UNIVERSIDADE ESTADUAL DE SANTA CRUZ
PRO-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM COMPUTACIONAL EM CIÊNCIA E TECNOLOGIA

ANUSIO MENEZES CORREIA

APRIMORAMENTO DO CÓDIGO CYRANO DE SIMULAÇÃO DE AQUECIMENTO DE PLASMA POR ONDAS RF UTILIZANDO TÉCNICAS DE PROCESSAMENTO PARALELO

ILHÉUS-BA
2018

ANUSIO MENEZES CORREIA

APRIMORAMENTO DO CÓDIGO CYRANO DE SIMULAÇÃO DE AQUECIMENTO DE PLASMA POR ONDAS RF UTILIZANDO TÉCNICAS DE PROCESSAMENTO PARALELO

Dissertação apresentada ao Programa de Pós-Graduação em Modelagem Computacional em Ciência e Tecnologia da Universidade Estadual de Santa Cruz, como parte das exigências para obtenção do título de Mestre em Modelagem Computacional em Ciência e Tecnologia.

Orientador: Prof. Dr. Esbel Tomas Valero Orellana

Coorientador: Prof. Dr. Gesil Sampaio Amarante Segundo

ILHÉUS-BA
2018

C824 Correia, Anusio Menezes.
Aprimoramento do código Cyrano de simulação de aquecimento de plasma por ondas RF utilizando técnicas de processamento paralelo / Anusio Menezes Correia. – Ilhéus : UESC, 2018.
66f. : il. anexos.
Orientador : Esbel Tomas Valero Orellana.
Co-orientador : Gesil Sampaio Amarante Segundo.
Dissertação (Mestrado) – Universidade Estadual de Santa Cruz. Programa de Pós-graduação em Modelagem Computacional em Ciência e Tecnologia.

Inclui referências.

1. Dispositivo – Controle termonuclear. 2. Modelagem por computador. 3. Simulação (computadores). 4. Aquecimento do plasma. 5. Ondas (física). I. Orellana, Esbel Tomas Valero. II. Amarante Segundo, Gesil Sampaio. III. Título.

CDD – 621.044

ANUSIO MENEZES CORREIA

APRIMORAMENTO DO CÓDIGO CYRANO DE SIMULAÇÃO DE AQUECIMENTO DE PLASMA POR ONDAS RF UTILIZANDO TÉCNICAS DE PROCESSAMENTO PARALELO

Ilhéus-BA, 16/03/2018

Comissão Examinadora

Prof. Dr. Esbel Tomas Valero Orellana
UESC
(Orientador)

Prof. Dr. Gesil Sampaio Amarante Segundo
UESC
(Coorientador)

Prof. Dr. Prof. Félix Mas Milian
UESC

Prof. Dr. Ernesto Augusto Lerche
Laboratory for Plasma Physics, Ecole Royale Militaire

Agradecimentos

- Agradeço primeiramente a minha mãe e meu pai, minha família e meus amigos que me incentivaram a completar este trabalho.
- Agradeço ao Prof. Dr. Esbel Tomas Valero Orellana e ao Prof. Dr Gesil Sampaio Amarante Segundo pela paciência, disponibilidade e orientação, ao longo destes dois anos. Sem o qual não teria sido possível realizar este projeto.
- Agradeço aos Doutores Ernesto Augusto Lerche e Dirk Van Eester por cederem o código para estudo e pelo apoio.
- Pesquisa desenvolvida com o auxílio do Núcleo de Biologia Computacional e Gestão de Informações Biotecnológicas - (NBCGIB), com recursos FINEP/MCT, CNPQ e FAPESB e da Universidade Estadual de Santa Cruz - UESC.

"Sempre chame as coisas pelo nome que têm, o medo de um nome aumenta o medo da coisa em si." Harry Potter e a Pedra Filosofal

APRIMORAMENTO DO CÓDIGO CYRANO DE SIMULAÇÃO DE AQUECIMENTO DE PLASMA POR ONDAS RF UTILIZANDO TÉCNICAS DE PROCESSAMENTO PARALELO

Resumo

A fusão termonuclear controlada (FTC) se apresenta como uma fonte de energia limpa e promissora do ponto de vista econômico e ambiental. No entanto, esta ainda é uma tecnologia em desenvolvimento. Os estudos sobre como obter a fusão de forma viável e controlada possuem mais de 80 anos, e desenvolveram ideias como os reatores de fusão conhecidos como *Tokamaks*. Apesar de existirem *Tokamaks*, atualmente, nenhum é capais de manter a fusão, de forma auto sustentável. Os estudos nesta areá requerem o uso intensivo de modelos computacionais. Dentre os vários modelos utilizados, destaca-se o CYRANO, que simula o aquecimento do plasma por ondas de RF. No entanto, o CYRANO tem entre suas limitações o elevado consumo de recursos computacionais e a impossibilidade de utilizar as capacidades das arquiteturas paralelas atualmente disponíveis. Neste documento, apresentamos uma análise do CYRANO e do seu desempenho. Desta forma foi possível destacar que as rotinas GENERA3 e OUTPOW são responsáveis por mais de 90% do tempo total utilizado pelo código. A análise demostrou que, apesar de ter uma importância maior em execuções menores, a função OUTPOW perde em relevância para a rotina GENERA em problemas de grande porte. Então foram substituídas as funções que mais consumiam tempo em GENERA, especificamente as sub-rotinas derivadas da (BLAS), por versões mais atuais, otimizadas e algumas com versões paralelas, como a ATLAS, BLIS e OpenBLAS. Foi aperfeiçoada também a rotina OUTPOW que teve a chamada da função POWABS incorporada para então, ser reescrita e ter o laço principal paralelizado através de diretivas de OpenMP. Os resultados das otimizações demonstraram uma melhora significativa no desempenho geral do tempo de execução. Isto, com resultados do código bem próximos dos obtidos com o código original, apesar de ocorrer uma inconsistência quando executados com 512 modos poloidais. Apesar desta inconsistência, os ganhos significativos, de quase 4 vezes mais rápido que o código original, demonstraram que o uso de OpenMP, de forma geral, obteve um ótimo desempenho, tendo o tempo de execução total significativamente reduzido.

Palavras-chave: CYRANO. OpenMP. BLAS. OpenBLAS. ATLAS. BLIS.

IMPROVEMENT OF THE CYRANO CODE OF PLASMA HEATING BY WAVES RF USING PARALLEL PROCESSING TECHNIQUES

PPGMC - UESC

Abstract

The controlled thermonuclear fusion (CTF) presents itself as clean and promising energy source in the economic and environmental point of view. However its technology is still in development. The study of how to obtain a fusion in the way that it is controlled and practicable have more than 80 years, and developed many ideias like the fusion reactors known as Tokamak. Although of Tokamaks already exist, currently, none of them was able of keep the fusion in the way that is self sustainable. The studies in this area require the intensive use of computational models. Among various utilized models, stand-out CYRANO code, who simulate the plasma heating by RF waves. However, CYRANO has among these limitations, the high consumption of computational resources and the non use of capacities of the parallel architectures currently available. In this document, we present an analysis of CYRANO and its performance. In this way was possible to highlight GENERA3 and OUTPOW routines are responsible for more than 90 % of the total time used by the code. The analysis showed that, although its greater importance in smaller executions, OUTPOW routine loses relevance to GENERA routine in large problems. Then, the time-consuming functions in GENERA were replaced, specifically the sub-routines from BLAS, by more current and optimized, some with parallel versions, such as ATLAS, BLIS and OpenBLAS. Also was improved OUTPOW routine in which had the built-in POWABS function call in the body of OUTPOW, to then be rewritten and have the main loop parallelized by means of OpenMP directives. The results of the optimizations demonstrated a significant improvement in the overall execution time performance. This, with code results very close to those obtained with the original code, unless an inconsistency when executed with 512 poloid modes. Despite this inconsistency, the significant gains, almost 4 times faster than the original code, showed that the use of OpenMP, in general, obtained an excellent performance, with the total execution time significantly reduced.

Keywords: CYRANO. OpenMP. BLAS. OpenBLAS. ATLAS. BLIS.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

UESC	Universidade Estadual de Santa Cruz
DCET	Departamento de Ciências Exatas e Tecnológicas
PPGMC	Programa de Pós-Graduação em Modelagem Computacional em Ciência e Tecnologia
CACAU	Centro de Armazenamento de dados e Computação Avançada da UESC
FTC	Fusão termonuclear controlada
ICRH	*Ion-Cyclotron Ressonance Heating*
CYRANO	*CYclotron Resonance Analysis with No Obfusca-tion*
Tokamak	*toroidal'naya kamera magnitnoi katushki*
JET	*Joint European Torus*
ITER	*International Thermonuclear Experimental Reactor*
OpenMP	*Open Multi-Processing*
BLAS	*Basic Linear Algebra Subprograms*
ATLAS	*Automatically Tuned Linear Algebra Software*
BLIS	O BLIS é uma estrutura de software portátil, que serve para instanciar bibliotecas densas de álgebra linear (BLAS) de alto desempenho.
FLOPS	*floating point operations per second*
GEMM	nome da rotina da BLAS para multiplicação de matrizes

Lista de símbolos

$\forall$ para todo;

$\in$ pertencente a;

$\mathbb{R}$ conjunto dos números reais;

$\mathbb{C}$ conjunto dos números complexos;

W representa a contribuição do plasma e a contribuição parcial de cada partícula;

R representa a contribuição do operador de onda no vácuo;

A representa a contribuição das fontes do sistema (Antena RF);

$\boldsymbol{E}$ representa o vetor do campo elétrico $\boldsymbol{E}(\boldsymbol{r})$;

$\boldsymbol{F}$ representa o conjugado complexo de um arbitrário, porém suficientemente bem comportado, campo vetorial $\boldsymbol{F}(\boldsymbol{r})$;

$\boldsymbol{J_S}(\boldsymbol{r})$ representa a densidade da corrente da fonte RF;

ω representa a frequência angular da fonte RF;

β representa a indexação das diferentes especies de partícula, cuja carga é q_β e a massa é m_β;

ε representa a permissividade dielétrica do plasma;

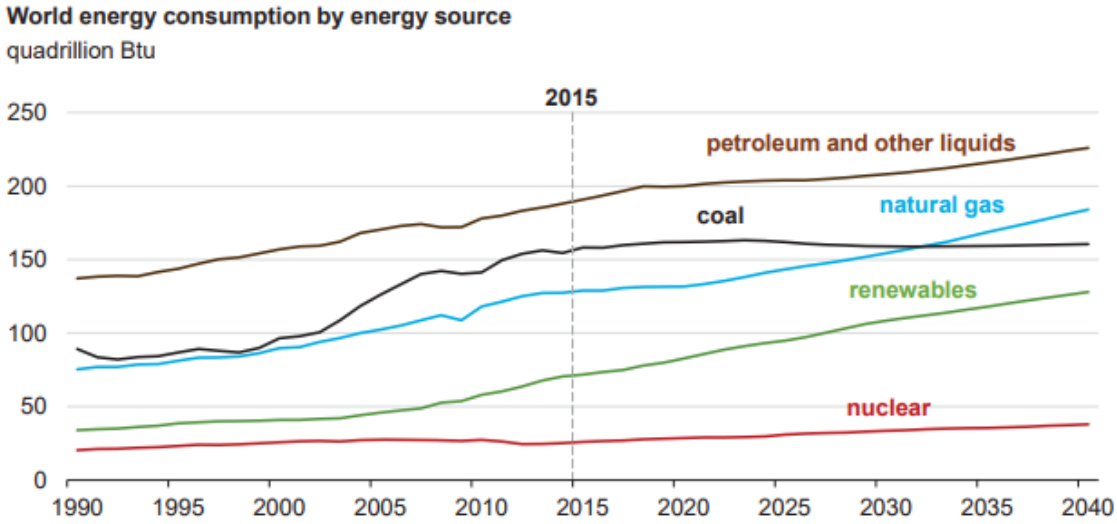
μ_0 representa a permeabilidade magnética do vácuo;

Sumário

1 Introdução

Nas últimas décadas o consumo de energia da humanidade vem crescendo continuamente (U.S. Energy Information Administration, 2016), e a tendência é continuar aumentando nas próximas décadas, como pode ser visto na figura 1. As fontes de energia mais difundidas atualmente são derivadas de combustíveis fósseis, como petróleo, gás e o carvão mineral. Estes recursos não são renováveis e portanto, apesar de existir uma grande reserva dos mesmos, a disponibilidade é finita e um dia irão acabar. Por outro lado, a queima de combustíveis fosseis envolve a emissão de CO_2 e outros gazes que agridem o meio ambiente (BAUER et al., 2015).

Figura 1 – Consumo de energia mundial por fonte.



Fonte: (U.S. Energy Information Administration, 2016).

Neste contexto vários pesquisadores, em diversas instituições e áreas, trabalham no desenvolvimento de tecnologias que utilizem fontes de energia renováveis, que não agridam o meio ambiente e sejam economicamente viáveis. Como parte destas pesquisas estão sendo propostos métodos que geram energias de formas inovadoras, tais como, retirar energia do sol, do movimento das ondas do mar, do fluxo de movimento dos carros nas pistas, dentre muitas outras (U.S. Energy Information Administration, 2016). Nem todas as propostas são atualmente viáveis e estão em diferentes estágios de desenvolvimento.

Uma das propostas em desenvolvimento atualmente é a de retirar energia gerada pela fusão de átomos, como ocorre nas estrelas. A Fusão Termonuclear Controlada (FTC) pode ser empregada com a finalidade de transformar a energia gerada neste tipo de reação, em energia elétrica. Desta forma pode ser utilizada como utilizando

como combustível, isótopos de hidrogênio, o material mais encontrado no universo (Kruecken, T., 1988). A principal vantagem da fusão controlada é que, teoricamente, é possível produzir uma grande quantidade de energia com um baixo consumo de "combustível". Gerar energia desta forma não emitiria CO_2 na atmosfera e apesar de produzir resíduos radioativos estes se decompõe rapidamente.

A FTC já teve sua viabilidade teórica comprovada (LAWSON, 1957), e para que possa ser explorada como fonte de energia, é necessário que o plasma aquecido no reator chegue a uma temperatura da ordem de 150.000.000 (cento e cinquenta milhões) Kelvin. Isto se deve ao chamado "produto triplo de Lawnson" (LAWSON, 1957), que relaciona o tempo de confinamento e a densidade esperada para que ocorra a fusão de forma continuada, permitindo o consumo desta energia.

Apesar de a FTC ser comprovadamente possível em laboratório, ainda não há um protótipo capaz de gerar a reação de fusão por muito tempo. Por este motivo, pesquisadores estão trabalhando no desenvolvimento de projetos de reatores, que comprovem sua viabilidade. Um destes projetos, e o mais importante atualmente, é o ITER (*International Thermonuclear Experimental Reactor*) (ITER, 2017), que se propõe desenvolver um reator de fusão do tipo *Tokamak*, com capacidade para gerar 500 MWh de potência de saída com apenas 50 MWh de potência de entrada. Este protótipo esta sendo construído atualmente em Cadarache, na França.

Devido às limitações tecnológicas e aos custos envolvidos nas pesquisas necessárias para o desenvolvimento deste reator, são utilizadas de forma intensiva, modelos computacionais. Muitos modelos físico/matemáticos foram criados para simular partes distintas de uma FTC. Estes modelos foram implementados computacionalmente, pois permitem, dentre outras coisas, prever o comportamento dos materiais envolvidos na construção do reator. Desta forma, utilizando apenas o computador para efetuar os cálculos necessários, podem ser evitados custos desnecessários com os materiais envolvidos.

Dentre os vários modelos matemáticos empregados, um dos principais são os que modelam o comportamento da interação antena ICRH(*Ion-Cyclotron Ressonance Heating*) - plasma. A antena é responsável por aquecer o plasma até a temperatura adequada, sem o qual não ocorreria a fusão de forma continuada no confinamento do reator, e utiliza materiais de alto custo no seu desenvolvimento. Vários modelos computacionais foram propostos para calcular o comportamento desta relação antena-plasma, entre os que pode-se destacar o ICANT (AMARANTE-SEGUNDO G. S. ELFIMOV, 2006), o FISIC (Kruecken, T., 1988), o EVE(DUMONT, 2009) e o CYRANO (LAMALLE, 1994), dentre outros.

Estes modelos computacionais utilizam uma grande quantidade de recursos computacionais, tanto em processamento quanto em uso de memória, de modo que

levam muito tempo para obter-se resultados de suas execuções. No entanto, muitos destes códigos são antigos e escritos de forma procedural de modo que não utilizam técnicas de otimização ou paralelismo para melhorar o desempenho, o que permite uma análise e uma reescrita destes códigos para introduzir estas técnicas.

Com o intuito de obter resultados com melhor qualidade sobre o modelo proposto pelo CYRANO, e diminuir o tempo demandado para efetuar os cálculos do atual código, foi proposto desenvolver uma versão aprimorada, que possibilitasse a execução de modelos mais complexos que requeiram maior quantidade de processamento e memória. Para tal, foi utilizado técnicas de processamento paralelo em arquitetura de memória compartilhada.

No entanto, antes de efetuar alterações no código se faz necessário uma análise previa sobre o CYRANO, o que envolve uma análise de desempenho, com o objetivo de identificar os gargalos no código. Desta forme é possível definir estratégias e propor otimizações para melhorar a implementação. Levando em consideração a amplo uso de rotinas de álgebra linear, que são utilizadas no CYRANO, é importante também fazer um estudo mais detalhado das sub-rotinas da (BLAS) e das diversas implementações otimizadas da mesma.

Este documento, esta organizado da seguinte forma. No primeiro capítulo há a introdução ao tema abordado. O segundo capítulo aborda uma revisão literária dos temas relacionados, como os modelos computacionais e suas aplicações na ciência, a fusão termonuclear controlada e seus possíveis benefícios a humanidade, um dos modelos computacionais que simulam determinadas partes de um reator de fusão como os códigos de onda completa, o uso de programação paralela e as consequências de introduzir estas técnicas e os benefícios de utilizar a BLAS nos códigos, além de uma apresentação a algumas implementações da BLAS. No terceiro capítulo e feita uma abordagem sobre o CYRANO, de sua modelagem ao código fonte na versão utilizada como base para modificações, conhecida neste texto como versão original. No quarto capítulo há uma descrição da metodologia e das ferramentas utilizadas na pesquisa. Já no quinto capítulo, uma discussão sobre os resultados obtidos, como o desempenho das rotinas da BLAS, mais precisamente da ZGEMM e DGEMM, os resultados das execuções obtidas com as novas implementações do CYRANO, além dos ganhos de desempenho obtidos. Finalmente no capítulo 6 são apresentadas as conclusões e recomendações para trabalhos futuros. Deve-se destacar também a inclusão do Anexo A, onde se faz uma descrição do arquivo de entrada do CYRANO. No Anexo B são apresentados alguns conceitos importantes na hora de se tratar códigos Fortran e sua interação com códigos desenvolvidos em C.

2 Revisão da literatura

Com o avanço na tecnologia dos computadores, que aumenta a cada ano, oferecendo um poder computacional cada vez maior e permitindo que computadores mais baratos efetuem grandes quantidades de operações por segundo. Assim, muitos pesquisadores de diversas áreas da ciência, utilizam atualmente o computador como ferramenta em seus estudos. Isto acontece devido à facilidade de executar as modelagens matemáticas, que resolvem problemas relacionados as pesquisas, utilizando o computador para efetuar os cálculos. Então, para utilizar o computador como ferramenta é necessário o uso de modelos computacionais do objeto de estudo.

2.1 Modelos Computacionais

A modelagem de fenômenos conhecidos matematicamente é um procedimento cognitivo, que é primeiramente aplicado na ciência da observação das coisas, mas também envolve significantemente as teorias da ciência formal. As modelagens, em geral, podem ser divididas em reais e teóricas, enquanto as teóricas consistem num conjunto de suposições simplificadas e suas possíveis derivações, as reais são realizações de suposições e teses adotadas. A modelagem computacional, que tem a capacidade de executar o modelo matemático num computador, utilizando de algum método numérico é amplamente utilizada para fins científicos, devido a grande quantidade de operações que um computador pode realizar (STACEWICZ; WLODARCZYK, 2010).

Como exemplo de modelos que podem ser construídos e utilizados em diversas áreas, pode-se citar:

- Em física - modelos de átomos como o modelo atômico de Bhor;
- Em biologia - Modelos de neurônios como o McCulloch-Pitts modelo neural linear;
- Em psicologia - Modelos computadorizados de memória semântica;

Estes exemplos estão referenciados em (STACEWICZ; WLODARCZYK, 2010).

Um modelo computacional de algum fenômeno tem que ser bem elaborado, tanto matematicamente para resultados precisos, como computacionalmente. Pois, vale lembrar que o computador tem suas limitações, e uma modelagem computacional, com escolhas ruins de representação das estruturas de dados e dos processos de execução, podem levar a um código que consome bastante memória e/ou tempo de execução para entregar algum resultado. Resultado este que dependendo da abordagem utilizada,

pode não somente levar um grande tempo em sua execução e/ou consumir uma quantidade de memória absurda ou até impraticável, como também, pode acabar entregando resultados errados.

Ainda assim, com todas as vantagens que uma modelagem computacional pode oferecer, modelar problemas como os necessários para estudo da fusão termonuclear controlada evitariam custos com materiais e dariam mais segurança nos experimentos.

2.2 Fusão Termonuclear Controlada (FTC)

A fusão termonuclear controlada já teve a sua viabilidade científica e técnica comprovada, esta comprovação foi obtida através de experiências realizadas em *Tokamaks* de grande porte como o JET (*Joint European Torus*) que conseguiu gerar a fusão em laboratório (Llewellyn Smith; WARD, 2007).

O *Tokamak*, cujo nome é um acrônimo das palavras russas ***to***roidal'naya ***ka***mera ***ma***gnitnoi ***k***atushki, que significam câmara toroidal com bobinas magnéticas, foi inicialmente pensado para provar a viabilidade da fusão como uma fonte de energia baseada no mesmo principio que gera a energia das estrelas, se comprovada, esta seria uma "fonte de energia limpa", ou seja, livre de emissão de carbono.

Para atingir a fusão, o plasma composto de núcleos de átomos leves como Hidrogênio ou seus isótopos Deutério e Trítio, é preso numa câmara em forma de toroide, por um campo magnético para que o plasma seja aquecido, até que os átomos colidam com energia suficiente para vencer a repulsão coulombiana, se fundindo e se tornando em núcleos de Hélio e neste processo liberar energia que pode ser aproveitada para gerar energia elétrica (BUTLER, 2013).

Um reator de fusão do tipo *Tokamak* poderia produzir energia barata e quase inesgotável, que pouco agride o meio ambiente, sendo mais segura e eficiente que uma usina de fissão nuclear, pois, caso haja alguma falha, não afetariam a população ou o meio ambiente de forma impactante já que sem a pressão do confinamento a fusão cessaria imediatamente, o que não ocorreria no caso de uma falha num reator de fissão, cuja reação é espontânea e o controle consiste em conter o material utilizado dentro de um reator, desacelerando o processo e canalizando a reação na geração de energia.

Existem outros projetos de maquinas que pretendem realizar a FTC na terra, como o NIF (*National Ignition Facility*) que pretende atingir a fusão com a chamada fusão inercial confinada, utilizando lazers poderosos para aquecer e comprimir uma pequena quantidade de hidrogênio até ocorrer a reação de fusão (PERKINS et al., 2009) ou os Stellarators que também fazem o uso de confinamento magnético (MAX. . . , 2017).

O material que seria utilizado como combustível na fusão é o hidrogênio e seus

isotopos, por serem os que requerem menos esforços para se fundirem. Eles existem em abundância na natureza e apesar de o processo produzir alguns resíduos radioativos, estes possuem um ciclo de vida curto durando algumas dezenas de anos, contra milhões de anos dos resíduos gerados pela fissão.

