UNIVERSIDADE FEDERAL FLUMINENSE
ESCOLA DE ENGENHARIA
MESTRADO EM ENGENHARIA ELÉTRICA E DE TELECOMUNICAÇÕES

SIMULAÇÃO EFICIENTE DE CONTROLADORES DE SISTEMAS DE POTÊNCIA DE GRANDE PORTE UTILIZANDO COMPILAÇÃO EM LINGUAGEM C

Thiago José Barbosa da Rocha

Niterói
Julho de 2020

UFF - PROGRAMA DE PÓS-GRADUAÇÃO DE ENGENHARIA ELÉTRICA E DE TELECOMUNICAÇÕES

SIMULAÇÃO EFICIENTE DE CONTROLADORES DE SISTEMAS DE POTÊNCIA DE GRANDE PORTE UTILIZANDO COMPILAÇÃO EM LINGUAGEM C

Thiago José Barbosa da Rocha

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e de Telecomunicações da Universidade Federal Fluminense, como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica e de Telecomunicações.

Orientador: Prof. Sergio Gomes Junior, D.Sc.

Niterói
Julho de 2020

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

R672s Rocha, Thiago José Barbosa da
Simulação Eficiente de Controladores de Sistemas de Potência de Grande Porte Utilizando Compilação em Linguagem C / Thiago José Barbosa da Rocha ; Sergio Gomes Junior, orientador. Niterói, 2020.
185 f. : il.

Dissertação (mestrado)-Universidade Federal Fluminense, Niterói, 2020.

DOI: http://dx.doi.org/10.22409/PPGEET.2020.m.10618812709

1. Dinâmica de Sistemas de Potência. 2. Controladores. 3. Computação de Alto Desempenho. 4. Transitórios Eletromagnéticos. 5. Produção intelectual. I. Gomes Junior, Sergio, orientador. II. Universidade Federal Fluminense. Escola de Engenharia. III. Título.

CDD -

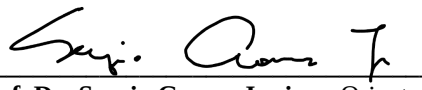
Bibliotecário responsável: Sandra Lopes Coelho - CRB7/3389

THIAGO JOSÉ BARBOSA DA ROCHA

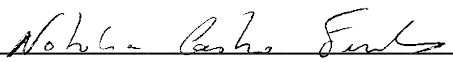
SIMULAÇÃO EFICIENTE DE CONTROLADORES DE SISTEMAS DE POTÊNCIA DE GRANDE PORTE UTILIZANDO COMPILAÇÃO EM LINGUAGEM C

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e de Telecomunicações da Universidade Federal Fluminense como requisito parcial para a Obtenção do Grau de Mestre em Engenharia Elétrica e de Telecomunicações.
Área de concentração: Sistemas de Energia Elétrica.

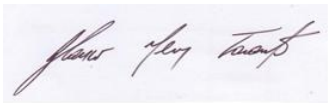
BANCA EXAMINADORA



Prof. Dr. Sergio Gomes Junior - Orientador
Universidade Federal Fluminense - UFF



Profª. Dra. Natalia Castro Fernandes
Universidade Federal Fluminense - UFF



Prof. Dr. Glauco Nery Taranto
Universidade Federal do Rio de Janeiro - UFRJ

Dr. Thiago José Masseran Antunes Parreiras
Centro de Pesquisas de Energia Elétrica - CEPEL

Niterói
(julho/2020)

Dedicatória

Por todos aqueles que se dedicam diuturnamente em ajudar os mais necessitados e por todos os que durante essa missão tiverem suas vidas ceifadas. Que seus exemplos de coragem e de determinação sejam um farol de esperança na construção de um novo mundo.

Agradecimentos

A Deus, sobre todas as coisas. Que eu seja digno de ser Seu filho.

Aos meus pais, que estiveram comigo desde os meus primeiros passos e sempre me apoiaram.

À minha irmã. Minha antílope favorita!

Ao Sergio Gomes, meu orientador. Uma rara e preciosa combinação de uma elevada inteligência e de uma grande humildade.

Ao Tiago Amaral. É uma honra e uma felicidade poder aprender todos os dias com você. Obrigado pela amizade, meu amigo. Você sempre terá minha profunda admiração e respeito.

Ao Thiago Masseran. Sua didática é impressionante. Obrigado pela paciência explicando minhas muitas dúvidas. Minha hora chegou!

Ao Nicolas Abreu. Obrigado por sua disponibilidade para me explicar os detalhes de funcionamento do Anatem.

Ao Bruno Borba. Obrigado pela confiança em mim ao me indicar ao Sergio Gomes como bolsista. Sem você não estaria onde estou hoje.

Aos meus amigos do Cepel: Rodrigo Godim, Thomas Campello, Nicholas Leite, Vinícius Lopes, Milon Pereira e José Guilherme.

Ao CEPEL, um Centro de Pesquisa que é um exemplo de excelência.

Ao Programa de Pós-Graduação em Engenharia Elétrica e de Telecomunicações da UFF pela oportunidade concedida.

“No man is free who is not master of himself”

Autor Desconhecido

“E t*endo amado os seus que estavam no mundo, amou-os até o fim”*

João 13,1

Resumo

Nesta dissertação desenvolveu-se em linguagem C++ um compilador de Controladores Definidos pelo Usuário (CDUs), denominado UDCC, que pudesse ser integrado a outros programas. A partir de um arquivo de entrada contendo CDUs, o UDCC gera um código em linguagem C equivalente em termos de dados, que é então compilado em linguagem de máquina e gerada uma DLL. Na sequência, essa DLL é executada pelo programa no qual o UDCC foi integrado e, desse modo, viabilizando a simulação eficiente desses controladores. Nos algoritmos convencionais de programas computacionais de simulação dinâmica de sistemas de potência, os dados dos controladores são armazenados em memória e cada bloco do controlador, contendo uma ou mais equações algébricas e/ou diferenciais, é processado sequencialmente na solução iterativa como se fosse uma linguagem de programação interpretada. Diferentemente, a metodologia deste trabalho utiliza o conceito de compilação de uma linguagem de programação, para transformar os dados de controladores em código C.

Um dos objetivos é o ganho de eficiência computacional, pois a simulação é realizada por um código dedicado exclusivamente à solução das equações, o que exclui boa parte do processamento que era necessário na forma interpretada convencional. Outro objetivo é facilitar a inclusão do recurso de simulação de controladores em múltiplos programas computacionais, com um esforço reduzido de integração do UDCC em cada programa.

Utilizou-se o UDCC inicialmente de forma independente, em um simulador protótipo simplificado contendo apenas a solução dos controladores e sem os demais componentes do sistema elétrico. Em seguida, o UDCC foi integrado ao AnaHVDC, que é o novo programa computacional do Cepel para simulação fasorial de transitórios eletromagnéticos, com ênfase na análise de elos de corrente contínua. O AnaHVDC não possuía o recurso de simulação de controladores e, graças à integração realizada neste trabalho, isto passou a ser possível.

Utilizou-se o programa Anatem, que utiliza a metodologia convencional, para comparações de desempenho computacional e de validação dos resultados de simulação dos controladores isolados utilizando as topologias dos controladores presentes no Sistema Interligado Nacional (SIN). Apresentou-se ainda resultados de simulação do AnaHVDC com o UDCC integrado em um sistema de pequeno porte (BenchCA) e de um caso de Plano de Ampliações e Reforços do SIN para o ano de 2023 (PAR2023), com validações envolvendo os programas Anatem, PacDyn e PSCAD. Os resultados consistentes e o bom desempenho computacional corroboram a importância prática do trabalho.

Palavras-chaves: Dinâmica de sistemas de potência, Controladores, Computação de alto desempenho, Fasores dinâmicos, Transitórios eletromagnéticos, Estabilidade.

Abstract

In this work, an optimized compiler of User Defined Controllers (CDU), called UDCC ("User Defined Controller Compiler"), was developed in C ++, in a way that it could be easily integrated into another program. The compiler translates the data from a file of CDUs into a C code that is then compiled to generate a DLL. After that, this DLL can be used by the program in which UDCC was integrated to efficiently solve these controllers. In conventional algorithms of computer programs for dynamic simulation of power systems, the data from the controllers is stored in memory and each block, containing one or more algebraic and/or differential equations, is processed sequentially during the iterative solution as if it were an interpreted programming language. In contrast, the methodology of this work uses the concept of compilation to a programming language, in order to transform the controller data into C code.

One of the objectives is to increase the computational efficiency, since the simulation is performed by a code that is dedicated exclusively to the solution of the equations, which excludes much of the processing that was necessary in the conventional interpreted approach. Another objective is to facilitate the inclusion of the controller simulation feature in multiple computer programs, with a small effort to integrate the UDCC into each program.

Initially, UDCC was used independently in a simplified prototype simulator containing only the solution of the controllers without the other components of the electrical system. In a second stage, UDCC was integrated into AnaHVDC, which is Cepel's new computer program for phasor simulation of electromagnetic transients, with emphasis on the analysis of the dynamic behavior of direct current links. AnaHVDC did not have the resource to simulate controllers and, thanks to the integration carried out in this work, it became possible.

Cepel's Anatem program, which relies on the conventional methodology, was used to compare computational performance and also to validate the simulation results of isolated controllers using the topologies of the controllers of the Brazilian Interconnected Power System (SIN). With UDCC integrated into AnaHVDC, two other simulations were also carried out: one considering a small system (BenchCA) and another one of the case of Planning for Expansion and Reinforcement of the SIN for the year 2023 (PAR2023). Both of them were validated against Anatem, PacDyn and PSCAD. The consistent results and the good computational performance corroborate the practical importance of this work.

Keywords: Power system dynamics, Control systems, High Performance Computing, Dynamic Phasors, Electromagnetic Transients, Stability.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

Anafas	–	*Software* desenvolvido pelo CEPEL para Análise de Faltas Simultâneas
AnaHVDC	–	*Software* desenvolvido pelo CEPEL para Análise de elos HVDC
AnaRede	–	*Software* desenvolvido pelo CEPEL para Análise de Redes Elétricas
Anatem	–	*Software* desenvolvido pelo CEPEL para Análise de Transitórios Eletromecânicos
ANEEL	–	Agência Nacional de Energia Elétrica
ATP	–	*Alternative Transient Program*, *software* de simulação de transitórios eletromagnéticos.
CA	–	Corrente Alternada
CC	–	Corrente Contínua
CDU	–	Controlador Definido pelo Usuário
CDUEdit	–	*Software* desenvolvido pelo CEPEL para edição de CDUs
CE	–	Controlador Estático
Cepel	–	Centro de Pesquisas de Energia Elétrica
CPU	–	*Central Processment Unit*
CV	–	Coeficiente de Variação
DLL	–	*Dynamic-Link Library*
DP	–	Desvio Padrão Populacional
EPE	–	Empresa de Pesquisa de Energia Elétrica
FACTS	–	*Flexible AC Transmission Systems*
FT	–	Função de Transferência
HVDC	–	*High Voltage Direct Current*
IGBT	–	*Insulated Gate Bipolar Transistor*
JIT	–	*Just-in-time*
ONS	–	Operador Nacional do Sistema Elétrico
PacDyn	–	*Software* desenvolvido pelo CEPEL para Análise e Controle de Oscilações Eletromecânicas

PAR	–	Plano de Ampliações e Reforços nas Instalações de Transmissão do SIN
PDE	–	Plano Decenal de Expansão de Energia Elétrica
PLT	–	Arquivo de Plotagem em formato do Cepel gerado pelos simuladores
PSCAD	–	*Power System Computer Aided Design, software* de simulação de transitórios eletromagnéticos.
PSS	–	Estabilizador de Sistema de Potência
RAT	–	Regulador Automático de Tensão
RAV	–	Regulador Automático de Velocidade
SEP	–	Sistema Elétrico de Potência
SIN	–	Sistema Interligado nacional
STATCOM	–	*Static synchronous compensator*
TCC	–	*Tiny C Compiler*
TCSC	–	*Thyristor Controlled Series Capacitor*
UDCC	–	*User-Defined Controller Compiled*
UDCrun	–	Simulador protótipo desenvolvido para simulação de CDUs isolados
DRAM	–	*Dynamic Random Access Memory*
IDE	–	*Integrated Development Environment*

Capítulo 1 - Introdução

O Brasil é o quarto maior país do mundo em área [1] e possui uma rede de transmissão de energia elétrica altamente complexa e ramificada, conforme Figura 1, que foi obtida no site do ONS. As linhas de transmissão atuais são representadas por linhas cheias e as que se espera que sejam incorporadas até o ano de 2024 são indicadas por linhas pontilhadas.

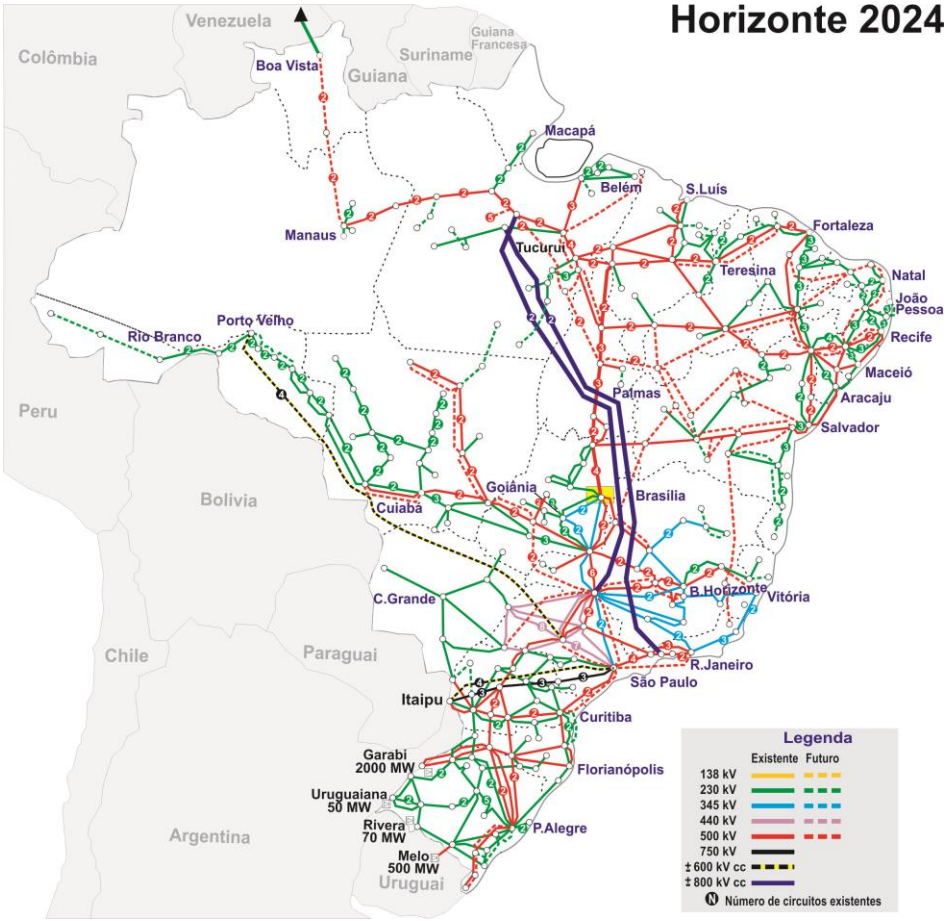


Figura 1 – Mapa do SIN [2]

Nos últimos anos a rede brasileira vem experimentando forte expansão de sua estrutura, sob coordenação da EPE por meio dos Planos Decenais de Expansão de Energia Elétrica (PDE), sendo o último o de 2029 [3]. O objetivo dos Planos é indicar as perspectivas de expansão do SIN considerando um horizonte de 10 anos e comtemplando tanto a geração como a transmissão de energia elétrica [3].

Um dos marcos dessa expansão é a integração da usina hidrelétrica de Belo Monte ao SIN. Localizada na bacia do Rio Xingu, próxima da cidade de Altamira (PA), ela possui potência instalada de 11.233 MW, constituindo-se a maior usina integralmente nacional [4].

Além disso, é notável a crescente importância da transmissão em corrente contínua no Brasil. Na última década se destacam dois projetos de largas proporções. O primeiro é o elo HVDC do Rio Madeira em tensão de ± 600 kV. Ele interliga Porto Velho (RO) à Araraquara (SP) e estende-se por 2.350 km [5]. O segundo projeto, mais recente, é referente às duas linhas HVDC que escoam a energia de Belo Monte, localizado no Rio Xingu, próximo à Altamira (PA). Uma das linhas se estende até Ibiraci (MG) e a outra até o Rio de Janeiro (RJ) [6]. A tensão de cada linha é de ± 800 kV e a potência de 4.000 MW [3].

Dessa forma, é essencial avaliar o impacto da inserção dos elos HVDC no SIN, especialmente no que tange à estabilidade do sistema. Um importante estudo nessa área é quanto à análise da falha de comutação, que acontece quando uma das válvulas do conversor HVDC que deveria desligar permanece conduzindo [7]. Como consequência a transmissão de energia é interrompida temporariamente e transitórios podem ser gerados no sistema [8]. Essas falhas de comutação são geralmente causadas pelo afundamento da tensão [9].

Nos estudos dinâmicos tradicionais, devido à complexidade e à extensão da rede brasileira, se recorre a programas computacionais de simulação, que podem ser divididos esquematicamente em dois grupos: aqueles voltados para simulação de transitórios eletromecânicos e aqueles que tratam da simulação de transitórios eletromagnéticos. Nos eletromecânicos é viável considerar todas as barras e componentes do sistema. Uma das desvantagens é que as válvulas de conversores de elos de corrente contínua e, consequentemente, a representação de suas falhas de comutação são simplificadas. Para uma melhor modelagem desses equipamentos e para detecção mais precisa de falhas de comutação é necessária uma representação mais completa, que é usada em estudos de transitórios eletromagnéticos. Entretanto, devido ao nível de detalhamento, é inviável preencher o número altamente extenso de dados necessários para representar todas as barras e componentes do sistema.

Dessa forma, essas duas classes de programas são insuficientes para uma análise adequada das falhas de comutação de elos. Nesse sentido, o Cepel buscou desenvolver um novo programa considerando um modelo completo da rede com todos as suas barras e componentes e, conjuntamente, considerando transitórios de natureza eletromagnética para

simulação trifásica de múltiplos elos HVDC[1], voltado para a identificação de falhas de comutação.

Esse novo programa, denominado AnaHVDC, se vale de uma modelagem por fasores dinâmicos, que se diferenciam dos fasores tradicionais por considerarem equações diferenciais na modelagem da rede elétrica e por serem variantes no tempo [10][2]. Um sistema CA em regime permanente pode ser descrito por ondas puramente senoidais ou, de forma equivalente, por fasores tradicionais. Quando se aplica um distúrbio, a forma de onda deixa de ter a forma senoidal, podendo ser interpretada como representada transitoriamente por fasores que variam no tempo [11]. A partir dessa nova perspectiva cada modelo da rede precisa ser equacionado seguindo sua representação por fasores dinâmicos [12]. A partir do AnaHVDC, contorna-se o problema de preenchimento de dados mencionado anteriormente[3]. Isso porque a base de dados dos equipamentos pode ser importada de outros programas, como o Anarede [13] e o Anatem [14]. Quanto à inicialização, no AnaHVDC ela se dá automaticamente a partir do arquivo de fluxo de potência do Anarede [11]. Isso também é uma vantagem importante em relação a outros programas em que a inicialização consiste em simular o sistema até ser encontrado o seu regime permanente e apenas depois começar a simulação propriamente dita. Em sistemas instáveis, acaba que este tipo de inicialização é inviável, havendo esta dificuldade de precisar ajustar os parâmetros até atingir a estabilidade.

Um aspecto importante do AnaHVDC é a possibilidade de utilização tanto de modelos de topologia fixa (*built-in*) como também de topologia genérica, definida pelo usuário do programa. A partir de um Controlador Definido pelo Usuário (CDU) é possível que o analista modele o seu controle com o grau de detalhamento adequado ao seu estudo. Dessa forma, por exemplo, a partir de um CDU é possível modelar um regulador de tensão que pode ser acoplado a uma máquina síncrona. O mesmo acontece para reguladores de velocidade, equipamentos estabilizadores e elos CC. Há mais de 400 controladores definidos pelo usuário na base do SIN, o que demonstra a importância dos CDUs no contexto dos estudos dinâmicos e, particularmente, no AnaHVDC.

A possibilidade de modelagem de controles por CDUs é uma ideia importada do Anatem. Neste programa a solução dos CDUs constitui-se a etapa mais exigente

[1] Destaca-se que, no presente estágio de desenvolvimento, o AnaHVDC apenas possui modelagem trifásica da rede elétrica, mas ainda não utiliza modelos trifásicos de elos de corrente contínua e ainda não realiza simulações de falha de comutação, que se encontram em desenvolvimento.
[2] O Anatem, por exemplo, possui apenas equações algébricas na modelagem da rede.
[3] Refere-se à virtual inviabilidade do preenchimento do número elevado de dados nos programas tradicionais de simulação de transitórios eletromagnéticos.

computacionalmente e, por isso, a mais demorada, o que cria um gargalo de tempo, uma vez que os CDUs precisam ser lidos em tempo de execução, interpretados e solucionados.

1.1 Motivação

O Anatem por trabalhar com transitórios eletromecânicos possui um passo de integração da ordem do milissegundo. Esse é um passo suficientemente alto para que a solução dos CDUs se dê em um tempo ainda aceitável em termos absolutos apesar do gargalo de tempo criado. Entretanto, essa questão é crítica no caso do AnaHVDC, que trabalha com transitórios eletromagnéticos e, por isso, com um passo de integração da ordem da dezena de microssegundos. O resultado é que o passo de integração no AnaHVDC é da ordem do centésimo do passo do Anatem, o que indica a dimensão do impacto da solução dos CDUs.

Portanto, em síntese, a principal motivação para este trabalho foi o fato do método usado pelo Anatem para solução de CDUs não ser suficientemente eficiente para ser aplicado ao AnaHVDC. Além disso, a otimização do Anatem, em face do crescente tamanho do arquivo de CDUs do SIN, também foi uma motivação. Por fim, o desenvolvimento de um solucionador eficiente de CDUs que fosse independente e integrável a outros futuros programas computacionais adiciona uma terceira motivação.

1.2 Objetivo

Esta dissertação tem como objetivo desenvolver um compilador de CDU otimizado, denominado UDCC (“*User Defined Controller Compiler”*), e que possa ser usado integrado a outros programas de simulação. Conforme ilustrado na Figura 2, o UDCC pode ser integrado, por exemplo, ao Anatem e ao AnaHVDC, onde os resultados da solução dos controladores atuarão nos equipamentos associados, ou pode ser acoplado ao simulador simplificado UDCrun, que foi desenvolvido nesta dissertação, com o objetivo de simular os controladores de forma isolada, sem atuação em equipamentos controlados. Em programas de simulação, o uso de CDUs constitui uma importante funcionalidade, o que justifica o esforço do desenvolvimento do UDCC. Além disso, o Anatem é um programa largamente usado no Brasil, de modo que a otimização de seu desempenho é de grande relevância. Por sua vez, o AnaHVDC constitui-se um programa único em sua área e é essencial que seus

componentes, incluindo o solucionador de CDUs, estejam construídos de forma altamente eficiente e confiável. Enquanto o UDCC já se encontra integrado ao AnaHVDC, sua integração ao Anatem cabe a trabalhos futuros.

