e paralelização do MOZYME, abrindo caminho para a análise de sistemas ainda maiores e mais complexos.

A capacidade de lidar com sistemas de grande escala, como proteínas e polímeros, torna o MOZYME uma ferramenta valiosa para pesquisas em química quântica e áreas afins. A continuidade na otimização e paralelização do MOZYME, juntamente com o desenvolvimento de novas técnicas e algoritmos, promete impulsionar ainda mais o progresso na aplicação de métodos de química quântica em sistemas moleculares complexos.

Em relação a trabalhos futuros, atualmente está sendo feita a paralelização da sub-rotina `buildf`. Em nossas análises com sistemas contendo mais de trinta mil átomos, a `buildf` tem um impacto relevante chegando a ser responsável por 35% do tempo total do MOPAC quando executado com um cálculo MOZYME.

Referências

[1] Clary, David C: *Quantum chemistry of complex systems*. Science, 314(5797):265–266, 2006.

[2] Stewart, James JP: *MOPAC: a semiempirical molecular orbital program*. Journal of computer-aided molecular design, 4(1):1–103, 1990.

[3] Stewart, James JP: *Application of localized molecular orbitals to the solution of semiempirical self-consistent field equations*. International journal of quantum chemistry, 58(2):133–146, 1996.

[4] Maia, Julio Daniel Carvalho, Gabriel Aires Urquiza Carvalho, Carlos Peixoto Mangueira Jr, Sidney Ramos Santana, Lucidio Anjos Formiga Cabral
e Gerd B Rocha: *GPU linear algebra libraries and GPGPU programming for accelerating MOPAC semiempirical quantum chemistry calculations*. Journal of chemical theory and computation, 8(9):3072–3081, 2012.

[5] Maia, Julio Daniel Carvalho, Lucidio dos Anjos Formiga Cabral e Gerd Bruno Rocha: *GPU algorithms for density matrix methods on MOPAC: linear scaling electronic structure calculations for large molecular systems*. Journal of Molecular Modeling, 26:1–12, 2020.

[6] Fukushima, K, M Wada e M Sakurai: *An insight into the general relationship between the three dimensional structures of enzymes and their electronic wave functions: Implication for the prediction of functional sites of enzymes*. Proteins: Structure, Function, and Bioinformatics, 71(4):1940–1954, 2008.

[7] Seaton, MJ: *Hartree–Fock method*, 1977.

[8] Stewart, JJP, P Császár e P Pulay: *Fast semiempirical calculations*. Journal of Computational Chemistry, 3(2):227–228, 1982.

[9] OpenMP: *The OpenMP API specification for parallel programming*, 2024. https://www.openmp.org, Acesso em: 16 set. 2024.

[10] Intel Corporation: *Intel VTune Profiler*, 2024. https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html, Acesso em: 16 set. 2024.

[11] AMD: *AMD μProf*, 2024. https://www.amd.com/pt/developer/uprof.html, Acesso em: 16 set. 2024.

[12] Taudul, Bartosz: *Tracy Profiler*, 2024. https://github.com/wolfpld/tracy, Acesso em: 16 set. 2024.

[13] NVIDIA Corporation: *NVIDIA Nsight Compute*, 2024. https://developer.nvidia.com/nsight-compute, Acesso em: 16 set. 2024.

[14] Fortran, GNU: *The GNU Fortran Compiler*, 2022. https://gcc.gnu.org/fortran/, Acesso em: 12 out. 2024.

[15] ErgoSCF: *ErgoSCF XYZ Files*, 2021. http://www.ergoscf.org/xyz.php, Acesso em: 12 out. 2024.