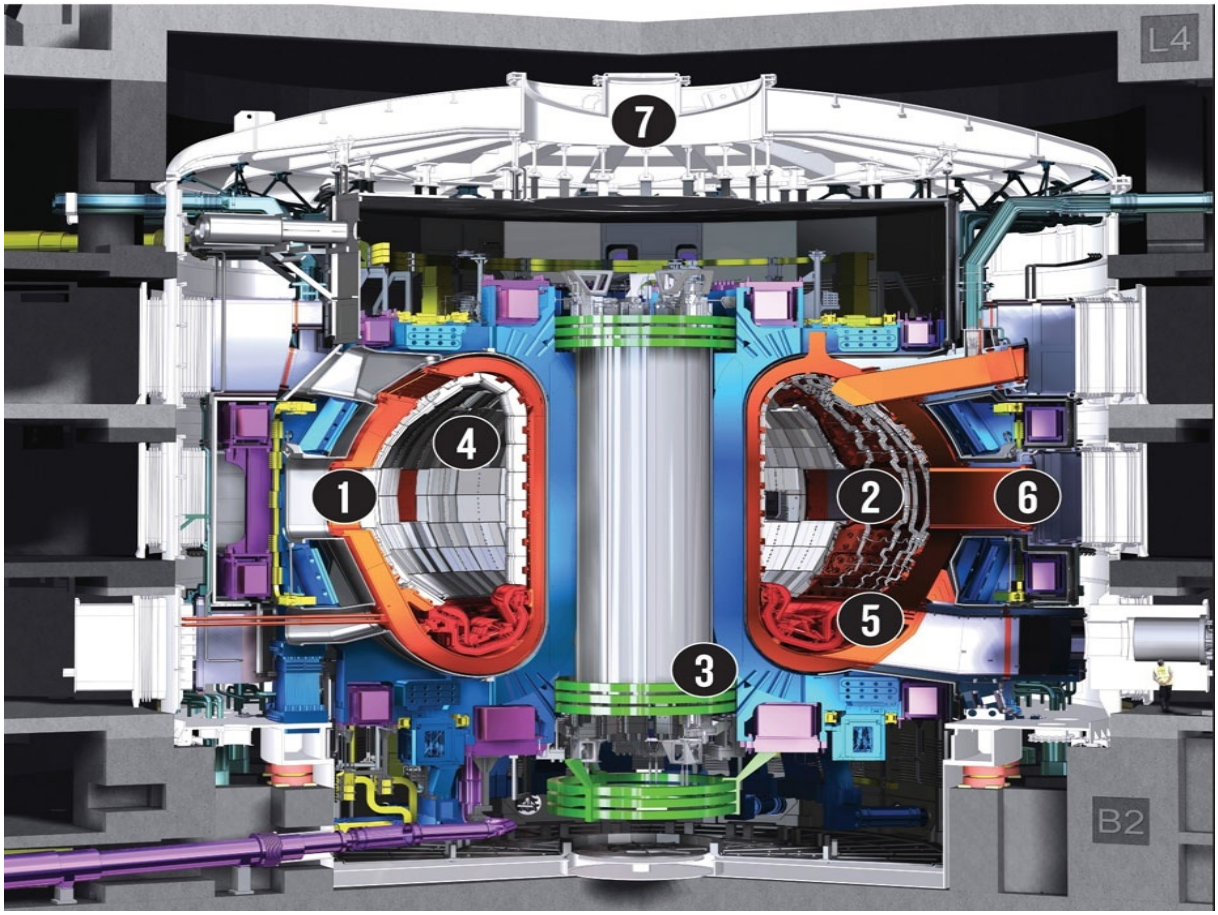
Atualmente, o maior projeto que visa criar um reator experimental de fusão nuclear é o Reator Experimental Termonuclear Internacional (ITER, em inglês) (ITER, 2017). O ITER é financiado e executado pelas nações que formam a União Europeia mais Rússia, Estados Unidos, Inglaterra, China, Coreia do Sul e o Japão. É um *Tokamak* (ver Figura 2) que está sendo atualmente construído em Cadarache, junto a cidade de Marselha na França, a área de todo o complexo equivale a sessenta campos de futebol.

O principal objetivo do ITER é demonstrar a viabilidade tecnológica da fusão ao produzi-la de forma autossustentável, o que significa que o calor gerado pela própria reação, se tornaria responsável por manter a temperatura adequada para que a fusão se mantenha, retirando, desta forma, a necessidade da continuidade de um aquecimento externo contínuo, assim, utilizando a energia para gerar energia elétrica num processo que pouco agride o meio ambiente. O ITER foi projetado para gerar 500 MW de potência de saída com apenas 50 MW de potência de entrada, ou seja, dez vezes a quantidade de energia de entrada $Q = 10$. O atual recorde para geração de energia por FTC é de 16 MW (obtido na instalação JET localizada em Culham , Reino Unido) (ITER, 2017).

Dos componentes que formam o *Tokamak* ITER, representado na Figura 2, temos:

- 1 - Recipiente de vácuo: Um grande contêiner de aço inoxidável que irá segurar o plasma e abrigar a reação de fusão;
- 2 - Sistema de aquecimento auxiliar: Onde injeções de feixes neutros e ondas eletromagnéticas de radiofrequência irão aquecer o plasma a 150.000.000 ^{o}C;
- 3 - Magnetos: dez mil toneladas de bobinas supercondutoras gerando campo 200.000 vezes maior que o campo magnético da terra, irão confinar e moldar o plasma;
- 4 - *Blanket*: Placas que pesam até 4 toneladas irão proteger o recipiente de vácuo e os Imãs do calor e dos nêutrons;
- 5 - Divertor: uma série de placas de tungstênio na parte inferior do recipiente de vácuo que recebe o calor e os gases escapados do *Tokamaks*;
- 6 - Sistemas de diagnóstico: ferramentas experimentais que são importantes (incluindo medidores de pressão e câmeras de nêutron) para a medição da física de plasma;

Figura 2 – Modelo do ITER.



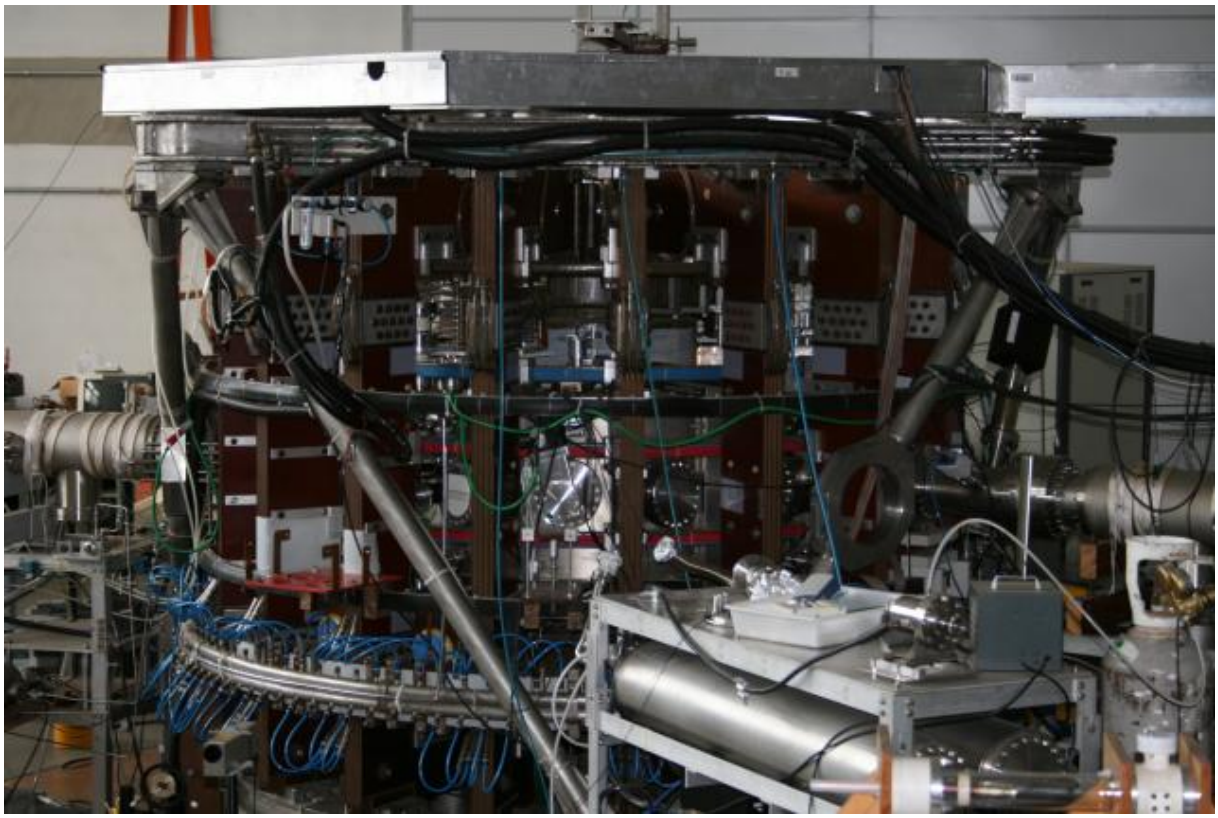
Fonte: (BUTLER, 2013).

- 7 - Criostato: um grande refrigerador cercando o recipiente de vácuo, protegendo as bobinas super condutoras e outros equipamentos do calor;

Existem outros *Tokamaks* além do ITER e do JET, alguns atualmente em funcionamento, servindo principalmente para realizar pesquisas mais fundamentais sobre plasma e outros em construção. Alguns foram descontinuados como os ISTTOK em Portugal, ALCATOR (ALto CAmpo TORus) nos EUA, JT-60 SA no Japão, entre outros. No Brasil existem os *Tokamaks Chauffage Alfvén Brasilien* (TCA-BR) nos departamentos de Física Aplicada da USP (ver Figura 3) e o ETE (Experimento *Tokamak* Esférico), possui formato esférico, está situado na Unicamp em São José dos Campos (ver Figura 4), este veio da Universidade de Kyoto para o Laboratório Associado de Plasma do INPE. Alguns *Tokamaks*, como os brasileiros, não são do porte necessário para produzir fusão nuclear, mas são utilizados em pesquisas sobre plasma.

Um importante componente do sistema de aquecimento de um *Tokamak* , é o sistema de aquecimento por *Ion-cyclotron range of frequency* (ICRF), principalmente em projetos como o ITER, que visam a produção de energia utilizando a fusão, portanto, simulações do aquecimento por ICRF, perfis de corrente e torque são necessários para

Figura 3 – Imagem do TCABR, *Tokamak* brasileiro no departamento de Física Aplicada da USP.



Fonte: (TCABR. . . , 2017)

estimar a eficácia do sistema ICRF ajudando a criar e sustentar alta potência necessária para a fusão (BUDNY et al., 2012).

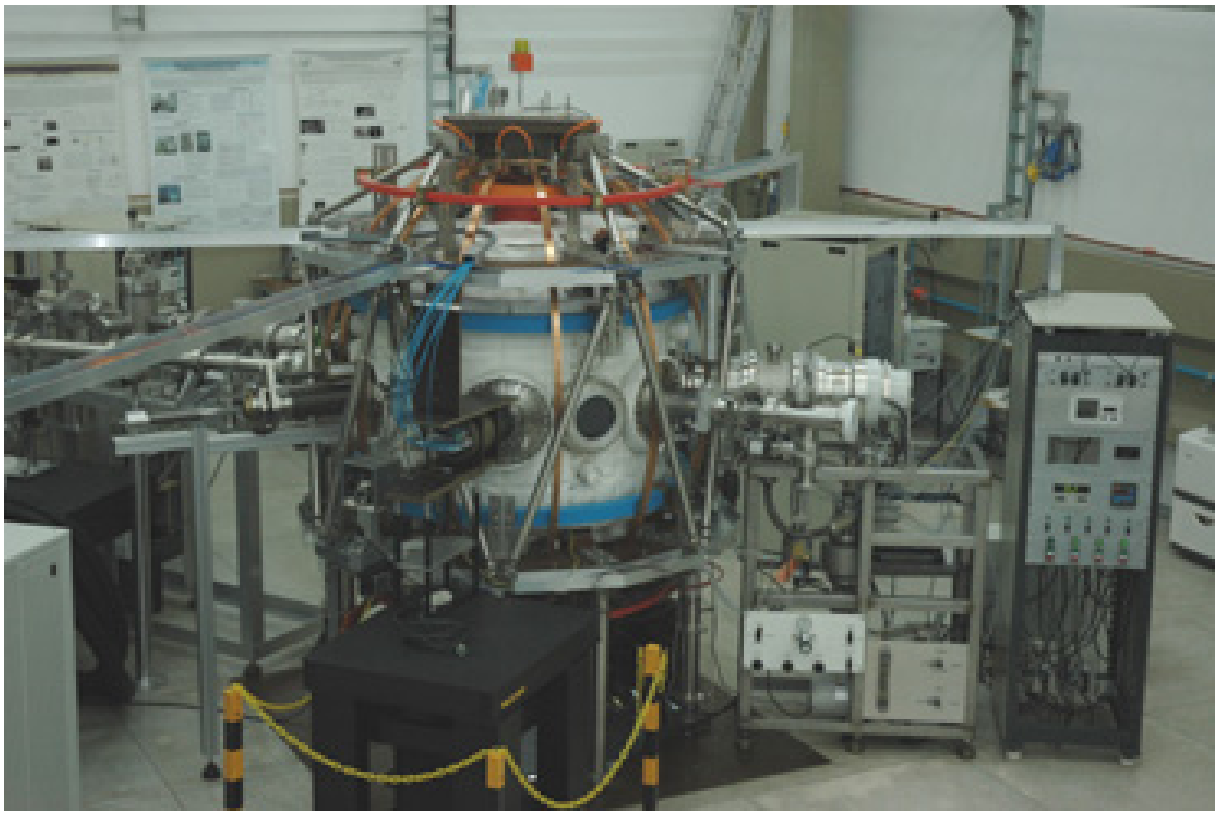
2.3 Códigos de onda completa

A modelagem de propagação de onda em plasma quente num campo magnético disforme é uma tarefa difícil. Uma das principais dificuldades em tal tarefa é obter um cálculo acurado da resposta dielétrica de um plasma linear em um campo RF oscilando numa determinada frequência. Pois, os cálculos acurados da cinética que relacionam a resposta dielétrica do plasma, envolvem soluções numéricas da equação Vlasov linearizadas na presença do campo RF. Tais simulações estão se tornando possíveis de serem feitas devido ao avanço na capacidade de efetuar muitos cálculos dos atuais e futuros computadores. Assim, um número de aproximações são normalmente feitos em códigos de onda completa ou *Full-Wave Codes*, para calcular a resposta dielétrica do plasma quente (SVIDZINSKI et al., 2016).

As aproximações feitas pelos códigos de onda completa normalmente incluem:

- a - Pequenos raios Larmor relativos ao comprimento de onda;

Figura 4 – Imagem do ETE, *Tokamak* brasileiro no o Laboratório Associado de Plasma do INPE.



Fonte: (INPE. . . , 2017)

- b - Desvio das órbitas de partículas, que desviam pouco de uma superfície magnética durante o tempo da contribuição dominante para a integral da órbita;
- c - São feitas simplificações significantes nas integrais da órbita;

Nas quais, a restrição do item (a) é validado para os harmônicos de baixo ciclotrônio quando se toma cuidado especial sobre o modo de ondas de Bernstein convertidas. A restrição do item (b) mantém, com acurácia razoável, devido ao numero relativamente pequeno de partículas que se afastam distante da superfície magnética durante o tempo mencionado. A restrição do item (c) é acurada apenas em um limitado domínio de números de onda (SVIDZINSKI et al., 2016).

Ao longo das últimas décadas, foram criados diversos códigos de onda completa que utilizam modelos numéricos tais como o TORIC (BRAMBILLA, 1999), o ALCYON (PETROV; MONAKHOV, 1999) ou o PENN (JAUN K. APPERT, 1995), estes em especial, possuem simplificações significativas nas integrais da órbita, estas simplificações são feitas para tornar o cálculo computacionalmente viável.

A simulação de onda completa de campo RF em Plasma quente com disformidades 3-D ainda possui uma limitação muito grande, mesmo utilizando programação

paralela massiva em supercomputadores. Entretanto, o código PSTELION é capais de resolver os campos RF de onda completa em uma geometria de um Stellarator utilizando modelagem RF 3-D na faixa de frequência ICRF, para tal, a paralelização do código foi feita utilizando a biblioteca ScaLAPACK, isto devido ao tamanho das matrizes envolvidas no cálculo, no entanto, ouve uma limitação significante no numero de harmônicas poloidais e toroidais que este código pode resolver (SVIDZINSKI et al., 2016).

Para verificar e avaliar a confiança nas simulações e nas previsões geradas, o projeto ITER utiliza *benchmarking* de códigos que simulam o aquecimento e a transmissão atual, os códigos utilizados nos *benchmarking*s são os AORSA, CYRANO, EVE, PSTELION, TASK/WM e TORIC (BUDNY et al., 2012) dos quais:

A equação de onda em cada ponto de grade espacial é formulada em termos de harmônicas de Fourier do campo elétrico de RF. O conjunto resultante de equações lineares para esses harmônicos de Fourier, com uma matriz densa, é então resolvido usando a biblioteca paralela ScaLAPACK11.

- AORSA - Resolve a representação integral da equação de onda, na geometria do *Tokamak* , usando o método de colocação, então, a equação da onda, a cada grade, é formulada em termos das harmônicas de Fourier do campo elétrico RF, resolvendo as matrizes resultantes com a biblioteca ScaLAPACK (SVIDZINSKI et al., 2016). Devido à limitações internas, no tempo em que o *benchmarking* foi efetuado, apenas seis especies puderam ser usados (JAEGER L. A. BERRY, 2001).

- CYRANO - O código CYRANO é similar ao TORIC. Esse usa elementos finitos na direção radial e também usa a representação de Fourier nas dimensões poloidais e toroidais, apenas tratando o equilíbrio "axisymétrico", assim, os modos toroidais são independentes. Além de resolver a equação na forma fraca variacional do formalismo Galerkin. O modelo de antena é idealizado como sendo cintos infinitamente finos na direção radial, possuindo correntes de antena homogêneas e inclui alimentadores radiais, foi modificado para incluir funções numéricas de distribuição geral para calcular a resposta dielétrica. No tempo em que o *benchmarking* foi efetuado, CYRANO não foi paralelizado (LAMALLE, 1994) e (AL., 2009).

- EVE - È baseado na formulação variacional do sistema Maxwell–Vlasov. A interação partícula onda é descrita como plasma quase local funcional, no qual uma expansão de primeira ordem de partícula hamiltoniana foi implementada, com o uso da formulação hamiltoniana associada as variáveis de ângulo de ação, dão ao EVE o título de principal elemento nos pacotes de onda adicionada à cinética. O código foi escrito em fortran 90 e foi paralelizado, sendo capais de rodar em

múltiplos clusters, seu pós processamento está escrito em Python (DUMONT, 2009).

- PSTELION - É o mais ressente da lista, foi desenvolvido para ser um código de onda completa para *stellarator* utilizando ICRF em 3D. O código resolve a excitação, propagação e absorvição da onda num *stellarator*. O problema do valor limite denominado Maxwell-Vlasov, na faixa de frequência ICRF, é resolvido em equilíbrio realista numa geometria de plasma toroidal alongada. O método numérico usa uma expansão da solução em séries de Fourier e resolve a equação resultante de forma radial por um método de diferencial finito (V.L., 2001).

- TASK/WM - Resolve as equações de Maxwell como problemas de valores de borda com coordenadas de fluxo magnético em uma configuração 3D(AL., 2007).

- TORIC - Resolve as equações de Maxwell em plasmas toroidais "axisymétricos", assumindo uma relação formativa, que é obtida através da equação Vlasov linearizada, expandindo o campo de onda em componentes tanto toroidais quanto poloidais, de Fourier(BRAMBILLA, 1999).

Um dos principais problemas do CYRANO é que este código é um dos que não possuem uma versão paralela. No entanto, paralelizar um código não é uma tarefa automática ou simples de ser realizada.

2.4 Programação de algoritmos paralelos

A lei de Moore, como ficou conhecida, afirma que a densidade dos transistores iria dobrar a cada 18 meses (MOORE, 1965). Apesar de isto significar uma melhoria na velocidade de clock dos processadores, existe uma limitação para o tamanho mínimo que a densidade de um transistor pode atingir. Esta limitação no tamanho dos transistores levaram as fabricantes de processadores a aumentar a quantidade de núcleos de um processador e utilizar uma memória compartilhada, o que significa que os múltiplos núcleos tem acesso compartilhado a memória global. Estas maquinas são conhecidas atualmente como *shared-memory multicore machines* ou maquinas com múltiplos processadores de memória compartilhada (LEISERSON; MIRMAN, 2008).

Uma arquitetura *multi-core* implica que esta terá no mínimo três aspectos relevantes, haver múltiplos núcleos de processamento, existir um modo no qual estes núcleos se comunicam e os do processador devem se comunicar com o mundo exterior (VAJDA, 2011).

Desde a chegada dos processadores dual-core, aproximadamente no início do século XXI, o número de cores vem aumentando a cada geração, e a chegada do copro-

cessador Xeon Phi introduziu um novo mundo de muitos cores. O Xeon Phi tem 61 cores com 4 threads cada, isto requere aplicações e APIs que explorem essa capacidade (XE, 2013).

Contudo, utilizar maquinas com multipolos núcleos não significa que todos os softwares exploram os núcleos disponíveis automaticamente. Para tal, na maioria das linguagens, é necessário reescrever o código fonte de modo a adaptar para o uso destes múltiplos núcleos.

Além disto, devido à latência do acesso à memória principal, os processadores modernos possuem pequenas memórias que suportam acessos de leitura e escrita muito mais rápidos. Estas memórias são conhecidas como memórias cache. Maquinas multi processadas podem ter múltiplos níveis de cache, cada um com diferentes tamanhos e velocidades de acessos, que podem ser compartilhadas ou utilizadas individualmente por cada núcleo de processamento. Assim, projetar algoritmos que fazem uso eficiente da memória cache pode aumentar significativamente o desempenho das aplicações.

Quando se pensa em paralelismo, ou algoritmos paralelos, lembramos da regra de três, onde para a pergunta "se com 1 trabalhador, a obra leva 10 dias para ser completada, com 5 trabalhadores, quantos dias são necessários para a obra acabar?" se responde "2 dias". Com computadores não é bem assim. Segundo a lei de Amdals (AMDAHL, 1967), existe um limite de ganho na redução de tempo, um *speedup*[1] máximo, em função das partes do código que não posem ser paralelizadas. No entanto, hoje em dia sabe-se que este limite depende da escalabilidade do problema, segundo (GUSTAFSON, 1988), ou seja, embora o limite de Amdals exista, ele muda conforme o tamanho da entrada e dos dados, do problema a ser resolvido.

O principal objetivo na computação paralela é desenvolver soluções que diminuem o tempo de computo na mesma proporção da quantidade de núcleos de processamento, ou seja, problemas escaláveis. Um dos desafios ao se escrever um código paralelo que atinja este objetivo, é tratar o acesso concorrente das linhas de execução aos mesmos dados, sendo que o acesso tem que possuir no mínimo, uma escrita no mesmo dado por linha de execução, ou seja, tratar as condições de corrida. As condições de corrida, impõem dificuldade em depurar o código podendo diminuir o desempenho ou mesmo fazer com que o código gere resultados completamente errados.

Uma das possibilidades de se obter códigos, cujo tempo de computo seja proporcional a quantidade de núcleos de processamento em programas paralelos, é não ter condições de corrida alguma, porém, enquanto que para alguns problemas seja possível evitar as condições de corrida, para outros, pode se tornar uma restrição, uma vez que é

[1] É a medida que quantifica o quanto um programa, em sua versão paralela, consegue ser mais rápido que sua versão serial, em função do tamanho do problema e da quantidade de processadores utilizados na versão paralela.

conveniente ter dados compartilhados num programa.

Além disto, embora existam problemas que podem ser divididos em tarefas que igualem a carga de processamento de acordo com os dados de entrada e a quantidade de núcleos disponíveis altamente escaláveis, ou seja, problemas escaláveis, existem os que não são escaláveis, cuja paralelização se torna impraticável.

A exemplo, um problema que em um dado eixo temporal, permita a avaliação explicita ou implícita de algum campo, em um dado instante de tempo, é chamado de problema temporal. Na maioria dos casos, os problemas temporais, possuem uma restrição de passo de tempo para assegurar a estabilidade do sistema. E mesmo para os que não possuem restrições de passo de tempo, geralmente não é possível calcular diretamente o dado no tempo final sem utilizar os resultados dos tempos anteriores. Deixando claro que qualquer método de integração temporal não possui propriedades paralelas no algoritmo. Em alguns casos se pode transformar um problema temporal de modo que a partir da equação original se produza uma simplificação que dependa apenas de um conjunto paramétrico, tornando o problema escalável com paralelismo (KOKULAN; LAI, 2012).