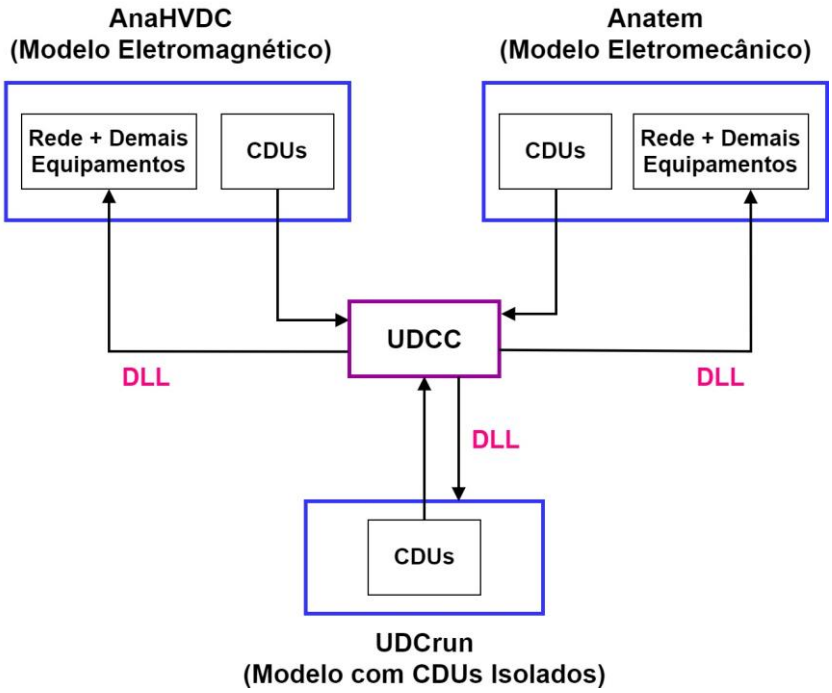


Figura 2 – Inter-relação do UDCC com o AnaHVDC, o Anatem e o UDCrun

Um solucionador de CDUs otimizado pressupõe que ele produza resultados corretos e de modo eficiente. Portanto, primeiramente é necessário demonstrar que o UDCC produz resultado idêntico ao Anatem quanto à solução isolada de CDUs[4]. Essa validação, que se encontra automatizada, foi realizada utilizando o UDCrun e comparando-se cada bloco individualmente e, na sequência, comparando-se cada CDU do SIN[5]. A descrição desse processo se encontra no tópico 3.2.

Além disso, um solucionador otimizado também implica em uma solução em tempo reduzido e, em termos comparativos, significativamente menor do que aquela obtida por meio do Anatem. Essa demonstração foi realizada considerando-se a solução dos CDUs do SIN, uma vez que este caso envolve um número elevado de CDUs. Isso é apresentado no Capítulo 4, referente aos resultados.

[4] O comparativo deve-se dar quanto a solução de um sistema composto apenas por CDUs. Quando integrado ao AnaHVDC os resultados destoam do Anatem, uma vez que os modelos dos equipamentos usados em cada programa são diferentes.
[5] O caso do SIN simulado foi o referente ao ano de 2016. Portanto, convenciona-se que toda vez que houver referência ao SIN é deste caso específico de que se trata.

Além disso, também é necessário demonstrar o sucesso da integração do UDCC ao AnaHVDC. Desse modo, são apresentados os resultados da simulação e validações do AnaHVDC em um caso de pequeno porte, o BenchCA, adaptado de [15], e em um caso de grande porte adaptado do PAR2023, que é o Plano de Ampliação e Reforços do ONS, disponibilizado publicamente pelo ONS ao setor elétrico, em que consta a rede e seus equipamentos atuais e também as modificações que se espera que ocorram em um futuro de 3 anos.

1.3 Publicações Originárias da Dissertação

Durante o XXV SNPTEE 2019 foi apresentado o artigo "Simulação Eficiente de Controladores Definidos pelo Usuário Utilizando Compilação em Tempo Real" na mesma área temática desta dissertação [16]. Além disso, como mencionado, o UDCC já foi integrado ao AnaHVDC, que também foi tema para um dos artigos apresentados no evento [11].

1.4 Estrutura do Trabalho

Esta dissertação estrutura-se da forma a seguir.

Capítulo 1: Introdução. Discorre-se brevemente sobre o tema da dissertação, incluindo a motivação, o objetivo e as publicações originárias dela.

Capítulo 2: Conceituação Teórica. São apresentados os conceitos básicos de estabilidade eletromecânica e a inserção do programa Anatem nesta área de estudo com foco nos Controladores Definidos pelo Usuário. Em seguida, são descritos os tipos de controladores usados no SEP e os que estão presentes nos CDUs do Sistema Interligado Nacional. Também são discutidas as diferenças entre compilação e interpretação e em qual delas a implementação do Anatem se enquadra. Nesse sentido, são abordadas as otimizações que os compiladores são capazes de realizar. Na sequência, é fornecida uma introdução conjunta à linguagem assembly e à arquitetura de computadores. Por fim, são abordados os conceitos básicos sobre processamento paralelo.

Capítulo 3: Metodologia. Primeiramente é apresentado o método empregado para solução de controladores. Em seguida, discute-se como foi realizada a validação da solução

do UDCC. Também são apresentados os compiladores embutidos no UDCC e o comparativo qualitativo de seu desempenho com ao do Anatem. Na sequência, é abordado o procedimento de modificação dos controladores do SIN para que o comparativo com o Anatem pudesse ser realizado. Por fim, discorre-se sobre a integração do UDCC ao AnaHVDC e sobre detalhes da implementação do processamento paralelo.

Capítulo 4: Resultados. São apresentados os resultados do comparativo de desempenho do UDCC em relação ao Anatem com base na solução dos controladores do SIN de forma isolada, utilizando o UDCrun com o UDCC. Na sequência, é simulado no AnaHVDC um caso adaptado baseado no PAR2023 e o resultado é comparado com aquele fornecido pelo PacDyn. São também apresentados resultados de um sistema exemplo de menor porte, com comparações de simulações não lineares do AnaHVDC com UDCC e os programas Anatem e PSCAD, incluindo resultados de curto fase-terra, que provoca desbalanços no sistema, em modelagem trifásica. Com isso, demonstra-se a integração do UDCC ao AnaHVDC

Capítulo 5: Conclusões. Apontam-se os trabalhos futuros a serem realizados e são expostas as contribuições do trabalho.

Capítulo 2 - Conceituação Teórica

Nesse capítulo é apresentado brevemente como se insere o Anatem nos estudos de estabilidade eletromecânica e o que são os Controladores Definidos pelo Usuário disponíveis neste programa. São discutidos também as principais classes de equipamentos modelados por CDUs, como reguladores de tensão e reguladores de velocidade. Na sequência, é dada uma introdução à linguagem de programação assembly, que será usada para justificar a eficiência do UDCC no Capítulo 3[6]. Também nesse sentido, são apresentadas as diferenças principais entre compilação e interpretação. Por fim, são descritos os principais conceitos referentes a processamento paralelo.

2.1 Dinâmica de Sistemas de Potência

Dentre os estudos de dinâmica de Sistemas de Potência se encontra o de estabilidade eletromecânica. Uma vez que é um estudo de estabilidade, o objetivo é determinar se após a ocorrência do distúrbio o sistema será capaz de alcançar um novo ponto operação ou se perderá a estabilidade[7] [17].

A estabilidade eletromecânica está relacionada à capacidade das máquinas de manterem o sincronismo após o sistema ser submetido a um distúrbio [18]. As turbinas de um gerador de uma usina térmica ou hidroelétrica possuem uma inércia considerável. Desse modo, após um distúrbio, o torque eletromagnético pode se alterar de modo quase instantâneo, enquanto o torque mecânico não é capaz de variar com a mesma rapidez. Caso o torque mecânico seja maior do que o magnético o rotor irá acelerar. Caso aconteça o inverso, o rotor irá desacelerar. A fase do rotor em relação a uma frequência síncrona é conhecida como ângulo de rotor (ou ângulo de carga) e é avaliada no estudo de transitórios eletromecânicos para determinar se haverá perda de estabilidade do sistema [17].

[6] É importante reforçar que o UDCC foi escrito em C++ e ele gera, a partir do arquivo CDU lido, um código em C, que na sequência é compilado pelo TCC ou pelo Clang, dois compiladores comerciais embutidos no UDCC e que serão discutidos em detalhes no tópico 3.3. O assembly foi usado nesta dissertação apenas para efeitos de comparação de eficiência entre o código do UDCC e do Anatem.

[7] Obter um novo ponto de operação não significa necessariamente retornar ao valor do ponto inicial e sim alcançar um regime permanente.

No estudo de estabilidade o Anatem se insere no que é conhecido como estabilidade eletromecânica a grandes perturbações, ainda que possa ser utilizado como ferramenta em outros estudos. Nesse caso, os distúrbios são de grande intensidade e são geralmente causados por curtos-circuitos nas linhas de transmissão [18].

Para simulação, como mencionado no Capítulo 1, o Anatem possui modelos pré-definidos (*built-in*) e também CDUs. Os *built-in* são aqueles que já se encontram modelados internamente no programa. Neste caso, o usuário precisa apenas fornecer os parâmetros do equipamento [15]. Isso é interessante especialmente por dois motivos. O motivo imediato é que aumenta a eficiência do trabalho do analista, que não precisa dedicar tempo modelando o controlador. Porém, há um segundo motivo, que é de ordem computacional. Os modelos *built-in* estão equacionados no código fonte do Anatem, que foi otimizado durante sua compilação. Isso significa que o executável do Anatem já possui esses modelos impressos nele. Isso aumenta a eficiência na solução desses modelos. Este ponto é retomado e abordado em mais detalhes no tópico 3.4.

Os CDUs, por outro lado, são modelos criados pelo usuário [19]. A vantagem é que o analista tem flexibilidade para realizar a modelagem que melhor convier ao estudo. Isso, entretanto, apresenta um revés importante. Os CDUs, uma vez que são modelados pelo usuário, não estão equacionados no código fonte do Anatem. Com isso, é necessário que o programa solucione esses CDUs de modo interpretado, o que será descrito em detalhes ao longo deste trabalho. O importante, neste ponto, é o fato de a interpretação ser significativamente custosa do ponto de vista computacional. De fato, a solução de CDUs constitui o principal gargalo de tempo de simulação no Anatem.

Na Figura 3 é mostrado um exemplo de um arquivo CDU com seus principais campos destacados. O formato do arquivo CDU da figura é comum ao Anatem e ao UDCC. Passa-se, portanto, à descrição de cada campo [19].

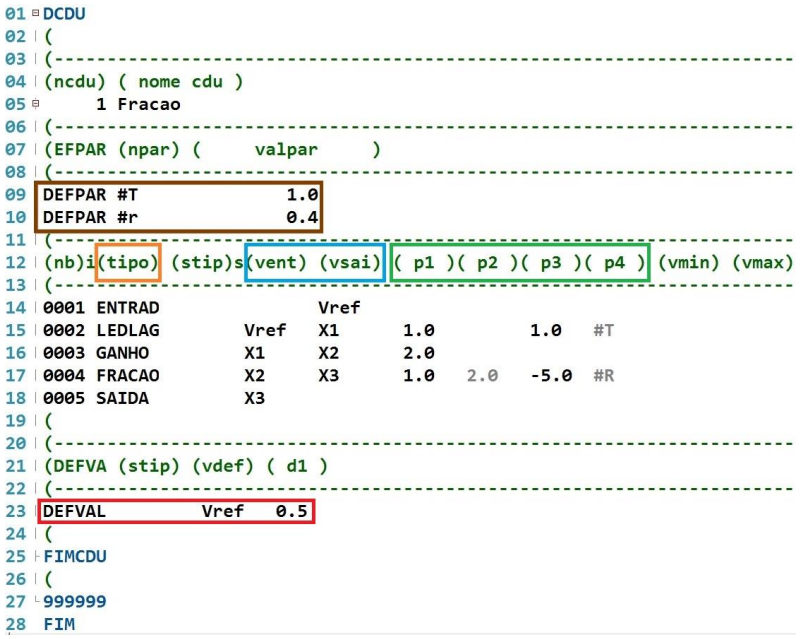


Figura 3 – Recorte de um CDU com destaque para os principais campos

- Retângulo laranja: tipo de bloco. O Anatem dispõe de dezenas de blocos a disposição do usuário, cuja lista e descrição podem ser encontradas no manual do programa [19], reproduzido no Apêndice F.
- Retângulo azul: entrada e saída de cada bloco. Todo bloco possui uma única saída e uma ou mais entradas. As únicas exceções são os blocos ENTRAD/IMPORT (sem entrada) e SAIDA/EXPORT (sem saída).
- Retângulo verde: parâmetros associados ao bloco. Cada bloco pode possuir entre nenhum parâmetro e 4 parâmetros, com exceção do bloco POL(S), que possui 8 parâmetros, e o bloco PONTOS, que pode possuir uma quantidade variável de pares de parâmetros, um par para cada ponto, tendo um limite máximo de 30.000 parâmetros no Anatem [15].
- Retângulo marrom: DEFPAR. São os dados referentes à definição de parâmetros e tradicionalmente possuem como primeiro caractere o símbolo de cardinalidade (#). Ou seja, os parâmetros podem ser definidos diretamente por um valor numérico ou por meio de um símbolo definido no DEFPAR.
- Retângulo vermelho: DEFVAL. Usado para definição de valores iniciais de variáveis, que são as entradas, as saídas e os limites (*vmin* e *vmax*) dos blocos. Portanto, o DEFVAL auxilia a inicialização do CDU. No caso da Figura 3 o usuário forneceu o valor da saída do bloco ENTRAD por meio de um DEFVAL. A partir desse auxílio, o Anatem é capaz de calcular o valor inicial da saída dos outros blocos. Caso o usuário não tivesse

fornecido nenhum DEFVAL, a inicialização seria inviável. Dependendo da dimensão do CDU é necessário fornecer diversos DEFVAL, mas também é bem comum que haja controladores que se inicializam automaticamente, sem a necessidade de qualquer DEFVAL.

2.2 Equipamentos de Controle

Em sistemas elétricos de potência há uma série de equipamentos de controle, que são necessários para controlar grandezas em valores desejados ou preservar a estabilidade do sistema. São exemplos desses equipamentos: reguladores automáticos de tensão (RATs), reguladores automáticos de velocidade (RAVs), estabilizadores (PSSs), controladores de compensadores estáticos (SVCs e STATCOMs), controladores de Capacitores Série Controlados a Tiristor (TCSCs) e controladores de conversores de elos de corrente contínua. Todos esses controladores podem ser modelados por meio de CDUs. Deve-se observar que em alguns casos, o modelo do atuador normalmente é incluído junto do modelo do controlador, representando-se este conjunto por um diagrama de blocos único. Assim, no modelo do RAT, normalmente está incluído o modelo da excitatriz e no modelo do RAV está incluído o modelo da turbina. Nos modelos dos controladores de equipamentos com eletrônica de potência, incluem-se também a modelagem dos circuitos de disparo dos elementos com chaveamento controlado.

O regulador de tensão é um equipamento acoplado à excitatriz de um gerador síncrono cujo objetivo é manter a tensão de saída em um nível aceitável apesar das variações de carga do sistema. O controle é realizado a partir da variação da corrente injetada no enrolamento de campo da máquina [18]. Para ilustrar, na Figura 4 é mostrado um diagrama de blocos em que o conjunto regulador de tensão-excitatriz é bastante simples, sendo modelado como CDU.

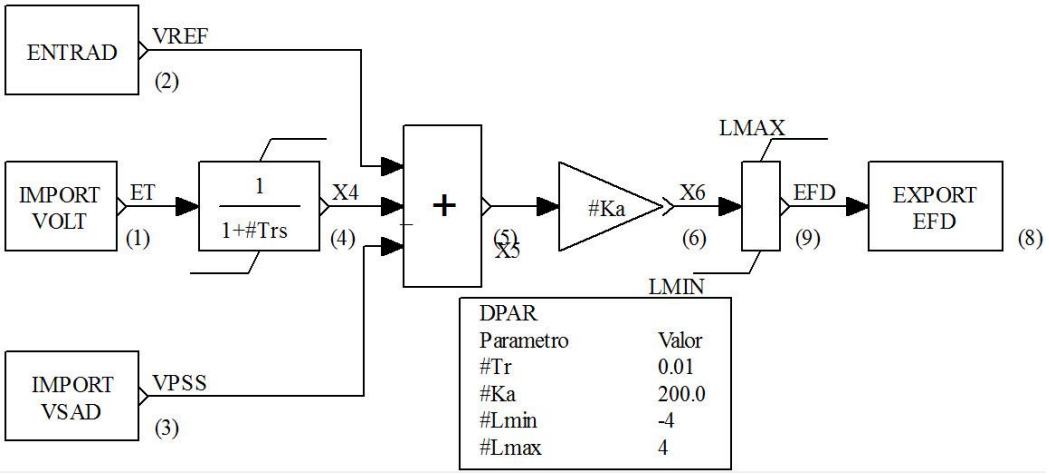


Figura 4 – Regulador de Tensão para um Sistema Máquina-Barra Infinita[8]

O bloco (2) fornece o valor de referência da tensão (VREF), enquanto no bloco (1) é importado o valor de tensão da barra (ET). Além disso, também é importado o sinal estabilizador aplicado no regulador de tensão da máquina (VPSS). Esses valores são somados de acordo com os sinais nas entradas do bloco (5) e o resultado é amplificado por meio do bloco GANHO (6). Na sequência, o sinal é limitado, caso viole algum de seus limites. Por fim, esse sinal é exportado como a tensão de campo da máquina síncrona (EFD), o que justifica os limites impostos pelo penúltimo bloco do regulador. Uma vez que o controlador possui poucos blocos, é possível perceber intuitivamente que de fato ele cumpre com o objetivo. Em regime permanente, o sinal estabilizador é normalmente zero, EFD é positivo, X4 é igual a ET e VREF será um pouco maior do que ET (VREF = ET + EFD/KA, em que KA geralmente é alto).

Como o controlador é proporcional, VREF e X4, em regime permanente, são diferentes. Após um distúrbio, em que a tensão ET se reduza, por exemplo, por um aumento da carga, X4 também diminui e, desse modo, a tendência é que X5 (erro) e, consequentemente, EFD aumentem. Com isso, a tensão de campo da máquina aumenta e ET se recupera, fazendo-se que o erro X5 diminua gradativamente até atingir um valor pequeno, mas suficiente para que multiplicado pelo ganho #Ka produza o EFD necessário para a recuperação da tensão. O regime permanente então é novamente atingido com a tensão ET recuperada ligeiramente inferior ao VREF.

[8] A partir de um arquivo CDU, como o da Figura 3, o programa CDUEdit é capaz de realizar a sua conversão para o diagrama de blocos correspondente, como o da Figura 4. Destaca-se que o arquivo CDU correspondente ao diagrama da Figura 4 não é mostrado.

Destaca-se que o sinal gerado pelo estabilizador precisa ser analisado para cada caso específico. De qualquer forma, considerando que o objetivo do estabilizador é amortecer as oscilações geradas pelo RAT, a tendência exposta é preservada. Em síntese, para uma redução da tensão ET espera-se que o RAT aumente a tensão de campo da máquina.

Os reguladores de tensão normalmente prejudicam a estabilidade a pequenas perturbações do sistema, reduzindo o amortecimento das oscilações eletromecânicas, pois introduzem um torque de amortecimento negativo. Nesse sentido, Estabilizadores de Sistemas de Potência (PSS) podem ser acoplados a esses reguladores, gerando um torque elétrico em fase com as variações de velocidade do rotor da máquina síncrona. O resultado é o amortecimento das oscilações produzidas pelos RATs e, consequentemente, a melhoria da estabilidade do sistema. Cita-se como exemplo de sinais de entrada do PSS a velocidade angular do rotor, a frequência terminal ou a potência elétrica da máquina. Outras variáveis também são possíveis [20].

Na Figura 5 é mostrado um tipo de PSS conhecido como PSS2A, que é caracterizado por ter duas portas de entrada, uma sendo a potência elétrica e a outra a variação do ângulo da máquina [21]. A saída é exportada para um regulador de tensão, conforme a Figura 4.

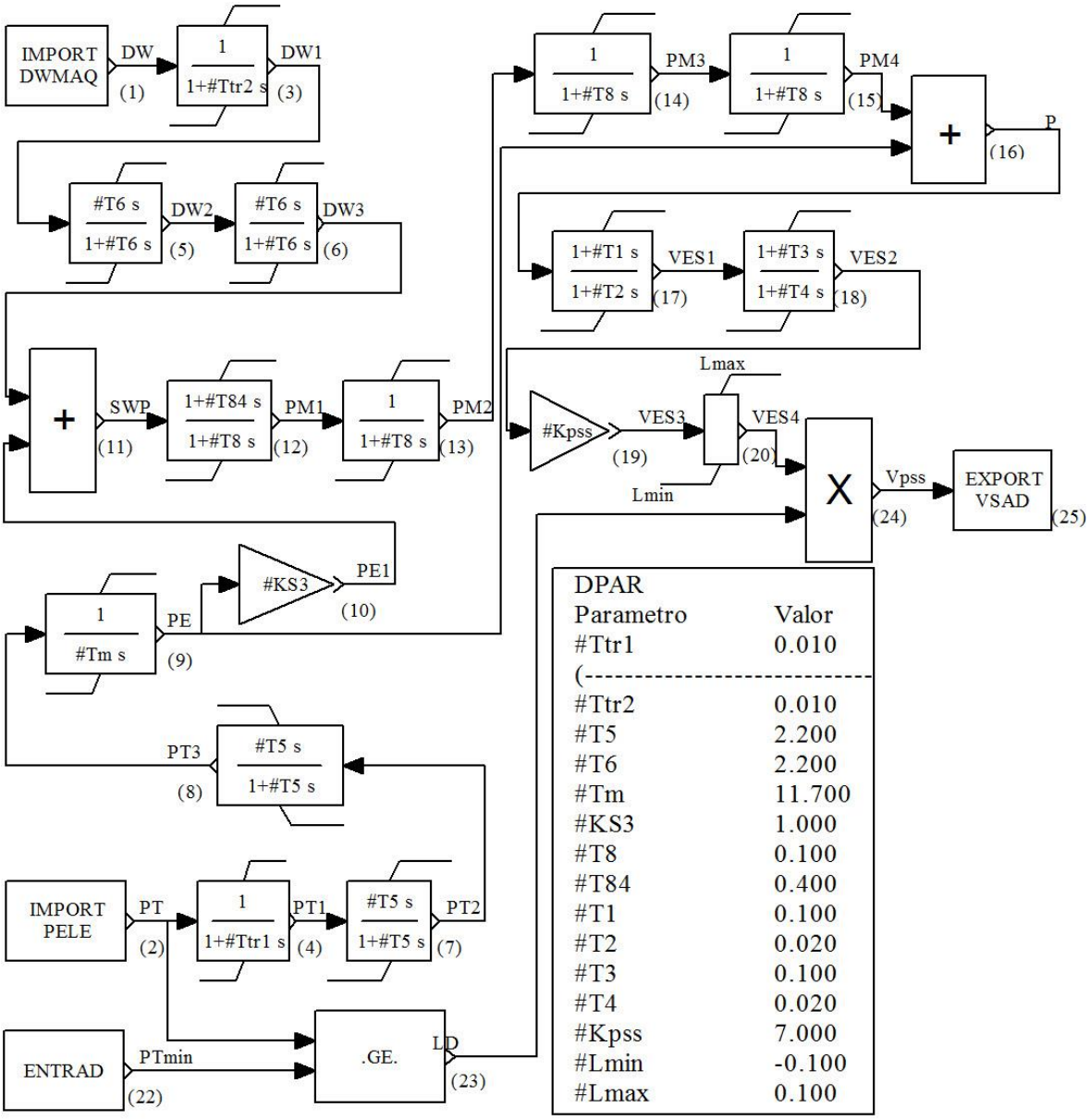


Figura 5 – Estabilizador de Sistema de Potência do tipo PSS2A

Descrito o impacto da variação de carga na tensão do sistema, passa-se a análise de sua influência na velocidade das máquinas síncronas. No caso de um aumento de carga, a tendência imediata é que as máquinas do sistema desacelerem e, consequentemente, ocorra uma redução da frequência [17]. Sem qualquer intervenção externa, a disparidade entre potência demandada e potência gerada iria forçar uma redução contínua da frequência e o sistema colapsaria. Dessa forma, é de suma importância a ação de equipamentos adicionais para retomar a frequência nominal do sistema. Vale destacar que a tolerância em regime

permanente para a variação de frequência no Brasil é de apenas 0,1 Hz, conforme estabelecido pela ANEEL [22].

Os equipamentos em questão são conhecidos como reguladores automáticos de velocidade (RAV). Eles podem, de modo geral, operar em dois modos: isócrono e com queda de velocidade. O que caracteriza o modo de operação isócrona é que a correção conduz a velocidade da máquina ao exato ponto em que se encontrava antes de sofrer alteração [23]. Entretanto, esse modo de operação é pouco usado, uma vez que não permite a coordenação dos despachos de múltiplas máquinas para atendimento das variações das cargas. Um caso em que ele pode ser usado é quando uma máquina está operando de modo isolado [24].

Mais comumente os reguladores operam em modo de queda de velocidade. Nesse modo, o controlador, ao observar uma redução na frequência, atua no sentido de aumentar a potência fornecida pelas unidades geradoras de modo a igualar a demanda exigida pelas cargas. No caso de aumento de frequência o regulador atua em sentido oposto, reduzindo a potência das unidades geradoras. Essa atuação se dá num espaço de tempo de algumas dezenas de segundos para máquinas hidráulicas e da ordem de alguns segundos para máquinas térmicas. Com isso, a frequência encontra um novo patamar de estabilidade, diferente do ponto de operação anterior [25]. Porém, como é essencial retornar à frequência original, é necessário que haja, além da regulação primária fornecida por esses reguladores, uma regulação secundária, denominado Controle Automático de Geração (CAG). Essa segunda regulação atua em alguns minutos após a alteração de frequência inicial e é baseada em controladores proporcionais integrais [25].

Na Figura 6 é apresentado o diagrama de blocos de um regulador automático de velocidade, incluindo neste diagrama o modelo de uma turbina hidráulica. A entrada é o desvio de velocidade angular da máquina em relação à velocidade síncrona, que é importada por meio do bloco IMPORT e dada pela variável DWMAQ. Por sua vez, a saída é a potência mecânica da máquina, que é exportada por meio do bloco EXPORT e dada pela variável PMEC [19].

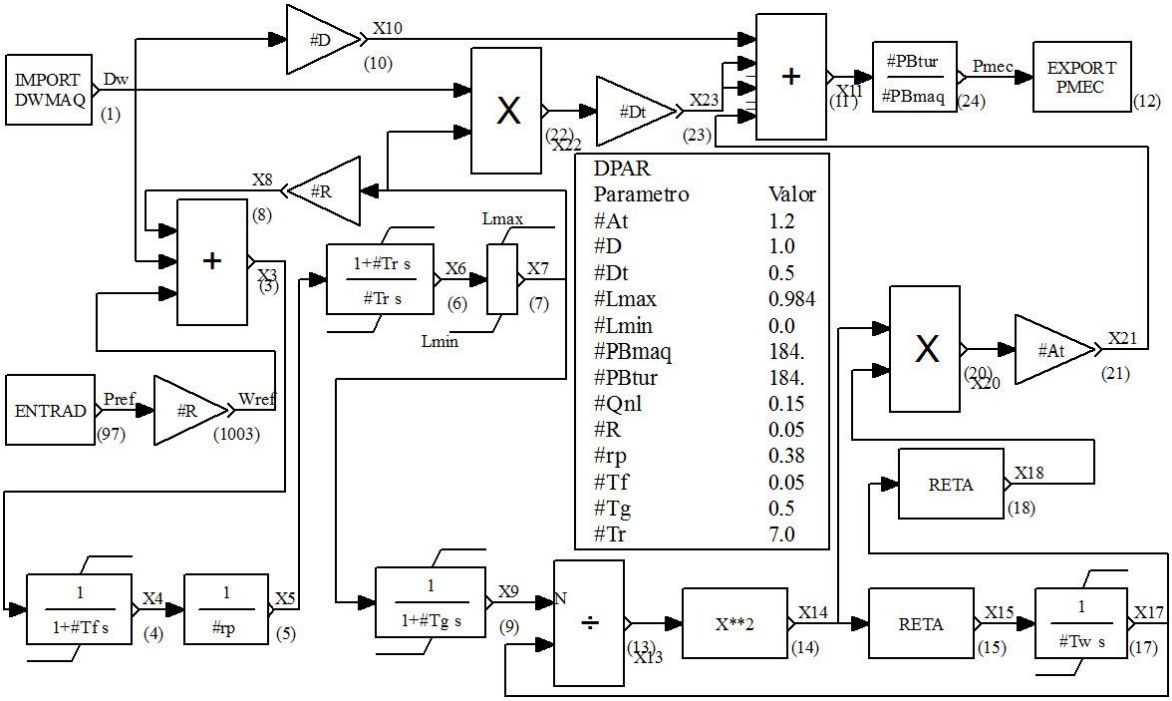


Figura 6 – Regulador de Velocidade Modelo MD01

Embora a análise do RAV do exemplo seja mais complexa que a do RAT da Figura 4, ainda assim é possível compreender a essência de seu funcionamento. Se DWMAQ aumenta, isso significa que a velocidade angular da máquina está ultrapassando a velocidade síncrona. Isso pode ser causado, por exemplo, devido a uma redução da carga do sistema. A estratégia de correção é reduzir a potência da máquina. Pelo diagrama de blocos percebe-se que se DWMAQ aumentar, a saída do multiplicador (22) também tenderá a aumentar. O bloco GANHO seguinte (23) apenas reduz esse valor e o sinal resultante é invertido (positivo para negativo) constituindo-se uma das entradas do bloco SOMA. Portanto, a saída do bloco SOMA tende a ser menor do que antes do aumento de DWMAQ. Por fim, PMEC tende a diminuir, o que era o esperado.

Outro ponto importante a considerar em Sistemas Elétricos de Potência é a variação de reativos, que pode gerar variações de tensão além do aceitável. Nesse sentido, compensadores estáticos são usados para controlar a potência reativa no SEP, com a flexibilidade de poder tanto injetar como absorver reativo. Esses equipamentos são compostos fundamentalmente por uma combinação de tiristores ou IGBTs, dependendo da tecnologia adotada (reator controlado a tiristores ou STATCOM) [26]. Compensadores estáticos possuem CDUs com muitos blocos e, por isso, não é mostrado um diagrama de

blocos. Entretanto, pode-se citar como exemplo o compensador estático de Anastácio do SIN, de número 1482 [19][9] .

Por fim, os TCSCs são equipamentos conectados em série com uma linha de transmissão, que são constituídos por um reator controlado em paralelo com um banco de capacitores. O controle nesses equipamentos pode ser realizado a partir da variação do ângulo de disparo do par de tiristores que estão localizados logo antes do reator. Dessa forma, o TCSC é capaz de alterar a reatância equivalente de ramos de transmissão entre duas subestações. Isso conduz à função desse equipamento, que é a de aumentar a capacidade de transmissão de potência de uma linha de transmissão. [27]. Outra função, utilizada no SIN, é o de amortecimento de oscilações de potência, pelo uso de um estabilizador aplicado a este equipamento, denominado "Power Oscillation Damping" (POD).

Como mencionado, todos os equipamentos de controle descritos podem ser modelados por meio de CDUs. De fato, no arquivo de CDUs do Sistema Interligado Nacional há, além de todos os controladores mencionados (RATs, RAVs, PSSs, CEs e TCSCs), também CDUs de geradores eólicos e fotovoltaicos e dos conversores de elos de corrente contínua. Não serão mostrados exemplos desses controladores uma vez que esses CDUs são muito extensos.

2.3 Arquitetura de Computador e Linguagem Assembly

O UDCC teve como norte o desenvolvimento de um solucionador de CDUs de modo altamente otimizado e, comparativamente, substancialmente mais rápido do que o Anatem. Dessa forma, um dos focos deste trabalho, além da exposição dos resultados comprovando que o objetivo foi alcançado, é fornecer uma justificativa para o desempenho superior do UDCC. Para isso, é necessário se recorrer ao assembly de cada código de modo a realizar o comparativo.

Portanto, cabe uma breve exposição sobre a linguagem de programação assembly e, sendo a linguagem de mais baixo nível no qual ainda se programa, também sobre a arquitetura de computadores. Um computador pode ser dividido esquematicamente em cinco partes: unidade de entrada, unidade de saída, unidade de memória, unidade de controle e

[9] O CDU de Anastásio pertencente ao SIN pode ser encontrado nos exemplos disponíveis no programa Anatem no caminho "..\ EXEMPLOS\11_SIN\FEV2016.cdu".

unidade lógica aritmética (ALU) [27]. Para os propósitos deste trabalho a análise será restrita às três últimas, que se relacionam conforme a Figura 7.

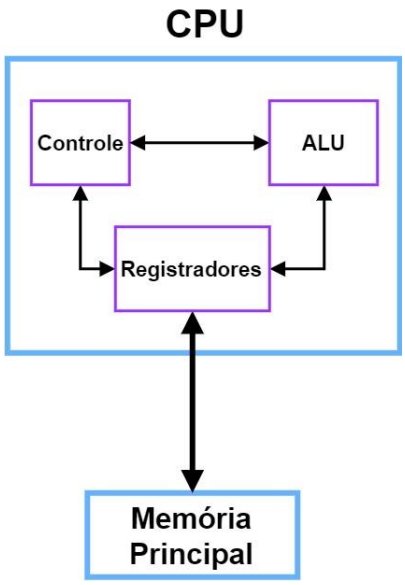


Figura 7 – CPU e Memória Principal

Primeiramente, destaca-se o fato de a CPU ser composta pela unidade de controle e pela unidade lógica aritmética. Além disso, os registradores também se encontram na CPU. De modo geral, uma instrução para ser executada é importada da memória principal para os registradores e a partir daí enviada para a ALU [28]. O resultado é colocado em um registrador de destino, que pode ser um dos registradores fontes ou não. Os registradores são a memória de acesso mais rápido em um computador e isso só é possível pela proximidade física deles com a ALU [28]. Uma vez que a CPU possui tamanho reduzido, não é viável na prática ter muitos registradores.

A Figura 8 mostra um programa simples escrito em C++ para soma de dois números. A função recebe apenas um argumento: um array de números. A única operação realizada é a soma dos dois primeiros números dessa lista e o resultado é retornado. Com isso, se podem ilustrar as instruções básicas de assembly. Diferentemente de linguagens de alto nível, em assembly as instruções recorrem aos registradores envolvidos em cada operação. Não serão tratadas instruções condicionais (*if*, *else if*, *else*, *switch*, *while*), pois para os propósitos desta dissertação não são relevantes.

```
1  int sumNumbers (int *p){
2    int sum;
3    sum = p[0] + p[1];
4    return sum;
5  }
```

Figura 8 - Função para Soma de Dois Números

O código correspondente em assembly é mostrado na Figura 9 [29].[10]

```
1   sumNumbers(int*):                          # @sumNumbers(int*)
2   push    rbp
3   mov     rbp, rsp
4   mov     qword ptr [rbp - 8], rdi
5   mov     rax, qword ptr [rbp - 8]
6   mov     ecx, dword ptr [rax]
7   mov     rax, qword ptr [rbp - 8]
8   add     ecx, dword ptr [rax + 4]
9   mov     dword ptr [rbp - 12], ecx
10  mov     eax, dword ptr [rbp - 12]
11  pop     rbp
12  ret
```

Figura 9 – Código para Soma de Dois Números

As instruções em azul são conhecidas em assembly como *opcode*. *Rbp*, *rsp*, *rdi*, *rsi*, *rdx* e *rax* são Registradores de Uso Geral que são exclusivos de arquiteturas de 64 bits. Além disso, são do tipo *qword* (com espaço de armazenamento de 8 bytes, ou seja, do tamanho de uma variável do tipo *double*) [30]. Nota-se que em toda instrução existem registradores envolvidos, como mencionado anteriormente. Por sua vez, também estão presentes registradores do tipo *dword* (com espaço de armazenamento de 4 bytes, ou seja, do tamanho de uma variável do tipo *int*[11]). Importante destacar que toda descrição é restrita para arquiteturas Intel® 64 e IA-32, que são as mais usuais [30].

Dessa forma, passa-se a descrição do código.

Linha 2 e Linha 3. São apenas convenções de chamada, ou seja, protocolos de instrução de como deve ser feita a passagem de argumentos para a função e o gerenciamento da pilha. *rbp* (register base pointer) aponta para a base da pilha enquanto *rsp* (register stack pointer) aponta para o topo da pilha. Portanto, na Linha 2 o registrador *rbp* é colocado na

[10] Nesta dissertação, todas as conversões de código C, C++ e Fortran para Assembly foram realizadas por [29].
[11] A rigor, em C++, a única restrição é que sejam alocados ao menos 2 bytes para um *int* [60]. Esse valor depende do compilador e obrigatoriamente deve ser compatível com o sistema operacional. Tipicamente possui 4 bytes [61].

pilha e na Linha 3 ele recebe o valor de *rsp*, de modo que ele passa a apontar para o topo da pilha.

Linha 4. *Mov* é a instrução de cópia em que o segundo operador é copiado para o primeiro [31]. Em assembly não há diferenciação na instrução se a cópia é de valores ou de um endereço de memória[12]. No caso, o registrador *rdi* guarda o endereço de memória do parâmetro da função. Afinal, em C/C++, a chamada de um ponteiro pelo seu nome retorna o endereço de memória onde ele se inicia. Portanto, na linha 4 o endereço de memória de *rdi* é copiado e colocado no endereço *rdp - 8*. Em processadores de arquitetura do tipo x86, a pilha cresce para "baixo", o que justifica o negativo em *[rdp – 8]*, ainda na linha 4 [31]. É na pilha que as variáveis locais, como *sum* da Figura 8, são armazenadas. Por sua vez, nesse caso, o "8" se refere ao fato de a arquitetura do processador ser de 64 bits e não ao tipo de variável alocada. Se o processador fosse de 32 bits a alocação se daria a partir de uma variável do tipo *dword*.

Linha 5. O endereço *rdp - 8* é copiado para o registrador *rax*. Isso é equivalente a *&list[0]* ou *list*.

Linha 6. O conteúdo do registrador (o equivalente a *list[0]*) é copiado para o registrado *ecx*. Como *rax* é um endereço, *ptr[rax]* é o elemento que se encontra no endereço *rax*, o que difere da notação em C/C++. Em síntese, o primeiro elemento da soma já está no registrador *ecx*.

Linha 7. Repetição da linha 5.

Linha 8. O conteúdo armazenado no endereço *rax + 4* (*list[1]*) é somado com o conteúdo do registrado *ecx* e o resultado é armazenado no próprio registrador *ecx*. Como o segundo elemento é um *int*, ele se encontra 4 bits acima do primeiro elemento. Dessa forma, acontece de fato a soma da Linha 3 da Figura 8. Percebe-se que nesse caso o ponteiro p está apontando para outra região de memória: o *heap*. Nesse caso, o *heap* cresce pra cima, ao contrário da pilha. Por isso o elemento *list[1]* se encontra 4 bytes acima do elemento *list[0]*. Isso é o oposto da pilha.

Linha 9. O conteúdo do registrador *ecx* é copiado para o endereço *rbp - 12*. Isso demonstra a pilha de fato crescendo para baixo. O registrador *rax* foi armazenado em *rbp – 8* e agora *ecx* é armazenado em rbp – 12.

Linha 10. O conteúdo no endereço *rbp – 12* é copiado para o registrador *eax* e o valor é retornado pela função.

[12] Em assembly, a cópia não pode ser endereço de memória para endereço de memória.

A Linha 11 e a Linha 12 também são convenções de chamada.

Muitas instruções podem parecer desnecessárias e não naturais. E de fato o são. Como será visto no tópico 3.4 é possível que o compilador otimize o código. Nesse caso, diversas instruções seriam eliminadas e o código seria reduzido a apenas 4 linhas.

2.4 Linguagens de Programação na Área de Engenharia

Muitos programas na área de engenharia desenvolvidos durante a década de 70 e de 90 foram estruturados em Fortran, que se consagrou como a linguagem padrão para aplicações envolvendo cálculos matemáticos computacionalmente pesados [32].

O Fortran ainda é usado em aplicações específicas, especialmente na área de física e engenharia, e também mantido em programas originalmente desenvolvidos nessa linguagem. Embora seja uma linguagem antiga, de modo algum o Fortran congelou no tempo. Sua última versão é de 2018, contando com processo paralelo nativo e suporte para orientação a objetos (herança, polimorfismo e alocação dinâmica) [32]. Além disso, atualmente também há compiladores altamente otimizados para Fortran, como o Intel Fortran Compiler.

Por outro lado, o C++ foi desenvolvido na década de 80 [33]. Da mesma forma que o Fortran, o C++ também é uma linguagem orientada a objetos e com processamento paralelo nativo.

Em termos comparativos, alguns critérios para escolha de uma linguagem devem ser observados. O primeiro critério considerado foi o desempenho. Entretanto, comparativos de desempenho são controversos quanto ao estabelecimento de critérios de avaliação. Ao longo do tempo criou-se uma imagem de que o Fortran seria incontestavelmente a linguagem mais rápida para cálculos matemáticos. Primeiramente é importante destacar que uma linguagem ser mais rápida do que outra em uma determinada aplicação é um abuso de linguagem. O que se pode afirmar é que um compilador específico de uma linguagem X produz a partir de determinado código fonte um código de máquina mais eficiente do que aquele gerado pela linguagem Y a partir de seu respectivo código fonte equivalente[13].

Uma vantagem do Fortran seria não permitir o que se conhece como *aliasing*. O conceito se refere à possibilidade de se acessar a mesma região de memória a partir de

[13] Os códigos fontes são equivalentes e não iguais, uma vez que cada um se encontra em sua respectiva linguagem de programação.

diferentes variáveis. Provavelmente o caso mais comum seja o de *pointer aliasing*, conforme ilustrado na Figura 10.

```
1  #include <iostream>
2  #include <string>
3
4  int main (int argc, char** argv){
5    std::string name = "Garfield";
6
7    std::string *cat = &name;
8    std::string *dog = &name;
9
10   *cat = "Hobbes";
11
12   std::cout << "Cat's name: " << *cat << std::endl;
13   std::cout << "Dog's name: " << *dog << std::endl;
14
15   return 0;
16 }
17
18 // Resultado
19 // Cat's name: Hobbes
20 // Dog's name: Hobbes
21
```

Figura 10 – Exemplo de *pointer aliasing*

Como se percebe, uma alteração em uma das variáveis irá alterar a outra. Isso acontece porque ambas apontam para a mesma região de memória. Esse caso simples ilustra o conceito de *pointer aliasing*, porém para se analisar a relação com o desempenho, considere-se o caso da Figura 11.

```
1  void foo(double* firstInput, double* secondInput, double* select,
2  double* output){
3    if (*select > 0){
4        *output = *firstInput;
5    } else if (*select < 0){
6        *output = *secondInput;
7        *select += 1;
8    }
9  }
```

Figura 11 – *Pointer aliasing* e perda de desempenho

Neste caso, é modelado um bloco que aceita duas entradas (*firstInput* e *secondInput*) e possui uma única saída (*output*), que é igual à primeira entrada caso a variável *select* seja

positiva e igual à segunda entrada caso *select* seja negativo. Além disso, caso *select* seja negativo soma-se 1 ao próprio *select*[14]. Na Figura 12 é apresentado o código em assembly resultante compilado com otimização –O3.

```
1   .LCPI0_0:
2           .quad   0x3ff0000000000000      # double 1
3   foo(double*, double*, double*, double*):
4           movsd   xmm0, qword ptr [rdx]   # xmm0 = mem[0],zero
5           xorpd   xmm1, xmm1
6           ucomisd xmm0, xmm1
7           jbe     .LBB0_2
8           mov     rax, qword ptr [rdi]
9           mov     qword ptr [rcx], rax
10          ret
11  .LBB0_2:
12          ucomisd xmm1, xmm0
13          jbe     .LBB0_4
14          mov     rax, qword ptr [rsi]
15          mov     qword ptr [rcx], rax
16          movsd   xmm0, qword ptr [rdx]   # xmm0 = mem[0],zero
17          addsd   xmm0, qword ptr [rip + .LCPI0_0]
18          movsd   qword ptr [rdx], xmm0
19  .LBB0_4:
20          Ret
```

Figura 12 – Assembly resultante do código da Figura 11

Há instruções que não foram descritas na breve introdução à linguagem assembly do tópico 2.3. Porém, uma análise do código como um todo não é relevante para demonstrar a questão do desempenho relacionado ao *pointer aliasing*. O importante é apenas analisar a instrução da linha 4. Nela o valor da variável *select* é alocado no registrador *xmm0*. Entretanto, na linha 16 a instrução é repetida. Em uma primeira impressão pode parecer que a instrução é desnecessária, afinal em nenhum ponto o valor do elemento para o qual *select* aponta parece ter sido alterado. Entretanto, apenas com esse código não é possível o compilador garantir que essa alteração não aconteceu. E isso se deve exatamente ao *pointer aliasing*. Não é possível garantir que a linha 6 do código da Figura 11 não tenha alterado o valor do elemento para o qual *select* aponta. E isso acontece porque o elemento para o qual variável *output* aponta é alterado e essa variável pode estar apontando para o mesmo endereço de memória que a variável *select*. Por isso, é necessário realocar no registrador *rax*

[14] Este é um código meramente ilustrativo do conceito de *pointer aliasing* e do seu impacto na performance. Não existe bloco correspondente no Anatem.

o elemento para o qual a variável *select* aponta. Caso a linha 6 da Figura 11 fosse eliminada a realocação no registrador não aconteceria.

No Fortran *aliasing* não é permitido e a equivalente à linha 16 da Figura 12 não existe [34]. Com isso, há um ganho de desempenho. Entretanto, há um meio de se contornar a questão em C++. E isso pode ser feito a partir do código da Figura 13.

```
1 void foo(double* __restrict firstInput, double* __restrict
2 secondInput, double* __restrict select, double* __restrict output){
3   if (*select > 0){
4       *output = *firstInput;
5   } else if (*select < 0){
6       *output = *secondInput;
7       *select += 1;
8   }
9 }
```

Figura 13 – *Pointer aliasing* e perda de desempenho

A alteração consistiu na introdução da palavra-chave *__restrict* no cabeçalho da função antes de cada parâmetro [35]. Com isso, é sinalizado ao compilador que o desenvolvedor garante que duas variáveis não estão apontando para a mesma região de memória. A consequência é que o código resultante é exatamente igual ao da Figura 12, com exceção da Linha 16, que é eliminada (isso pode ser conferido em [29]).

Entretanto, há um detalhe. *__restrict* não é uma palavra-chave na ISO/IEC 14882:2017 (*Programming languages – C++*). Isso significa que não é obrigatório que os compiladores a implementem. Nos casos específicos, ela é suportada pelos compiladores Clang e GCC [35]. Por outro lado, o compilador TCC ignora *__restrict* [36]. O fato é que geralmente a questão do *pointer aliasing* é de pequeno impacto e, como mostrado, possível de ser contornada.

Na literatura é possível encontrar casos comparativos de desempenho entre diferentes linguagens de programação, a exemplo do artigo [37], no qual os autores consideraram dez algoritmos de problemas relevantes e dezenas de linguagens de programação. Esses dez algoritmos foram implementados em todas as linguagens escolhidas. Na sequência, foi medido o tempo de execução. Entretanto, os resultados apresentam grande variação dependendo do problema considerado. Na realidade, o importante é o desempenho da linguagem na solução específica do problema com o qual o desenvolvedor está lidando. Desse modo, não foram considerados esses resultados como um parâmetro relevante como

critério de desempenho entre o C/C++ e o Fortran[15]. Além disso, de modo algum o que foi discorrido anteriormente exaure a discussão. Outras questões poderiam ser elencadas. De toda forma, não é o objetivo defender uma posição específica e sim expor que ambas as linguagens são altamente eficientes.

Retoma-se um ponto frisado anteriormente. A eficiência de um código está atrelada ao assembly gerado pelo compilador usado e não propriamente à linguagem em si. Isso conduz ao segundo critério importante na escolha de uma linguagem: a sustentabilidade. Nesse quesito, o C++ é indubitavelmente melhor. Isso porque ele possui larga penetração na indústria, sendo o código base no desenvolvimento do navegador Chrome [38] e do próprio Windows [39]. Devido a isso, existe uma forte necessidade de que os compiladores em C/C++ estejam em seu estado da arte. Além disso, a migração de uma linguagem para a outra não costuma ser um procedimento simples. Tome-se como exemplo o kernel do Linux, escrito predominante em C, que em 2016 (versão 4.9) continha mais de 22 milhões de linhas de código [40]. A consequência é que, embora não haja garantias propriamente ditas, existe uma tendência forte de que as linguagens C/C++ se mantenham relevantes no futuro e com compiladores altamente eficientes. Entretanto, o mesmo não acontece com o Fortran, que é usado apenas em nichos específicos. Dessa forma, não há o mesmo indicativo de que os compiladores se mantenham no futuro em seu estado de arte. Por fim, a sustentabilidade também se traduz no C/C++ possuírem uma literatura bem mais extensa e também sua comunidade ser vastamente maior, de modo que é mais fácil encontrar ajuda para problemas específicos.