Embora muitos pesquisadores de diversas áreas da ciência utilizem o computador como ferramenta em seus estudos, devido à facilidade de executar as modelagens matemáticas para resolver problemas inerentes as pesquisas no computador. A tarefa de escrever um software que representa uma modelagem matemática, na maioria das vezes, não é trivial. Muitos dos códigos são desenvolvidos sem levar em consideração as técnicas de engenharia de software. Por este motivo, surgem dificuldades ao dar manutenção ou mesmo reescrever algumas partes destes códigos.

Além das maquinas multi-núcleos, a computação paralela pode ser obtida com sistemas distribuídos, onde múltiplas maquinas podem ser conectadas em rede e dividirem o trabalho, utilizando as placas gráficas com processamento de proposito geral, termo conhecido como GPGPU. Ha também a paralelização da memória, que pode ser adquirida em sistemas distribuídos, aumentando a quantidade de memória permitindo a computação de problemas mais complexos.

Mensurar a qualidade de uma aplicação paralelizada é difícil, principalmente porque a análise assintótica para aplicações paralelas é uma técnica ainda em desenvolvimento. Nem todos os algoritmos podem ser paralelizados, mas se um algoritmo pode ser, normalmente se diminui a quantidade de tempo (KOKULAN; LAI, 2012) gasto no cômputo. Entretanto, medir a eficiência de um algoritmo paralelo não é trivial, uma vez que analise assintótica não pode ser utilizadas em algoritmos paralelos. A principal métrica neste caso passa a ser o tempo de execução.

Uma forma de medir o quanto o tempo diminuiu, ou o quanto o algoritmo para-

lelo é proporcionalmente mais rápido que o serial, é calculando o chamado "*speedup*". Que pode ser calculado por meio da equação $S(n,p) = \frac{T_p(n,p)}{T_s(n)}$, onde S é o *speedup*, T_s é o tempo do serial, T_p é o tempo do paralelo, n é o tamanho do problema e p a quantidade de processadores utilizados.

Os computadores hoje em dia possuem no mínimo dois núcleos de processamento, e podem possuir placas gráficas que possuem muitos núcleos de processamento sendo verdadeiros supercomputadores, mas, para aproveitar estes hardwares, um software tem que ser escrito de forma a se adequar a esta grande quantidade de processadores, para isto existem APIs (*Application Programming Interface*) ou interface de programação para aplicações como OpenMP (DAGUM; MENON, 1998). O uso de placas gráficas para programação de proposito geral é chamado de GPGPU (*general purpose graphics processing unit*) e às duas APIs mais conhecidas são CUDA (MOROZOV; MATHUR, 2012) e OpenCL (STONE DAVID GOHARA, 2010).

2.4.1 OpenMP

Das diversas APIs que auxiliam no processo de paralelização de algoritmos em linguagens imperativas, como C/C++ e Fortran, destacam-se aquelas que implementam o padrão OpenMP. Desde sua aparição em 1997, o OpenMP emergiu como um padrão para programação paralela de memória compartilhada. Seu uso se tornou popular nas áreas técnicas e cientificas, por ter suporte a muitas linguagens, incluindo C/C++ e Fortran, ser um padrão aberto e pela facilidade de implementação de códigos baseados nele (XE, 2013).

OpenMP consiste em um conjunto de diretivas, que descrevem como o código deve ser paralelizado pelo compilador, e também é composto por uma API e variáveis de ambiente para controlar o comportamento do código. Estas diretivas são pragmas em C/C++ e comentários especiais em Fortran, o que significa que para compilar o código sem o OpenMP, para que ele execute de forma serial, basta dizer para o compilador ignorar os pragmas e comentários, não havendo necessidade de modificar o programa (XE, 2013).

Muitos pesquisadores assumem que a única forma de conseguir escalabilidade em software paralelo é com o modelo de passagem de memória, mas isto não é uma verdade absoluta. Atualmente os processadores possuem suporte a memória cache correlacionada. Estes são conhecidos como arquitetura de multiprocessadores de memória compartilhada escalável ou (scalable shaved memory multiprocess SSMP). Nestes sistemas o modelo é de memória compartilhada e o modelo de memória distribuída é construído em cima do modelo de memória compartilhada com o qual a escalabilidade esta diretamente relacionada. (DAGUM; MENON, 1998).

O modelo de memória compartilhada é construído em cima de um único processo, que pode se dividir em múltiplas linhas de execução, também conhecidas como *threads*. Todas as *threads* de um mesmo processo tem acesso ao espaço de memória do mesmo, ou seja, podem ler e escrever em qualquer espaço de memória compartilhada. Nestes casos o programador pode declarar variáveis privadas, que são espaços de memória disponíveis apenas para *threads* específicas e só elas têm acesso aos mesmos (GROPP; E, 2008).

OpenMP foi criado com o objetivo de tornar fácil e portável para as diversas linguagens e plataformas, já que antes de sua criação os distribuidores de sistemas de arquitetura de memória distribuída haviam criados suas próprias extensões de desenvolvimento para Fortran e C. Como não havia portabilidade, muitos desenvolvedores preferiam modelos de passagem de memória como *Message Passing Interface* (MPI) ou *Parallel Virtual Machine* (PVM) (DAGUM; MENON, 1998).

Projetado para explorar as características das arquiteturas de memória compartilhadas, OpenMP possui a habilidade de acessar a memória diretamente pelo sistema (com o minimo de latência e nenhum mapeamento explicito de memória). O OpenMP também permite aos desenvolvedores que precisam paralelizar um código já existente, atingir seus objetivos com pouquíssimas modificações no código original, o que não é possível na maioria dos outros padrões de linguagens (GROPP; E, 2008).

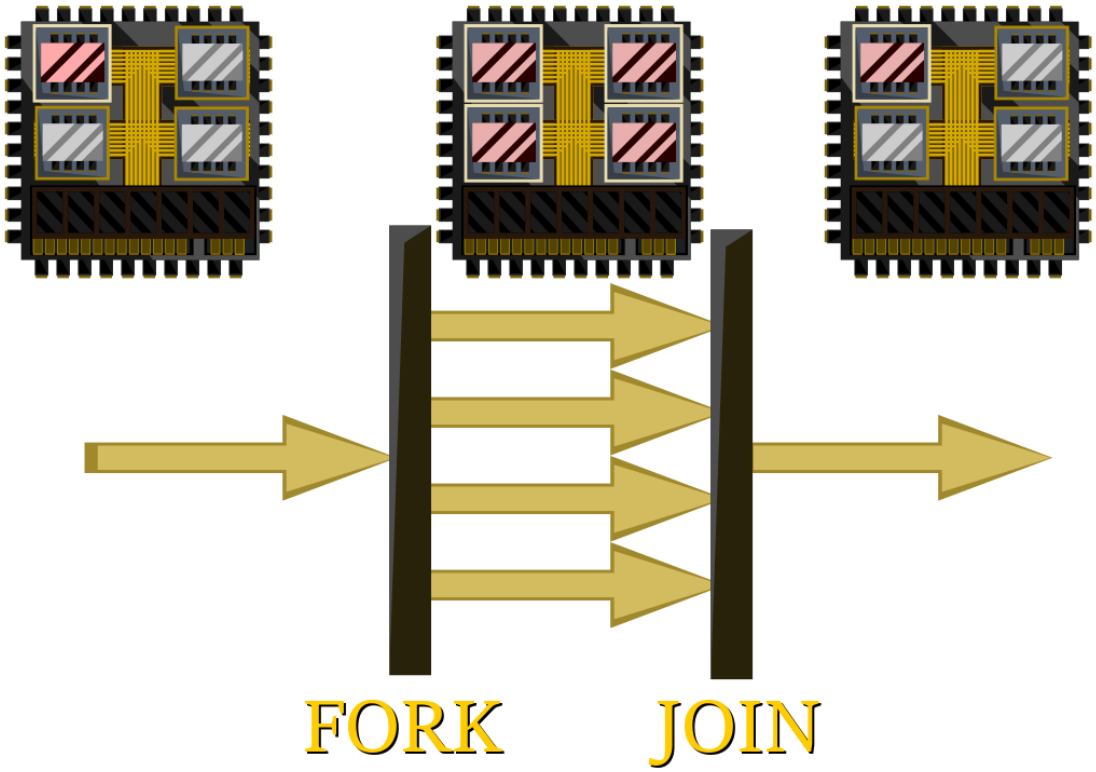
No nível mais elementar, OpenMP é um conjunto de diretivas de compilação e rotinas que estendem Fortran e separadamente C/C++ que expressam paralelismo de memória compartilhada. Isto permite deixar a linguagem básica sem especificações, desta forma os distribuidores podem implementar OpenMP em qualquer compilador Fortran. Naturalmente, para suportar ponteiros e alocáveis, Fortran 90 e Fortran 95 é necessário que a implementação inclua semânticas adicionais em Fortran 77.

OpenMP utiliza a abordagem de paralelismo do tipo "*fork-join*", que consiste em separar (*fork*) os procedimentos que podem ser executados em paralelo e juntar (*join*) ou sincronizar os dados e os procedimentos em alguns determinados pontos da execução, idem a ilustração da Figura 5. Onde os construtores de junção e separação podem ser aninhados, o que torna este tipo de paralelismo muito útil para algorítimos do tipo dividir e conquistar.

Apesar de efetivo e simples, paralelismo de laços é usualmente limitado em escalabilidade, por que deixa alguma fração constante da parte sequencial do programa, o que pela lei de Amdahl pode rapidamente reter os ganhos da execução paralela (AMDAHL, 1967).

Em uma comparação realizada com aplicações de física irregulares, que aderem ao modo de computação SPMD (Single Process Multiple Data), como os encontrados

Figura 5 – Representação figurativa do termo *Fork-Join*.



Fonte: elaborada pelo autor.

em sistemas de partículas de fluidos acoplados, OpenMP mostrou que sua flexibilidade pode ter uma grande vantagem no desempenho sobre MPI (AMRITKAR et al., 2012).

Apesar das versões paralelas, em geral, efetuarem os cálculos em menos tempo do que as versões seriais, os resultados podem apresentar algumas diferenças quando envolvem cálculos complexos com variáveis de ponto flutuante, de precisão simples ou dupla. Isto ocorre, muitas vezes, devido à aritmética de ponto flutuante ser diferente da aritmética matemática em alguns aspectos.

2.4.2 Aritmética de ponto flutuante

Simular um conjunto contínuo infinito (neste caso, os números reais) com um conjunto finito (os "números de máquina") não é uma tarefa simples. Pensando no computador, para se obter uma conversão de um conjunto infinito a um conjunto finito, os números convertidos para a maquina, devem oferecer velocidade, precisão, acurácia, alcance dinâmico, facilidade de uso e implementação e custo de memória baixo. Desde o começo da era computacional, muitas formas de aproximar os números reais nos computadores foram criadas, dentre elas aritmética de ponto fixo, logarítmica e semi logarítmica, sistema numérico, frações continuadas, números racionais, sistemas de nú-

meros de níveis indexados, sistema de número de barra fixa e barra flutuante, números "2-adic" e ate mesmo "strings" de números infinitos. No entanto, a aritmética de ponto flutuante é de longe o método de representação mais utilizado nos computadores da atualidade para representar os números reais (MULLER et al., 2010).

Desta forma, é importante ressaltar que utilizando a aritmética de ponto flutuante os computadores não conseguem representar todos os números reais. Isto acontece devido à limitação da memória, que apesar de na atualidade existir em grande escala, ainda tem um limite físico de dados que ela consegue suportar. Ainda assim, o sistema de ponto flutuante consegue representar uma grande quantidade de números.

No padrão IEEE 754-1985 (e em outros padrões também), uma exceção pode ser sinalizada com o resultado de uma operação, esta exceção pode ser uma "flag" de status e/ou algum mecanismo de armadilha (MULLER et al., 2010), ela pode sinalizar:

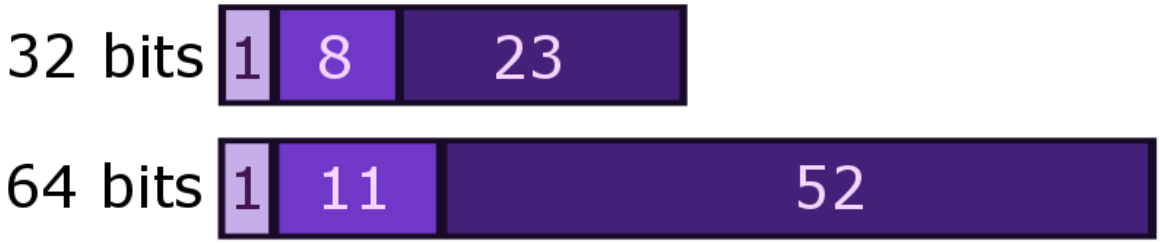
- **Invalid:** sinalizada quando uma entrada é invalida para a função, neste caso o resultado é NaN (*Not A Number* ou não é um número) quando suportado. Exemplo $(+\infty) - (-\infty)$, $\frac{0}{0}$, $\sqrt{-1}$;
- **Overflow:** sinalizada quando o resultado arredondado possui o expoente maior que e_{max};
- **Underflow:** sinalizada quando um resultado pequeno (menor que e_{min}) e não zero é detectado. Isto pode ser quando ocorre, ou antes do arrendondamento ou depois do arrendondamento;
- **Inexact:** sinalizada quando o resultado de y não é exatamente representável ($arredonda(y) \neq y$, y não sendo NaN);

Atualmente, na representação de ponto flutuante os números são representados no computador de forma binaria em sequências de, normalmente. 32 (precisão simples) ou 64 (precisão dupla) bites.

A exemplo, sabendo que o padrão adotado no computador segue o IEEE 754 (revisado em 2008), que especifica que os dados binários do tipo ponto flutuante (float) sejam divididos em 3 campos, o sinal, o expoente e a mantissa, e também definem os formatos conhecidos como float, e double de 32 e 64 bits respectivamente como sendo, 1 bit para o sinal em ambos os casos, 8 e 11 bites para o expoente e 23 e 52 para a mantissa, como na Figura 6. O padrão também define como as operações são executadas e como se comportam os bits para representar zero, menos infinito, mais infinito e NaN, dentre outras coisas (WHITEHEAD; FIT-FLOREA, 2011).

Apesar de ser amplamente aceito, este padrão introduz algumas limitações, como a perda de propriedades importantes, em alguns casos. Temos, por exemplo,

Figura 6 – Representação binaria de ponto flutuante.



Fonte: elaborada pelo autor.

situações de cálculos com aritmética de ponto flutuante de precisão simples nos quais não somente $(a + b) + c$ é diferente de $a + (b + c)$ no valor real, como também são diferentes um do outro, o que mostra que as operações aritméticas com ponto flutuante não são 100% precisas (WHITEHEAD; FIT-FLOREA, 2011).

Como os códigos mais complexos executam milhares de cálculos, este erro pequeno pode se propagar nos cálculos posteriores, fazendo com que os resultados divirjam de operações realizadas em ordens diferentes. O que não significa que estão errados, apenas que possuem um erro esperado devido a essa limitação.

2.5 BLAS (Basic Linear Algebra Subprograms)

Muitos dos modelos computacionais dependem da álgebra linear, que é uma poderosa ferramenta matemática para resolver diversos problemas. Em 1973 foi escrito um artigo no boletim de notícias SIGNUM que descrevia as vantagens de adotar um conjunto básico de rotinas para resolver problemas de álgebra linear em computadores (HANSON F. T. KROGH, 1973).

As sub-rotinas básicas de álgebra linear (*Basic Linear Algebra Subprograms*) que ficaram mais conhecidas pelas siglas em inglês BLAS, foram criadas originalmente na década de 70. Este conjunto de sub rotinas ficaram bem conhecidas ate os tempos atuais, sendo utilizadas em vários softwares, como as bibliotecas LIMPACK e muitos algoritmos publicados pelas transações ACM em software matemático (DONGARRA et al., 1985).

A BLAS é um conjunto de rotinas que realiza operações básicas com matrizes e vetores. Este conjunto de sub-rotinas foram desenvolvidas inicialmente pelos pesquisadores Chuck Lawson e Dick Hanson, no *Jet Propulsion Laboratory* JPL, inicialmente escrita em Fortran com versões também em linguagem Assembly para o IBM 360/67, o CDC 660, o CDC 7600 e o Univac 1108. Em seguida, em maio de 1974, na "*Matematical Software II conference, Purdue University*" e em maio de 1975 na "*National Computer Conference, Anahein*" ocorreram os primeiros encontros abertos do projeto BLAS, estes encontros, que contaram com a participação de 30 pessoas, resultaram em extensivas modificações das especificações iniciais e mudanças adicionais do projeto. Estes encontros tiveram o objetivo de melhorar o design e fazer rotinas mais robustas no código em FORTRAN da BLAS (LAWSON et al., 1979).

Na época do desenvolvimento da BLAS, quando os designers de software tinham que utilizar álgebra, era necessário fazer suas próprias bibliotecas, pois, códigos que envolviam álgebra linear não tinham um padrão estabelecido.

Assim, a BLAS permitiria uma diminuição no tempo de desenvolvimento de softwares que utilizam álgebra linear, pois, o desenvolvedor não perderia tempo implementando aplicações que já existem. E também existiria uma certa portabilidade, de forma que, se uma implementação mais eficiente da BLAS fosse criada, bastaria trocar a biblioteca sem reescrever uma linha de código do software que a utiliza. E já que um tempo significante da execução em programas complicados de álgebra linear devem ser gastos em operações para solucioná-las, uma redução no tempo de execução que é gasto nestas operações, irá diminuir a carga computacional reduzindo o custo para rodar os programas que utilizam a BLAS(LAWSON et al., 1979).

2.5.1 Descrição

Escrita em Fortran, atualmente quase todas as linguagens podem utilizar a BLAS e suas versões otimizadas. É atualmente mantida pelo grupo netlib (NETLIB, 2017) e é comumente usada por sua eficiência, portabilidade, e viabilidade.

A BLAS divide as suas rotinas em 3 níveis. Os níveis definem a relação matriz/vetor e são clasificados pela complexidade assintótica ($O(n)$,$O(n^2)$,$O(n^3)$) das operações envolvidas, divididos da seguinte forma:

- Nível 1 - Realiza operações com escalar, vetor e vetor-vetor;
- Nível 2 - Realiza operações com escalar, vetor-matriz;
- Nível 3 - Realiza operações com escalar, matriz-matriz;

Os nomes das funções da BLAS representam suas operações, a exemplo, a função "axpy" executa a operação $Y = aX + Y$ (*ax plus y*), sendo Y e X vetores e a um escalar. A primeira letra da função representa o tipo de dado utilizado na rotina, podendo ser:

- s para ponto flutuante de precisão simples;
- d para ponto flutuante de precisão dupla;
- c para complexos de precisão simples;
- z para complexos de precisão dupla;

Assim, para multiplicar um vetor por um escalar de precisão dupla a função é "daxpy". Vale lembrar que os nomes das funções não utilizam mais que 6 caracteres, devido à limitação de nome de variáveis na época da criação do código.

2.5.2 Otimizações da BLAS

Em várias aplicações, o desempenho das operações de álgebra linear é o que mais impede a modelagem de problemas mais complexos. Complexos, no sentido que expressam mais precisamente a realidade. No entanto, a álgebra linear é rica em operações que são altamente aperfeiçoáveis e paralelizáveis, assim, um código altamente adaptado e/ou paralelizado pode rodar muitas vezes mais rápido do que um código genérico (WHALEY; DONGARRA, 1997).

Atualmente existem diversas implementações de bibliotecas que seguem o padrão da BLAS, estas implementações são desenvolvidas com base em arquitetura específicas. Este é o caso da ATLAS (WHALEY; DONGARRA, 1997), OpenBLAS (OANCEA,

2015) e BLIS (WILLENBRING; LABORATORIES, 2015) que focam na otimização do código com base na utilização dos recursos disponíveis nas CPUs comumente usadas nos computadores pessoais (PCs). Estas implementações conseguem aumentar o desempenho utilizando recursos que conseguem executar de forma paralela em memória distribuída e compartilhada nos modernos processadores que possuem múltiplos núcleos (multi cores). Para este fim estas bibliotecas utilizam recursos como OpenMP ou Pthreads (WHALEY; DONGARRA, 1997). Podem se destacar também a ScaLAPACK que visa implementar as rotinas da BLAS em memória distribuída e hibrida, utilizando as biblióticas MPI e OpenMP. Pode se mencionar Também as implementações a CUBLAS e CLBLAS para GPGPUs que disponibilizam as sub rotinas da BLAS para serem executadas em arquiteturas baseadas em GPGPUs.

A ATLAS, OpenBLAS, e a BLIS são bibliotecas que tentam encontrar o código binário mais eficiente para uma arquitetura específica. Para tal, vários executáveis são testados e o mais rápido é escolhido esta técnica é chamada de auto-tuning (KELEFOURAS et al., 2014). Como essas bibliotecas são melhorias da BLAS e portanto, usam os protótipos de suas funções, elas podem ser substituídas e acordo com a necessidade e disponibilidade de hardware de quem usa a BLAS, sem muito esforço.

2.5.3 Uso da BLAS no CYRANO

Para o CYRANO foi trabalhada as implementações BLAS, ATLAS, OpenBLAS e BLIS. A função GENERA do código CYRANO, faz o uso intensivo das rotinas da BLAS. Principalmente da ZGEMM e ZAXPY. Com o intuito de substituir pela melhor versão otimizada da BLAS testes de performance foram executados em três bibliotecas otimizadas, a ATLAS (WHALEY; DONGARRA, 1997), a OpenBLAS (OANCEA, 2015) e a BLIS (WILLENBRING; LABORATORIES, 2015), com a finalidade de mensurar a performance destas bibliotecas.

2.5.4 Calculando FLOPS

Um número complexo é composto de uma parte real e uma parte imaginaria, onde $a + bi; \forall a, b \in \mathbb{R}$, se somarmos dois números imaginários temos:

$c = x + yi$, $d = u + vi$ então $c + d = (x + u) + (y + v)i, \forall c, d \in \mathbb{C}, x, y, u, v \in \mathbb{R}$

Assim, $c + d$ são duas somas, ou seja, duas operações de ponto flutuante. Se multiplicarmos dois números complexos temos:

$$c * d = (x + yi) \times (u + vi) = (xu - yv) + (xv + yv)i$$

Então $c \times d$ são 4 multiplicações, 1 soma e 1 subtração de números reais, ou seja, para computar $c + d$ o computador tem que realizar duas operações de soma, e para computar $c \times d$ quatro operações de multiplicações, uma soma e uma subtração.

O computador para realizar operações de subtração, utiliza o complemento do número negativo para representa-lo, pois, nessa abordagem a soma de números positivos e negativos podem ser efetuadas normalmente como a soma de dois números positivos.

Considere A, B e C como matrizes de tamanho $A_r \times A_c$, $B_r \times B_c$ e $C_r \times C_c$ respectivamente. GEMM calcula $C = \alpha C + \beta(AB)$ sendo A, B, C matrizes de números reais, e α, β sendo escalares. Para cada elemento de C calculado são A_c de multiplicações entre A e B mais 1 por αC mais 1 por $\beta(AB)$, além de $B_r - 1$ somas mais 1 por $\alpha C + \beta(AB)$ que dão:

$(A_c + 2) \times (C_c \times C_r)$ multiplicações mais:

$B_r \times (C_c \times C_r)$ somas.

Se considerarmos soma e multiplicação como operações e considerarmos $A_c = B_r, A_r = C_r$ e $B_c = C_c$ temos:

$(B_r+2)\times(C_c\times C_r)+B_r\times(C_c\times C_r) = (B_r+2+B_r)(C_c\times C_r) = (2B_r+2)\times(C_c\times C_r)$ operações por GEMM.

Para matrizes quadradas de tamanho N, temos:

$O(N) = (2N + 2) \times N^2$ operações.

ZGEMM calcula $C = \alpha C + \beta(AB)$ para matrizes de números complexos, cada soma de número complexos são duas somas de números reais e cada multiplicações de números complexos são quatro multiplicações e duas somas. Então temos:

$[(A_c + 2) \times 4 \times (C_c \times C_r)]$ multiplicações mais:

$[(A_c + 2) \times 2] \times (C_c \times C_r)$ somas das multiplicações mais:

$(B_r \times 2) \times (C_c \times C_r)$ somas.