Diante do exposto, conclui-se que as linguagens C/C++, na atualidade, são uma escolha bastante apropriada para o desenvolvimento de um programa na área de engenharia, sendo eficientes, confiáveis e com grande suporte da comunidade. Além disso, pelos motivos apresentados, são linguagens cujo futuro, em termos comparativos, tende a ser particularmente promissor.

2.5 Compilação e Interpretação

Uma CPU é composta essencialmente por transistores, que funcionam como interruptores, podendo estar ligados ou desligados [41]. Essa dualidade é tradicionalmente

[15] Embora os resultados não sejam considerados relevantes, os autores encontraram que em média o C foi 1,87 vezes mais rápido que o Fortran.

representada por 1 (excitado) e 0 (não excitado). Dessa forma, toda instrução, em algum ponto, deve ser reduzida à uma instrução binária. Naturalmente, a linguagem de mais baixo nível possível é uma que seja estritamente numérica, que é conhecida como código de máquina, podendo ser executada diretamente pela CPU. Porém, uma linguagem nesses moldes é cansativa de ser programada e difícil de ser entendida.

Para contornar essa questão, a linguagem assembly fornece uma equivalência de 1:1 (com algumas poucas exceções) entre instruções em forma de palavras mnemônicas e as instruções em código de máquina que estão na forma numérica [31]. Destaca-se que é necessário que o código assembly seja convertido pelo assembler em código de máquina antes de ser repassado à CPU [42].

A proximidade com a linguagem de máquina permite classificar as linguagens em dois tipos: alto nível e baixo nível. Quanto maior a proximidade com o hardware (entendida como a distância para a equivalência de 1:1), mais baixo nível será uma linguagem [43]. Sendo assim, linguagens como C e C++ são classificadas como de alto nível, enquanto o assembly é considerado como de baixo nível (na verdade, o mais baixo possível no qual ainda se programa).

No que se refere à implementação de uma linguagem de programação há duas técnicas envolvidas: compilação e interpretação. Essas técnicas não são atributos intrínsecos das linguagens em si, uma vez que uma linguagem de programação pode ser implementada seguindo qualquer uma das duas técnicas [44]. Entretanto, as linguagens ao longo do tempo evoluíram privilegiando uma técnica ou outra, de forma que é usual se referir às próprias linguagens como compiladas ou interpretadas. Nesse sentido, por exemplo, é dito recorrentemente que o C++ é uma linguagem compilada.

Em implementações compiladas, existem três etapas independentes: compilação propriamente dita, linkagem e por fim a execução [45]. Isso é sintetizado na Figura 14.

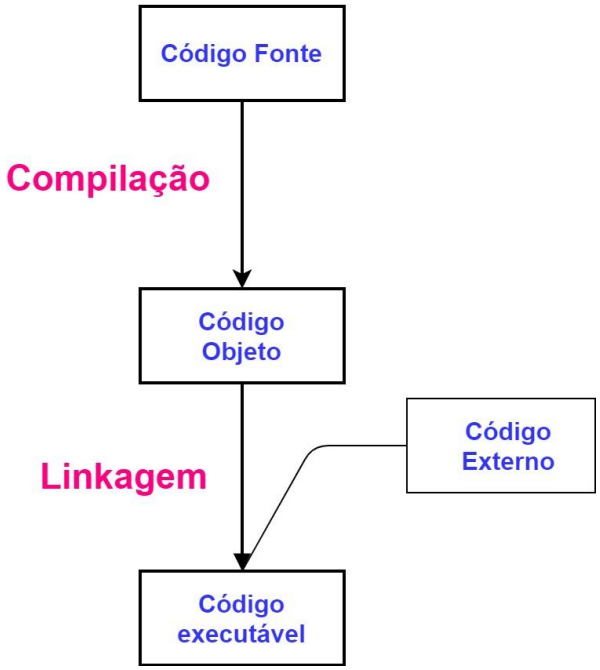


Figura 14 – Processo de Compilação

Dessa forma, primeiramente acontece a leitura por completo do código fonte (*name.cpp*[16] no caso do C++) e, a partir de um compilador, é realizada a conversão para código de máquina e gerado um código objeto (*name.obj*). Entretanto, usualmente existe uma dependência do programa com códigos externos, como bibliotecas. Dessa forma, na segunda etapa o código objeto é linkado a esses códigos externos e gerado o código executável (*name.exe* ou *name.dll*), que pode ser convencional ou de uma biblioteca de link dinâmico DLL ("*Dynamic Link Library*"). A diferença entre os dois tipos, é que o executável convencional possui uma função de entrada principal, que sempre é chamada em sua execução, enquanto que a DLL possui várias funções que podem ser chamadas por outro programa em tempo de execução (*runtime*). A rigor ainda existe uma etapa intermediária entre a compilação e a linkagem que nesse momento não é relevante. Ela será tratada no tópico 3.4.

Por outro lado, em implementações interpretadas essas etapas são fundidas em uma única. Nesse sentido, em interpretadores simples, o código fonte é lido linha a linha e executado em *runtime* pelo interpretador. Cabe lembrar que o interpretador em si é compilado. O exemplo da Figura 15 ilustra isso.

[16] Entende-se por *name.cpp* um arquivo .cpp que se queira compilar em que *name* é o nome do arquivo.

```
1   #include <fstream>
2   #include <iostream>
3   #include <string>
4   #include <vector>
5
6   using namespace std;
7
8   enum Command { SET, ADD };
9
10  struct CommandValue {
11    Command command;
12    int value;
13
14    CommandValue(const Command s, const int v) : command(s),
15  value(v) {}
16  };
17
18  std::vector<CommandValue> readCommandValues(std::istream &f) {
19    std::vector<CommandValue> commandValues;
20    std::string command;
21    int value;
22    while (f) {
23      f >> command >> value;
24
25      Command c;
26      if (command == "set") {
27        c = SET;
28      } else if (command == "add") {
29        c = ADD;
30      }
31
32      commandValues.emplace_back(c, value);
33
34      if (f.eof())
35        break;
36
37    }
38
39    return commandValues;
40  }
41
42  int interpretCommandValues(const CommandValue* commandValues,
43    const size_t commandValuesLen) {
44    int ans;
45
46    for (size_t i = 0; i < commandValuesLen; ++i) {
47      const auto &commandValue = commandValues[i];
48
49      if (commandValue.command == SET) {
50        ans = commandValue.value;
51      } else if (commandValue.command == ADD) {
```

```
52        ans += commandValue.value;
53      }
54    }
55
56    return ans;
57  }
58
59  int main(int argc, char** argv) {
60    ifstream calc(argv[1]);
61
62    // Leitura do arquivo
63    const auto commandValues = readCommandValues(calc);
64
65    // Interpreta estrutura
66    const auto ans1 = interpretCommandValues(commandValues.data(),
67  commandValues.size());
68    cout << "Answer is " << ans1 << endl;
69
70    puts("ok");
71    return 0;
72  }
```

Figura 15 – Intepretador Simples

Por sua vez, a Figura 16 mostra o arquivo de dados fornecido para o interpretador.

```
1  set 1
2  add 3
3  add 4
```

Figura 16 – Arquivo de Dados

Na Figura 15 é mostrado o princípio básico de funcionamento de um interpretador[17], que recebe um arquivo de dados e realiza a soma dos valores contidos nele. O arquivo de dados da Figura 16 contém em sua primeira coluna instruções sobre qual operação o interpretador deve realizar. *Set* se refere a atribuir à variável o valor corresponde na segunda coluna[18]. Por sua vez, *add* indica que o valor associado deve ser somado à variável criada.

Dessa forma, passa-se à descrição propriamente dita do programa da Figura 15. Primeiramente, a função *main*, na linha 63, chama a função *readCommandValues* passando como argumento o arquivo de dados *calc* (no caso exemplo esse arquivo é o da Figura 16).

[17] Os interpretadores atuais, que são altamente sofisticados, se valem de estratégias de otimização complexas não descritas neste trabalho. Este exemplo ilustra a lógica básica de interpretação, o que é suficiente para os propósitos desta dissertação, uma vez que é precisamente aquela usada pelo Anatem.

[18] Analisando-se a linha 50 se percebe que a variável não é criada neste ponto. Nessa linha acontece a atribuição de valor a variação. A sua criação se dá na linha 44.

Essa função realiza a leitura do arquivo de modo que cada linha lida corresponda a um elemento do vetor *CommandValue*. Esse é um vetor de elementos do tipo *struct*, sendo que elemento contém duas variáveis: a instrução da primeira coluna e o valor associado a ela na segunda coluna. A estrutura do vetor *commandValues* aplicada ao caso exemplo é sintetizada na Figura 17.

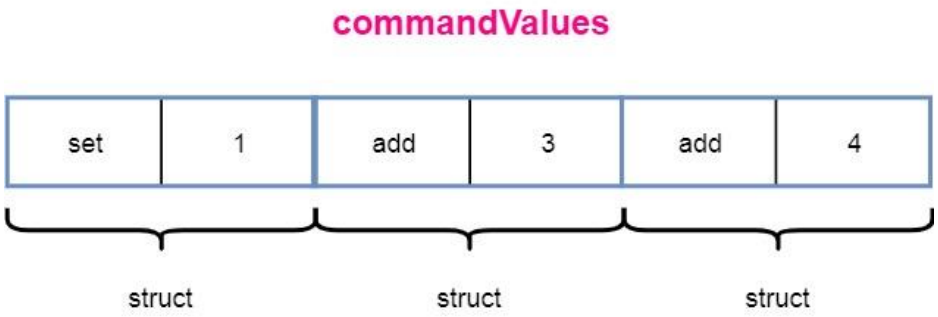


Figura 17 – Vetor commandValues

Ao retornar para a função *main*, é chamada a função *interpretCommandValue*. É nesta função em que as operações são de fato executadas, ou seja, acontece a atribuição da variável a um valor inicial e a soma dos valores subsequentes. Por fim, o valor resultante das somas é exibido para o usuário.

Em síntese, primeiramente acontece a leitura de um arquivo de dados, em que a primeira coluna se refere a uma instrução e, na sequência, a busca por essa instrução em um loop de modo que aconteça a interpretação apropriada. Esse processo acontece até que o arquivo seja lido por completo. Essa descrição de funcionamento é importante, uma vez que reflete precisamente a lógica da estrutura interna do Anatem, cujos detalhes específicos serão tratados no tópico 3.4.

Portanto, implementações compiladas tendem a ser mais velozes, uma vez que a execução se dá diretamente a partir de código de máquina altamente otimizado, considerando que a tradução tenha sido realizada por um compilador de alto desempenho [46].

A eficiência em termos de tempo de execução da implementação compilada é demonstrada no artigo [37] já mencionado no tópico 2.4. Das linguagens analisadas 11 eram com implementações compiladas, 10 com implementação interpretada e 6 do tipo JIT (Just-in-time). A compilação JIT será abordada adiante. Buscou-se utilizar os melhores compiladores, interpretadores e bibliotecas disponíveis. Em termos de tempo de execução,

nas dez primeiras colocações constaram 8 implementações compiladas, 2 implementações JIT e nenhuma interpretada.

Entretanto, a lógica seguida pelo interpretador da Figura 15 não é atualmente usual. Linguagens como Python e Java possuem uma implementação híbrida. Um compilador típico compila todo o código fonte para código de máquina antes da execução. Por outro lado, em Python o código de alto nível é convertido para um código intermediário, conhecido como *bytecode*, o qual é interpretado e executado em uma máquina virtual (VM)[19] [47]. Linguagens como Java vão um passo além. Além da conversão para *bytecode*, a máquina virtual determina quais partes do código são mais usadas, como um loop com um número muito grande de ciclos, e converte o *bytecode* para linguagem de máquina e realiza a execução a partir dela [47]. Nas partes de código menos usadas essa conversão não é vantajosa e não acontece. Isso é conhecido como compilação JIT. O importante no âmbito deste trabalho é o fato de que o Anatem segue uma implementação puramente interpretada, a semelhança do exemplo da Figura 15. A consequência é que o desempenho é significativamente penalizado. A estrutura interpretada específica do Anatem é abordada no tópico 3.4.

Entretanto, a compilação apresenta um importante revés. Como mencionado, um código de máquina é específico para cada arquitetura de processador e ainda dependente do sistema operacional usado. Portanto, um programa compilado para Windows não irá funcionar em um Linux, por exemplo. Neste caso, a consequência é que é necessário manter dois programas equivalentes cada um para um sistema operacional. Isso não acontece com implementações interpretadas.

É importante destacar que a linguagem Fortran, na qual o Anatem é escrito, é compilada, como descrito no tópico anterior. Portanto, o código fonte do Anatem é compilado e gerado um arquivo executável do programa. Entretanto, o Anatem em si realiza a solução de arquivos CDUs de forma interpretada. Isso significa que o CDU é lido linha a linha pelo Anatem em tempo de execução e as informações são armazenadas em estruturas internas, como vetores. Apenas depois dessa etapa ser concluída é que a solução de fato se inicia processando de forma interpretada cada dado armazenado.

[19] Importante não confundir com máquina virtual no sentido de “simular um computador dentro de um computador”. Mesmo termo, porém em sentidos diferentes e não correlatos.

2.6 Compiladores e otimizações

Como mencionado, o Anatem possui o seu código fonte compilado e altamente otimizado, porém a solução dos CDUs se dá de modo interpretado. Enquanto isso, no UDCC todo o processo é compilado e, dessa forma, a solução dos CDUs é realizada a partir da DLL gerada. Portanto, o desempenho do UDCC está atrelado à otimização do código de máquina gerado pelo compilador usado.

Compiladores simples, a exemplo do TCC, realizam apenas otimizações básicas do código. Dessa forma, a compilação se dá de modo rápido, entretanto a execução do código é lenta. Por outro lado, compiladores sofisticados, como o Clang e o GCC, empregam diversas estratégias para otimização do código, o que implica uma compilação lenta, porém uma execução rápida. Essas estratégias são de extrema importância para o desempenho do UDCC e serão descritas brevemente. Um tipo básico de otimização é conhecido como *constant folding* [48], que costuma ser realizada inclusive por compiladores simples, a exemplo do TCC [49]. Nesse caso, o compilador calcula expressões cujo resultado pode ser determinado em tempo de compilação. A Figura 18 ilustra isso.

```
1  // sem otimização
2  int hours(int years){
3    double hours;
4    hours = 24*365*years;
5    return hours;
6  }
```

```
1  // com otimização
2  int hours(int years){
3    double hours;
4    hours = 8760*years;
5
6    return hours;
7  }
```

Figura 18 – Constant Folding

Uma segunda classe de otimizações é conhecida como *strength reduction* e abriga um amplo leque de estratégias, dentre elas a substituição de divisões por multiplicações, a substituição de multiplicações por somas e uso de operadores *bitwise*. Considere-se o exemplo da Figura 19 de uma função que recebe dois inteiros, ambos por referência, em que um é divido por dois e o outro multiplicado por dois. Importante ressaltar que esse não é apenas um exemplo do que se supõe que aconteça, e sim o código assembly de fato gerado pelo compilador Clang x86-64 versão 10.0.0 com otimização -O3 a partir de [29].

```
1  void operations(int &a, int &b){
2    a = a/2;
3    b = 2*b;
4  }
```

```
1  operations(int&, int&):
2  mov     eax, dword ptr [rdi]
3  mov     ecx, eax
4  shr     ecx, 31
5  add     ecx, eax
6  sar     ecx
7  mov     dword ptr [rdi], ecx
8  shl     dword ptr [rsi]
9  ret
```

Figura 19 – Strength Reduction (Multiplicação e Divisão por 2)

Nesse caso, optou-se por mostrar o código assembly gerado, de modo a tornar mais claro o que acontece. Sem entrar em detalhes do código assembly, é possível perceber que a operação de divisão foi substituída por uma operação de *right shift*, indicada pelo *opcode shr*, seguida por uma soma simples e por fim uma operação *shift arithmetic right*. Por sua vez, a operação de multiplicação da linha 4 foi substituída por uma única operação de *left shift*, conforme a linha 8. Essas substituições têm como objetivo otimizar o código gerado. A otimização da Figura 19 também é realizada pelo TCC [49].

Um terceiro tipo de otimização, presente no Clang e no GCC, porém ausente no TCC, é *inlining*. Nesse caso, a chamada da função é substituída pelo corpo da função. Desse modo, não há o *overhead* da chamada da função.

Existem diversas outras otimizações mais sofisticadas. Neste tópico apresenta-se apenas a ideia básica do funcionamento de otimizações. No tópico 3.4 são mostradas otimizações na prática, aplicadas ao caso específico do UDCC e que estão fora do alcance do TCC. A discussão sobre otimizações é interessante uma vez que no UDCC está embutido um compilador simples, apenas com otimizações básicas, e um compilador sofisticado. Ambos possuem sua importância, o que é exposto no tópico 3.4.

2.7 Otimizações do Uso de Cache

No tópico anterior foram exploradas as otimizações que os compiladores realizam. Neste tópico serão expostas técnicas visando à otimização do uso da memória cache, que é utilizada para armazenar os dados ou código da região da memória RAM convencional que estão sendo acessados no momento para um acesso mais rápido pelo processador. Uma importante estratégia de otimização do uso de cache baseia-se no que se conhece como *branch prediction*. Um exemplo clássico é o comparativo de tempo entre o processamento

de um array aleatório e o processamento de um array ordenado. Também foi incluído o processamento de um array binário preenchido alternadamente com zero e um. A Figura 20 ilustra o caso.

```
#include <iostream>
#include <algorithm>
#include <chrono>
using namespace std;

static const int length = 50001;

void unsortedArray(int* arr) {
  int count = 0;

  // Inicializa contagem de tempo
  std::chrono::steady_clock::time_point begin = std::chrono::stead
y_clock::now();

  // Caso o número seja menor do que a metade de length, ou seja,
  // igual ou menor do que 25000 o contdor é incrementado
  for (int i = 0; i < length; ++i) {
    if (arr[i] < length / 2) {
      ++count;
    }
  }
  // Finaliza contagem de tempo
  std::chrono::steady_clock::time_point end = std::chrono::steady_
clock::now();
  cout << "Tempo para processamento de um array aleatorio: "
  << std::chrono::duration_cast<std::chrono::microseconds>(end - b
egin).count() << "[us]" << std::endl;
}

void sortedArray(int* arr) {
  int count = 0;

  std::chrono::steady_clock::time_point begin = std::chrono::stead
y_clock::now();
  for (int i = 0; i < length; ++i) {
    if (arr[i] < length / 2) {
      ++count;
    }
  }
  std::chrono::steady_clock::time_point end = std::chrono::steady_
clock::now();
  cout << "Tempo para processamento de um array em ordem crescente
: "
  << std::chrono::duration_cast<std::chrono::microseconds>(end - b
egin).count() << "[us]" << endl;
}
```

```
47
48  void binaryArray(int* arr) {
49    int count = 0;
50    std::chrono::steady_clock::time_point begin = std::chrono::stead
51  y_clock::now();
52    for (int i = 0; i < length; ++i) {
53      // Caso o valor seja 0 (zero) o contador é incrementado
54      if (arr[i] == 0) {
55        ++count;
56      }
57    }
58    std::chrono::steady_clock::time_point end = std::chrono::steady_
59  clock::now();
60    cout << "Tempo para processamento de um array binario: "
61      << std::chrono::duration_cast<std::chrono::microseconds>(end -
62   begin).count() << "[us]" << endl;
63  }
64
65  int main()
66  {
67    int arr[length];
68
69    // Gera números randomicos para preenchimento do array
70    for (int i = 0; i < length; ++i) {
71      arr[i] = rand() % length;
72    }
73
74    // Chama função para marcar o tempo com array randômico
75    unsortedArray(arr);
76
77    // O array é organizado em ordem crescente
78    sort(arr, arr+length);
79
80    // Chama função para marcar o tempo com array em ordem crescente
81    sortedArray(arr);
82
83    // Preenchimento array apenas com 0/1 alternados: 0, 1, 0, 1...
84    for (int i = 0; i < length - 1; i+=2) {
85      arr[i] = 0;
86      arr[i + 1] = 1;
87    }
88
89    // Chama função para marcação de tempo com array binario
90    binaryArray(arr);
91
92    return 0;
93  }
```

Figura 20 – Comparativo de Tempo (*Branch Prediction*)

O resultado da compilação da Figura 20 é mostrado na Figura 21.

```
Tempo para processamento de um array aleatorio: 438[us]
Tempo para processamento de um array em ordem crescente: 160[us]
Tempo para processamento de um array binario: 156[us]
```

Figura 21 – Comparação de tempo: resultado

O importante do código exemplo anterior é mostrar que o processamento de arrays que contenham um padrão é muito mais eficiente. No caso do array desordenado e do array em ordem crescente o contador *count* é incrementado apenas quando o valor do elemento contido no array é superior 25000. Além disso, percebe-se na linha 72 que os valores aleatórios escritos no array variam entre 0 e 50000. Dessa forma, no array ordenado espera-se, numa aproximação rudimentar, que os elementos iguais ou inferiores a 25000 estejam na primeira metade do array, enquanto os elementos superiores 25000 estejam na segunda metade. O resultado é que os primeiros 25000 valores lidos não resultarão em acréscimo do contador, e sim apenas a segunda metade dos valores. Portanto, o processador percebe esse padrão e começa a realizar predições. Ao ser processada a segunda metade do array o processador percebe que eles sempre resultam em acréscimo do contador. Portanto, o processador intui que o elemento seguinte também irá resultar em acréscimo. Dessa forma, já é realizada a operação de incremento do contador (linha 19) antes mesmo de analisar a instrução condicional *if* (Linha 18), que verifica se de fato o contador deve ser incrementado. Se o processador tiver errado a predição, ele corrige o próprio erro. Num array que segue um padrão claro, como neste exemplo, a taxa de acerto do processador é altíssima, o que possibilita o ganho verificado. No caso do array binário, também há um padrão claro, embora seja distinto. O objetivo é contar o número de zeros e os números do array, que são apenas binários, são dispostos de modo alternado. Novamente o processador percebe o padrão é realiza a predição baseada na mesma lógica exposta anteriormente. Por sua vez, no caso do array desordenado, a predição não é útil uma vez que não há um padrão. Portanto, o *branch prediction* conduz a uma regra geral em relação à otimização empregada pelos processadores: escrever códigos que contenham padrões.

Outro importante ponto é em relação às otimizações dos caches dos processadores. No tópico 2.3 foi exposto brevemente a arquitetura de um computador. Toda informação que é resgatada da memória até o processador consome um tempo, que é usualmente medida em ciclos do processador, o que é apresentado na Tabela 1 [50].