Isto de números reais. Considerando multiplicações e somas como operações temos:

$[(B_r + 2) \times 4 \times (C_c \times C_r)] + [(B_r + 2) \times 2] \times (C_c \times C_r) + (B_r \times 2) \times (C_c \times C_r)$ $= [(B_r+2)\times 4 + (B_r+2)\times 2 + B_r \times 2] \times (C_c \times C_r) = [(B_r+2)\times 6 + (B_r \times 2)] \times (C_c \times C_r)$ $= [(B_r + 2) \times 7] \times (C_c \times C_r)$ operações por ZGEMM.

Para matrizes quadradas de tamanho N, temos:

$O(N) = (N + 2) \times 7N^2$ operações.

A quantidade de operações de ponto flutuante realizadas durante a chamada a uma sub-rotina ZGEMM pode ser utilizada então para determinar o desempenho da mesma. De forma geral o desempenho pode ser medido em FLOPS, acrônimo de (*floating point operations per second*) ou operações de ponto flutuante por segundo. Para

se obter a quantidade de FLOPS das funções DGEMM e ZGEMM, pode ser utilizada a seguinte expressão:

$$FLOPS = \frac{O(N)}{T} \tag{1}$$

Onde T é tempo gasto e $O(N)$ é a quantidade de operações realizada pela função. Os resultados obtidos em computadores atuais são da ordem de 10^9 FLOPS ou seja GFLOPS (gigaflops). Basta então multiplicar o resultado de 1 por 10^{-9} para obtermos o resultado em GFLOPS.

3 CYRANO

O código CYRANO, acrônimo de (*CYclotron Resonance Analysis with No Obfuscation* ou análise de ressonância ciclotrônica sem ofuscação em português) foi escrito por Philipe Lamalle em sua tese de doutorado (LAMALLE, 1994), tem como objetivo, analisar e modelar o comportamento de plasmas quando expostos à radiação RF, principalmente em Tokamaks, auxiliando em pesquisas relacionadas a fusão nuclear.

Para satisfazer as características do problema e resolver as equações completas de Maxwell dentro do confinamento de um *Tokamak*, foi utilizada uma das muitas abordagem para resolver o problema de elementos finitos, cuja abordagem escolhida é conhecida como método Galerkin (LAMALLE, 1994). Desta forma o problema de resolver a equação diferencial ordinária se resume em resolver um sistema linear.

3.1 Modelagem

O CYRANO modela a equação de Maxwell-Vaslov em uma configuração toroidal de plasma com seção transversal poloidal na faixa de frequência ciclotrópica iônica utilizando a formulação de Galerkin. Esta formulação é explicada com mais detalhes em (LAMALLE., 1997). De forma simples a formulação pode ser representada através da formulação fraca:

$$W(\boldsymbol{F},\boldsymbol{E}) + iR(\boldsymbol{F},\boldsymbol{E}) = A(\boldsymbol{F},\boldsymbol{j}s); \tag{2}$$

onde:

- W representa a contribuição do plasma e a contribuição parcial de cada partícula;

$$W(\boldsymbol{F},\boldsymbol{E}) = \sum_{\beta}\left(\frac{q_\beta}{2}\int_V dr^3\int dv^3 f_\beta \boldsymbol{F}^* \cdot \boldsymbol{v}\right) \tag{3}$$

- R representa a contribuição do operador de onda no vácuo;

$$R(\boldsymbol{F},\boldsymbol{E}) = \frac{1}{2}\int_V\left[\frac{1}{\omega\mu_0}\nabla\times\boldsymbol{F}^*\cdot\nabla\times\boldsymbol{E} - \omega\varepsilon_0\boldsymbol{F}^*\cdot\boldsymbol{E}\right]dr^3 \tag{4}$$

- A representa a contribuição das fontes do sistema (Antena RF);

$$A(\boldsymbol{F},\boldsymbol{j}s) = -\frac{1}{2}\int_V \boldsymbol{F}^*\cdot\boldsymbol{j}_s dr^3 \tag{5}$$

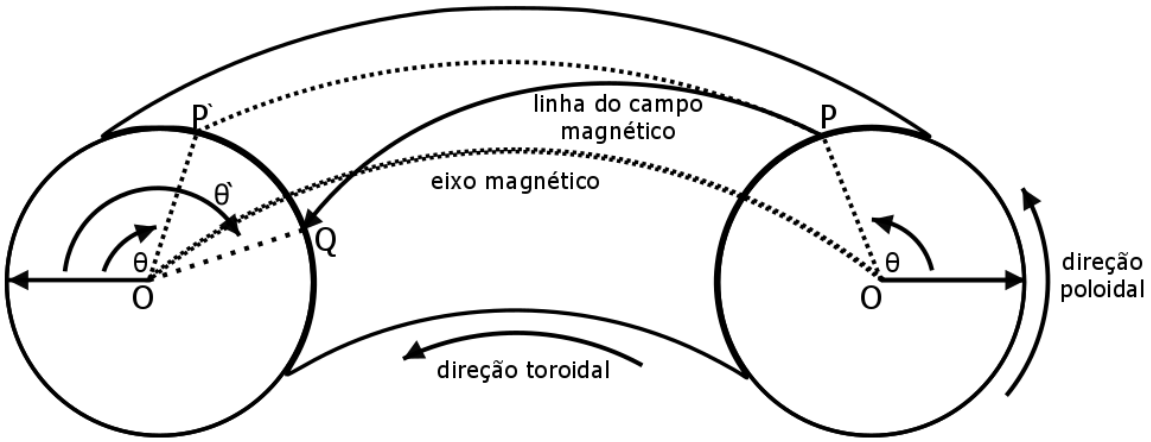
Sendo que:

- $\boldsymbol{E}$ é o vetor do campo elétrico $\boldsymbol{E}(\boldsymbol{r})$;
- $\boldsymbol{F}$ é o conjugado complexo de um arbitrário, porém, suficientemente bem-comportado, campo vetorial $\boldsymbol{F}(\boldsymbol{r})$;
- $\boldsymbol{J_S}(\boldsymbol{r})$ é a densidade da corrente da fonte RF;
- ω é a frequência angular da fonte RF;
- β é a indexação das diferentes espécies de partícula, cuja carga é q_β e a massa é m_β;
- ε é a permissividade dielétrica do plasma;
- μ_0 é a permeabilidade magnética do vácuo;

O método Galerkin permite converter problemas de valores de contorno em um sistema de equações lineares que podem ser resolvidos por métodos como o método de Gauss ou fatoração LU.

O domínio, na parte da geometria, é composto de camadas toroidais aninhadas, como na figura 7. As regiões elementais são subseções ao longo do raio na direção poloidal. Estas regiões são decompostas em sub-regiões com propriedades numéricas especificas e cada Sub-Região é cortada radialmente em elementos finitos (LAMALLE, 1994). Em *Tokamaks* maiores a seção transversal costuma ser de formato em D, por questões de estabilidade, como na figura 8.

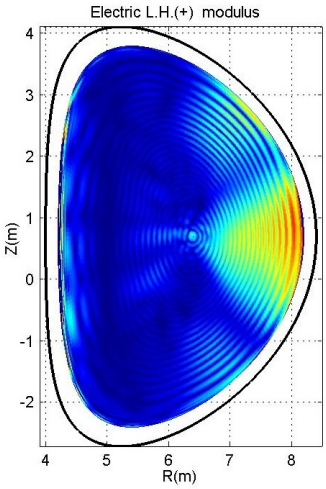
Figura 7 – Visualização das orientações poloidais e toroidais em um *Tokamak*.



Fonte: elaborada pelo autor.

Com o objetivo de fornecer um conjunto de códigos que podem analisar as descargas de plasma no projeto ITER de forma flexível, modular e integrada, o *EUROfusion*

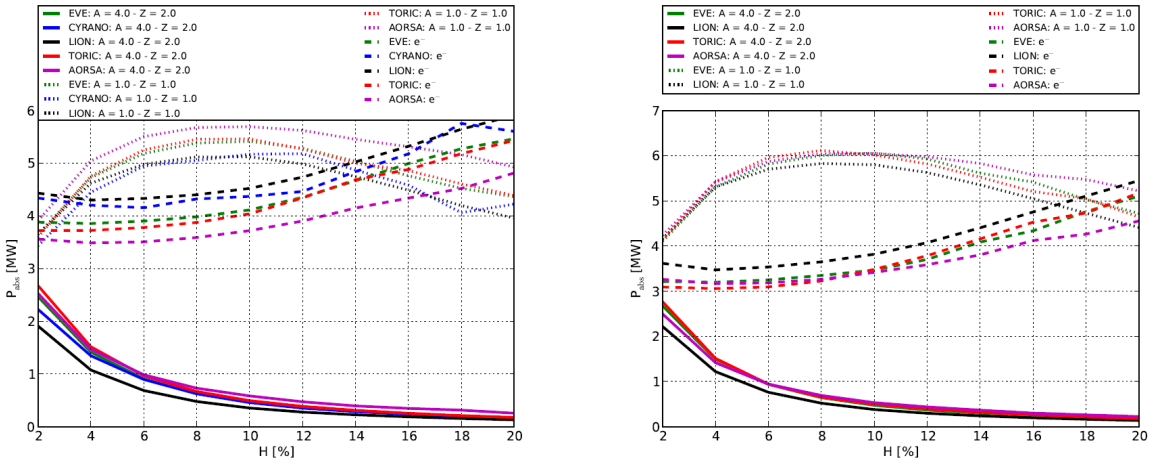
Figura 8 – Imagem gerada a partir de dados do CYRANO, que demonstra uma visualização transversal no toroide de um *tokamak*.



Fonte: elaborada pelo autor.

Code Development for Integrated Modelling project (WPCD) atualmente conta com quatro códigos de ondas completas, sendo estes os códigos chamados de CYRANO, EVE, LION e TORIC (R. Bilato, N. Bertelli, M. Brambilla, R. Dumont, E.F. Jaeger, T. Johnson, E. Lerche, O. Sauterk; VILLARD, 2015). A Figura 9 mostra uma comparação dos códigos de onda completa utilizados pelo WPCD nos testes de performance e validação dos quatro códigos, além do AORSA no artigo (R. Bilato, N. Bertelli, M. Brambilla, R. Dumont, E.F. Jaeger, T. Johnson, E. Lerche, O. Sauterk; VILLARD, 2015).

Figura 9 – Repartição de energia para as primeiras fatias de dez minutos dos casos circulares 40/1 (esquerda) e elíptica 40/11 (direita).



Fonte: (R. Bilato, N. Bertelli, M. Brambilla, R. Dumont, E.F. Jaeger, T. Johnson, E. Lerche, O. Sauterk; VILLARD, 2015).

3.2 O Código fonte

O código fonte está escrito em Fortran 70, com alguns poucos elementos de Fortran 90, mas não possui memória dinâmica, sendo que para executar é necessário modificar, ainda no código fonte, os valores de tamanhos das matrizes e vetores envolvidos em todos os processos de cálculo efetuados pelo software, mais precisamente no arquivo "pardin.copy".

Para resolver os cálculos com maior granularidade nos dados, o CYRANO consome muito tempo de processamento e memória, sendo que para alguns casos, o tempo demandado e a memória consumida são muito grandes. Neste contexto, para diminuir o tempo de computo, é necessário utilizar técnicas que permitam o aperfeiçoamento do código, como utilizar o paralelismo.

O código possui 236 arquivos com terminação ".f", abrigando as porções principais do código fonte FORTAN, 64 arquivos com terminação ".copy", geralmente com a especificação de valores de flags e tamanhos máximos de vetores. O arquivo principal do código, (responsável por iniciar e terminar o programa) encontra-se no arquivo "Cyrano.f". As variáveis podem então, estarem definidas blocos chamados de COMMONS e/ou NAMLIST, os definidos como NAMLIST são os que serão lidos do arquivo de entrada, enquanto que os definidos como COMMONS irão compartilhar a memória nas diferentes funções que os incluírem.

São 1126 variáveis definidas nos arquivos ".copy" das quais 257 estão definidas como NAMLIST e 884 no grupo dos COMMONS. Estar definida no grupo COMMONS significa que a variável tem escopo global no código, devido a grande quantidade de funções é difícil mapear os valores destas variáveis, a exemplo, a variável "ithoma", que é um iterador (variável que tem um valor específico por iteração dentro de um laço), é inclusa em "genera3_" e em "regjum", em "genera3_" começa com o valor do primeiro elemento 1 e é incrementado a cada iteração, em regjum (função que é chamada dentro da iteração em genera3) ele é incrementado dependendo da região. Ao chamar "ithoma" dentro de uma rotina que modifica o valor, é difícil mapear o valor de ithoma na iteração individual dentro de genera. Mapear o valor de iteradores por iteração individual é um dos processos que auxiliam na paralelização de loops, uma vez que cada thread é executada em ordem diferente. Como "ithoma" depende da iteração e das chamadas de funções, não é fácil mapear o valor desta variável.

O arquivo fonte "cyrano.f" inicialmente carrega os dados das variáveis para a memória para então chamar explicitamente as rotinas "iset", "int_to_string2", "longeurs", "dset", "zset", "fbini", "intbfp", "autcut", "cutec", "longeurs", "presol", "dfftci", "dfftri", "tables", "outgrid", "outtab", "groots", "filant", "anjump", "genera3", "fbtclo", "rdsol", "outdis", "qufetc2", "outrff", "outpow", "outrf2" e "outbit". Rotinas que começão com

"out" no código, são rotinas que salvam dados em arquivo.

- iset - Rotina da linpack, coloca todos os valores de um vetor de "integer" para um determinado valor passado por parâmetro;
- int_to_string2 - entra com um dado do tipo "integer" e retorna uma "string" de até 3 caracteres com a representação numérica do "integer" com zeros a esquerda ou "XXX" caso seja maior que 999;
- longeurs - Constrói o tamanho de intervalo para a abcissa;
- dset - Rotina da linpack, coloca todos os valores de um vetor de "double" para um determinado valor passado por parâmetro;
- zset - Rotina da linpack, coloca todos os valores de um vetor de "double complex" para um determinado valor passado por parâmetro;
- fbtini - Inicializa o bloco tri diagonal frontal;
- intbfp - Calcula a integral analítica dos produtos de duas básicas funções pela potência da variável reduzida "ksi" no intervalo $[0, 1]$;
- autcut - gera uma malha miníma de acordo com a relação da dispersão local;
- cuteq - salva num vetor N valores que variam de X0 a X1;
- presol - salva em algumas variáveis globais alguns indexes de vetores;
- dfftci - Rotina da IMSL, calcula os parâmetros necessários para as rotinas FFTCF (que calcula os coeficientes de Fourier de uma sequência periódica de números complexos) e FFTCB (que calcula a sequência periódica de números complexos de seus coeficientes complexos);
- tables - Gera as tabelas de perfis de equilíbrio;
- outgrid - Constrói a grade radial e poloidal para todas as rotinas de saída bidimensional;
- outtab - Salva em arquivo as tabelas de equilíbrio;
- groots - Gera raízes da dispersão local nos, nos de elementos num plano Equatorial, exceto no eixo magnético;
- filant - Preenche o vetor de excitação modular de acordo com a configuração poloidal da antena;
- anjump - Calcula os saltos na antena para todos os modos "dofs";

- genera3 - Responsável por montar e resolver o problema GALERKIN. Gera o problema de elemento finito, Nesta rotina existe um loop para interagir com a abstração do que seria cada elemento poloidal do plasma no qual para cada elemento fbstep é chamado para efetuar uma fatoração lu, e para tal, cada elemento tem a rotina da BLAS ZGEMM chamada três vezes dentro de fbstep. Além disto, no primeiro elemento apenas, ZAXPY é chamada para uma parte do codigo que calcula gaus.
- fbtclo - Encerra a resolução f.b.t;
- rdsol - Carrega para a memória a solução do bloco IBLO da solução do problema IPRO e armazena na matriz complexa X;
- outdis_ern - Constrói as tabelas de dispersão para serem plotadas em qualquer plotador;
- qufetc2 - Calcula as formas residuais e quadráticas para cada elemento usando a solução linear;
- outrff - Constrói as tabelas de saída para serem plotadas em qualquer plotador;
- outpow - Constrói as tabelas unidimensionais relacionadas com a força do sistema, para serem plotadas em qualquer plotador. Nesta rotina há um cálculo de potência em algumas variáveis, dentro de quatro um loops, com cálculos que são repetidos dentro dos loops desnecessariamente.
- outrf2 - Constrói mais tabelas de saída para serem plotadas em qualquer plotador;
- outbit - Constrói e salva mais tabelas de saída para serem plotadas em qualquer plotador;

Nos arquivos fonte, também se encontram rotinas da BLAS principalmente para ajudar na fatoração LU da biblioteca LINPAK, que também se encontra entre os arquivos do código fonte. Tais rotinas são chamadas principalmente na rotina “fbstep” no arquivo “SOLVER_D4.f”.

4 Metodologia

Com base nas características do modelo, foi definido um conjunto de casos de estudos com o objetivo de estabelecer um referencial para as novas implementações em termos de resultados e tempo de execução. Os casos escolhidos foram executados no CACAU, plataforma computacional que será apresentada nas próximas seções, e seus resultados analissados com ferramentas especificas para *profiling* de código. O objetivo desta anáçise é determinar quais as rotinas do CYRANO responsáveis pelo maior consumo de recursos computacionais, principalmente tempo de processamento.

Uma vez detetado o papel significativo das chamadas às rotinas da BLAS no desempenho do CYRANO, foram feitos testes de desempenho com as rotinas DGEMM e ZGEMM utilizqando diferentes versões da BLAS. Especificamente foram comparadas a própria BLAS oficial, a ATLAS, a OpenBLAS e a BLIS, em seguida, estas bibliotecas foram substituídas no CYRANO, mais precisamente, nas chamadas da BLAS que são feitas pelo código.

Então, a função "OUTPOW" foi otimizada e paralelizada com a API OpenMP, apos estes passos, foram feitos os testes com as implementações da BLAS no CYRANO junto a nova versão executada de forma paralela e serial.

4.1 Materiais

Para efetuar os testes foram utilizados um cluster de computadores para computação de alto desempenho chamado CACAU. Os detalhes do CACAU aparecem descrito na Tabela 1. O compilador gfortran (parte do GCC versão 5.4.0), a linguagem Fortran70, a ferramenta de análise de performance "GPROF" e a API de paralelismo OpenMP foram utilizadas para o desenvolvimento desta pesquisa. Além destes, nos experimentos foram utilizados as bibliotecas ATLAS, BLAS, OpenBLAS e BLIS que são diferentes implementações da BLAS.

Tabela 1 – Descrição do hardware do CACAU (Centro de Armazenamento de Dados e computação avançada da UESC - <http://nbcgib.uesc.br/cacau>). 01/2018

	"Fila gpu"	"Fila Long"
Processador	2x Intel(R) Xeon(R) CPU E5-2440 @ 2.40GHz	2x Intel(R) Xeon(R) CPU E5-5430 @ 2.66GHz
memória RAM	6x 8GB DIMM DDR3 Synchronous 1600 MHz (48 GB)	16 GB
SO	linux Server release 6.3	linux Server release 6.3
Versão do Kernel	2.6.32-279.el6.x86_64	2.6.32-279.el6.x86_64

Para realizar as análises de resultados foi utilizado a linguagem python, junto as bibliotecas MATPLOTLIB e PANDAS para gerar os gráficos com os resultados do código.

4.2 Análise do código

Para analisar o desempenho do CYRANO, foi definido um conjunto de casos de estudos baseados nas caraterísticas do modelo, estes casos conseguiram expressar na simulação computacional o aumento do consumo de recursos que são utilizados conforme o tamanho dos dados de entrada da simulação aumentam. Desta forma, foram definidos dez casos de testes com o objetivo de avaliar o desempenho do código, cinco com 281 elementos e cinco com 481 elementos radiais e mais cinco casos com 151 elementos radiais para avaliar a precisão dos resultados, cada um dos casos de estudo com 32, 64, 128, 256 e 512 modos poloidais considerados. Os modos poloidais variam de -16 a 16, -32 a 32, -64 a 64, -128 a 128 e -256 a 256 acoplamentos poloidais. Os diferentes casos foram executados nas duas filas do CACAU.

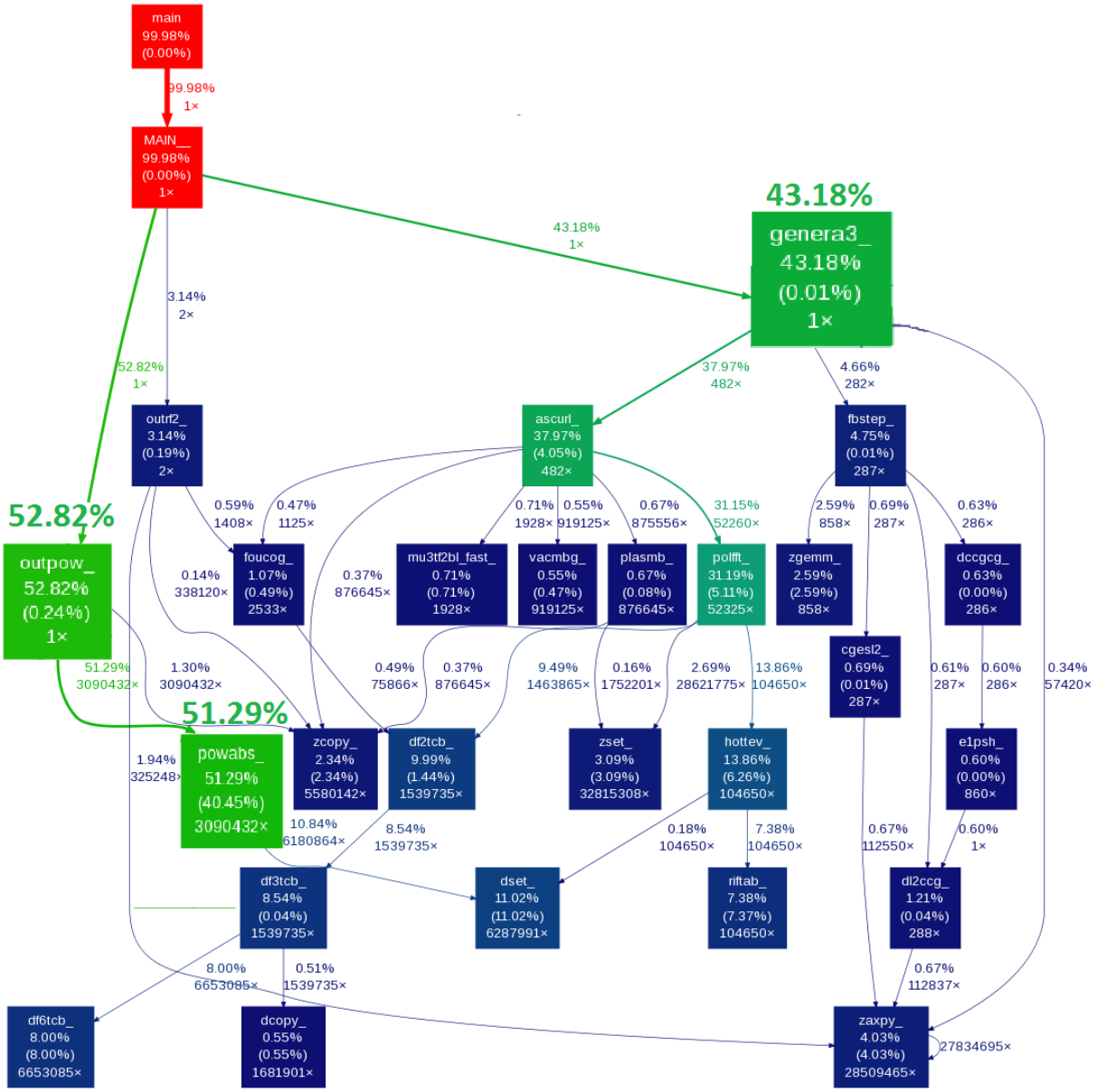
A ferramenta GPROF foi então utilizada para perfilar o código com intenção de definir quais sub-rotinas consomem mais tempo de processamento. Para processar os resultados do GPROF foi utilizado o *script* GPROF2dot, que apresenta o resultado do perfil em um gráfico. Este *script* exibe na árvore gerada apenas as funções relevantes, de modo que muitas funções chamadas pelo código não são exibidas, pois, o consumo no processamento ficou próximo de 0%.

Os resultados obtidos com o GPROF demostram que as rotinas GENERA3 e OUTPOW, das sub-rotinas chamadas diretamente na rotina principal, ocupam a maior parte do tempo de computo. O que pode ser observado nas Figuras 10 e 11 onde são apresentados os casos para 32 e 512 modos poloidais.