Tabela 1 – Hierarquia de Memória

Memória	Ciclos	Armazenamento
Cache L1	1 - 4	128 kB - 512 kB
Cache L2	3 - 10	512 kB - 2 MB
Cache L3	3 - 10	3 MB - 16 MB
DRAM	20 - 100	2 GB - 64 GB
Disco Rígido	1.000.000	500 GB - 2000 GB

A Tabela 1 mostra a importância de que a informação resgatada já esteja armazenada no cache, evitando-se a latência significativa da importação a partir da memória. Destaca-se também que os ciclos são cumulativos, ou seja, a informação armazenada na memória deve ser transportada para o cache e, na sequência, para os registradores [51][20]. Nesse sentido, a Figura 22 ilustra o que é conhecido como *spatial locality*.

```
1   #include <iostream>
2   #include <algorithm>
3   #include <chrono>
4   using namespace std;
5
6   static const int dim = 1000;
7   int arrOne[dim][dim];
8   int arrTwo[dim][dim];
9
10  void rowMajor() {
11    int i, j;
12    // Inicializa contador de tempo
13    std::chrono::steady_clock::time_point begin =
14  std::chrono::steady_clock::now();
15    for (int i = 0; i < dim; ++i) {         // Varre a linha
16      for (int j = 0; j < dim; ++j) {       // Varre a coluna
17        arrOne[i][j] = i + j;               // i: linha. j: coluna
18      }
19    }
20
21    // Finaliza contador de tempo
22    std::chrono::steady_clock::time_point end =
23  std::chrono::steady_clock::now();
24    cout << "Tempo para processamento por coluna de uma matriz : "
25
```

[20] A descrição é uma simplificação do que de fato acontece. No transporte de informação há outros componentes intermediários envolvidos, como a *Northbridge bus* e a *Southbridge bus*. Entretanto, a versão apresentada é suficiente para os propósitos desta dissertação.

```
26      << std::chrono::duration_cast<std::chrono::microseconds>(end -
27  begin).count() << "[us]" << std::endl;
28
29  }
30
31  void columnMajor() {
32    int i, j;
33    std::chrono::steady_clock::time_point begin =
34  std::chrono::steady_clock::now();
35    for (int i = 0; i < dim; ++i) {          // Varre a coluna
36      for (int j = 0; j < dim; ++j) {        // Varre a linha
37        arrTwo[j][i] = i + j;                // i: coluna. j: linha
38      }
39    }
40    std::chrono::steady_clock::time_point end =
41  std::chrono::steady_clock::now();
42    cout << "Tempo para processamento por linha de uma matriz : "
43      << std::chrono::duration_cast<std::chrono::microseconds>(end -
44  begin).count() << "[us]" << std::endl;
45  }
46
47  int main() {
48    rowMajor();
49    columnMajor();
50    return 0;
      }
```

Figura 22 – Comparativo de Tempo (*Spatial Locality*)

O resultado da compilação da Figura 22 é mostrado na Figura 23.

```
Tempo para processamento por coluna de uma matriz: 3236[us]
Tempo para processamento por linha de uma matriz: 574[us]
```

Figura 23 – Comparação de tempo: resultado

É possível perceber que quando a matriz é preenchida horizontalmente (por linha) o desempenho é significativamente melhor. Quando o primeiro elemento da matriz ([0][0]) é preenchido, os elementos espacialmente próximos (*spatial locality*) são alocados no cache [51]. Entretanto, a proximidade é medida percorrendo-se a matriz horizontalmente. Dessa forma, deve-se considerar o elemento [0][1] ao lado do zero [0][0]. Por outro lado, o elemento [1][0] está a uma distância significativa. Percorrendo-se a matriz por linha, quando for atribuído o valor 1 ao elemento [0][1] (segundo elemento a ser preenchido), ele já estará no cache. A isso se dá o nome de *cache hit*. Ao contrário, percorrendo-se a matriz por coluna, quando for atribuído o valor 1 ao elemento [1][0] (segundo elemento a ser preenchido), ele muito provavelmente não estará no cache. Com isso, é necessário que ele seja importado da

memória principal, o que implica uma perda de tempo considerável. A isso se dá o nome de *cache miss*. Portanto, o *cache hit* é muito alto quando a matriz é preenchida por linha e muito baixo quando ela é preenchida verticalmente.

Técnicas como *spatial locality* estão abrigadas em um conjunto maior conhecido como *locality of reference*, ou seja, a tendência do processador de acessar lugares de memória próximos entre si [52]. O uso conjunto dessas técnicas permite ganhos de desempenho consideráveis e, por isso, é uma otimização relevante a ser considerada.

2.8 Paralelismo

Primeiramente é importante distinguir os conceitos de concorrência e de paralelismo. Paralelismo (*hardware concurrency*) ocorre quando duas ou mais *threads* estão executando de forma simultânea. Dessa forma, é necessário que o processador tenha ao menos dois núcleos [53]. Isso é ilustrado na Figura 24, em que cada *thread* está sendo executada em núcleos diferentes.

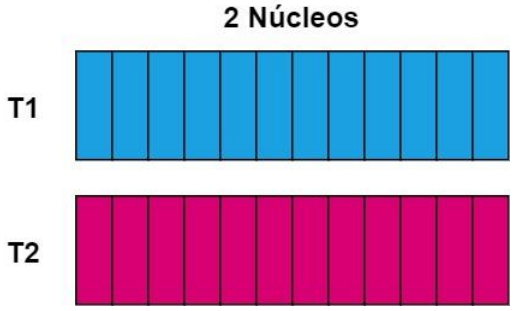


Figura 24 – Paralelismo em Dois Núcleos

Por sua vez, a concorrência sem paralelismo (*task switching*) ocorre quando duas ou mais *threads* estão em progresso durante um mesmo intervalo de tempo, porém não de forma simultânea. Em um processador com um único núcleo não é possível se obter uma execução verdadeiramente em paralelo [53]. Entretanto, o processador é capaz de alternar rapidamente entre as *threads* em execução, dando a ilusão de simultaneidade, o que é ilustrado na Figura 25. Neste caso, a *thread 1* e *thread 2* estão sendo executada em alternância.

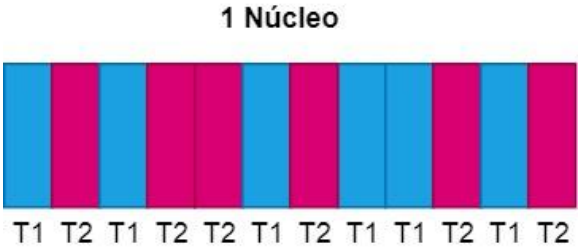


Figura 25 – Concorrência em um núcleo

Atualmente, é comum os processadores disporem de múltiplos núcleos. Além disso, em um computador diversos processos estão em execução simultaneamente, o que implica centenas ou até milhares de *threads* concomitantes. Portanto, o que ocorre é uma combinação da Figura 24 e da Figura 25, o que é retratado na Figura 26. Neste caso, a t*hread* 1 e a *thread* 2, que são consideradas as *threads* de interesse, estão sendo executadas em paralelo, porém em concorrência com diversas outras *threads*.

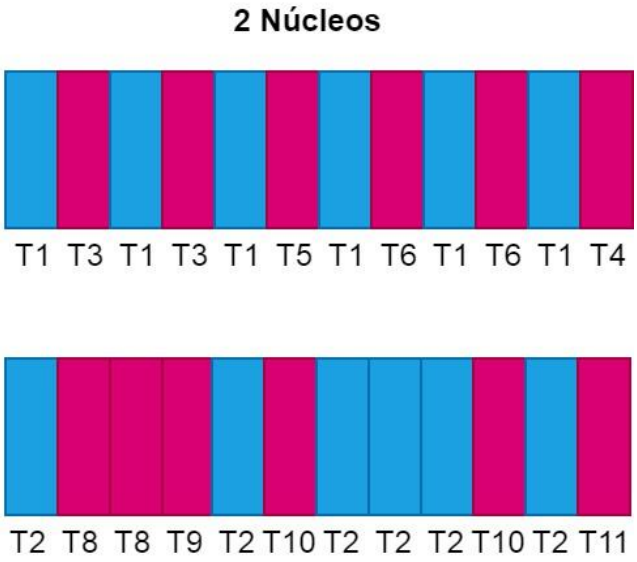


Figura 26 – Paralelismo com Concorrência

Em um arquivo CDU, uma possibilidade é instruir a CPU a lançar 2 *threads*, de modo que cada uma seja responsável pela solução de um conjunto diferente de CDUs. É importante ressaltar que neste exemplo, ainda que o computador disponha de 2 núcleos, não é obrigatório que as *threads* sejam alocadas em núcleos diferentes, uma vez que essa distribuição é responsabilidade do sistema operacional. Entretanto, a distribuição igualitária de *threads* pelos núcleos é o caminho natural. Considerando-se o caminho natural, a situação

se assemelha novamente a da Figura 26. Neste caso a *thread* 1 é responsável pela solução de um conjunto de CDUs e a *thread* 2 por um segundo conjunto.

No caso do Anatem, ainda que o computador disponha de vários núcleos não é possível a estratégia de solução paralelizada[21]. Desse modo, a solução dos CDUs é realizada por uma única *thread*, a que foi criada no início da execução do programa. Dessa forma, a situação possível em um computador, por exemplo, com dois núcleos, é representada pela Figura 27, em que *T1* é a thread do Anatem. Nota-se que a *thread* pode alternar entre os núcleos.

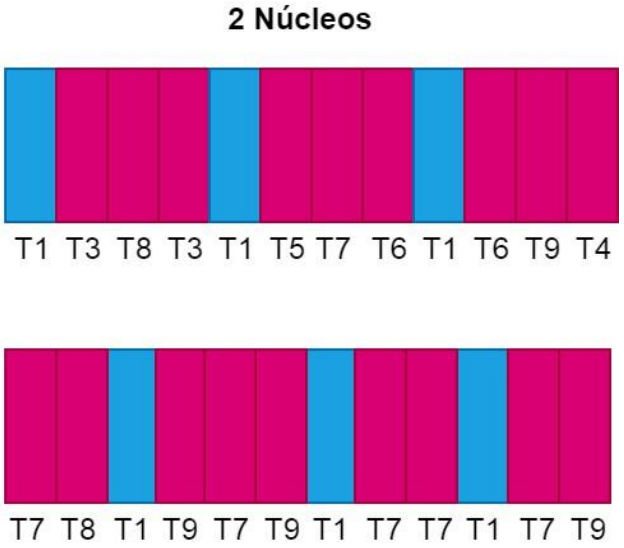


Figura 27 – *Thread* de Solução do Anatem

É importante uma ressalva. A partir do programa *Process Explorer* é possível visualizar o número de *threads* alocadas por cada aplicação. Na Figura 28 é indicado que estavam 16 *threads* alocadas para o Anatem em um momento em que o programa estava apenas realizando a abertura de um arquivo sem executar qualquer solução. Essas *threads* não foram lançadas por um comando inserido no código do Anatem escrito pelo desenvolvedor. Possivelmente essas *threads* foram lançadas por bibliotecas externas as quais o Anatem usa.

[21] Cabe notar, entretanto, que no presente momento a equipe do Anatem está trabalhando na paralelização do programa.

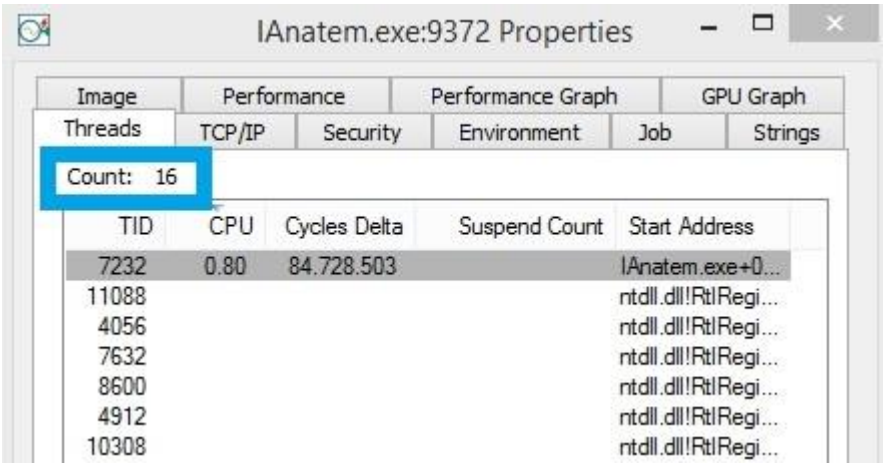


Figura 28 – *Thread* na Execução do Anatem

Portanto, o que se afirma é que a solução do Anatem é executada em uma única *thread*, representada por T1 na Figura 27, enquanto as outras *threads* podem ser de outros processos ou inclusive do próprio Anatem. Seja como for, a solução de CDUs no Anatem acontece de forma sequencial.

Capítulo 3 - Metodologia

Neste capítulo é apresentado como acontece a solução dos CDUs, explorando cada uma das etapas envolvidas. Na sequência, é discutido como se deu a validação inicial do programa desenvolvido, tendo-se como base os CDUs do SIN, e também a automatização dessa validação. São abordados, em seguida, os compiladores disponíveis, que são usados para geração da DLL a partir do código escrito pelo UDCC. Após isso, é apresentado um comparativo entre a filosofia de solução interpretada do Anatem e a solução compilada utilizada pelo UDCC e, desse modo, justificando o melhor desempenho do UDCC. Além disso, para que a validação mencionada anteriormente pudesse acontecer, diversas alterações precisaram ser realizadas, que também são discutidas. Por fim, é mostrado como aconteceu o processo de integração do UDCC ao AnaHVDC e também como isso impactou na eficiência do processamento paralelo.

3.1 Solução dos Controladores

Considere-se como exemplo o CDU da Figura 29, em que é aplicado em degrau de -10% e a variável X3 é inicializada em 2.0. O resultado desse CDU é mostrado na Figura 30.

```
01 DCDU
02 (ncdu) ( nome cdu )
03     1  EXEMPLO
04 (
05 (EFPAR (nome) (     valor      )
06 DEFPAR #KI      2.0
07 DEFPAR #KP      1.7
08 DEFPAR #T       1.5
09 (-------------------------------------------------------
10 (nb)i(tipo)o(stip)s(vent) (vsai) ( p1 )( p2 )( p3 )( p4 )
11 (-------------------------------------------------------
12   1  ENTRAD                VREF
13   2  SOMA          VREF    X1
14                   -  X3    X1
15   3  PROINT           X1    X2       #KI   #KP   1.0
16   4  LEDLAG           X2    X3       1.0   0.0   1.0   #T
17   5  SAIDA            X3
18 (
19 (EFVAL (stip) (vdef) ( d1 )
20 DEFVAL          X1     0.0
21 DEFVAL          X3     2.0
22 (
23 FIMCDU
24 999999
25 FIM
```

Figura 29 – Arquivo CDU do Controlador Exemplo

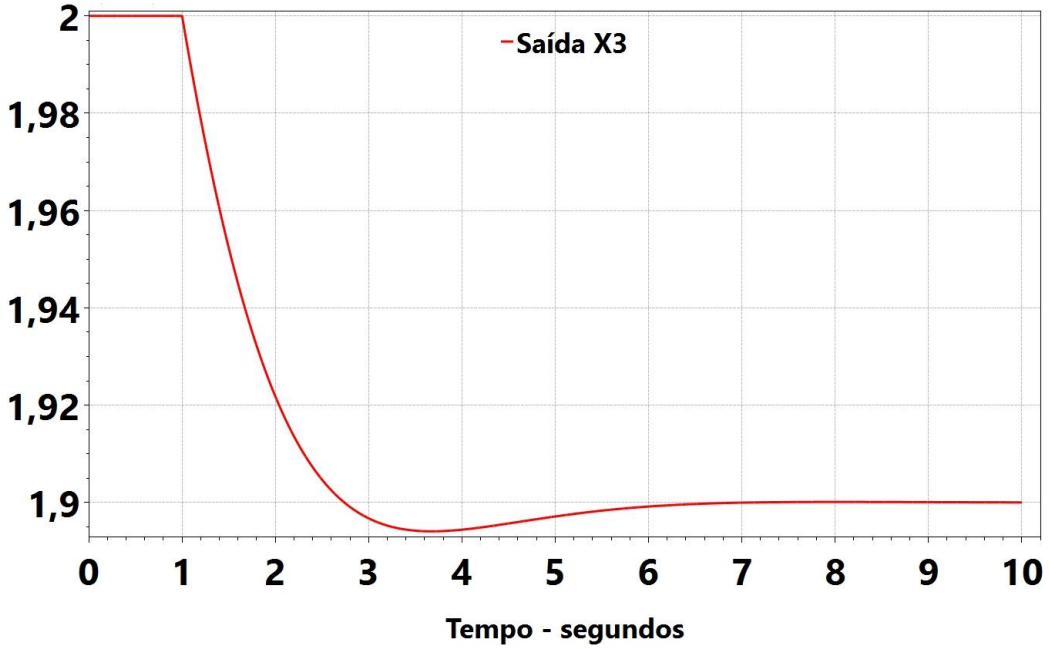


Figura 30 – Sinal da Saída do Controlador Exemplo

Esquematicamente esse CDU pode ser representado pelo seu diagrama de blocos de acordo com a Figura 31. A partir dele será descrita a lógica de solução empregada no UDCC.

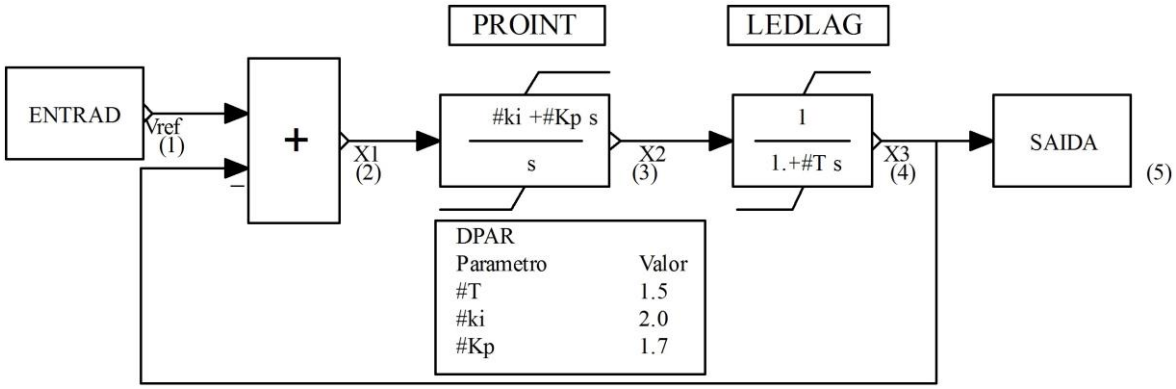


Figura 31 – Controlador Exemplo

Deve-se observar que na metodologia proposta, o UDCC será um gerador de códigos em C, a partir dos dados dos CDUs. Na atual implementação, todos os códigos são gerados em memória e o compilador, seja ele TCC ou Clang, é chamado ao final do processo para gerar uma DLL cujas funções podem ser chamadas por um simulador externo. No caso da simulação isolada dos CDUs, ou seja, sem interação com outros equipamentos, foi desenvolvido o programa UDCrun nesta dissertação, que possui os loops de tempo e

iterativos de um simulador convencional, porém apenas chama as funções de solução do UDCC para os CDUs isolados a serem simulados.

Na metodologia proposta, é possível realizar a segurança do uso não autorizado do código C gerado pelo programa UDCC, uma vez que a geração do código C pode ser feita só em memória. A alternativa convencional é a geração em arquivo, que pode ser inclusive compilado por qualquer outro compilador de forma não automática. A forma em arquivo é particularmente útil para o processo de depuração (*debug*) disponível em ambientes integrados de desenvolvimento (IDEs).

A primeira etapa do processo de solução dos blocos é a inicialização das variáveis (VREF, X1, X2 e X3). Sem qualquer informação adicional, além do esquema do controlador da Figura 31, a inicialização é inviável. Portanto, é necessário que sejam fornecidos valores de inicialização, que são conhecidos como DEFVAL, conforme apresentado no tópico 2.1[22] e que se encontram na Figura 29. O processo de inicialização desse controlador está esquematizado na Figura 32. Destaca-se que o objetivo desse tópico é apresentar a filosofia geral de solução empregada no UDCC e não o procedimento exato. O código do UDCC emprega, por vezes, estratégias diferentes no sentido de melhorar a eficiência e a precisão da solução. Convém ressaltar que a função de inicialização não precisa ser tão otimizada computacionalmente uma vez que ela só é chamada uma única vez no início da simulação, sendo realizada em tempo imperceptível, comparada com a função de solução que é chamada por múltiplas iterações para todos os passos de integração.

```
1   #define par_T 1.5
2   #define par_Ki 2.0
3   #define par_kp 1.7
4   #define INI_LEDLAG(Y, U, P1, P3) (U) = ((P3) / (P1)) * (Y)
5   #define INI_PROINT()
6   #define INI_SOMA(Y, U1, U2) U1 = Y + U2
7   #define INI_ENTRADA()
8
9   void inicializaCdu(double* x) {
10    x[1] = 0.0;       // DEFVAL X1
11    x[3] = 2.0;       // DEFVAL X3
12    INI_LEDLAG(x[3], x[2], 1.0, 1.0);
13    INI_PROINT();
14    INI_SOMA(x[1], x[0], x[3]);
15    INI_ENTRADA();
16  }
```

Figura 32 – Código C para Inicialização do Controlador Exemplo

[22] Repete-se a informação referenciada: “DEFVAL é usado para definição de valores de variáveis, que são as entradas, as saídas e os limites (vmin e vmax) dos blocos. Portanto, o DEFVAL auxilia na inicialização do CDU.”

O primeiro bloco a ser inicializado é o LEDLAG conforme a Equação (1) [23]. O valor de X3 é fornecido via DEFVAL.

$$\underset{\text{Saída}}{X3} = \frac{P1}{P3} \times \underset{\text{Entrada}}{X2} \tag{1}$$

$$2 = \frac{1.0}{1.0} \times X2 \tag{2}$$

$$\boxed{X2 = 2.0} \tag{3}$$

Nota-se que a entrada do PROINT já é fornecida via DEFVAL e sua saída é X2, tendo sido determinado no passo anterior. Portanto, com a inicialização do LEDLAG, o PROINT é automaticamente inicializado. Desse modo, segue-se para o bloco SOMA, conforme a Equação (4)

$$X1 = VREF - X3 \tag{4}$$

$$\underset{\text{DEFVAL}}{0.0} = VREF - \underset{\text{DEFVAL}}{2.0} \tag{5}$$

$$\boxed{VREF = 2.0} \tag{6}$$

Inicializado o bloco SOMA, automaticamente o bloco ENTRAD é inicializado.

Nesse controlador há dois blocos dinâmicos, #3 (PROINT) e #4 (LEDLAG), que são resolvidos por meio do método trapezoidal de integração[24]. Como exemplo, será mostrada a solução para o bloco #4. Ressalta-se, novamente, que o equacionamento para a solução é exatamente aquele empregado pelo Anatem.

Na Figura 33 é mostrada uma representação esquemática desse bloco [19] .

[23] A inicialização apresentada na Equação (1) pode ser demonstrada recorrendo-se, novamente, ao teorema do valor inicial. A partir da forma geral do bloco Ledlag, apresentada na Figura 33 e fazendo-se "s" tender a infinito, conclui-se que na inicialização o bloco LEDLAG se resume a um GANHO de valor $P1/P3$.

[24] No apêndice A consta uma revisão do método trapezoidal de integração.