Desta forma foi possível determinar que para simulações de casos menores, Figura 10, a função OUTPOW tem grande importância no tempo total. No entanto, esta relevância decai na medida que se trabalha com simulações maiores, como visto na Figura 11, e nestes casos GENERA3 passa a representar uma parcela maior do tempo total.

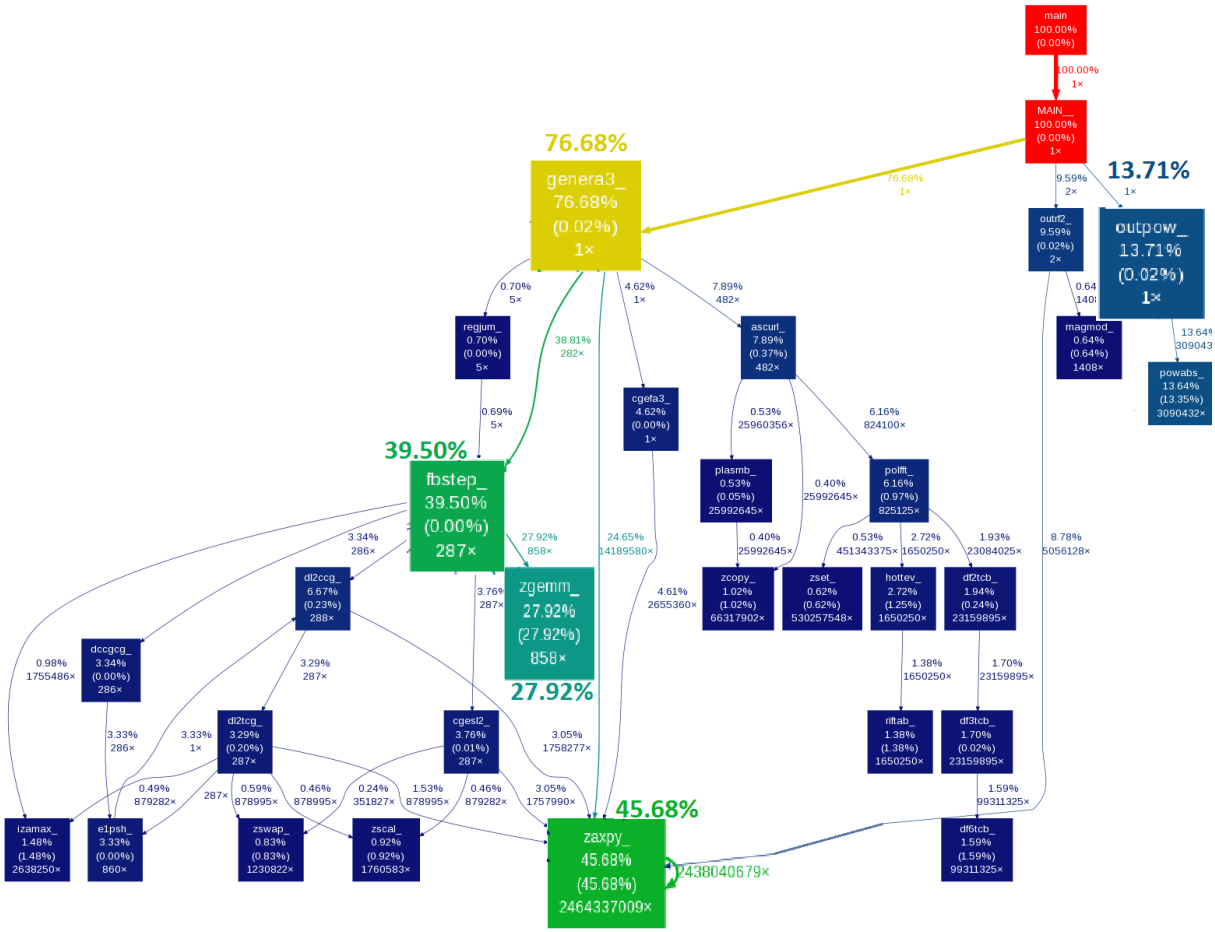
Analisando as Figuras 10 e 11, desta vez dando enfase as funções chamadas por GENERA e OUTPOW, observa-se que as chamadas a POWABS representam quase 100% do tempo de OUTPOW, e as chamadas ás funções ZGEMM e ZAXPY quase 50% de GENERA3.

Figura 10 – Imagem que representa a saída do GPROF para o código CYRANO utilizando um caso com 32 modos poloidais e 481 elementos. Em destaque as funções que ocupam maior porcentagem do tempo de execução, a função OUTPOW com 52,3% e GENERA3 com 42,1%. No casso de OUTPOW a maior parte do processamento se concentra POWABS com 51,29%.



Fonte: elaborada pelo autor.

Figura 11 – Imagem que representa a saída do GPROF para o código CYRANO utilizando um caso com 512 modos poloidais e 481 elementos. Em destaque as funções que ocupam maior porcentagem do tempo de execução, a função GENERA3 com 76,68% e OUTPOW com 13,71%. No casso de GENERA3 se destacam as funções ZAXPY com 45,68% e ZGEMM com 27,92%.



Fonte: elaborada pelo autor.

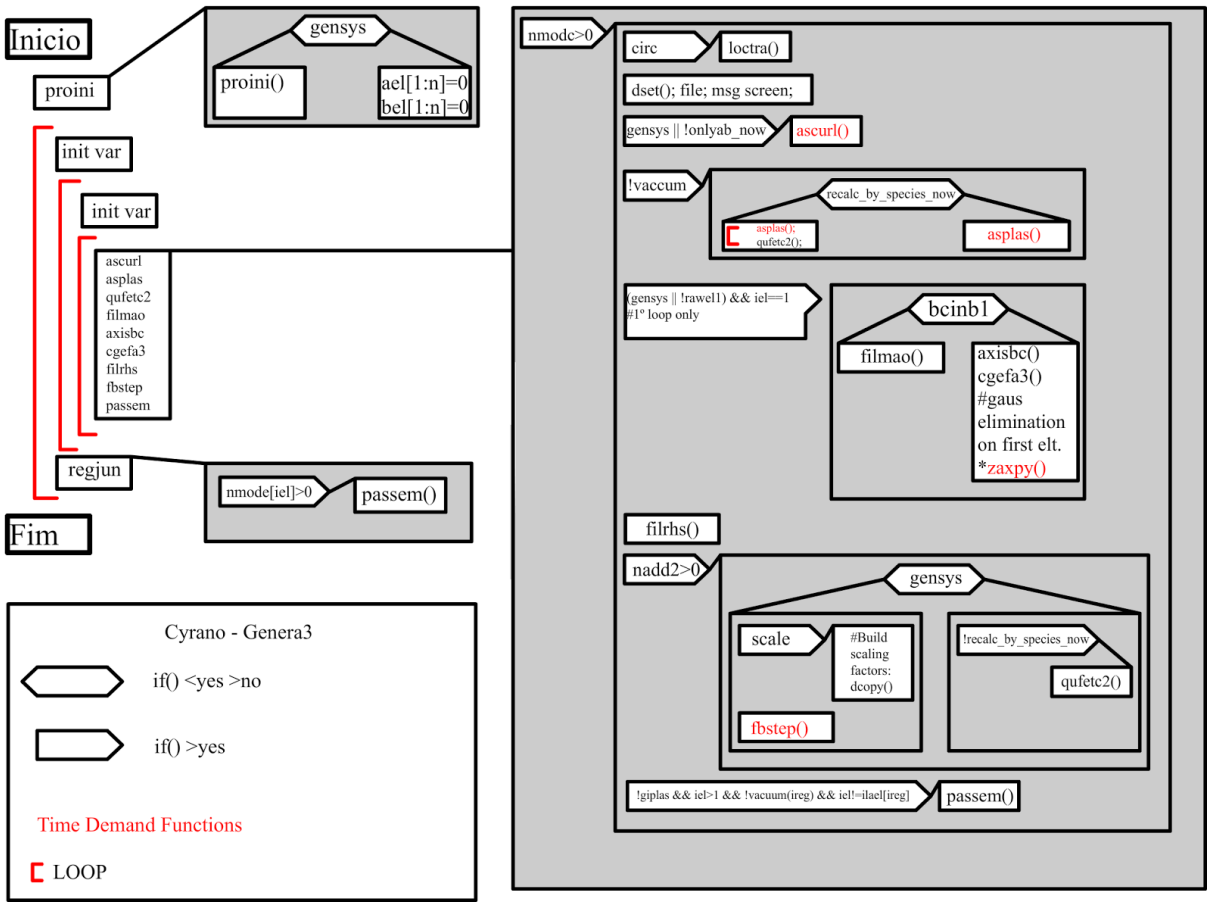
4.2.1 GENERA

No código, GENERA3 ou GENERA é a função responsável por montar e resolver o método Galerkin descrito no Capitulo 3.1. Esta rotina segue um fluxo de chamadas das rotinas como descrito na Figura 12.

Como, segundo os dados obtidos com GPROF quem mais consome tempo de execução em GENERA são as rotinas ZAXPY e ZGEMM, uma análise é necessária em como elas são utilizadas.

A ZAXPY é a rotina da BLAS que efetua multiplicação de vetor por um vetor com números complexos, é chamada apenas no primeiro elemento da região em GENERA3 para efetuar uma eliminação de gaus com picoteamento parcial, a quantidade de vezes em que é chamada depende diretamente com o tamanho do vetor ael, por tanto, do

Figura 12 – Fluxo de chamadas de funções pela rotina GENERA.



Fonte: elaborada pelo autor

tamanho do problema de entrada e é dada pela seguinte equação $\frac{N_1 * N_2}{2}$ onde $N_1 = lbl1$ e $N_2 = lblock_1 + lblock_2$, e portanto, conforme o tamanho da matriz aumenta mais a eliminação de gaus se torna relevante no tempo de computo.

A ZGEMM que é a rotina da BLAS responsável pela multiplicação de matrizes de números complexos, nos casos analisados, é chamada dentro da sub-rotina FBSTEP, que por sua vez, é chamada a quantidade de elementos mais a quantidade de regiões menos uma vez, FBSTEP então, chama 3 vezes a ZGEMM. Apesar de ter uma quantidade de vezes em que é chamada relativamente baixa a multiplicação de matrizes é altamente custosa para ser efetuada no computador, e conforme os tamanhos das matrizes aumentam, mais processamento é necessário o que torna esta função prioridade para uma melhoria na performance.

Além de GENERA, a outra rotina que consome mais tempo de computo, e em alguns casos, chega a ter um cossumo superior à própria GENERA, é a rotina OUTPOW.

4.2.2 OUTPOW

OUTPOW é uma rotina de pós-processamento, encarregada de gerar vários arquivos de saída do CYRANO, estes arquivos são referentes a energia absorvida pelo plasma, ou seja, a energia que é utilizada para aquecer o plasma.

Antes de escrever os arquivos, alguns dados são processados pela POWABS, o que consome muito tempo de execução, quase todo o tempo da rotina OUTPOW. Desta forma, ficou claro que a paralelização mais eficiente seria a executar em múltiplas linhas de execução (threads) as chamadas a rotina POWABS. Então, com o objetivo de paralelizar esta rotina, o código de POWABS foi transferido para o laço em que esta função era chamada, após tal feito, foram movidos os cálculos que se repetiam desnecessariamente para antes deste laço, além disto, foi preciso zerar o vetor "powquo", pois, compilando com as flags necessárias para utilizar o OpenMP, vetores não são iniciados com zeros quando declarados, com estes ajustes foi formando então, uma versão que reproduziu o mesmo resultado que a versão original.

Ao introduzir as diretivas de OpenMP foi necessário o uso de memória dinâmica para os vetores privados, como FORTRAN 70 não tem memória dinâmica, foi criada uma rotina com estas especificações, onde a memória privada é criada dentro da função e as memórias compartilhadas são passadas por parâmetro, gerando assim uma versão aprimorada do mesmo código. Após a realização destes procedimentos, foi necessário comparar os resultados para comprovar que a modificação não alterou os dados resultantes do código.

4.2.3 Comparando os resultados

Para analisar os resultados obtidos pelo código e avaliar a precisão dos resultados com a versão paralela, foram separados alguns casos de estudo, sendo eles, os com 151 elementos radiais cujos modos poloidais variam entre 32, 64, 128, 256 e 512 modos. Então estes resultados obtidos com a versão melhorada do código, foram comparados com os resultados da versão do código original, todas com a nova versão de OUTPOW, tanto serial quanto a paralela e utilizando as mesmas entradas. A comparação foi então efetuada ao analisar os gráficos gerados, cujos títulos são descritos a seguir:

- "Antena current and charge/Eps0 modes moduli" - A corrente e a carga por permissividade no vácuo;
- "Antena current toroidal spectrum" - A corrente da antena no espectro toroidal;
- "Radial derivative of inward Poyting flux (dPoynt/dr)" - A derivada radial do fluxo de energia para dentro;

- "Electric fiels poloidal harmonics (L2 norm)" - O campo elétrico poloidal harmônico ($\sqrt{\sum(|(X)^2|)}$;
- "L.H. eletric field" - O campo elétrico LH;
- "Poloidal eletric field" - O campo elétrico poloidal;
- "Radial eletric field" - O campo elétrico radial;
- "Toroidal eletric field - O campo elétrico Toroidal"
- "sp.absorbing in wave calc./dPoydr" - A absorvição da energia;
- "Average power absorption density per species (FLR0)" - Densidade média de absorção de energia por espécie (Raio de Larmor finito);
- "Average power absorption density per species (FLR2)" - Densidade média de absorção de energia por espécie (Raio de Larmor finito de 2ª ordem);
- "Tritium Individual power absorption contributions (averaged)" - Media das contribuições de absorção de energia individual do Trítio;
- "Inward Poyting flux" - Fluxo do campo eletromagnetico para dentro;
- "Wall poloidal current" - A corrente poloidal na parede de plasma;

Então, foram feitos alguns *script*s na linguagem python, para gerar os gráficos que utilizam os dados de saída dos códigos, estes *script*s são:

- Um *script* que gera dois gráficos comparativos entre um código e outro, gerando um terceiro gráfico com o erro absoluto segundo a formula $|C_1 - C_2|$ e um quarto gráfico com o erro relativo segundo a formula $\frac{|C_1-C_2|}{|C_1|}$, nos quais C_1 e C_2 são os dados do primeiro e do segundo código respectivamente.
- Um *script* que gera dois gráficos comparativos entre um código e outro, normalizando os dados segundo a formula $\frac{C-C_{min}}{C_{max}-C_{min}}$ no qual C são os dados, C_{min} é o menor valor existente em C e C_{max} é o maior valor em C, desta forma, os valores dos dados ficam entre 0 e 1, evitando divisões por números muito próximos de zeros. Este *script* também gera um terceiro e um quarto gráfico com o erro absoluto e relativo dos dados normalizados.
- Um *script* que gera os dois gráficos para comparação entre um código e outro, um terceiro gráfico com o erro absoluto e um quarto gráfico com o erro relativo formado pelos dados normalizados.

- Um *script* que gera os gráficos individuais dos dados contidos no arquivo "power-details_Trit.dat" para duas saídas de códigos.

Assim, foram analisados os gráficos chamados de "Err absoluto" e "Err realtivo", que representam o erro absoluto e o erro relativo dos dados com e sem a normalização para cada gráfico, resultado do 3º *script*, com objetivo de situar quais são as grandezas numéricas envolvidas no resultado. Assim, o "Err relativo" dos dados de cada gráfico, foram analisados para saber a media da discrepância percentual de cada resultado, considerando os dados do código original como "dados corretos", ou seja C_1. As figuras escolhidas para a análise são compostos por quatro gráficos, o gráfico resultado do código original (A), o gráfico resultado do código a ser analisado (B), os dados do gráfico (A) menos os dados do gráfico (B) que gera o Err Absoluto (C) e o Err relativo (D) normalizado.

Para entender a saída do CYRANO é preciso compreender, além de física de plasma e a fisionomia de um Tokamak, o arquivo dos dados de entrada.

4.3 O arquivo de entrada do CYRANO

O arquivo de entrada do código é definido no próprio código e não aceita entrada dinâmica, o arquivo definido pela versão disponibilizada do CYRANO não foi modificada, assim, o arquivo é o "./Data_Files/JET_data_tight.txt", este arquivo tem que estar no padrão para leitura do NAMELIST da linguagem FORTRAN, e também, não aceita o carácter '\r' utilizado junto com o '\n' para identificar quebra de linha no sistema operacional Windows, causando erro na execução do código caso o arquivo possua tal carácter, logo no começo da execução, sem alertar de alguma forma que o erro é este.

O padrão NAMELIST é um arquivo em formato de texto, que carrega as variáveis para o FORTRAN, como no exemplo do Algoritmo 1, neste exemplo, o nome do conjunto que será chamado no código fortran é NAMREG, assim ao efetuar a leitura com o comando "read(file,namreg,end=1001,err=1002)" o programa irá ler do arquivo já aberto na variável "file" os dados encapsulados no arquivo texto entre &NAMREG e &END (Fortran não é *case sencive*), em caso de final de arquivo (EOF) irá para o label 1001 e em caso de erro para o 1002. Os dados serão carregados nas variáveis que possuem o mesmo nome, no exemplo, a variável "nreg" terá o valor 3, "ns", que é um vetor terá os valores "[3,1,1]" a partir da posição 1, "rx0m" terá "[0.0,2.0,2.1,2.12]" a partir da posição 0 e "dobtyp" terá na posição 2 o valor -2. Apesar de em fortran os vetores começarem por padrão a partir da posição 1, é possível iniciar de qualquer posição, o que inclui posição negativa.

Algoritmo 1: Exemplo do NAMELIST.

```
&NAMREG
nreg=3, ns(1)=3,1,1,
rx0m(0) = 0., 2.0, 2.1, 2.12,
dobtyp(2) = -2,
&END
```

O arquivo "JET_data_tight.txt", possui os valores iniciais das variáveis que serão utilizados no cálculo, estes são divididos nos seguintes grupos de NAMELISTs:

- NAMOUT - Define alguns dados de saída serão escritos em arquivo;
- NAMPLO - Define quais dados e definições dos plots serão escritos em arquivo;
- NAMGEO - Define os dados relacionadas geometrias utilizada no calculo;
- NAMMAG - Define os dados relacionados ao campo magnético;
- NAMPLA - Define os dados relacionados ao plasma utilizado;
- NAMANT - Define os dados relacionados a antena;
- NAMREG - Define os dados relacionados as "Regiões";
- NAMSUB - Define os dados relacionados as "Sub-Regiões";
- NAMSYS - Define os dados relacionados ao sistema;

Os dados que aumentam a complexidade, e portanto, o tempo de computo são, "modva1" e "modva2" do NAMELIST "NAMANT", que representam o intervalo de modos poloidais considerados, "klim" do NAMELIST "NAMSYS", que representa o número de acoplamentos poloidais considerados e "iele" do NAMELIST "NAMSUB", que define a quantidade de elementos radiais por Sub-Região. Os dados de entrada mais importantes são:

- dshape - para habilitar ou desabilitar o plasma;
- b0 - define o centro do campo magnético;
- ipl - define a corrente no plasma;
- flrops - define o "truncamento" do raio de Larmor finito e do Raio de Larmor finito de 2ª ordem;
- nspec - define a quantidade de espécies de plasmas que serão utilizadas na simulação;

- zch - "Z" define a quantidade de prótons do átomo das espécies de plasma que serão utilizadas;
- amass - "A" define a massa dos átomos das espécies de plasma que serão utilizadas;
- t0 - define a temperatura inicial dos átomos das especies de plasma que serão utilizadas;
- n0 - define a densidade do plasma;
- fregag - define a frequência da emissão das ondas de rádio frequência RF;
- motoan - define a quantidade de modos toroidais;
- modva1 e modva2 - o intervalo de modos poloidais considerados [modva1:modva2];
- klim - o número de acoplamentos poloidais considerados;

Como o CYRANO faz uso da BLAS, foram feitos testes de performance nas diferentes versões disponíveis da mesma, principalmente com a ZGEMM, que é a função da BLAS no código que mais cresse no tempo de computo conforme a quantidade de modos poloidais e elementos radiais aumentam.

4.4 Teste de desempenho da BLAS

Antes de realizar os testes com as versões otimizadas do CYRANO foram avaliadas as diferentes opções de implementações da BLAS disponíveis. Para testarmos o desempenho das bibliotecas, o algoritmo DGEMM e ZGEMM da ATLAS, OpenBLAS, BLIS e da própria BLAS, teve o seu tempo de execução medido para realizar o calculo de GFLOPS descrito na seção 2.5.4.

Para medir o tempo gasto pela GEMM, foi utilizada a API OpenMP e sua função de medir o tempo gasto "get_wtime", o código foi escrito com auxílio da biblioteca GSL *Scientific Library* que permite encapsular o código de forma que, para executar as mais diferentes bibliotecas da BLAS, basta mudar na compilação do código. O compilador utilizado foi o gcc. Na linguagem C o sistema de matrizes é de linhas por colunas, diferente de fortran que é colunas por linhas, mas utilizando matrizes como vetores, o sistema coluna por linha não foi invertido. na execução dos testes foi utilizado o CACAU, cujas especificações estão descritas na Tabela 1.

No experimento, as matrizes são matrizes quadradas que variam de 32×32 até 4096×4096 de 32 em 32 passos. Para cada passo, o algoritmo é executado 3 vezes e então, o melhor tempo obtido destas é utilizado no cálculo dos GFLOPS. Uma vez obtido os resultados, uma análise se faz necessária.

5 Resultados e discussões

Antes de analisar os resultados obtidos com o CYRANO, é importante analisar os resultados obtidos com a BLAS, pois, o desempenho obtido com o CYRANO está relacionado com o desempenho principalmente das rotinas ZGEMM e ZAXPY, que fazem parte da BLAS.

5.1 Resultados de desempenho da BLAS

Os resultados da quantificação do tempo dos códigos obtidos estão em formato de gráfico na Figura 13. Como se observa, a BLAS é muito mais lenta do que as outras bibliotecas otimizadas, e para matrizes muito grandes o tempo consumido por ela é significativamente maior quando comparada as outras.

Quando se conhece a quantidade de operações efetuadas num procedimento, uma das formas de medir o desempenho deste procedimento de forma mais acurada é calcular a quantidade de operações realizadas em média por segundo. Esta medida é conhecida como FLOPS. A OpenBLAS obteve um desempenho significativo de aproximadamente 75 GFLOPS, no entanto, as versões paralelas da BLIS alancaram mais que o dobro desse desempenho.

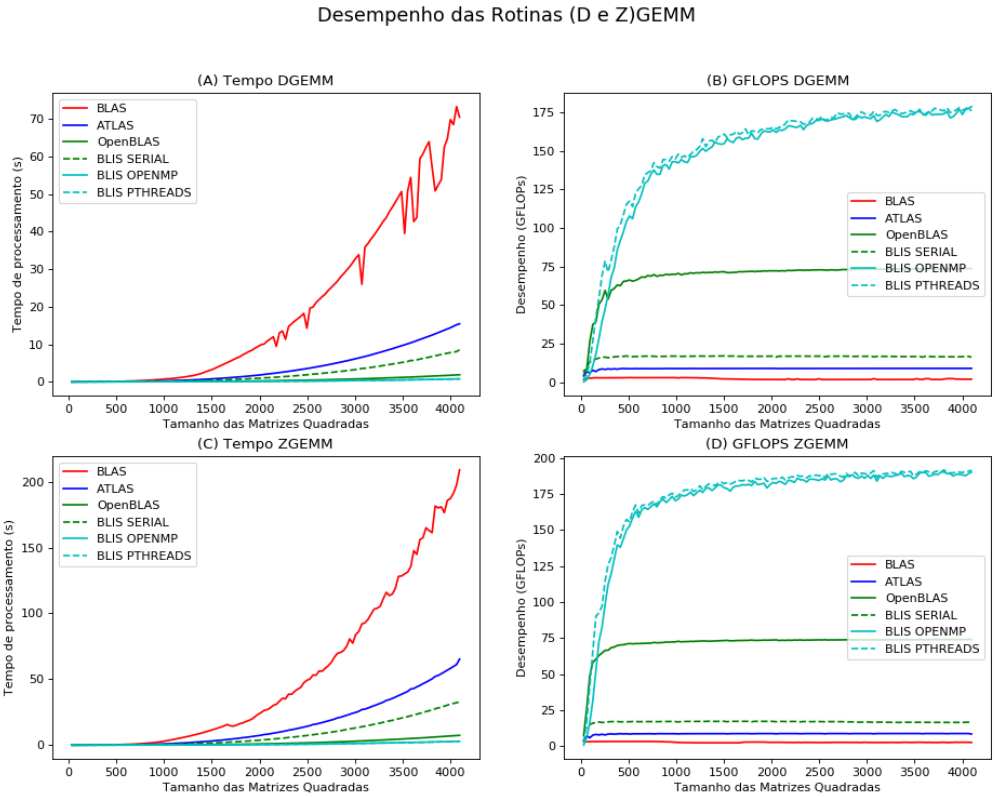
A quantidade de GFLOPS da DGEMM é bem próxima da ZGEMM, como se vê na Figura 13 **(B)** e **(D)** o que é normal, já que a quantidade de operações de ponto flutuante de precisão dupla depende do hardware tanto quanto do software. Um hardware tem um limite de FLOPS teóricos que pode atingir, para tal, é necessário um software bem desenvolvido que consiga alcançar este limite. As bibliotecas otimizadas tentam utilizar de técnicas, que estão relacionadas principalmente com o uso da memória cache e com o uso dos múltiplos núcleos, para aproveitar ao máximo os recursos destes hardwares.