:

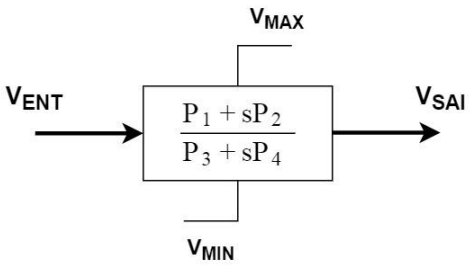


Figura 33 – Representação do Bloco LEDLAG

No caso específico do controlador da Figura 31 os parâmetros são:

$$Bloco\#4\begin{cases} P_1 = 1 \\ P_2 = 0 \\ P_3 = 1 \\ P_4 = T \end{cases} \tag{7}$$

Para simplificação durante o desenvolvimento dos cálculos considera-se a variável de entrada como “U” e a variável de saída como “Y”. Dessa forma, a solução geral para o bloco LEDLAG pode ser escrita conforme a Equação (8).

$$Y = \frac{P_1 + sP_2}{P_3 + sP_4} \times U \tag{8}$$

Substituindo-se a Equação (7) na Equação (8) obtém-se a Equação (9), que modela o bloco #4 específico do controlador exemplo.

$$Y = \frac{1}{1 + sT} \times U \tag{9}$$

$$\left(1 + sT\right)Y = U \tag{10}$$

Nesse momento, a Equação (10) se encontra no domínio da frequência. Portanto, aplica-se a transformada inversa de Laplace para realizar a conversão para o domínio do tempo considerando-se as condições iniciais nulas.

$$\mathcal{L}^{-1}\left\{\left(1+sT\right)Y\right\}=\mathcal{L}^{-1}\left\{U\right\} \tag{11}$$

$$\mathcal{L}^{-1}\left\{Y\right\}+\mathcal{L}^{-1}\left\{sTY\right\}=\mathcal{L}^{-1}\left\{U\right\} \tag{12}$$

$$y+T\dot{y}=u \tag{13}$$

Neste ponto, aplica-se o método trapezoidal à Equação (13). O apêndice A, que apresenta uma revisão do método trapezoidal, justifica a passagem da Equação (13) para Equação (14).

$$\frac{y\left(t\right)+y\left(t-\Delta t\right)}{2}+T\frac{y\left(t\right)-y\left(t-\Delta t\right)}{\Delta t}=\frac{u\left(t\right)+u\left(t-\Delta t\right)}{2} \tag{14}$$

$$2T\frac{y\left(t\right)-y\left(t-\Delta t\right)}{\Delta t}+y\left(t\right)+y\left(t-\Delta t\right)=u\left(t\right)+u\left(t-\Delta t\right) \tag{15}$$

$$\left(\frac{2T}{\Delta t}+1\right)y\left(t\right)=u\left(t\right)+u\left(t-\Delta t\right)+2T\frac{y\left(t-\Delta t\right)}{\Delta t}-y\left(t-\Delta t\right) \tag{16}$$

$$\left(\frac{2T}{\Delta t}+1\right)y\left(t\right)=u\left(t\right)+u\left(t-\Delta t\right)+\left(\frac{2T}{\Delta t}-1\right)y\left(t-\Delta t\right) \tag{17}$$

$$y\left(t\right)=\frac{1}{\left(\frac{2T}{\Delta t}+1\right)}\left[u\left(t\right)+u\left(t-\Delta t\right)\right]+\frac{\left(\frac{2T}{\Delta t}-1\right)}{\left(\frac{2T}{\Delta t}+1\right)}y\left(t-\Delta t\right) \tag{18}$$

$$y\left(t\right)=\frac{1}{\left(\frac{2T}{\Delta t}+1\right)}u\left(t\right)+\underbrace{\frac{1}{\left(\frac{2T}{\Delta t}+1\right)}u\left(t-\Delta t\right)+\frac{\left(\frac{2T}{\Delta t}-1\right)}{\left(\frac{2T}{\Delta t}+1\right)}y\left(t-\Delta t\right)}_{b\left(t-\Delta t\right)} \tag{19}$$

Todos os termos que sejam dependentes do instante de tempo anterior constituem o termo histórico, indicado por *b(t – Δt)*. Portanto, a Equação (19) pode ser reescrita como conforme a Equação (20).

$$y\left(t\right)=c_1u\left(t\right)+b\left(t-\Delta t\right) \tag{20}$$

Em que:

$$b(t-\Delta t)=c_1 u(t-\Delta t)+c_2 y(t-\Delta t) \tag{21}$$

$$c_1=\frac{1}{\left(\frac{2T}{\Delta t}+1\right)} \tag{22}$$

$$c_2=\frac{\left(\frac{2T}{\Delta t}-1\right)}{\left(\frac{2T}{\Delta t}+1\right)} \tag{23}$$

Analisando-se a Equação (20), é possível perceber que a solução é esquematicamente constituída por duas partes: cálculo das constantes e cálculo do termo histórico. Na Figura 34 é apresentada a função para cálculo das constantes. Cabe relembrar que o "T" da Equação (22) e da Equação (23) é o quarto parâmetro do bloco LEDLAG do controlador exemplo da Figura 31 e conforme a Equação (7).

```
1  #define par_T 1.5
2  #define par_Ki 2.0
3  #define par_kp 1.7
4
5  void constantesCdu(double* c, double step) {
6    double aux1 = 2 / step;
7    c[0] = ((par_Ki)+aux1 * (par_kp)) / aux1;    // C0 PROINT
8    c[1] = 1.0;                                  // C1 PROINT
9    c[2] = ((par_Ki)-aux1 * (par_kp)) / aux1;    // C2 PROINT
10   double aux2 = (2 / step) * (par_T);
11   c[3] = 1 / (aux2 + 1);                       // C0 LEDLAG
12   c[4] = (aux2 - 1) / (aux2 + 1);              // C1 LEDLAG
13 }
```

Figura 34 – Cálculo das Constantes do Controlador Exemplo

Na Figura 34 a função possui dois parâmetros: um ponteiro *c* que aponta para um vetor de constantes e o passo de integração *step*, que é equivalente ao Δt da Equação (23). É interessante comparar as variáveis c[3] e c[4] da Figura 34 com a Equação (22) e a Equação (23), respectivamente. Por sua vez, na Linha 7, na Linha 8 e na Linha 9 estão as constantes do bloco PROINT. A demonstração, embora não mostrada, segue o mesmo processo daquela do bloco LEDLAG.

O código para cálculo do termo histórico é dado pela Figura 35.

```
1  void memoriaCdu(double* b, double* c, double* x) {
2    b[0] = c[1] * x[2] + c[2] * x[1];          // PROINT
3    b[1] = c[4] * x[3] + c[3] * x[2];          // LEDLAG
4  }
```

Figura 35 – Cálculo do Termo Histórico do Controlador Exemplo

No caso da Figura 35 a função possui 3 parâmetros: um ponteiro *b* que aponta para um vetor de termos históricos, um ponteiro de constantes já descritos anteriormente e um ponteiro *x* que aponta para um vetor contendo os valores das saídas dos blocos do CDU no instante de tempo anterior. Destaca-se que são dois termos históricos, um para cada bloco dinâmico do controlador. Portanto, na Linha 2 é o termo histórico referente ao bloco PROINT e na Linha 3 o do bloco LEDLAG. Comparando-se a Linha 3 com a Equação (21) escreve-se a Equação (24).

$$\underbrace{b\left(t-\Delta t\right)}_{b[1]} = \underbrace{c_1}_{c[4]}\underbrace{u\left(t-\Delta t\right)}_{x[2]} + \underbrace{c_2}_{c[5]}\underbrace{y\left(t-\Delta t\right)}_{x[3]} \tag{24}$$

A entrada do bloco LEDLAG no passo anterior é indicada por $u\left(t-\Delta T\right)$, porém é resgatada no código como a saída do bloco PROINT, por meio de x[2], o que é equivalente.

Por fim, apresenta-se na Figura 36 o código que realiza a solução do CDU.

```
1  void solucionaCdu(double* b, double* c, double* x) {
2    x[1] = x[0] - x[3];                 // SOMA    X1=Vref-X3
3    x[2] = c[0] * x[1] + b[0];          // PROINT
4    x[3] = c[3] * x[2] + b[1];          // LEDLAG
5  }
```

Figura 36 – Cálculo da Solução do Controlador Exemplo

Para este exemplo específico, podemos a partir do apresentado desenvolver o programa completo. Desse modo, na Figura 37 é apresentado o ucd.h, composto pela junção da inicialização (Figura 32), do cálculo das constantes (Figura 34), do cálculo do termo histórico (Figura 35) e da solução (Figura 36). O udc.h é uma versão simplificada do código C gerado pelo UDCC, que, no caso do CDU Compilado, é na sequência compilado em uma DLL.

```
1   //udc.h
2   #define par_T 1.5
3   #define par_Ki 2.0
4   #define par_kp 1.7
5
6   #define INI_LEDLAG(Y, U, P1, P3) (U) = ((P3) / (P1)) * (Y)
7   #define INI_PROINT()
8   #define INI_SOMA(Y, U1, U2) U1 = Y + U2
9   #define INI_ENTRADA()
10
11  void inicializaCdu(double* x) {
12    x[1] = 0.0;       // DEFVAL X1
13    x[3] = 2.0;       // DEFVAL X3
14    INI_LEDLAG(x[3], x[2], 1.0, 1.0);
15    INI_PROINT();
16    INI_SOMA(x[1], x[0], x[3]);
17    INI_ENTRADA();
18  }
19
20  void constantesCdu(double* c, double step) {
21    double aux1 = 2 / step;
22    c[0] = ((par_Ki)+aux1 * (par_kp)) / aux1;    // C0 PROINT
23    c[1] = 1.0;                                  // C1 PROINT
24    c[2] = ((par_Ki)-aux1 * (par_kp)) / aux1;    // C2 PROINT
25    double aux2 = (2 / step) * par_T;
26    c[3] = 1 / (aux2 + 1);                       // C0 LEDLAG
27    c[4] = (aux2 - 1) / (aux2 + 1);              // C1 LEDLAG
28  }
29
30  void memoriaCdu(double* b, double* c, double* x) {
31    b[0] = c[1] * x[2] + c[2] * x[1];            // PROINT
32    b[1] = c[4] * x[3] + c[3] * x[2];            // LEDLAG
33  }
34
35  void solucionaCdu(double* b, double* c, double* x) {
36    x[1] = x[0] - x[3];                  // SOMA    X1=Vref-X3
37    x[2] = c[0] * x[1] + b[0];           // PROINT
38    x[3] = c[3] * x[2] + b[1];           // LEDLAG
39  }
```

Figura 37 – udc.h

Por sua vez, na Figura 38 é apresentado o código que gerencia a solução do caso exemplo. Ele é versão simplificada do UDCrun do CDU Compilado.

```
1   #include <iostream>
2   #include <fstream>
3
4   #include "udc.h"
5
6   using namespace std;
7
8   #define TOL 1e-10
9
10  int main() {
```

```
11    double x[4];
12    inicializaCdu(x);
13
14    double c[5];
15    double step = 0.001;        // passo da simulação
16    constantesCdu(c, step);
17
18    int timeSimulation = 10;
19    int points = timeSimulation / step + step / 2;
20
21    ofstream myFile("output.plt");
22    myFile << "5" << endl;
23    myFile << "Tempo - segundos" << endl;
24    myFile << "CDU         1    1 VREF    EXEMPLO" << endl;
25    myFile << "CDU         1    2 X1      EXEMPLO" << endl;
26    myFile << "CDU         1    3 X2      EXEMPLO" << endl;
27    myFile << "CDU         1    4 X3      EXEMPLO" << endl;
28    myFile.ios_base::setf(ios::scientific, ios::floatfield);
29
30    double b[2];
31    double xOld[4];
32
33    for (int time = 0; time <= points; ++time) {
34
35      double mismatch = 10 * TOL;
36      memoriaCdu(b, c, x);
37
38      if (time == 1001) {
39        x[0] += -0.1;             // Evento aplicado: degrau -
40  10%
41      }
42
43      while (mismatch > TOL) {
44        for (int i = 0; i < 4; ++i) {
45          xOld[i] = x[i];
46        }
47        solucionaCdu(b, c, x);
48        mismatch = abs(x[3] - xOld[3]);
49      }
50
51      myFile.precision(14);
52      myFile.width(24);
53      myFile << left << time * step;
54      for (int i = 0; i < 4; ++i) {
55        myFile.precision(14);
56        myFile.width(24);
57        myFile << left << x[i];
58      }
59      myFile << endl;
60    }
61    return 0;
62  }
```

Figura 38 – Solucionador de CDU

3.2 Validação

3.2.1 Bloco a Bloco

Todos os 85 blocos disponíveis no SIN foram implementados e validados, com exceção do bloco atraso, que atualmente é modelado simplificadamente como um bloco LEDLAG com constante de tempo igual ao valor do atraso[25]. Vale destacar que o bloco atraso possui implementação mais complexa, exigindo alocação de vetores adicionais para armazenamento dos valores anteriores das suas variáveis de entrada e, também, tratamento matemático da interpolação de valores quando os atrasos não são múltiplos do passo [61]. Como este bloco não aparece no SIN, a sua ausência não interfere nas simulações realizadas. Sua implementação será realizada em breve. Na validação bloco a bloco, foi criado um CDU pequeno e executados três casos para cada bloco. São eles:

1) Aplicação de degrau no primeiro bloco (ENTRAD) e valor de inicialização (DEFVAL) fornecido para o mesmo bloco

2) Sem aplicação de degrau e valor de inicialização fornecido para o primeiro bloco (ENTRAD)

3) Sem aplicação de degrau e valor de inicialização fornecido para o último (SAIDA).

É importante destacar que em blocos específicos, como os lógicos, a inicialização só é possível em um dos sentidos e, por isso, foram simulados para esses blocos apenas o caso 1 e o caso 2.

A seguinte padronização foi adotada para simulação:

1) Duração de 10 segundos para a simulação;

2) Evento com amplitude de -0,1 aplicado no instante de 1 s;

3) Passo de integração de 0,001 e frequência de plotagem igual a 1. Por frequência de plotagem, entende-se o intervalo de tempo entre dois pontos plotados em termos de passos de integração. Portanto, 1 significa que será plotado 1 ponto a cada passo de integração.

[25] A descrição matemática de cada um destes blocos encontra-se no Apêndice F.

Para efeitos de ilustração, na Figura 39 é apresentado o arquivo CDU que foi usado para validação do bloco WSHOUT e na Figura 40 o diagrama de blocos desse CDU. A padronização anterior foi adotada. Esse corresponde ao caso 1, ou seja, com aplicação de degrau e DEFVAL na variável de entrada. O valor do DEFVAL pode ser encontrado na linha 25 da Figura 39. A instrução para aplicação do degrau aparece em outro arquivo, que não é mostrado (arquivo .stb do Anatem).

```
01 DCDU
02 (
03 (--------------------------------------------------------------------
04 (ncdu) ( nome cdu )
05      1 Wshout
06 (--------------------------------------------------------------------
07 (
08 (--------------------------------------------------------------------
09 (EFPAR (npar) (     valpar     )
10 (--------------------------------------------------------------------
11 DEFPAR #T              0.1
12 (
13 (--------------------------------------------------------------------
14 (nb)i(tipo) (stip)s(vent) (vsai) ( p1 )( p2 )( p3 )( p4 ) (vmin) (vmax)
15 (--------------------------------------------------------------------
16 0001 ENTRAD                Vref
17 0002 LEDLAG         Vref   X1      1.0         1.0   #T
18 0003 GANHO          X1     X2      2.0
19 0004 WSHOUT         X2     X3     -1.0   1.0   2.0
20 0005 SAIDA          X3
21 (
22 (--------------------------------------------------------------------
23 (DEFVA (stip) (vdef) ( d1 )
24 (--------------------------------------------------------------------
25 DEFVAL          VREF    0.5
26 (
27 FIMCDU
28 (
29 999999
30  FIM
```

Figura 39 – CDU de Teste do bloco WSHOUT

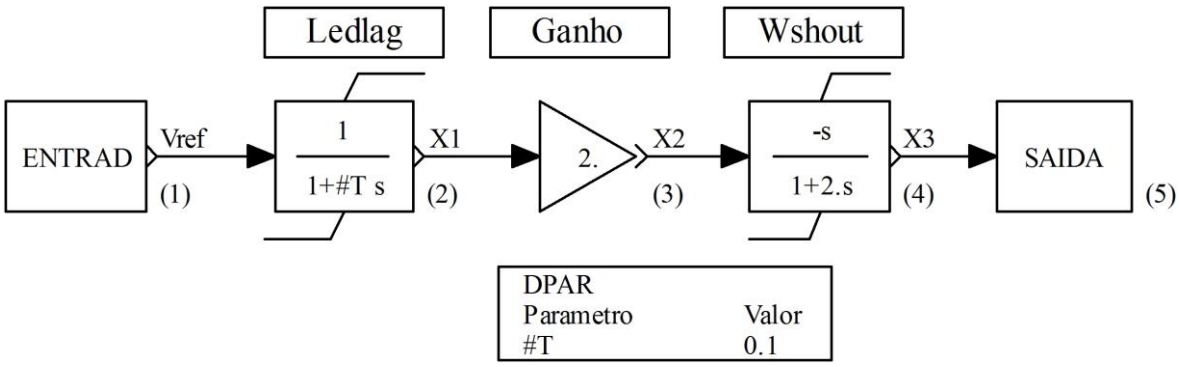


Figura 40 – Diagrama de blocos do CDU de Teste do Bloco WSHOUT

Na Figura 41 é mostrado o perfeito acoplamento entre a resposta do UDCC com o resultado do Anatem. Todos os blocos foram modelados rigorosamente da mesma forma, com exceção do bloco NOISE, que é um bloco de teste cujo objetivo é produzir um ruído

em torno do sinal e não aparece em nenhum CDU do SIN. Dessa forma, o ruído produzido pelo Anatem e aquele gerado pelo UDCC se baseiam em equações diferentes.

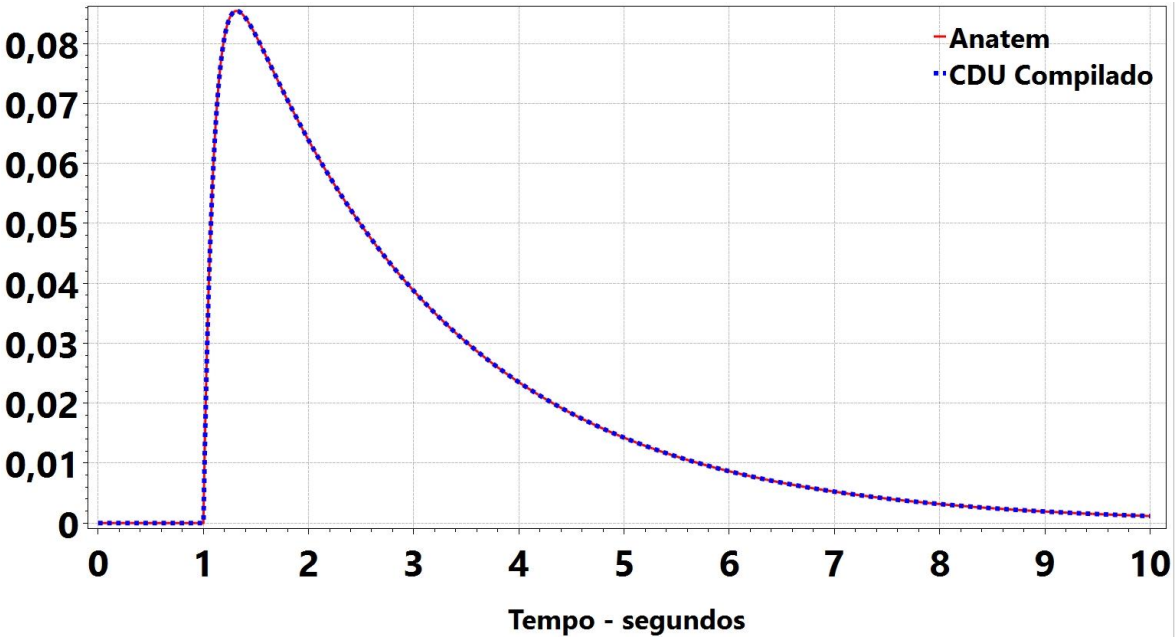


Figura 41 – Saída do Caso Teste do Bloco WSHOUT

3.2.2 Simulação Isolada (UDCrun)

Além da comparação bloco a bloco, o UDCC também foi submetido a uma validação considerando o caso real do SIN e realizando a comparação com o Anatem. Entretanto, há a uma dificuldade nisso. A comunicação de cada CDU do SIN com o sistema é realizada a partir dos blocos IMPORT (recebe o sinal da rede) e EXPORT (envia o sinal para a rede). Porém, para esta simulação o UDCC operaria de forma isolada sem considerar a rede[26]. Portanto, algumas modificações precisaram ser feitas nos CDUs do SIN para que o comparativo pudesse ser feito.

Primeiramente, os blocos IMPORT e EXPORT foram removidos do CDU. Porém, é justamente a perturbação da rede que impõe uma maior exigência sobre a solução de um

[26] Nota-se que o UDCC pode ser usado de forma isolado ou integrado a outros programas, como o AnaHVDC. Dessa forma, pode-se questionar o motivo de não se realizar a validação considerando o AnaHVDC, uma vez que não seria necessário modificar os CDUs para o comparativo. Entretanto, cabe lembrar que o AnaHVDC realiza a modelagem da rede por fasores dinâmicos e, dessa forma, considerando os transitórios eletromagnéticos. Por outro lado, o Anatem despreza esses transitórios eletromagnéticos. Portanto, o resultado dos programas seria diferente devido à própria natureza de cada um. E isso pode ser comprovado analisando-se os resultados dos tópicos 4.2.1, 4.2.2 e 4.3.2.

CDU. De modo a contornar essa simplificação indesejada que foi introduzida, criou-se uma malha de retroalimentação. Com isso, a saída do CDU busca reproduzir uma dinâmica da rede que normalmente é transmitida por meio do bloco IMPORT.

Os CDUs do SIN são, em sua maioria, reguladores de tensão e reguladores de velocidade, cada qual possuindo uma dinâmica própria. Por isso, foram criadas duas funções de transferência padrão, uma para cada tipo de regulador.

O esquema geral para os reguladores de tensão está ilustrado na Figura 42. Entre a função de transferência e o Regulador foi introduzido um bloco SOMA, de modo que o evento pudesse ser aplicado. Na Figura, V_T é a tensão terminal e EFD é a tensão de campo da máquina associada ao controlador.

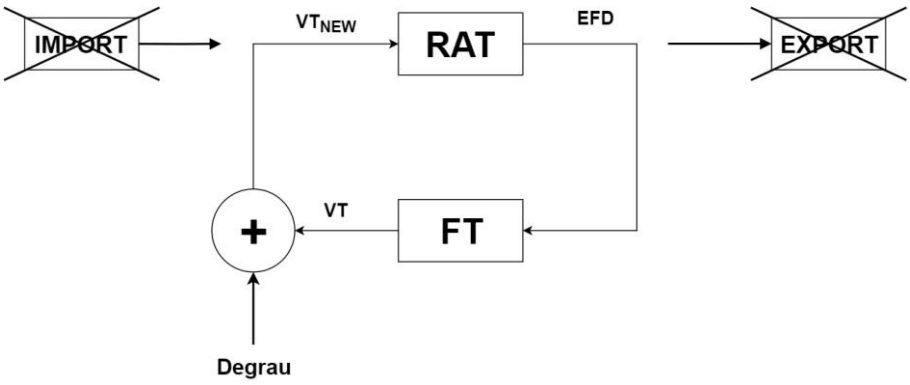


Figura 42 – Diagrama de Blocos dos Reguladores de Tensão

Na Figura 43 é exemplificado como esse procedimento foi implementado no regulador de tensão de Itumbiara. A função de transferência acoplada ao Regulador está destacada em azul.

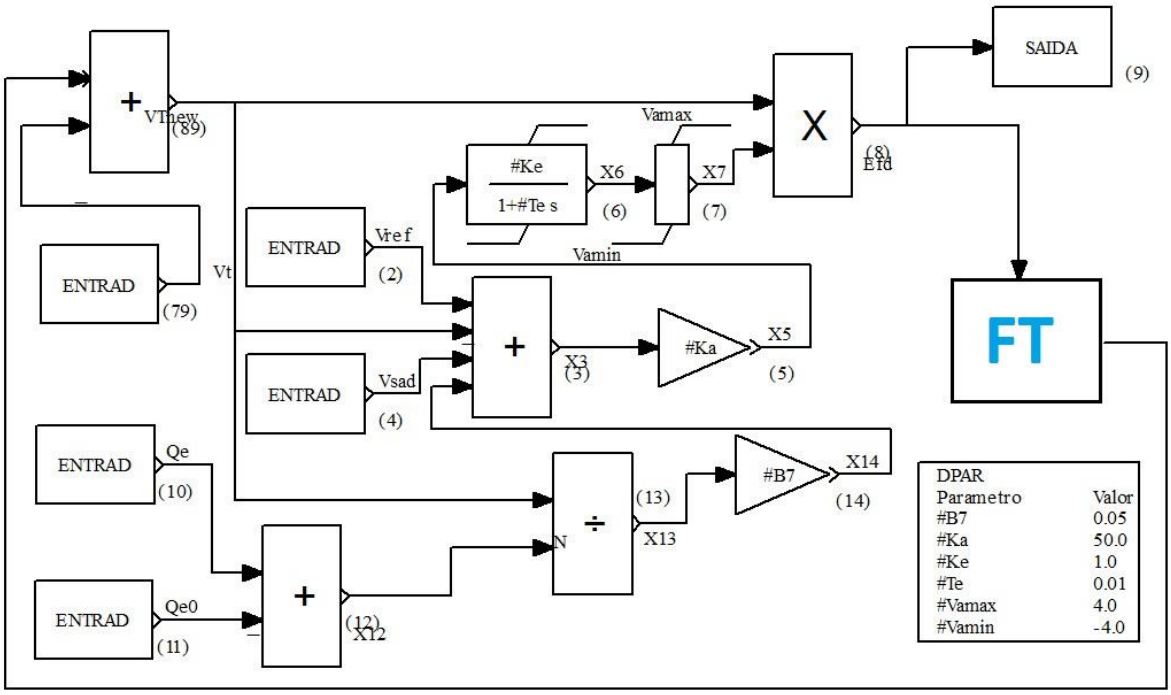


Figura 43 – Regulador de Tensão Itumbiara (Modificado)

Nos reguladores de tensão, em geral, o sinal recebido é a tensão terminal VT, que é comparada com um valor de referência e, a partir disso, o regulador ajusta a corrente de campo, de modo que V_T se iguale à tensão de referência. Nesse caso, é aplicado um degrau de amplitude -0,1 diretamente na tensão terminal. Diante disso, VT aumenta e o controlador atua no sentido de reduzir a tensão de campo, de modo a conduzir VT de volta ao seu valor inicial. Essa lógica pode ser percebida analisando-se o somador (3). Vsad e X14 são iguais à zero, valor que foi introduzido a partir de DEFVALs não representados na figura visando à simplificação da análise. Com isso, o somador se reduz a $Vref - VT_{new}$. Com aumento de VT_{new} (Figura 43) a saída do somador diminui e a tensão de campo (Figura 44) atinge valores próximos a zero. É interessante observar que em máquinas com excitatrizes rotativas a corrente de campo, em tese, pode atingir valores negativos. Isso não acontece em máquinas com excitatrizes estáticas, uma vez que isso implicaria nos tiristores da ponte da armadura conduzir corrente no sentido inverso, o que não é possível. Como a corrente no caso apresentado não atinge valores negativos, essa distinção não é relevante.

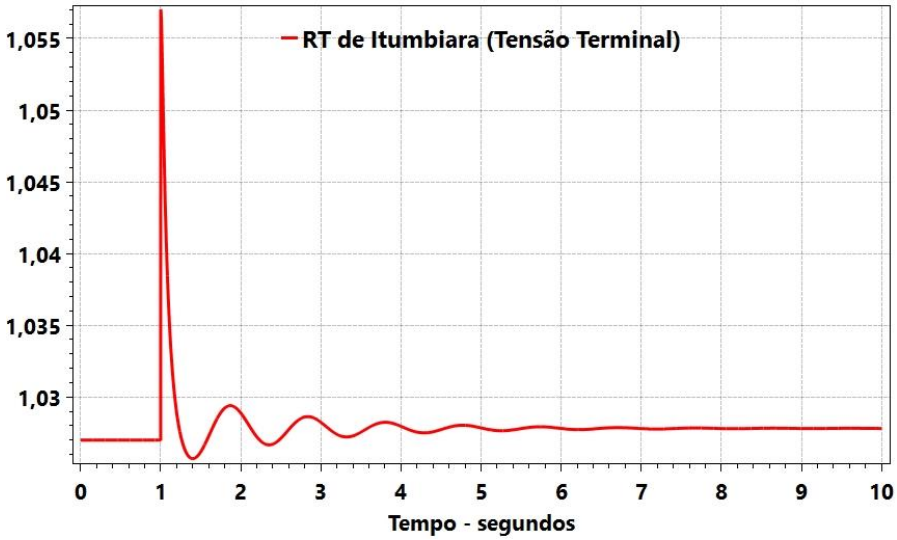


Figura 44 – Regulador de Itumbiara – Tensão Terminal (VT)

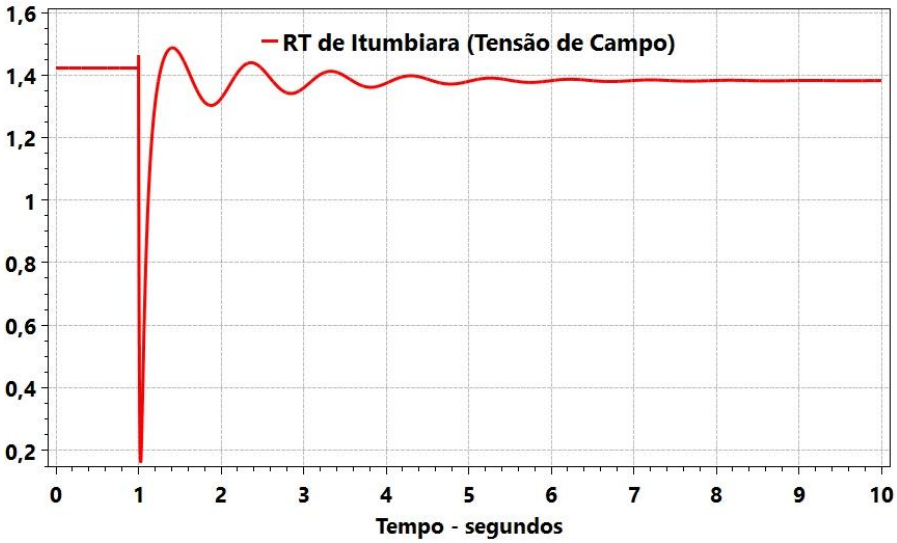


Figura 45 – Regulador de Itumbiara – Tensão de Campo (EFD)

Por sua vez, na Figura 46 está representada a Função de transferência que foi inserida no regulador de tensão de Itumbiara para simular a dinâmica da rede. Essa função de transferência é comum a todos os reguladores de tensão. Para síntese desta função, considerou-se primeiro um filtro de primeira ordem para emular de forma simplificada a relação entre tensão e corrente em malha aberta (modo da excitatriz) que normalmente possui uma redução de um ganho e constante de tempo da ordem de alguns segundos (no caso ajustou-se o ganho para 0,7218 e a constante de tempo para 2,5 s). Em uma malha paralela, é sintetizada a oscilação eletromecânica por uma função de segunda ordem realimentada, para produzir uma oscilação de frequência em torno do 1 Hz e algum amortecimento, que

pode até ser negativo dependendo do regulador de tensão que está sendo testado. Os pesos 0,8 e 0,2, que neste caso foram na razão de 4:1, foram ajustados para dosar a proporção do modo da excitatriz e do modo eletromecânico na resposta da tensão, que são somados ao final obedecendo estes pesos.

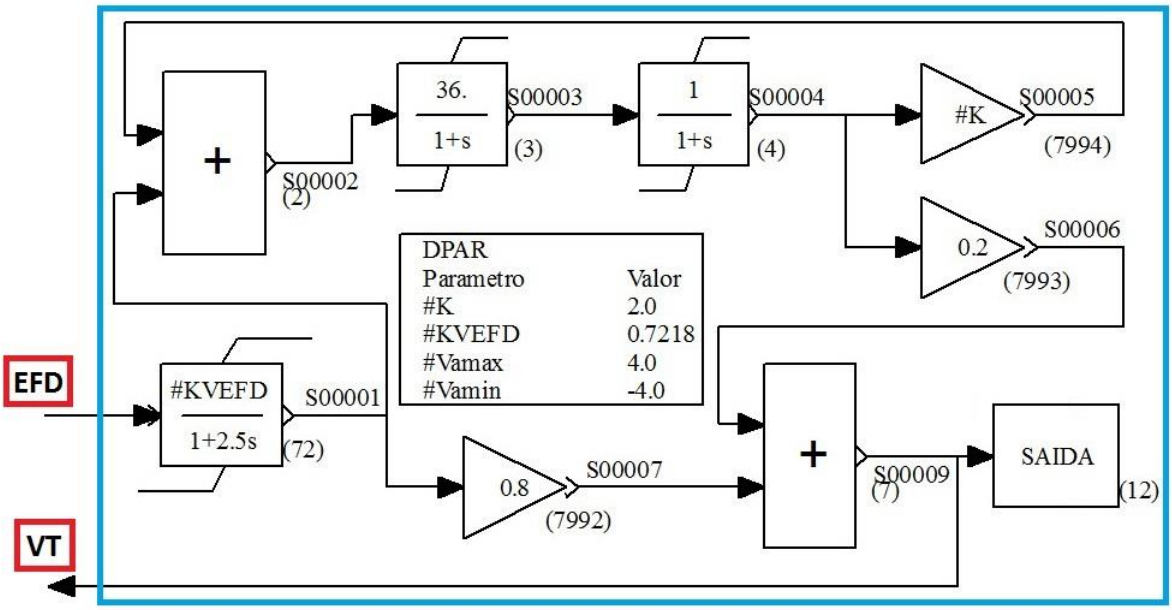


Figura 46 – Função de Transferência do Regulador de Tensão de Itumbiara Modificado

Em termos de código, as modificações realizadas no CDU de Itumbiara podem ser vistas na Figura 47. As linhas em negrito iniciadas por "(" são aquelas que foram comentadas em relação à versão original, enquanto as demais em negrito foram acrescentadas.

```
1   ANAC
2   DARQ
3   (Tipo) (C) ( Nome do Arquivo
4   OUT        ANA_000108_RT_ITUMB_GER.out
5   LOG        ANA_000108_RT_ITUMB_GER.log
6   PLT        ANA_000108_RT_ITUMB_GER.plt
7   999999
8   DCDU
9   (ncdu) ( nome cdu )
10     108 RT_ITUMB_GER
11  (-------------------------------------------------------------------
12  (EFPAR (npar) (     valpar     )
13  (-------------------------------------------------------------------
14  DEFPAR #B7                  0.05
15  DEFPAR #Ka                  50.0
16  DEFPAR #Ke                   1.0
17  DEFPAR #Te                  0.01
18  DEFPAR #Vamax                4.0
19  DEFPAR #Vamin               -4.0
```

```
(----------------------------------------------------------------
(nb) (tipo) (stip)s(vent) (vsai) ( p1 )( p2 )( p3 )( p4 ) (vmin)
(vmax)
(----------------------------------------------------------------
(EFPAR (npar) (     valpar     )
DEFPAR #KVEFD  0.721848347288135
DEFPAR #KKKKK  0.972222222222222
(EFVAL (stip) (vdef) ( d1 ) ( d2 )
DEFVAL        S00008     0.
(nb) (tipo) (stip)s(vent) (vsai) ( p1 )( p2 )( p3 )( p4 ) (vmin)
7999 ENTRAD                S00008
7998 LEDLAG         Efd    S00001 #KVEFD    0.    1.   2.5
7997 SOMA           S00001 S00002
                   -S00005 S00002
7996 LEDLAG         S00002 S00003    36.    0.    1.    1.
7995 LEDLAG         S00003 S00004     1.    0.    1.    1.
7994 GANHO          S00004 S00005 #KKKKK
7993 GANHO          S00004 S00006    0.2
7992 GANHO          S00001 S00007    0.8
7991 SOMA           S00006 S00009
                    S00007 S00009
7990 SOMA           S00009 Vt
                   -S00008 Vt
7989 SAIDA          S00009
(nb) (tipo) (stip)s(vent) (vsai) ( p1 )( p2 )( p3 )( p4 ) (vmin)
(vmax)
(0001 IMPORT VTR            Vt
0001 ENTRAD                Vt
(EFPAR (npar) (     valpar     )
DEFPAR #XX001       1.0270001335
(EFVAL (stip) (vdef) ( d1 ) ( d2 )
DEFVAL        Vt     #XX001
0002 ENTRAD               Vref
0003 SOMA          -Vt     X3
                    Vref   X3
                    Vsad   X3
                    X14    X3
(0004 IMPORT VSAD           Vsad
0004 ENTRAD                Vsad
(EFPAR (npar) (     valpar     )
DEFPAR #XX002                  0
(EFVAL (stip) (vdef) ( d1 ) ( d2 )
DEFVAL        Vsad   #XX002
0005 GANHO          X3     X5     #Ka
0006 LEDLAG         X5     X6     #Ke      0.0   1.0#Te
0007 LIMITA         X6     X7                              Vamin
0008 MULTPL         Vt     Efd
                    X7     Efd
(0009 EXPORT EFD    Efd
0009 SAIDA          Efd
(EFPAR (npar) (     valpar     )
DEFPAR #XX003       1.4227366972
(EFVAL (stip) (vdef) ( d1 ) ( d2 )
DEFVAL        Efd    #XX003
(0010 IMPORT QELE           Qe
0010 ENTRAD                Qe
(EFPAR (npar) (     valpar     )
DEFPAR #XX004     0.010773408631
(EFVAL (stip) (vdef) ( d1 ) ( d2 )
DEFVAL        Qe     #XX004
0011 ENTRAD               Qe0
```

```
0012 SOMA          Qe     X12
                  -Qe0    X12
0013 DIVSAO        X12    X13
                   Vt     X13
0014 GANHO         X13    X14    #B7
(-------------------------------------------------------------------
(EFVAL (stip) (vdef) ( d1 ) ( d2 )
(-------------------------------------------------------------------
(EFPAR (npar) (     valpar     )
DEFPAR #XX005     0.010773408631
(EFVAL (stip) (vdef) ( d1 ) ( d2 )
DEFVAL        QE0    #XX005
DEFVAL        Vamax  #Vamax
DEFVAL        Vamin  #Vamin
(
FIMCDU
999999
DEVT
(Tp) ( Tempo)( El )( Pa)Nc( Ex) ( % ) (ABS ) Gr Und          (Bl)P (
Rc ) ( Xc ) ( Bc ) (Defas)
TCDU      1.0   108                    -0.100                 7999
999999
DPLT
(Tipo)M( El ) ( Pa) Nc Gp ( Br) Gr ( Ex) (Bl) P   (
Nome da Variável                 )
CDU       108                              7989
999999
DSIM
( Tmax ) (Stp) ( P ) ( I ) ( F )
    10.0 0.001     1
EXSI
FIM
```

Figura 47 – Código CDU do Regulador Modificado de Itumbiara

Por outro lado, os reguladores de velocidade apresentam uma dinâmica diferente e, por isso, foi usada outra função de transferência, que nesse caso se resume apenas a um bloco PROINT que sintetiza a relação entre a variação de velocidade da máquina e a potência mecânica de forma simplificada. Na Figura 48 é mostrado o diagrama para reguladores desse tipo. DW é o desvio da velocidade angular da máquina em relação à velocidade síncrona enquanto P_{MEC} é a potência mecânica da máquina. O evento aplicado se dá no sentido de simular um aumento de frequência da rede, o que implica um aumento da velocidade síncrona da máquina [54].

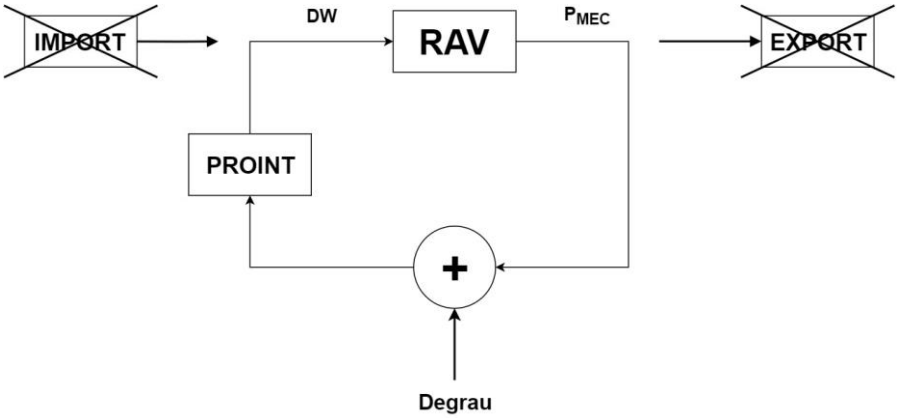


Figura 48 – Diagrama de Blocos dos Reguladores de Velocidade

A função primária de o controle é atenuar a perturbação reconduzindo o sistema ao regime permanente. Isso é feito alterando-se a potência mecânica da máquina. No caso apresentado, a frequência sofre um aumento. Dessa forma, a potência deve ser diminuída, o que é possível ser realizado diminuindo-se a velocidade mecânica da máquina. Portanto, se ajustando a potência mecânica da máquina o controle é capaz de reduzir o DW.

Como exemplo, é mostrado na Figura 49 o regulador de velocidade de Angra modificado, com a adição do bloco PROINT, que está destacado em azul.

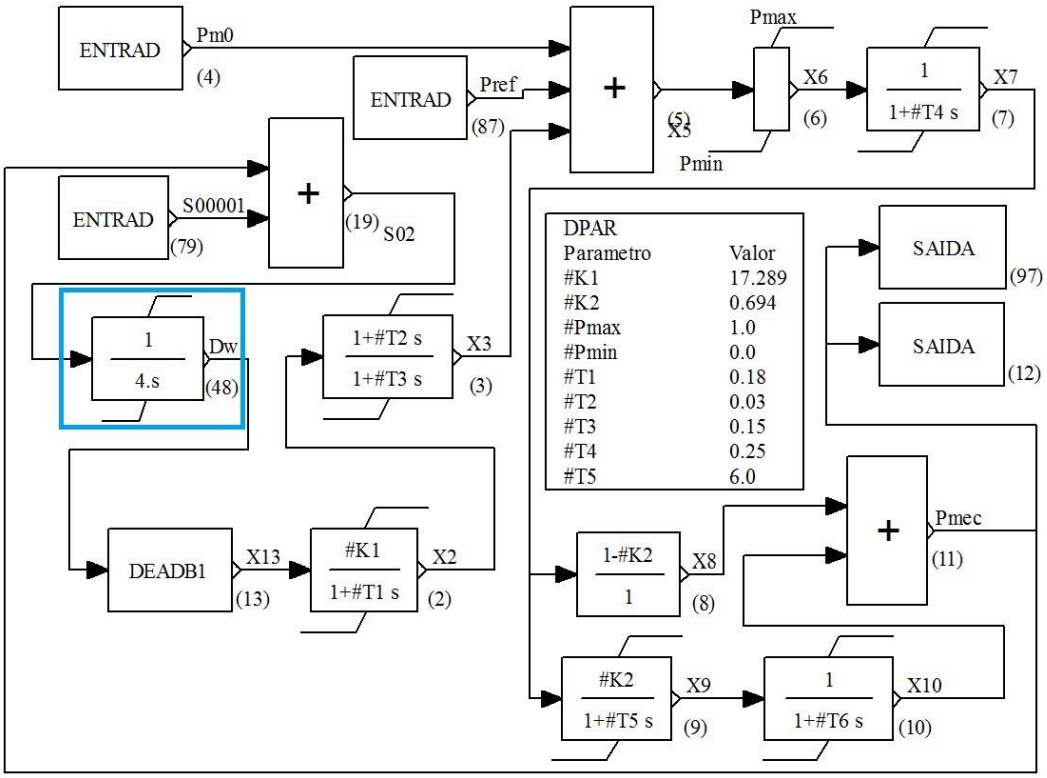


Figura 49 – Regulador de Velocidade de Angra (Modificado)

Na Figura 50 é mostrado o sinal do DW e na Figura 51 o sinal do PMec. Importante destacar que não é função do controlador zerar o DW, e sim reduzir o seu valor final.

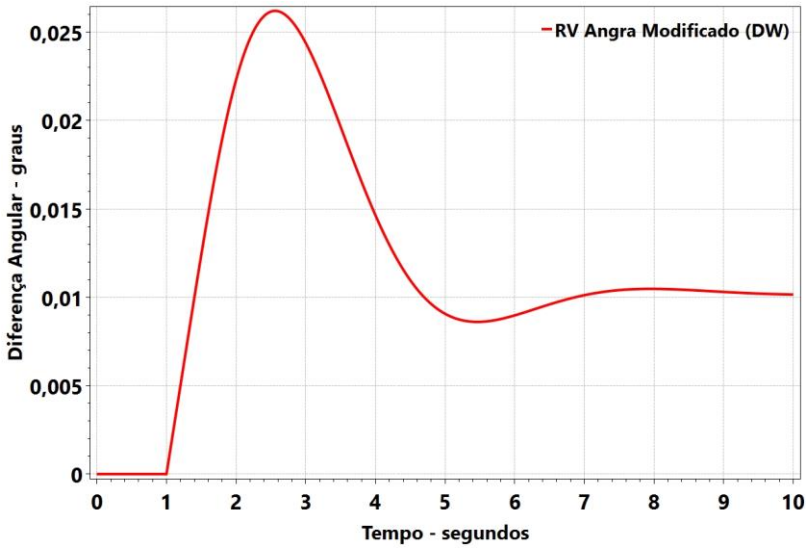


Figura 50 – Regulador de Velocidade de Angra Modificado – DW

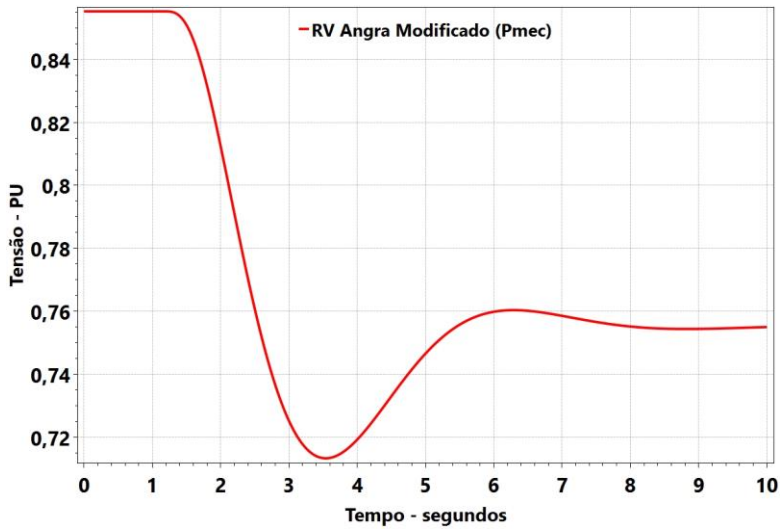


Figura 51 – Regulador de Velocidade de Angra Modificado – Potência Mecânica (P_{MEC})

Os estabilizadores não foram testados, uma vez que exigiriam um tratamento mais complexo para conexão nos reguladores de tensão e os ajustes poderiam estar inadequados para a função de transferência sintetizada nos testes. Além disto, os estabilizadores possuem normalmente topologias mais simples e não utilizam tantos blocos especiais quanto os reguladores de tensão e de velocidade. Por este motivo, acredita-se que os testes realizados

com estes dois últimos tipos de controladores já cobrem bem a variedade de blocos disponíveis no Anatem e implementados no UDCC, cumprindo o objetivo da validação desejada.

3.2.3 Automatização da Validação

O número de simulações necessárias para realização de todas as validações descritas é muito alta. Cabe relembrar que na validação bloco a bloco constam 85 blocos, cada qual com no mínimo 3 testes. Por sua vez, a do SIN abarcou mais de 400 CDUs. Conjuntamente isso implicou aproximadamente 600 simulações. Dessa forma, uma validação comparando-se manualmente as plotagens resultantes do UDCC e do Anatem seria impraticável.

A fim de contornar esse problema, foram criados testes automáticos. Primeiramente, foi criado um arquivo batch, de modo a executar automaticamente, no Anatem, todos os CDUs usados como base para validação[27] e, com isso, gerando os arquivos de plotagem respectivos. Um recorte do batch é mostrado na Figura 52.

```
@echo off
set ANATEM=C:\CEPEL\Anatem\v12_0_26dev\ANATEM.EXE

cd C:\Projetos\AnaHVDC\Cpp\trunk\Udc\tests\data\Blocos\Abs\Degrau\
%ANATEM% /i AbsStep.stb

cd C:\Projetos\AnaHVDC\Cpp\trunk\Udc\tests\data\Blocos\Abs\Entrada\
%ANATEM% /i AbsInput.stb

cd C:\Projetos\AnaHVDC\Cpp\trunk\Udc\tests\data\Blocos\Abs\Saida\
%ANATEM% /i AbsOutput.stb

cd C:\Projetos\AnaHVDC\Cpp\trunk\Udc\tests\data\Blocos\Acos\Degrau\
%ANATEM% /i AcosStep.stb

cd C:\Projetos\AnaHVDC\Cpp\trunk\Udc\tests\data\Blocos\Acos\Entrada\
%ANATEM% /i AcosInput.stb
```

Figura 52 – Arquivo batch para plotagem automática de CDUs via Anatem

Portanto, no arquivo batch é criada uma variável de ambiente chamada “ANATEM”. Com isso, é possível executar diretamente por linha de comando o programa apenas pela expressão %ANATEM%.

[27] Isso inclui tanto os CDUs referentes à validação bloco a bloco (tópico 3.2.1) como os CDUs do SIN (tópico 3.2.2).