Quanto aos tempos de execução, a ZGEMM consome muito mais tempo que a DGEMM, apesar dos FLOPS serem bem próximos, isto ocorre porque a quantidade de operações também são maiores para a ZGEMM, como explicado no Capítulo 2.5.4.

Observando novamente a Figura 13 com os gráficos, identifica-se que a biblioteca BLIS paralela é a que teve o melhor desempenho, sendo que a versão com pthreads é ligeiramente melhor que a com OpenMP, seguida pela OpenBLAS.

Como o CYRANO utiliza a BLAS nas partes criticas do cálculo, principalmente as rotinas “ZAXPY” e “ZGEMM”, foram realizados testes utilizando as bibliotecas ATLAS, OpenBLAS e BLIS com a finalidade de aumentar a performance do CYRANO diminuindo o tempo de computo total para efetuar as simulações.

Figura 13 – Performance das rotinas ZGEMM e DGEMM das bibliotecas BLAS, ATLAS, OpenBLAS e BLIS



Fonte: elaborada pelo autor.

Apesar de mais eficientes no consumo de recursos para realizar estas operações, os resultados das operações variam levemente de uma versão da BLAS para a outra, o motivo pelo qual isto ocorre foi explicado no Capitulo 2.4.2, mas estas pequenas diferenças podem conduzir os resultados numéricos a aproximações diferentes, devido à alta quantidade de operações.

5.2 Comparação dos resultados gerados

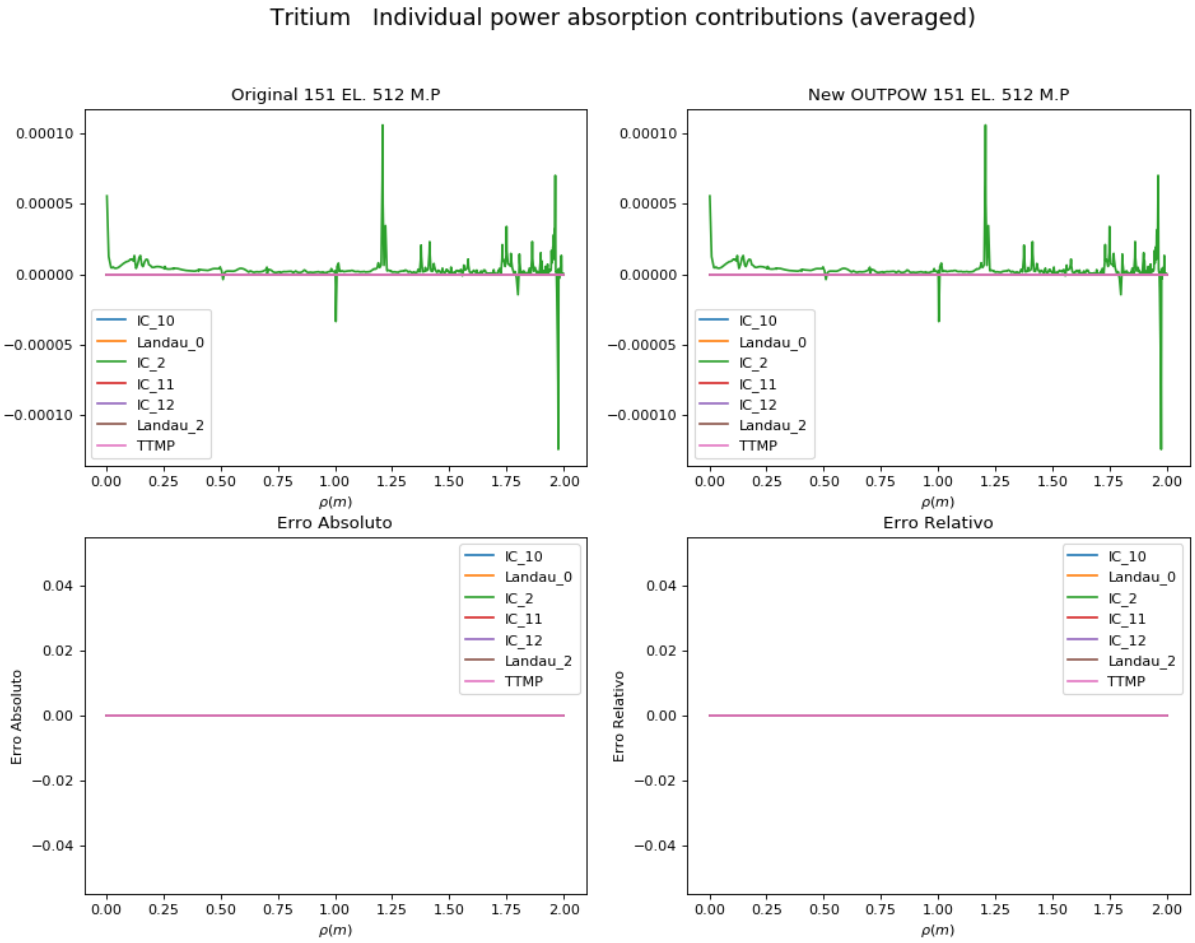
Os resultados gerados após a execução do CYRANO estão em arquivos com a extensão ".dat", estes são gerados na pasta "Plot_data\runXXX" onde XXX é um número entre 000 e 999. No código que foi utilizado como base, ou seja, o código original, este número começa a partir de 105 e vai incrementando dependendo o arquivo de entrada. Os arquivos ".dat" contem tabelas com vários números separados por tabulação, cada tabela tem seu próprio significado para os números contidos nelas.

Como a quantidade de dados gerados nestas tabelas são muito grande, a melhor forma de analisar-los é gerar gráficos com estas tabelas, assim, nos gráficos gerados o

eixo X é normalmente a primeira coluna desta tabela, enquanto que as outras colunas são os dados e o eixo Y os valores destes dados.

Comparando os resultados gerados pela versão original do código com a que utiliza a nova versão do OUTPOW serial, como na Figura 14, e sabendo que todos os outros gráficos gerados com todos os diferentes modos para 151 elementos também tiveram erros relativos iguais a zero, é possível concluir que ambos os códigos geram resultados absolutamente idênticos.

Figura 14 – Comparação de resultados. Código original comparado com o código utilizando a nova versão de OUTPOW serial.



Fonte: elaborada pelo autor.

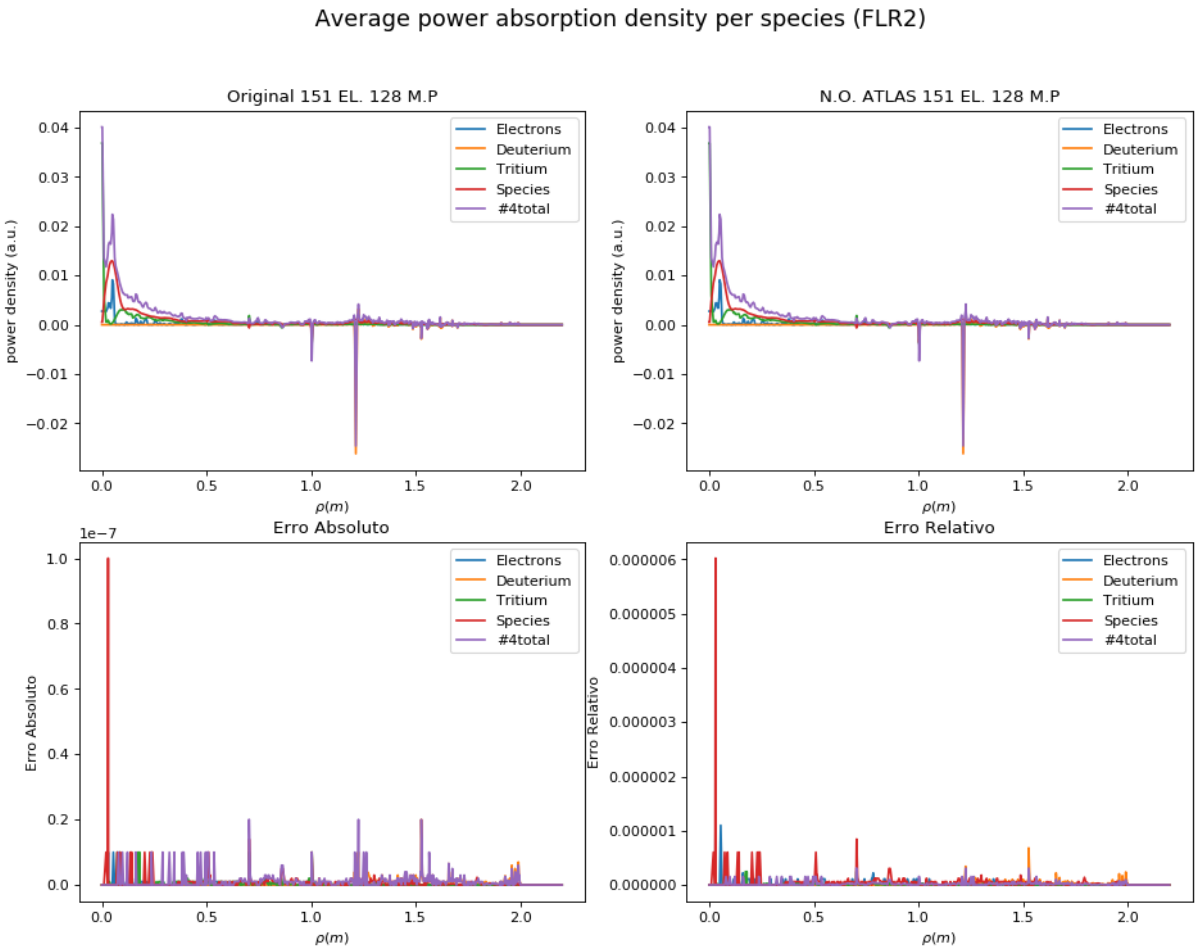
O gráfico da Figura 14 representam uma "saída" direta da rotina OUTPOW, já que, são os valores das variáveis nas quais, na versão paralela da nova versão do OUTPOW, são efetuadas um procedimento conhecido como *reduction*[1], sendo seus valores salvos em arquivo logo após este procedimento, para que então, possam ser plotadas no gráfico. Vale lembrar que os dados utilizados para efetuar os cálculos

[1]Reduction é um procedimento bem conhecido que serve para realizar uma operação (soma, subtração, multiplicação, etc) sucessivamente de um vetor ou lista em uma variável, de forma paralela.

destas variáveis são previamente calculados em GENERA, que utiliza a BLAS em seus procedimentos.

Ao utilizar outras bibliotecas da BLAS, mantendo o OUTPOW serial, para até 128 modos poloidais, os resultados são semelhantes, com erros de 0,07% no caso de 151 elementos radiais com 128 modos poloidais, como se observa nos gráficos da Figura 15, que é o gráfico comparativo da versão original e a que utiliza a ATLAS com a nova versão do OUTPOW serial. A atlas obteve o pior erro relativo neste gráfico, enquanto o maior pico da Atlas foi de $0,0007$ o da OpenBLAS tem o maior pico de $0,000007$ e a BLIS ficou com a maior escala de $0,0000016$.

Figura 15 – Comparação de resultados. Código original comparado com o código utilizando a nova versão de OUTPOW serial com a ATLAS.



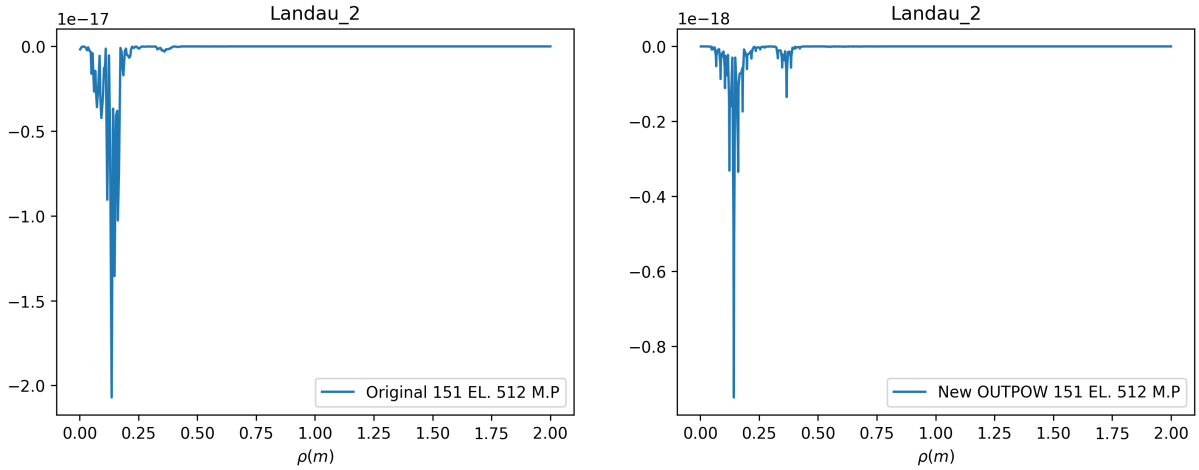
Fonte: elaborada pelo autor.

Como os resultados obtidos com as outras bibliotecas da BLAS, apesar de ligeiramente diferentes, geram resultados satisfatórios, é necessário analisar os resultados da paralelização de OUTPOW. Para tal, foram gerados os gráficos da versão com apenas o OUTPOW paralelizado, mantendo a BLAS original do código, os gráficos gerados que são "saídas diretas" do OUTPOW, ou seja, resultado de cálculos efetuados nesta função, são: "sp.absorbing in wave calc./dPoydr", "Average power absorption density

per species (FLR0)", "Average power absorption density per species (FLR2)" e "Tritium Individual power absorption contributions (averaged)".

O gráfico "Tritium Individual power absorption contributions (averaged)" tem um conjunto de legendas que representam as variáveis "IC_10", "Landau_0", "IC_2", "IC_11", "IC_12", "Landau_2", "TTMP", "TTMP_2", que são os rótulos das variáveis em que são feitas o procedimento *reduction*. Os resultados do código exibido nos gráficos relacionados a nova versão do OUTPOW com o paralelismo, que incluem o "Tritium Individual power absorption contributions (averaged)", demostram gráficos aparentemente similares, mas os erros relativos grandes, isto ocorre porque estes valores são muito pequenos, próximos de zero, para averiguar isto, basta observar os gráficos dos valores individuais, como o de "Landau_2" na Figura 16, onde se vê que apesar de terem formas semelhantes, a ordem dos números é de 1e-17 e 1e-18, este tipo de relação ocorre com o caso de 151 elementos radiais, também com os gráficos das variáveis IC_2 e IC_11.

Figura 16 – Comparação de resultados. Código original comparado com o código utilizando a nova versão de OUTPOW paralelo que utiliza a OpenBLAS.



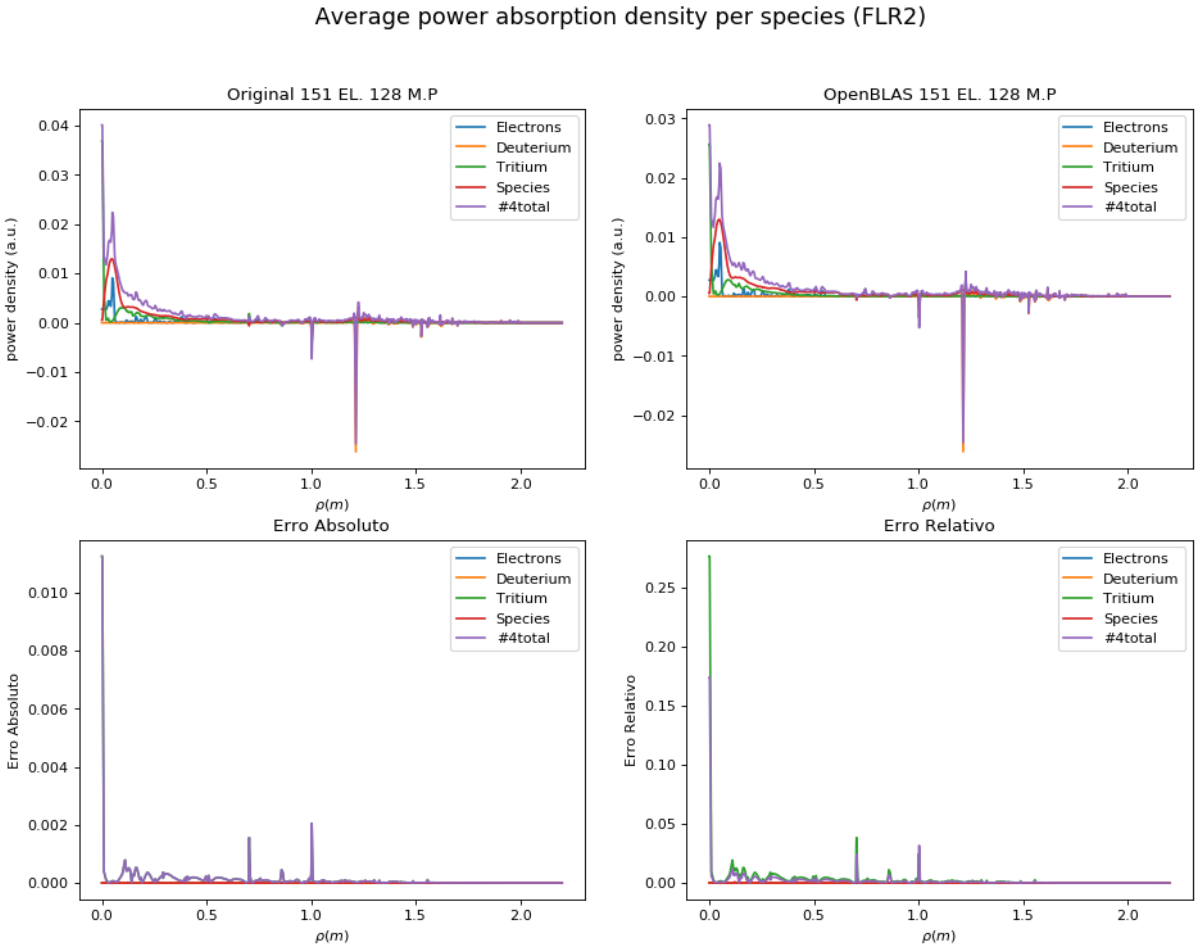
Fonte: elaborada pelo autor.

Estes erros ocorrem, aparentemente, por causa de erros de aritmética de ponto flutuante que são inerentes a paralelização do código e a grandeza dos dados envolvidos no cálculo.

Para analisar os resultados obtidos ao utilizar as outras bibliotecas da BLAS junto a nova rotina OUTPOW paralelizada, foram construídos também, os gráficos dos erros relativos com os dados normalizados, este procedimento foi necessário, pois, o erro relativo possui picos bastantes relevantes devido a divisões por números que são quase zero.

O gráfico na Figura 17 representa o quanto os dados são semelhantes ao comparar a versão original com a versão paralela utilizando a OpenBLAS. Apesar de iniciar com

Figura 17 – Comparação de resultados com dados normalizados. Código original comparado com o código utilizando a nova versão de OUTPOW paralelo que utiliza a OpenBLAS.



Fonte: elaborada pelo autor.

25% de diferença, sendo este apenas o dado inicial no erro relativo, em média, a maior parte dos dados possuem valores menores que 5% de diferença, percebe-se então, que os dados são relativamente aproximados entre os resultados dos códigos original e paralelo com OpenBLAS paralelo.

As descontinuidades observadas nos perfis radiais de absorção de potencia (Fig. 17) são inerentes da aproximação adotada para o calculo da densidade de potencia local em OUTPOW:

$$p(\rho,\theta) = real(\boldsymbol{E}(m_1) \times \boldsymbol{M}_{12}(m_2 - m_1, \frac{(m_1+m_2)}{2}) \times conj[\boldsymbol{E}(m2)]) \tag{6}$$

Onde $E(m_1)$ e $E(m_2)$ representam as componentes (poloidais) de Fourier do campo elétrico local, e M_{12} é a matriz de acoplamento dos respectivos modos poloidais, que contem a resposta dielétrica do plasma (propriedades de absorção). Como a matriz

de acoplamento eh tabulada em função da diferença $m_1 - m_2$ e do valor médio $\frac{(m_1+m_2)}{2}$ dos modos considerados, enquanto que os campos são representados pelas respectivas componentes poloidais m_1 e m_2, a aproximação utilizada na Equação 6 perde em precisão quando um número elevado de modos poloidais eh considerado. Essa fraqueza do modelo esta sendo revisada e técnicas numéricas para evitar esse efeito, baseadas em um gradeamento adaptativo da representação da matriz M_{12} em torno das regiões criticas, deve ser implementada em breve.

Contudo, ao analisar os gráficos para quantidades de modos 256 e 512, o comportamento do código se demostrou divergente entre as versões, ficando evidente que, quanto menos modos poloidais mais aproximados são os resultados dos códigos, e quanto mais modos poloidais mais divergentes.

Então o seguinte experimento com o arquivo de entrada de 151 elementos radiais e 512 modos poloidais foi realizado para averiguar esta possível instabilidade nos resultados do código.

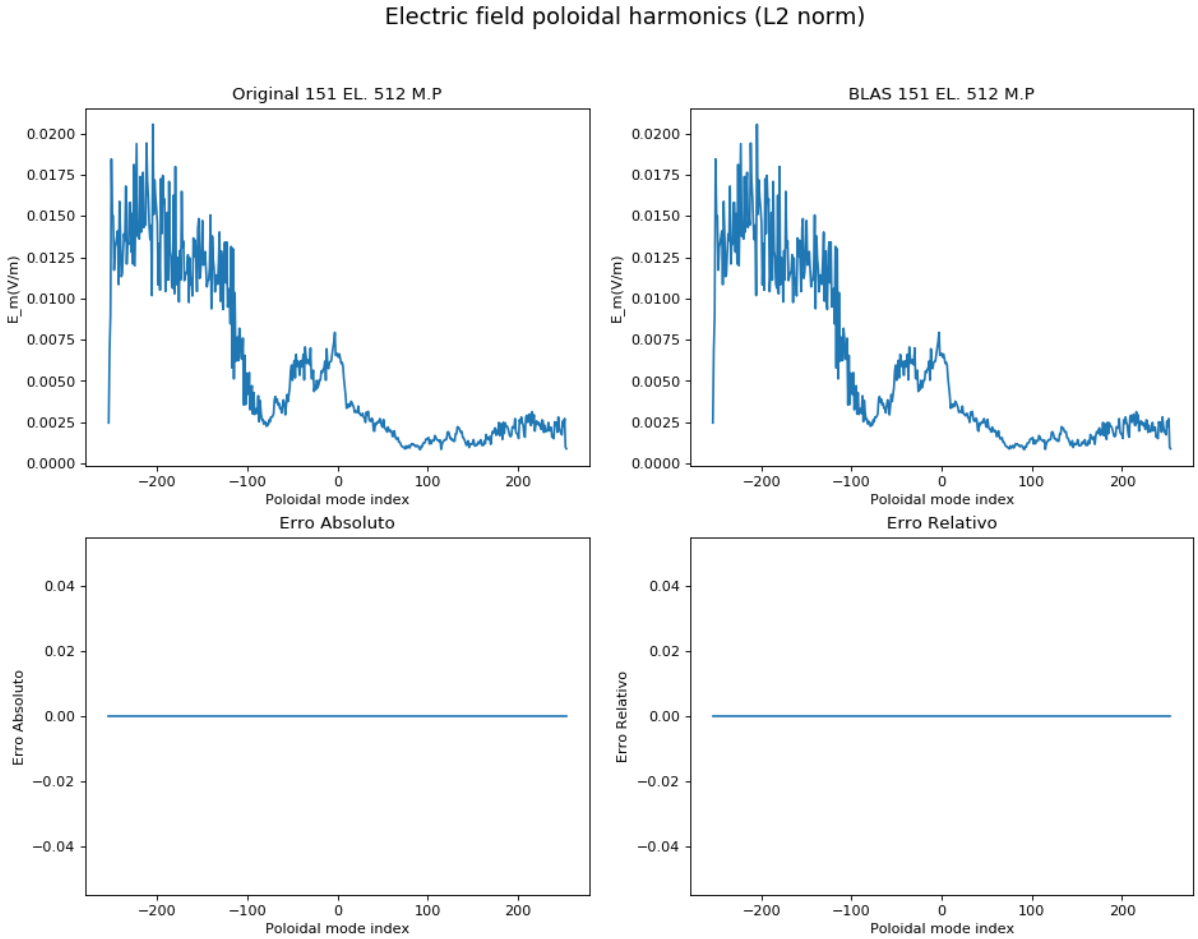
O experimento consiste em executar o código com a biblioteca BLAS oficial, para então, comparar o resultado obtido com o mesmo código compilado com as mesmas flags de compilação, só que com uma das versões da BLAS paralela, que idem a BLAS seja, testada, bem conhecida e amplamente utilizada no meio acadêmico, para tal a OpenBLAS foi a escolhida. Vale lembrar que, nestes casos, a única diferença entre os códigos, é a biblioteca BLAS utilizada na compilação, já que a nova versão do OUTPOW, quando executada sem paralelismo com OpenMP, de forma serial, gera resultados idênticos aos do OUTPOW original. Desta forma, é garantido que a única diferença entre os códigos, é a versão da BLAS que foi utilizada na compilação.

Ao observar a Figura 18, percebe-se que ambos os resultados são idênticos, pois, tanto o erro absoluto quanto o erro relativo entre ambos os resultados são de 0% entre os dados. Comparando os resultados gerados pelo código compilado com a BLAS oficial contra os resultados do código original, o erro relativo se mantém em 0% para todos os gráficos gerados. Ou seja, a BLAS que faz parte do código fonte do CYRANO é idêntica a versão oficial da BLAS, o que retira as dúvidas sobre o código alterado está proporcionando as divergências encontradas nos resultados quando a simulação é efetuada com 512 modos.

Os resultados que são obtidos quando o mesmo código é compilado alterando apenas a biblioteca BLAS pela OpenBLAS na link edição, são, como se vê nas Figuras 19 e 20, diferentes do código original, e portanto, também do código que utiliza a BLAS oficial, possuindo erros reativos significativos.

Quando se observa os gráficos superiores na Figura 20, a primeira vista são semelhantes, porém, prestando atenção no erro absoluto, os valores da diferença entre

Figura 18 – Comparação de resultados. Código original comparado com o código utilizando a BLAS oficial.



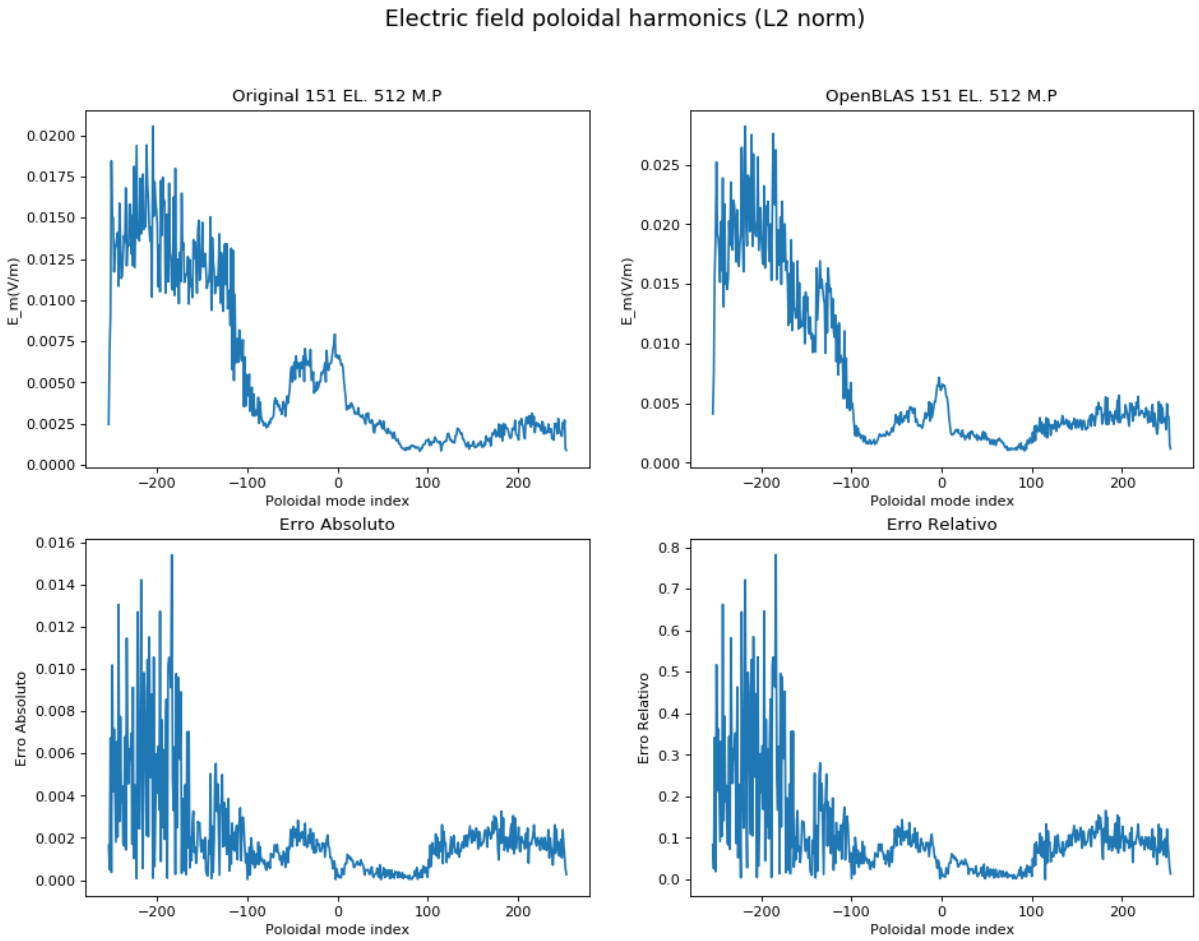
Fonte: elaborada pelo autor.

os gráficos são relativamente grande, e como se observa do erro relativo, que possui valores maiores que 40% de diferença, e a maioria dos dados está a cima dos 10%. E em todos os outros gráficos gerados, a diferença entre os gráficos mostraram erros relativos significativos, similares aos apresentados neste documento.

Este experimento demostra que existe uma instabilidade do método numérico utilizado no código, sendo que para determinados dados de entrada, esse se torna mais instável conforme se aumenta a quantidade de modos poloidais. E considerando que o código compilado para a versão paralela está correto, como ficou evidente após os dados resultantes do código com a versão da BLAS original serem idênticos aos do CYRANO original, é possível observar que os dados são diferentes para cada uma das versões da BLAS que foram utilizadas nos testes, apesar de serem similares.

Acreditamos que os erros estão relacionados as matrizes mal condicionadas presentes na execução do código, o que torna mais evidente o efeito causado pela aritmética de números finitos, desta forma tornando o código instável ao ter que realizar

Figura 19 – Comparação de resultados. Código original comparado com o código utilizando a OpenBLAS.



Fonte: elaborada pelo autor.

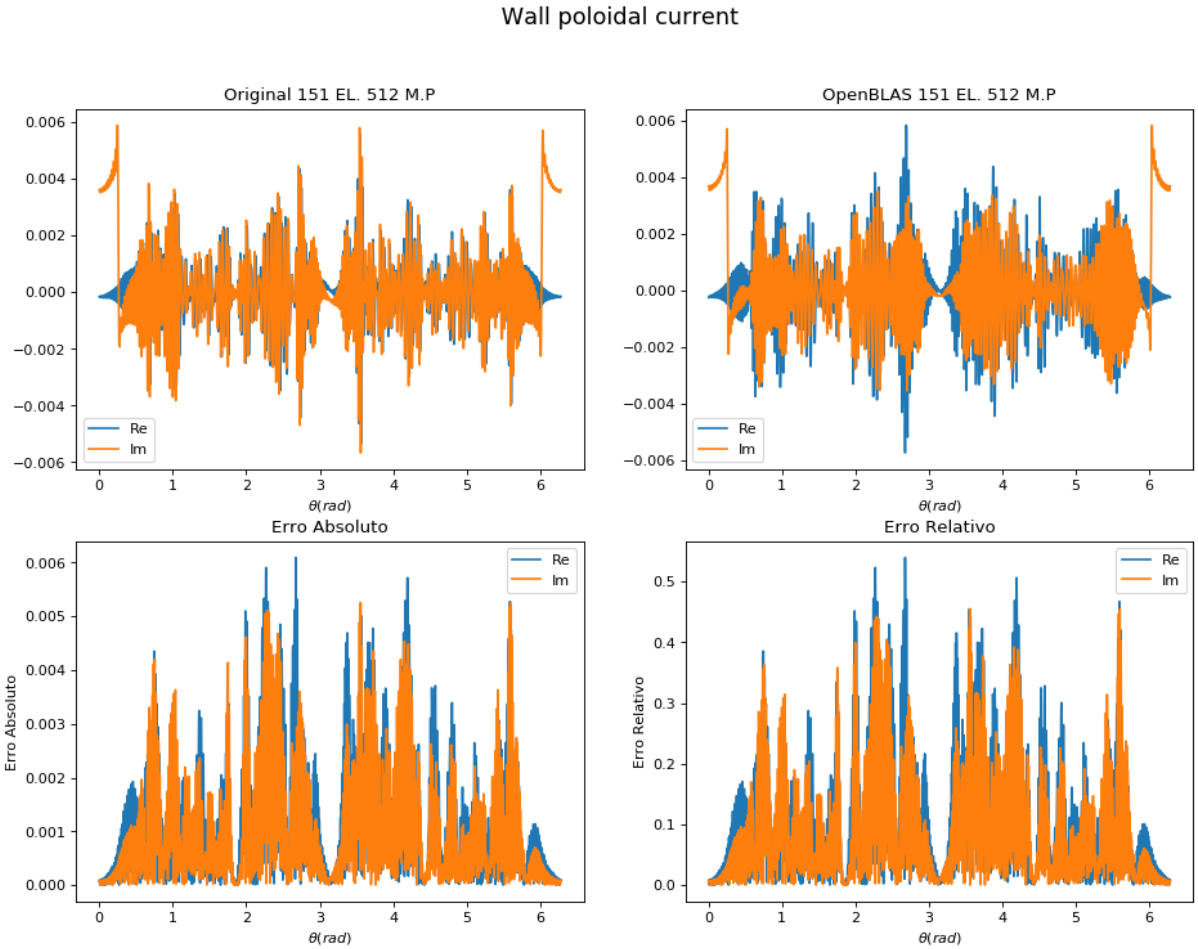
operações com matrizes maiores para alguns casos. No entanto, com os resultados próximos do esperado, é necessário saber o quão eficiente são estas novas versões, com as diferentes bibliotecas da BLAS e o novo OUTPOW paralelizado.

5.3 Desempenho

Ao medir o tempo utilizado pelas funções GENERA e OUTPOW, no código utilizado como base sem modificações (ou o código **original** completamente serial), observa-se que GENERA aumenta o tempo de computo, como se percebe nos Gráficos 21 **A** e **B**. Esta diferença entre GENERA e OUTPOW se destaca ainda mais na fila LONG, que apesar de possuir processadores com mais "clock" que os da fila GPU (2.66GHz contra 2.40GHz), são da geração passada e por tanto possuem menos recursos tecnológicos o que os deixa com menos poder de processamento.

Após rodar os casos com os algoritmos paralelos, neste caso com a OpenBLAS, tendo como objetivo medir o tempo de execução entre GENERA e OUTPOW, ficou fácil

Figura 20 – Comparação de resultados com normalização dos dados. Código original comparado com o código utilizando a OpenBLAS.

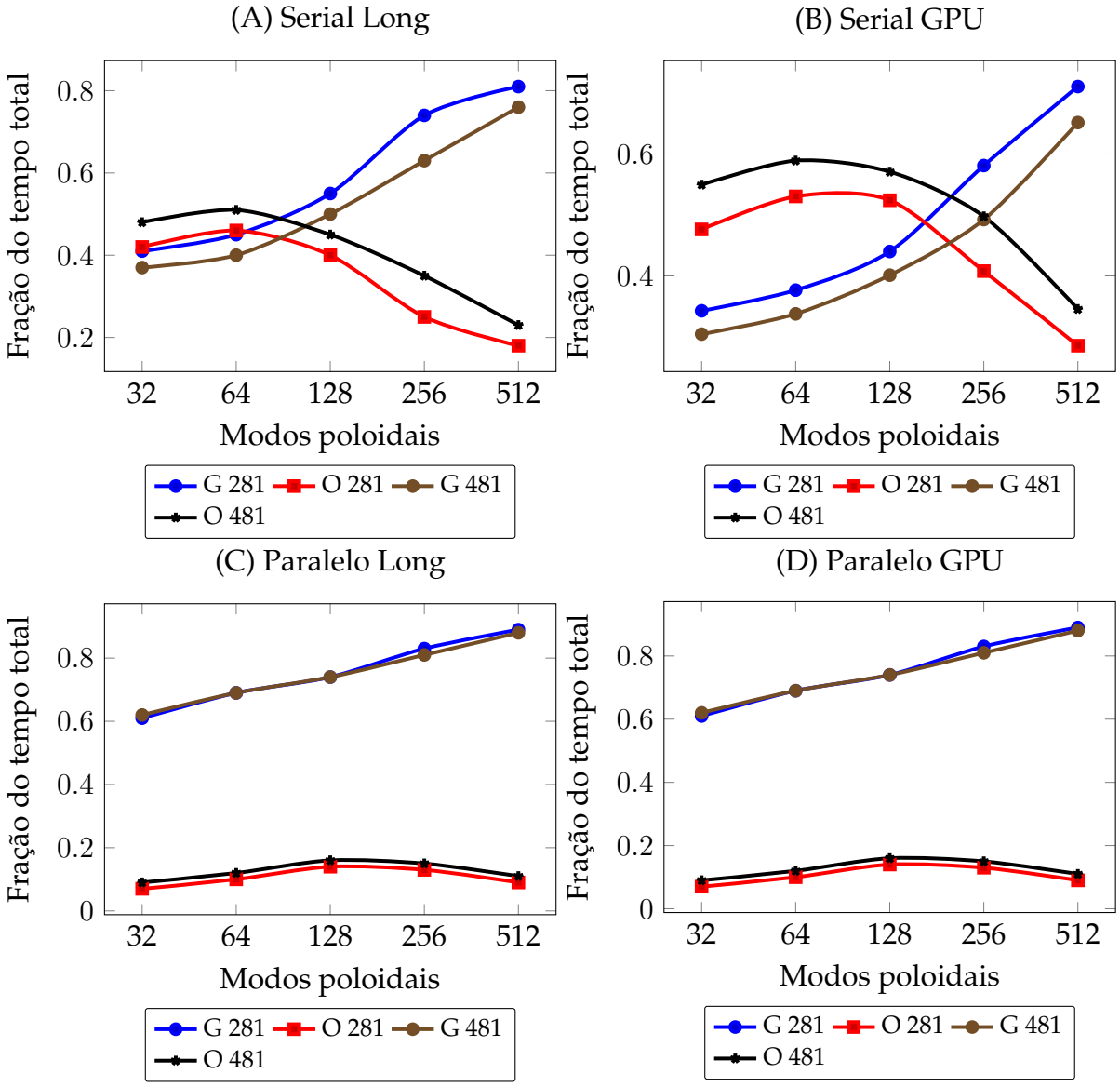


Fonte: elaborada pelo autor.

observar no gráfico da Figura 21 (**C**) e (**D**) que, como já era esperado, GENERA passa a predominar no tempo de computo do código, este comportamento é crescente, mesmo com GENERA diminuindo o tempo de execução conforme se aumenta o número de modos poloidais, o tempo diminuído não acompanha o comportamento quase exponencial do tempo necessário para efetuar os cálculos na versão serial. Neste contexto, OUTPOW não passa de 20% do tempo total, o que demostra que uma paralelização em OUTPOW é atualmente menos prioritária para melhorar a performance do CYRANO.

Ao comparar versões seriais com versões paralelas, uma das formas de medir o desempenho é o *speedup* que é $\frac{T_p}{T_s}$, onde T_p e T_s são o tempo da execução pelo código paralelizado e sequencial respectivamente. O *speedup* pode não ser uma boa comparação em alguns casos, a exemplo, imagine comparar um algoritmo executado numa GPGPU (*General-Purpose Graphics Processing Units*) com o equivalente serial num processador da linha Atom de baixo custo da Intel, não é o mesmo valor se a comparação for feita com processadores mais poderosos como um i7 ou i9. Neste caso o *speedup* é uma boa escolha,

Figura 21 – Relação do consumo do tempo das funções GENERA3 e OUTPOW com o tempo total, tendo 281 e 481 elementos. Sendo comparados os códigos original e Paralelo com OpenBLAS utilizando nova rotina OUTPOW Paralela. Nas filas GPU e LONG do CACAU. O eixo Y representa a fração do tempo total do código e o eixo X a quantidade de modos poloidais considerados. G representa GENERA e O, OUTPOW.



Fonte: O próprio autor.

pois, é comparado um núcleo de processamento contra múltiplos do mesmo modelo de núcleo de processamento, então se a comparação for feita em um processador diferente, o *speedup* deve se manter a proporção se a quantidade de núcleos forem as mesmas, já que o paralelismo é feito com OpenMP.

As Tabelas 2 e 3 exibem os tempos de execução para os casos de 281 e 481 elementos radiais, e ao observar o comportamento de OUTPOW paralelo, percebe-se

Tabela 2 – Tempo em segundos, gasto pelo código nas rotinas OUTPOW, GENERA3 e o tempo total, medido no cacau, fila gpu, caso com 281 elementos radiais.

	Tempos em segundos				*Speedup*		
Modos	Original	OpenBLAS	BLIS	Atlas	OpenBLAS	BLIS	Atlas
			GENERA3				
32	189,44	190,89	186,90	186,68	0,99	1,01	1,01
64	455,25	367,19	355,46	411,02	1,24	1,28	1,11
128	1.705,80	856,19	859,78	1.590,59	1,99	1,98	1,07
256	10.518,28	3.786,95	3.796,31	8.068,04	2,78	2,77	1,30
512	69.938,47	21.470,46	20.794,57	51.242,83	3,26	3,36	1,36
			OUTPOW				
32	263,27	21,76	22,21	21,95	12,10	11,85	12,00
64	641,19	51,62	52,42	51,34	12,42	12,23	12,49
128	2.031,61	158,95	161,92	159,64	12,78	12,55	12,73
256	7.382,96	580,04	600,85	579,86	12,73	12,29	12,73
512	28.098,32	2.210,14	2.237,09	2.210,64	12,71	12,56	12,71
			TOTAL				
32	552,76	312,55	336,36	310,35	1,77	1,64	1,78
64	1.208,78	530,91	550,61	575,43	2,28	2,20	2,10
128	3.876,20	1150,37	1.213,81	1.889,50	3,37	3,19	2,05
256	18.102,27	4.558,44	5.028,26	8.847,58	3,97	3,60	2,05
512	98.418,68	24.049,36	23.924,12	53.839,58	4,09	4,11	1,83

que a rotina ficou em média 12 vezes mais rápida que a versão original, mantendo um *speedup* linear, na quantidade de *threads*, o que mostra que a versão paralela do OUTPOW é eficiente.

Apesar de a BLIS possuir um *speedup* duas vezes maior que a OpenBLAS, como demostrado no começo deste capitulo, o uso da versão paralela da BLIS no CYRANO não foi tão eficiente quando comparado com o uso da OpenBLAS no CYRANO, ficando tão boa quanto a OpenBLAS.

A ATLAS, teve ganhos até 256 modos, mas com 512 modos diminuiu significantemente o desempenho adquirido nas execuções com modos menores, vale lembrar que a ATLAS utilizada neste experimento é serial e otimizada.

Na Tabela 4, que representa os tempos gastos para o caso de 151 elementos radiais, a versão paralela com a OpenBLAS obteve um *speedup* no tempo total de 4,63, o que indica que foi 4,6 vezes mais rápido executar o CYRANO com a OpenBLAS e a nova versão do OUTPOW, apesar de os resultados obtidos com o código ter as divergências já explicadas, executar o CYRANO com 512 elementos utilizando a OpenBLAS levou quase o mesmo tempo que a execução com o código original utilizando 256 elementos, gerando uma aproximação melhor que a versão de 256 elementos, e como esta instabilidade esta relacionada ao método numérico utilizado pelo CYRANO, então, ainda assim, por estes motivos, a versão paralela é mais eficiente.

Tabela 3 – Tempo em segundos, gasto pelo código nas rotinas OUTPOW, GENERA3 e o tempo total, medido no cacau, fila gpu, caso com 481 elementos radiais.

	Tempos				*Speedup*		
Modos	Original	OpenBLAS	BLIS	Atlas	OpenBLAS	BLIS	Atlas
			GENERA3				
32	320,49	328,59	326,91	315,48	0,98	0,98	1.02
64	809,48	635,38	649,63	752,56	1,27	1,25	1.08
128	3.172,55	1.573,22	1.605,92	2.579,99	2,02	1,98	1.23
256	16.419,44	6.880,12	6.772,87	13.842,19	2,39	2,42	1.19
512	118.669,84	38.165,74	38.922,91	219.699,85	3,11	3,05	0.83
			OUTPOW				
32	578,39	47,01	54,66	47,05	12,30	10,58	12.29
64	1.412,16	110,80	116,49	113,01	12,74	12,12	12.50
128	4.513.97	350,05	354,77	349,70	12,90	12,72	12.91
256	16.592,59	1.287,44	1.337,59	1.286,39	12,89	12,40	12.90
512	62.947,85	4.923,88	5.011,12	4.937,07	12,78	12,56	12.75
			TOTAL				
32	1.052.54	527,16	762,73	514,70	2.00	1,38	2.04
64	2.396.87	917,99	1.109,90	1.039,47	2,28	2,16	2.31
128	7.907.16	2.137,97	2.417,55	3.150,26	3.70	3,27	2.51
256	33.344.97	8.489,57	9.098,93	15.459,52	3.93	3,66	2.16
512	182.142.54	43.614,20	44.840,48	214.217,76	4.18	4,06	0.55

Tabela 4 – Tempo em segundos, gasto pelo código nas rotinas OUTPOW, GENERA3 e o tempo total, medido no cacau, fila gpu, caso com 151 elementos radiais.

	Tempos				*Speedup*		
Modos	Original	OpenBLAS	BLIS	Atlas	OpenBLAS	BLIS	Atlas
			GENERA3				
32	129,15	128,52	122,05	123,59	1,00	1,06	1,05
64	311,90	250,72	271,13	284,60	1,24	1,15	1,10
128	1.163,59	598,92	861,96	931,47	1,94	1,35	1,25
256	6.261,70	2323,98	4.078,92	4.410,43	2,69	1,54	1,42
512	42.081,58	11977,02	23.882,97	26.604,60	3,51	1,76	1,58
			OUTPOW				
32	212,10	19,26	19,54	18,23	11,01	10,86	11,63
64	517,98	44,33	44,06	41,55	11,69	11,76	12,47
128	1.638,30	134,33	133,18	128,56	12,20	12,30	12,74
256	5.963,16	480,63	477,96	467,80	12,41	12,48	12,75
512	22.294,70	1.783,29	1.772,32	1.754,27	12,50	12,58	12,71
			TOTAL				
32	464,83	206,71	216,45	212,34	2,25	2,15	2,19
64	947,07	360,10	386,50	397,32	2,63	2,45	2,38
128	2.914,27	812,02	1.077,26	1.141,53	3,59	2,71	2,55
256	12.365,08	2.914,51	4.671,37	4.992,40	4,24	2,65	2,48
512	64.634,23	13.967,38	25.870,88	28.574,73	4,63	2,50	2,26

A BLIS, apesar de ter mantido a mesma performance que a OpenBLAS nos casos de 281 e 481, diminuiu pela metade se comparada com a OpenBLAS nos casos de 151 elementos. Como ela não ativa o paralelismo automaticamente, é possível que tenha sido executada todas as mais de três vezes em serial, pois, esta é a performance esperada se a execução dela fosse serial. Apesar disto, a BLIS se mantém melhor que a ATLAS e inferior a OpenBLAS, que conseguiu a melhor performance em todos os casos mo CYRANO.

6 Conclusão

A BLAS é uma excelente biblioteca para algoritmos que dependem da álgebra linear, sua interface possibilitou que ao longo dos anos, novas bibliotecas otimizadas a substituíssem, não por outra, mas por versões melhoradas da mesma, saindo de uma era de computadores de núcleos simples para computadores com multi-núcleos utilizando sempre o máximo do potencial de hardware de cada geração.

A ZGEMM manteve o desempenho esperado, que apesar de demorar significativamente mais no tempo de execução para matrizes de mesmo tamanho, manteve a quantidade de operações por segundo próxima aos da DGEMM. A BLIS se mostrou superior a ATLAS e a OpenBLAS neste caso, e por isto foi a implementada nos testes do CYRANO junto com a OpenBLAS e a Atlas.

Utilizar bibliotecas com padrões reconhecidos proporciona facilidades no aperfeiçoamento do código, de forma rápida fácil e eficiente, como foi obtido com a substituição da BLAS no CYRANO, então é importante observar a comunidade que estuda estes códigos conhecidos para atualizar estas rotinas e assim obter ganhos de desempenho sem muito esforço, no entanto, muitos programadores e pesquisadores desconhecem tais rotinas como a BLAS, LAPACK ou LINPACK e acabam fazendo suas próprias versões tornando trabalhoso uma futura atualização nestes procedimentos.

Os resultados dos códigos CYRANO, quando comparados com os da versão original, principalmente com os arquivos de entrada de ate 128 modos poloidais, se demonstraram relativamente iguais. Assim, a nova versão melhorada do código, que é até quatro vezes mais rápida para entregar os resultados, se torna uma opção viável para ganhar tempo nas simulações com aproximações próximas aos do código original.

Quanto ao desempenho, embora a nova rotina OUTPOW, quando paralelizada teve uma redução no tempo significativo, na ordem da quantidade de *threads* menor, ou seja, obteve um *speedup* linear, o *speedup* da execução total do código, em todos os casos estudados, ficou próximo ao obtido com a rotina GENERA, este comportamento foi previsto quando a análise do desempenho código foi executada, pois GENERA, conforme a quantidade de modos poloidais aumenta, passa a ter um consumo de tempo cada vez maior no tempo total do código. No entanto, utilizando as bibliotecas Atlas, OpenBLAS e BLIS, aliadas ao OpenMP, foi possível reduzir o tempo em ate quatro vezes menos, com reduções de tempo significativas, de dois dias e duas horas para doze horas, de um dia para seis horas e de dezoito horas para quase quatro horas de execução.

No entanto, um estudo mais acurado na rotina GENERA é necessário, pois,

apesar de existir um potencial de paralelismo nesta rotina, ela é muito complexa, e para paralelizar os procedimentos desta rotina de forma eficiente, é necessário executar os cálculos para cada elemento radial de forma paralela e independente, porém, o atual código possui muitas dependências nas iterações dos cálculos destes elementos.

Existe também uma necessidade de avaliar o método numérico para identificar os "limites" dos dados de entrada para o código, já que os resultados, principalmente no caso de 512 modos poloidais com 151 elementos radiais, demostraram uma instabilidade presente no código, que quando apenas a BLAS é substituída na compilação, as "saídas" em alguns casos são discrepantes.

Além disto, é necessária uma atualização para uma linguagem que permita o uso de memória dinâmica, pois, um dos principais limitações do código é o ajuste manual da memória que é utilizada em sua execução. Assim, com a melhoria obtida com a atualização da biblioteca BLAS no código e com os resultados obtidos, ainda existe espaço para paralelização e otimização, possibilitando a execução de modelos mais complexos que requeiram maior quantidade de memória.

Referências

AL., B. D. et. Plasma Sci. Technol. 9 312. 2007.

AL., E. A. L. et. Plasma Phys. Ctrl. Fusion 51(4). 2009.

AMARANTE-SEGUNDO G. S. ELFIMOV, A. G. R. L. G. R. M. O. K. R. L. A. M. A. RF antenna analysis with the ICANT code. **Fusion Engineering and Design**, v. 81, n. 19, p. 2205–2212, 2006. ISSN 09203796.

AMDAHL, G. M. Validity of the single processor approach to achieving large scale computing capabilities. **AFIPS spring joint computer conference, 1967**, p. 1–4, 1967.

AMRITKAR, A.; TAFTI, D.; LIU, R.; KUFRIN, R.; CHAPMAN, B. OpenMP parallelism for fluid and fluid-particulate systems. **Parallel Computing**, Elsevier B.V., v. 38, n. 9, p. 501–517, 2012. ISSN 01678191. Disponível em: <http://dx.doi.org/10.1016/j.parco.2012.05.005>.

BAUER, N.; BOSETTI, V.; HAMDI-CHERIF, M.; KITOUS, A.; MCCOLLUM, D.; Mé-JEAN, A.; RAO, S.; TURTON, H.; PAROUSSOS, L.; ASHINA, S.; CALVIN, K.; WADA, K.; VUUREN, D. van. Co2 emission mitigation and fossil fuel markets: Dynamic and international aspects of climate policies. **Technological Forecasting and Social Change**, v. 90, n. Part A, p. 243 – 256, 2015. ISSN 0040-1625. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0040162513002382>.

BRAMBILLA, M. The high-frequency constitutive relation of axisymmetric toroidal plasmas. **Plasma Physics and Controlled Fusion**, v. 41, n. 6, p. 775, 1999. Disponível em: <http://stacks.iop.org/0741-3335/41/i=6/a=307>.

BUDNY, R. V.; BERRY, L.; BILATO, R.; BONOLI, P.; BRAMBILLA, M.; DUMONT, R. J.; FUKUYAMA, A.; HARVEY, R.; JAEGER, E. F.; INDIRESHKUMAR, K.; LERCHE, E.; MCCUNE, D.; PHILLIPS, C. K.; VDOVIN, V.; WRIGHT, J. Benchmarking ICRF full-wave solvers for ITER. **Nuclear Fusion**, v. 52, n. 2, 2012. ISSN 00295515.

BUTLER, D. ITER keeps eye on prize. **Nature**, v. 502, n. 7471, p. 282–3, oct 2013. ISSN 1476-4687. Disponível em: <http://www.nature.com/news/iter-keeps-eye-on-prize-1.13957>.

DAGUM, L.; MENON, R. OpenMP: an industry standard API for shared-memory programming. **IEEE Computational Science and Engineering**, v. 5, n. 1, p. 46–55, 1998. ISSN 10709924. Disponível em: <http://ieeexplore.ieee.org/document/660313/>.

DONGARRA, J. J.; CROZ, J. D.; HAMMARLING, S.; HANSON, R. J. A proposal for an extended set of fortran basic linear algebra subprograms. **SIGNUM Newsl.**, ACM, New York, NY, USA, v. 20, n. 1, p. 2–18, jan. 1985. ISSN 0163-5778. Disponível em: <http://doi.acm.org/10.1145/1057935.1057936>.

DUMONT, R. Variational approach to radiofrequency waves in magnetic fusion devices. **Nuclear Fusion**, v. 49, n. 7, p. 075033, 2009. Disponível em: <http://stacks.iop.org/0029-5515/49/i=7/a=075033>.

GROPP, W.; E, L. (Ed.). **Using OpenMP: Portable Shared Memory Parallel Programming**. MIT Press, 2008. v. 10. 353 p. (Scientific and Engineering Computation Series, v. 10). Disponível em: <http://books.google.com/books?hl=en{\%}7B{&}{\%}7Dlr={\%}7B{&}{\%}7Did=MeFLQSK>.

GUSTAFSON, J. L. Reevaluating Amdahl's Law. **Communications of the ACM**, 1988.

HANSON F. T. KROGH, C. L. L. R. J. A Proposal for Standard Linear Algebra Subprograms. **JET PROPULSION LABORATORY, Pasadena, Calif**, 1973. Disponível em: <https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19740005175.pdf>.

INPE / Notícias - Pesquisadores do INPE integram comitê da Rede Nacional de Fusão. 2017. <http://www.inpe.br/noticias/noticia.php?Cod_Noticia=1155>. Acesso em: 03 de nov. 2017.

ITER. 2017. <https://www.iter.org/factsfigures>. Acesso em: 03 de nov. 2017.

JAEGER L. A. BERRY, E. D. e. a. E. F. Physics of Plasmas 8, 1573. 2001.

JAUN K. APPERT, J. V. L. V. A. Global waves in resistive and hot tokamak plasmas . **Computer Physics Communications**, p. 153–187, 1995.

KELEFOURAS, V.; KRITIKAKOU, A.; GOUTIS, C. A Matrix-Matrix Multiplication methodology for single/multi-core architectures using SIMD. **Journal of Supercomputing**, v. 68, n. 3, p. 1418–1440, 2014. ISSN 15730484.

KOKULAN, N.; LAI, C. H. Inducing temporal parallel properties into time dependent problems. **Proceedings - 11th International Symposium on Distributed Computing and Applications to Business, Engineering and Science, DCABES 2012**, n. 5, p. 5–8, 2012.

Kruecken, T. FISIC - a full-wave code to model ion cyclotron resonance heating of tokamak plasmas. **Max-Planck-Institut fuer Plasmaphysik, Garching (Germany, F.R.)**, n. IPP–5/23, 1988.

LAMALLE, P. **Nonlocal theoretical generalization and tridimensional numerical study of the coupling of an ICRH antenna to a tokamak plasma**. 192 p. Tese (Doutorado) — Faculté des Sciences de l'Université de Mons, École Royale Militaire - 1040 Brussels, Belgium - Koninklijke Militaire School, 1994.

LAMALLE., P. U. On the radiofrequency response of tokamak plasmas. **Plasma Physics and Controlled Fusion**, v. 39, n. 9, 1997.

LAWSON, C. L.; HANSON, R. J.; KINCAID, D. R.; KROGH, F. T. Basic linear algebra subprograms for fortran usage. **ACM Trans. Math. Softw.**, ACM, New York, NY, USA, v. 5, n. 3, p. 308–323, set. 1979. ISSN 0098-3500. Disponível em: <http://doi.acm.org/10.1145/355841.355847>.

LAWSON, J. D. Power from Nuclear Fusion. **Nature**, Nature Publishing Group, v. 180, n. 4590, p. 780–782, oct 1957. ISSN 0028-0836. Disponível em: <http://www.nature.com/nature/journal/v180/n4590/pdf/180780a0.pdf>.

LEISERSON, C. E.; MIRMAN, I. B. **How to survive the multicore software revolution (or at least survive the hype)**. 2008. Cilk Arts.

Llewellyn Smith, C.; WARD, D. The path to fusion power. **Philosophical transactions. Series A, Mathematical, physical, and engineering sciences**, v. 365, n. 1853, p. 945–56, apr 2007. ISSN 1364-503X. Disponível em: <http://rsta.royalsocietypublishing.org/content/365/1853/945>.

MAX Planck Institute For Plasma Physics. 2017. <http://www.ipp.mpg.de/3871973/ICRH>. Acesso em: 03 de nov. 2017.

MOORE, G. E. **Cramming more components onto integrated circuits**. 1965. Electronics Magazine, pages 82–85.

MOROZOV, S.; MATHUR, S. Massively Parallel Computation Using Graphics Processors with Application to Optimal Experimentation in Dynamic Control. **Computational Economics**, v. 40, n. 2, p. 151–182, 2012. ISSN 09277099.

MULLER, J.-M.; BRISEBARRE, N.; DINECHIN, F.; JEANNEROD, C.-P.; LEFÈVRE, V.; MELQUIOND, G.; REVOL, N.; STEHLÉ, D.; TORRES, S.; DINECHIN, F. de; JEANNEROD, C.-P.; LEFÈVRE, V.; MELQUIOND, G.; REVOL, N.; STEHLÉ, D.; TORRES, S. **Handbook of Floating-Point Arithmetic**. [s.n.], 2010. v. 7. 956–963 p. ISSN 15577317. ISBN 978-0-8176-4704-9. Disponível em: <http://link.springer.com/10.1007/978-0-8176-4705-6>.

NETLIB. 2017. <http://www.netlib.org>. Acesso em: 03 de nov. 2017.

OANCEA, B. Accelerating R with high performance linear algebra libraries. **CoRR**, abs/1508.0, 2015. Disponível em: <http://arxiv.org/abs/1508.00688>.

PERKINS, L. J.; BETTI, R.; LAFORTUNE, K. N.; WILLIAMS, W. H. Shock Ignition : A New Approach to High Gain Inertial Confinement Fusion on the National Ignition Facility. v. 045004, n. July, p. 1–4, 2009.

PETROV, A. B. Y.; MONAKHOV, I. Coupled full-wave and ray-tracing numerical treatment of mode conversion in a tokamak plasma. **Physics of Plasmas**, v. 7, n. 3, 1999.

R. Bilato, N. Bertelli, M. Brambilla, R. Dumont, E.F. Jaeger, T. Johnson, E. Lerche, O. Sauterk, D. V. E.; VILLARD, L. Status of the Benchmark Activity of ICRF Full-wave Codes within EUROfusion WPCD and beyond. **21st Topical Conference on Radio Frequency Power in Plasmas**, n. 633053, 2015.

STACEWICZ, P.; WLODARCZYK, A. Modeling in the Context of Computer Science - A Methodological Approach. **Studies in Logic, Grammar and Rhetoric**, v. 20, n. 33, p. 155–179, 2010.

STONE DAVID GOHARA, G. S. J. E. OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. **Comput Sci Eng.**, p. 66–72, 2010.

SVIDZINSKI, V. A.; KIM, J. S.; SPENCER, J. A.; ZHAO, L.; GALKIN, S. A.; EVSTATIEV, E. G. Hot plasma dielectric response to radio-frequency fields in inhomogeneous magnetic field. **Physics of Plasmas**, v. 23, n. 11, 2016. ISSN 10897674. Disponível em: <http://dx.doi.org/10.1063/1.4966638>.

TCABR Tokamak | TCABRJE : Joint Experiment on TCABR. 2017. <http://tcabrje.if.usp.br/node/35>. Acesso em: 03 de nov. 2017.

U.S. Energy Information Administration. **International Energy Outlook 2016**. [s.n.], 2016. v. 0484(2016). 1–2 p. ISSN 0163660X. ISBN 2025866135. Disponível em: <www.eia.gov/forecasts/ieo/pdf/0484(2016).pdf>.

VAJDA, A. **Programming Many-Core Chips**. [s.n.], 2011. ISSN 1098-6596. ISBN 978-1-4419-9738-8. Disponível em: <http://link.springer.com/10.1007/978-1-4419-9739-5>.

V.L., V. 3D FULL WAVE CODE MODELLING OF ICRF PLASMA HEATING IN LARGE STELLARATORS. v. 25, p. 1541–1544, 2001.

WHALEY, R. C.; DONGARRA, J. **Automatically Tuned Linear Algebra Software**. [S.l.], 1997. URL : `http://www.netlib.org/lapack/lawns/lawn131.ps`.

WHITEHEAD, N.; FIT-FLOREA, A. Precision & Performance : Floating Point and IEEE 754 Compliance for NVIDIA GPUs. **NVIDIA white paper**, v. 21, n. 10, p. 767–75, 2011. ISSN 03331024. Disponível em: <http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/Floating_Point_on_NVIDIA_GPU_White_Paper.pdf\nhttp://developer.nvidia.com/content/precision-performance-floating-point-and-ieee-754-compliance-nvidia-gpus>.

WILLENBRING, J. M.; LABORATORIES, S. N. Replicated Computational Results (RCR) Report for " BLIS : A Framework for Rapidly Instantiating BLAS Functionality ". v. 41, n. 3, p. 0–3, 2015.

XE, C. S. Full throttle. p. 1–42, 2013.

Anexos

ANEXO A – Arquivo de entrada exemplificado

```
'JET ' #nome do arquivo
#variáveis que identificam as saídas que irão ser escritas
&NAMOUT
machin='PC',
wridat=.false., wrplta=.false., wrisol=.false.,
wrifou=.false., nfouwr=0, ifouwr=3,4,5,6,8,10,
replay=.false.,
distes=.false.,
unitgl=40, unitrh=41, unitem=42,
numdis=.false.,
keepso=.false.,ploold=.false.,nolifi=60,nolofi=60,
unqlfp=50, undirk=51,
&END
#dados sobre os arquivos que irão ser plotados
&NAMPLO
igrdev=0,
transp=.false.,notrsp=25,
outgau=.true.,ninter=4,ncomp=4,spline=.false.,splfac=0.011,
xcm=24.,ycm=20.,
xmin=666.,xmax=666.,ymin=666.,ymax=666.,
nkurv=2,isk=2,jsk=2,
plosol=.true.,plopow=.true.,
plot2d=.true.,
nploth=256,
niso=15,
ploty(1,1)= .true., .true., .false., .true., .true., .false.,
.true., .true., .false.,
```

6*.false.,

.true., .true., .false.,

2*.false., .true., 2*.false.,.true.,

.false.,.false.,.true.,

nfiell=20,

ploana=.false.,

plosta=.false.,ploeve=.false.,

&END

For future implementation in namelists:

READ_GENERAL = .false.,

READ_FLUX = .false.,

READ_PROFILES = .false.,

SAME_TEMP = .false.,

SAME_DENS = .false.,

#Dados da geometria do Tokamak

&NAMGEO

dshape=.true., geneq=.false.,

cyl= .false.,

r0= 2.96d0,

z0= 0.d0,

polsym=.true.,

cokpco=.false.,

&END

#Dados sobre o campo magnético

&NAMMAG

kappa=1.00, shsh0=0.0, ishsht=1, delta=0.300,

b0=3.3, # Centro do campo magnético

ipl=2.0d6, alpha=3.6d0, # Corrente do plasma

nintps=10,

&END

```
# dados sobre o plasma
&NAMPLA
vacuum=.false.,2*.true.,
coldpl(1)=.false.,2*.true.,
flrops(1)=.true.,2*.false., # Raio de Larmor finito de ordem 0 e 2
plstep(1)=.true.,
sccol(1)=.false.,
damp(1,1)=3*0.d0,
quasis=.false.,
nspec=3, # Numero de especies
my_nspgdr=0, my_ispgdr(1)=2,
CONCFAC=1.d0
nspgdr=0, ispgdr(1)=2, isotro(1)=.true., maxwdr(1)=.true., rqlral=.false.,
relati(1)=3*.false.,
ispres=2,nha=1,xbou=8.d0,rratio=0.25,
zch(1)= -1., 1., 1., 1., 16., # numero atômico do átomo
amass(1)= 5.446623d-04, 2.000001d0, 2.d0, 1.d0, 40.d0, # massa do átomo
t0(1,1)= 5.5d3, 55.0d3, 5.5d3, 5.5d3, 5.5d3, # temperatura inicial
tb(1,1)= 0.1d3, 0.1d3, 0.1d3, 0.1d3, 0.9d2
itexp(1,1)= 2.0, 2.0, 2.0, 2.0, 2.4
v0alph= 1.84e6, udrift=1.d6,
equidn(1)= 3*0,
usenma=.false.,
n0(1,1)=0.25d0, # densidade do plasma (ne)
nb(1,1)=0.05d0,
spefra(1,1)= 1., 10d-2, 90d-2, 5d-2, 0.1d-2
idexp(1)= 1.,
i_special(1)= 0, 1, 0, 0, 1,
ninscr=0., tinscr=0.,
&END
```

```
#Dados da antena
&NAMANT
fregag=25.0d6, # frequencia RF
ntoant=4, samant=.true., antyn=4*5, anetyn=4*1, irbant(1)=4*2,
feeder(1)=4*.true.,
thea1(1)=-3.14159265358979323 85d0, thea2(1)=3.14159265358979323 85d0,
zaa(1)=-0.1, zaa(2)=0.1, dza(1)=0.04,
monoto=.true., motoan(1)=27, kparze=1.d-8, # modos toroidais N
motov1=0,motov2=0,motovs=-10,
ktoan(1)=3.,ktoans=0.,ntotor=1,
monomo=.false.,moant(1)=3,
modva1=-12,modva2=12, # modos poloidais considerados, de modva1 a modva2
tancur(1)=(1.d0,0.d0),(-1.d0,0.d0),(1.d0,0.d0),(-1.d0,0.d0),
normap=.false.,rfpow=1.0d6,
falen(1)=4*.false.,
raprio(1)=4*0.d0,laprio(1)=4*270.d-9,caprio(1)=4*141.d-12,
nscree=0,
walres=72.d-8,
&END
#Dados das regiões radiais
&NAMREG
nreg=3, ns(1)=3,1,1,
rx0m(0) = 0., 1.0, 1.05, 1.27, # raios: plasma, antena, parede
dobtyp(2) = -2,
&END
#subregioes
&NAMSUB
istyp(1) = 1,1,1,2,2, # onde estão as subregioes, 1 = plasma, 2 = vácuo
sx0m(0) = 0., 0.03, 0.9, 1.0, 1.05, 1.27, # plasma, antena, parede
iele(1) = 1, 100, 10, 20, 20 , # elementos radiais subdivididos, neste caso 151
```

```
elementos
# sendo 1+100+10 no plasma e 20+20 no vácuo
        &END
        #
        &NAMPOM
        mstud1=-2,mstud2=2,mstust=1,
        &END
        #dados sobre o sistema
        &NAMSYS
        soldoc=.true.,
        autome=.false., killmo=.true., elpwl=4., elpdl=4., damcut=1.d-4,
        lelmin=5.d-4, leledg=2.d-3,
        scale=.false.,
        ncrot=64,
        klim=24, # numero de acoplamentos poloidais considerados (modva2-modva1)
        nftnr=64,
        ncresp=800,
        ngauss=4,
        ngaux=5, ngauv=4,
        wripoi=.false., wriblo=.false.,
        wbc3=.false., bulfce=.true.,
        bcinb1=.true.,
        nabcap=.true., bccase=0,
        stomat=.false.,
        recalc_total=.false., onlyab=.false.,
        rawel1=.false.,
        recalc_by_species=.false.,
        giplas=.true., gicurl=.true., gigdr=.false.,
        &END
```

ANEXO B – Curiosidades

B.1 Crosscompilation

"Crosscompilation" ou compilação cruzada, é uma técnica suportada por diversas linguagens de programação, que permite a comunicação entre códigos escritos em diferentes linguagens.

As estruturas de C/C++ e fortran são muito semelhantes, não tendo apenas ponteiros de C/C++ em fortran, mas como fortran usa passagem de parâmetro por referência, todos os dados entregues por fortran a C/C++ é através de ponteiros.

Além disto, todas as rotinas de fortran terminam com "_" quando acessadas de C e todas as rotinas em C que serão chamadas em fortran devem estar em letras minusculas e terminar em "_", sendo acessadas sem o uso do "_" e em letras maiúsculas ou minusculas em fortran;

B.2 NAMELIST

É um padrão de arquivo definido para uso pela linguagem fortran70. O arquivo tem formato de texto do linux, e carrega para as variáveis no código em execução, com os mesmos nomes que estão nos arquivos, os valores também definidos nos mesmos arquivos.

Não existe oficialmente uma biblioteca para leitura destes dados em C/C++.

B.3 COMMONS

É uma forma de definir um conjunto de variáveis globais em fortran.

É equivalente a "struct" global de C, e de fato, se alguém quiser acesso a estas variáveis num "crosscompilation" com a linguagem C, é necessário definir as "structs" com o mesmo nome que os "commons" e com os mesmos nomes e tipos das variáveis em letras minusculas, pois, como fortran não é "case sencive" (não diferencia letras minusculas e maiúsculas) os "tokens" estão em letras minusculas no dicionário, o que implica que todas as rotinas e dados em fortran estão em letras minusculas para acesso em outras linguagens em compilação cruzada.