No caso do UDCC, os arquivos de cada CDU foram condensados num único arquivo denominado AnaHVDC.stb[28] e executados pelo UDCrun que, por sua vez, dispõe de diversas opções de simulação conforme a Figura 53.

```
-a Seta a amplitude do degrau
-c Seta o número de threads concorrentes a serem utilizadas
-d Seta o intervalo de tempo da simulacao (delta T)
-e Seta o tempo em que o evento deve ser aplicado
-f Seta o tempo final
-g Seta a granularidade da distribuição dos CDUs entre as threads
-h Mostra essa mensagem de ajuda
-i Seta o tempo inicial
-m Sinaliza insercao de CDUs por intervalo
-n Indica que não deve contabilizar nenhuma variavel para a plotagem (nem as saídas)
-o Seta o nome do arquivo de plotagem
-p Indica que deve plotar todas as variáveis (e não apenas as saídas)
-v Seta o indice da variavel em que o degrau sera aplicado
```

Figura 53 – Opções de Solução no UDCC

Para as simulações foram usadas como opções: -a-0.1 AnaHVDC (o último argumento é o nome da DLL gerada sem a sua extensão).[29] O degrau é ajustado para uma amplitude negativa de 0,1 pu (-a-0.1). Além disso, outras instruções não indicadas se encontram implícitas, o que significa que são valores *default* assumidos pelo programa. Portanto, o UDCC assume a aplicação do evento no tempo de 1 s (-e), o tempo final (-f) de 10 s, o tempo inicial (-i) de 0 segundo, a granularidade (-g) igual a 1, o uso de 4 *threads*, plotagem apenas da primeira variável de saída e aplicação do evento no primeiro bloco ENTRAD de cada CDU. Dessa forma, são gerados os arquivos de plotagem por meio do UDCC, que podem ser então comparados com aqueles gerados pelo Anatem. Para descrever como isso acontece, considere-se o arquivo de plotagem exemplo dado na Figura 54.

[28] O arquivo AnaHVDC.stb inclui todos os CDUs usados como base para validação.
[29] A partir do arquivo AnaHVDC.stb, o UDCC gera a DLL AnaHVDC.dll, que é usada como último argumento. A extensão .dll é assumida pelo programa.

```
    5
Tempo - segundos
CDU          1     1 VREF     wshout
CDU          1     2 X1       wshout
CDU          1     3 X2       wshout
CDU          1     4 X3       wshout
 0.000000E+00 0.500000E+00 0.500000E+00 0.100000E+01 0.000000E+00
 0.100000E-02 0.500000E+00 0.500000E+00 0.100000E+01 0.000000E+00
 0.200000E-02 0.500000E+00 0.500000E+00 0.100000E+01 0.000000E+00
 0.300000E-02 0.500000E+00 0.500000E+00 0.100000E+01 0.000000E+00
 0.400000E-02 0.500000E+00 0.500000E+00 0.100000E+01 -.111022E-15
 0.500000E-02 0.500000E+00 0.500000E+00 0.100000E+01 -.138778E-15
 0.600000E-02 0.500000E+00 0.500000E+00 0.100000E+01 -.166533E-15
 0.700000E-02 0.500000E+00 0.500000E+00 0.100000E+01 -.194289E-15
```

Figura 54 – Exemplo de arquivo PLT

O PLT da Figura 54 é o resultado da simulação do CDU da Figura 39. A partir da sétima linha, a primeira coluna corresponde ao instante de tempo e as colunas seguintes à saída de cada bloco na ordem em que aparecem no CDU. Dessa forma, um PLT é essencialmente um arquivo texto com uma formatação padronizada especificando o valor da saída de cada bloco para cada instante de tempo da simulação.

Portanto, foi baseando-se nisso que foi elaborada a estratégia para determinar se as curvas de plotagem são coincidentes, a semelhança das curvas do exemplo da Figura 41. Primeiramente verifica-se se o número de blocos em ambos os PLTs são iguais. Essa informação é dada na primeira linha. Na sequência é checado se os valores a partir da sétima linha de um PLT são coincidentes com os valores de mesma posição no outro PLT.

3.3 Compiladores do UDCC

Nos testes foram usados dois compiladores. O primeiro foi o TCC (Tiny C Compiler), que é um compilador de grande simplicidade, realizando apenas otimizações básicas como as descritas no tópico 2.6. Isso implica a geração de um código assembly pouco eficiente a ponto de ser, de modo geral, mais lento do que o próprio Anatem para os casos testados e que serão analisados no Capítulo 4. Entretanto, a importante vantagem é que a compilação se dá de modo extremamente rápido.

O outro compilador usado foi o Clang, que é altamente sofisticado. Isso faz com que tenha características inversas ao do TCC: é lento na compilação, porém o código assembly gerado é altamente otimizado. O TCC e o Clang encontram-se embutidos ao UDCC. Isso

significa que não é necessário que o usuário possua em sua máquina esses compiladores instalados para que possa realizar simulações com o UDCC.

No UDCC estão embutidos ambos os compiladores, cada um tendo sua importância específica. O CDU do Sistema Interligado Nacional é atualizado semestralmente pelo ONS. Se o analista for trabalhar com esse CDU é necessário que seja realizada a compilação do CDU e geração da DLL, que constitui a primeira etapa para qualquer CDU simulado no UDCC. A partir da DLL o usuário pode realizar diferentes simulações sem que seja necessária uma recompilação. O TCC gera a DLL a partir do CDU do SIN em apenas alguns segundos. Por outro lado, o Clang, com máxima otimização, requer aproximadamente meia hora. Entretanto, nesse caso, a opção pelo Clang é a mais vantajosa. Embora inicialmente haja o revés do tempo de compilação ele irá ocorrer uma única vez até que o CDU do SIN seja atualizado e seja necessária uma nova compilação. Porém, como mencionado, isso ocorre apenas semestralmente. Devido à compilação altamente otimizada do Clang, o analista poderá, a partir da DLL gerada, simular seus casos muito mais rapidamente do que se tivesse optado pelo TCC. Dessa forma, compensa-se em longo prazo o tempo de compilação do Clang. Além do mais, a compilação, após lançada pelo usuário, é executada sem necessidade de interferência externa. Portanto, durante a compilação o usuário pode trabalhar normalmente ou ainda lançar a compilação em seu tempo vago. Desse modo, a meia hora de compilação é irrelevante comparado ao ganho em longo prazo.

Por outro lado, o TCC também possui sua importância, o que justifica estar embutido no UDCC. Caso o analista precise compilar um CDU grande, porém que venha a ser usado poucas vezes, o tempo de compilação pode ser considerado uma desvantagem significativa, uma vez que não existe o ganho em longo prazo. Enquanto isso, pelo mesmo motivo, a perda de eficiência é pouco significativa.

3.4 Otimizações no UDCC

Um exemplo prático ilustra o grau de otimização que os compiladores são capazes de obter. Considere-se o CDU da Figura 55.

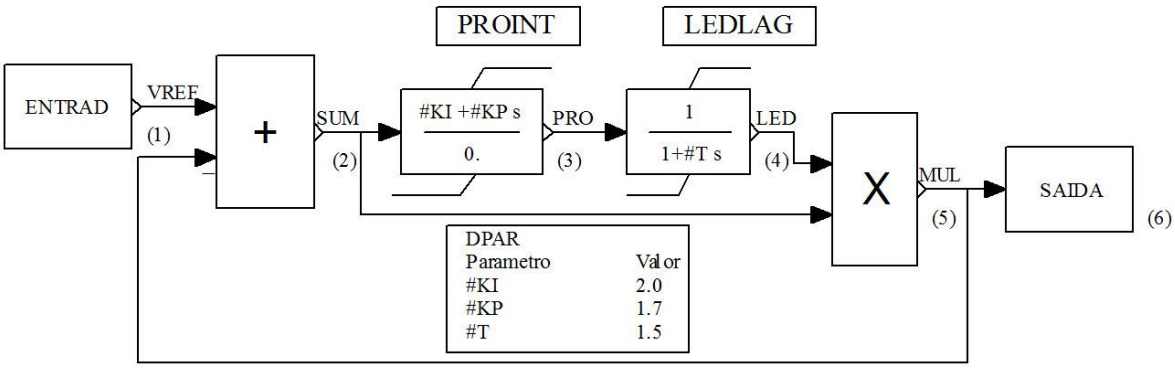


Figura 55 – CDU Exemplo

Na Figura 56 é apresentado um código C que representa a solução do CDU exemplo anterior.

```
1  void solcdu(double *b01, double *c01, double *x){
2    x[1]=x[0]-x[4];              // SUM=VREF-MUL
3    x[2]=c01[0]*x[1]+b01[0];     // PRO=c01[0]*SUM+b0[0]
4    x[3]=c01[1]*x[2]+b01[1];     // LED=c01[1]*PRO+b0[1]
5    x[4]=x[3]*x[2];              // MUL=SUM*LED
6  }
```

Figura 56 – Código C para Solução do CDU Exemplo

Para testes foi usado o compilador Clang x86-64 versão 10.0.0. Sem instruções explícitas para otimização, o Clang traduz o código C para o código assembly da Figura 57, que é próximo ao código gerado pelo TCC.

```
1  _Z6solcduPdS_S_: # @_Z6solcduPdS_S_
2  push rbp
3  mov rbp, rsp
4  mov qword ptr [rbp - 8], rdi
5  mov qword ptr [rbp - 16], rsi
6  mov qword ptr [rbp - 24], rdx                    // Linha 1
7  mov rax, qword ptr [rbp - 24]
8  movsd xmm0, qword ptr [rax]
9  mov rax, qword ptr [rbp - 24]
10 subsd xmm0, qword ptr [rax + 32]
11 mov rax, qword ptr [rbp - 24]
12 movsd qword ptr [rax + 8], xmm0                   // Linha 2
13 mov rax, qword ptr [rbp - 16]
14 movsd xmm0, qword ptr [rax]
15 mov rax, qword ptr [rbp - 24]
16 mulsd xmm0, qword ptr [rax + 8]
17 mov rax, qword ptr [rbp - 8]
18 addsd xmm0, qword ptr [rax]
19 mov rax, qword ptr [rbp - 24]
20 movsd qword ptr [rax + 16], xmm0                  // Linha 3
21 mov rax, qword ptr [rbp - 16]
22 movsd xmm0, qword ptr [rax + 8]
23 mov rax, qword ptr [rbp - 24]
24 mulsd xmm0, qword ptr [rax + 16]
25 mov rax, qword ptr [rbp - 8]
26 addsd xmm0, qword ptr [rax + 8]
27 mov rax, qword ptr [rbp - 24]
28 movsd qword ptr [rax + 24], xmm0                  // Linha 4
29 mov rax, qword ptr [rbp - 24]
30 movsd xmm0, qword ptr [rax + 24
31 mov rax, qword ptr [rbp - 24]
32 mulsd xmm0, qword ptr [rax + 16]
33 mov rax, qword ptr [rbp - 24]
34 movsd qword ptr [rax + 32], xmm0                  // Linha 5
35 pop rbp
36 ret                                              // Linha 6
```

Figura 57 – Código assembly (sem otimização)

Passa-se a análise do código instrução por instrução. Como o código é longo ele foi segmentado por uma linha contínua a qual corresponde a cada linha do código da Figura 56. O objetivo é facilitar a correlação entre os códigos.

Linhas 2–3. As duas primeiras linhas do código são as convenções de chamada já explicadas no tópico 2.3.

Linhas 4–6. O endereço de memória contido em *rdi, rsi e rdx,* que são os argumentos da função, são colocados na pilha.

Linha 7. Na linha 7 o conteúdo de *rbp -24*, que é o endereço de memória do terceiro argumento da função, é copiado para o registrado *rax*.

Linha 8. Neste instante, x[0] (o valor do primeiro elemento do terceiro argumento da função) é copiado para *xmm0*, um registrador que é capaz de armazenar até 128 bits. Isso é indicado na instrução *movsd*, usada quando os operadores possuem 128 bits [30].

Linha 9. Essa é repetição da linha 7. Sem otimização o compilador ignora o fato de que o endereço de *x* já está armazenado no registrado *rax* e realiza novamente a operação.

Linha 10. É realizada a subtração entre x[0] e x[4]. Nesse caso, o endereço de x[4] está 32 bits depois do endereço de x[0], que está armazenado no registrado *rax*. Por isso, a instrução *qword ptr [rax + 32]*.

Linha 11. Mesmo comentário feito sobre a linha 9.

Linha 12. O conteúdo de registrador *xmm0*, que neste ponto é o resultado de x[0] – x[4], é copiado para [*rax* + 8], ou seja, para x[1]. Vale relembrar que *rax* é o endereço de x[0].

Linha 13. O endereço do elemento *c01* é copiado para o registrado *rax*.

Linha 14. O valor de c01[0] é copiado para o registrador *xmm0*.

Linha 15. O endereço do elemento *x* é copiado para o registrado *rax*.

Linha 16. Acontece a multiplicação entre o valor em [*rax* + 8], que no caso é x[1] e o conteúdo do registrador *xmm0*, a saber c01[0]. O resultado é armazenado no próprio c[0].

O código prossegue seguindo a lógica descrita anteriormente e por isso a análise do assembly gerado pelo Clang sem otimização é encerrado neste ponto.

Por sua vez, na Figura 58 é mostrado o código assembly produzido pelo Clang com otimização -O3

```
1   solcdu(double*, double*, double*)
2   movsd xmm0, qword ptr [rdx]
3   subsd xmm0, qword ptr [rdx + 32]
4   movsd qword ptr [rdx + 8], xmm0
5   mulsd xmm0, qword ptr [rsi]
6   addsd xmm0, qword ptr [rdi]
7   movsd qword ptr [rdx + 16], xmm0
8   movsd xmm1, qword ptr [rsi + 8]
9   mulsd xmm1, xmm0
10  addsd xmm1, qword ptr [rdi + 8]
11  movsd qword ptr [rdx + 24], xmm1
12  mulsd xmm1, xmm0
13  movsd qword ptr [rdx + 32], xmm1
14  ret
```

Figura 58 – Código assembly (Clang com Otimização -O3)

A redução de código é significativa e por isso sua execução é muito mais veloz do que aquele produzida sem otimização. Passa-se à análise.

Linha 2. Acontece a cópia do valor armazenado no endereço do registrado *rdx* (terceiro argumento da função) para o registrador *xmm0*. Isso só é possível porque por convenção os registradores seguem uma ordem específica para armazenamento. *rax* é o primeiro parâmetro da função e, na sequência, *rcx*, *rdx*... [55]. Essa é a convenção no Linux, sistema operacional no qual ocorre a compilação de [29]. Dessa forma, está implícito que o primeiro argumento da função é atribuído a *rax* e a partir daí seguindo a ordem mencionada.

Linha 3. É realizada a subtração entre x[0] e x[5]. É exatamente igual à linha 10 do código sem otimização.

Linha 4. O conteúdo de registrador *xmm0*, que neste ponto é o resultado de x[0] - x[4], é copiado para [rax + 8], ou seja, para x[1]. É exatamente igual à linha 12 do código sem otimização.

Linha 5. Essa linha corresponde a linha 16 do código sem otimização. Importante notar, entretanto, que elas não são iguais. Como essa otimização é sutil, transcreve-se lado a lado o código otimizado (à direita) correspondente ao não otimizado (à esquerda).

`mov rax, qword ptr [rbp - 16]`
`movsd xmm0, qword ptr [rax]`
`mov rax, qword ptr [rbp - 24]`
`mulsd xmm0, qword ptr [rax + 8]`
	`mulsd xmm0, qword ptr [rsi]`

Figura 59 – Recorte dos códigos otimizado e sem otimização

O interessante é perceber que sem otimização o compilador atribui a *rax* o endereço de *c*. Depois atribui ao registrador *xmm0* o valor contido em *rax*, no caso c01[0]. Na sequência o endereço de *x* é atribuído a *rax*. Por fim, o valor no endereço de *rax + 8*, correspondente a x[1], é multiplicado com *xmm0*, correspondente o valor c01[0], e armazenado no próprio *xmm0*. Na versão otimizada, o compilador percebe que *xmm0*, por causa da linha 3, está ainda associado ao endereço de x[1] e, portanto, ele é usado diretamente na multiplicação com o valor armazenado em *rsi*, no caso c01[0].

3.5 Interpretação no Anatem e Comparativo com o UDCC

Como mencionado, o Anatem possui o seu código fonte compilado e, portanto, otimizado. Entretanto, a solução do CDU se dá de modo interpretado.

É útil descrever detalhadamente como funciona a interpretação no caso específico do Anatem. Cada CDU é escrito no que se pode chamar de linguagem CDU, ou seja, um conjunto de instruções que precisam ser executadas por um interpretador que as entenda. No caso, o interpretador é o Anatem. Dessa forma, o arquivo CDU é importado para vetores internos até que se atinja o fim do arquivo, em que cada vetor é responsável por armazenar um tipo de informação. Desse modo, há um vetor para os blocos, 4 vetores cada qual para um tipo de parâmetro (P1, P2, P3 e P4), um vetor para os termos históricos e 6 vetores para as constantes, dentre outros. Isso é mostrado na Figura 60[30]. As variáveis são as entradas e as saídas dos blocos. Na Figura 60, por simplificação, é representado apenas o vetor das constantes c01. Dessa forma, o bloco LEDLAG representado na figura também aponta para o primeiro elemento do vetor das constante c02, c03 e c04. O mesmo acontece para os parâmetros.

[30] A Figura 60 é uma representação simplificada da estrutura de armazenamento de um CDU no Anatem. Como será descrito ainda neste tópico o acesso aos dados não acontece de modo direto.

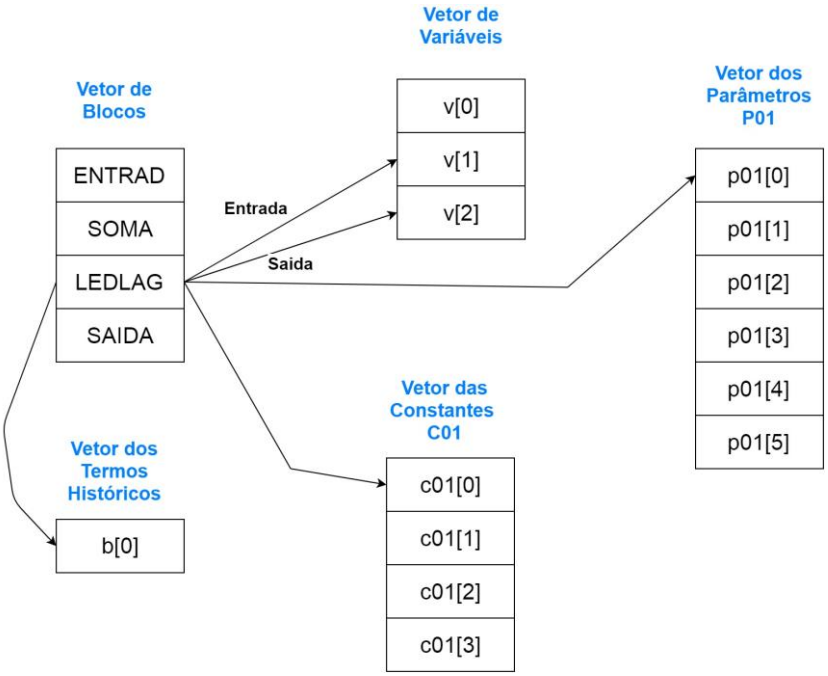


Figura 60 – Estrutura de Armazenamento de Dados no Anatem

A partir desse ponto se inicia a etapa de solução. Desse modo, cada elemento do vetor de blocos é percorrido e o Anatem sabe como realizar a interpretação. No caso, a interpretação é buscar em um loop o bloco correspondente ao do elemento do vetor, conforme a Figura 60. É importante perceber que isso é exatamente o que uma linguagem interpretada tradicional realiza, conforme o exemplo da Figura 15.

O loop do Anatem de busca de bloco é ilustrado na Figura 61, que é um recorte adaptado do código do Anatem.

```
1  do i=FirstBlock, lastBlock
2     block = blockType(i)
3  C.  Instrucoes do codigo original nao mostradas aqui
4      select case( block )
5      case( GANHO )
6        CALL solGanho (...)
7      case( PROINT )
8        CALL solProint(...)
9  C.  Instrucoes do codigo original nao mostradas aqui
10     case( LEDLAG )
11       CALL solLedlag (...)
12 C.  Instrucoes do codigo original nao mostradas aqui
13     end select
14 Enddo
```

Figura 61 – Código de Solução do Anatem

A solução equivalente no UDCC é mostrada na Figura 62.

```
1  void solution (...) {
2    for (...) {
3      ledlag (...);
4      ganho (...);
5      fracao (...);
6      ...
7    }
}
```

Figura 62 – Loop de Solução UDCC

Dessa forma, é possível perceber algumas diferenças entre as abordagens de solução do UDCC e do Anatem, que são elencadas a seguir. A quarta, quinta e sexta diferenças são as mais relevantes.

A primeira diferença pode ser compreendida analisando-se a linha 2 da Figura 61. Nesse ponto, o Anatem determina a partir do vetor de blocos o próximo bloco a ser resolvido, o que acontece em tempo de execução. Essa instrução não existe no UDCC, uma vez que o próximo bloco a ser resolvido é aquele que se encontra na linha seguinte.

A segunda diferença é que no Anatem é necessário determinar qual o tipo do bloco que foi importado do vetor de blocos para que seja chamada a função de solução respectiva. Essa determinação é realizada pela instrução *select case* (linha 4 da Figura 61). No caso do UDCC, essa determinação não é necessária uma vez que as funções de solução já estão ordenadas sequencialmente.

A terceira diferença é a existência, no Anatem, da instrução DO da linha 1 da Figura 61. O Anatem desconhece em tempo de compilação quantos blocos de CDU devem ser lidos. Portanto, é necessário checar em tempo de execução se foi percorrida toda a lista de blocos. Novamente essa instrução não existe no UDCC uma vez que o último bloco a ser lido é o fim da instrução FOR da linha 2 da Figura 62.[31]

A quarta diferença refere-se aos parâmetros. Para cada bloco solucionado, o Anatem precisa resgatar os valores dos parâmetros no vetor respectivo. No caso do UDCC a situação é diferente. Para essa análise retome-se a Figura 34. Nela constam diversas instruções do tipo *#define*, que são instruções do pré-processador. Isso significa que essa substituição é realizada antes de o compilador executar, o que é esquematizado na Figura 63.

[31] Destaca-se que o FOR da linha 2 da Figura 62 corresponde ao loop de convergência. A instrução correspondente no Anatem é um loop externo ao DO da linha 1 da Figura 61 e que não é mostrado.

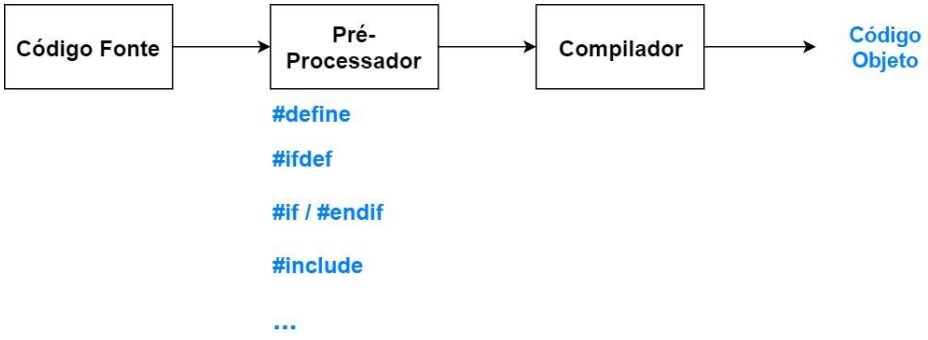


Figura 63 – Fluxo de um programa até a geração do código objeto

Portanto, quando a compilação acontece, o código que de fato está sendo compilado é o da Figura 64. Importante notar que a substituição do pré-processador é puramente textual.

```
1  void constantesCdu(double* c, double step) {
2    double aux1 = 2 / step;
3    c[0] = ((2.0)+aux1 * (1.7)) / aux1;    // C0 PROINT
4    c[1] = 1.0;                            // C1 PROINT
5    c[2] = ((2.0)-aux1 * (1.7)) / aux1;    // C2 PROINT
6    double aux2 = (2 / step) * (1.5);
7    c[3] = 1 / (aux2 + 1);                 // C0 LEDLAG
8    c[4] = (aux2 - 1) / (aux2 + 1);        // C1 LEDLAG
9  }
```

Figura 64 – Cálculo das Constantes na fase logo após pré-processador

Dessa forma, tome-se como exemplo a linha 3. A constante *par_Ki*, substituída por 2.0, é um dos parâmetros do bloco. No Anatem não existe a substituição do parâmetro pelo seu valor numérico, de modo que é necessário que esse valor seja resgatado no vetor dos parâmetros, indicado na Figura 60. Por outro lado, no UDCC esse *overhead* não existe. Esta diferença é significativa pelo seu efeito multiplicativo, uma vez que o número de acessos desse tipo é muito grande. Apenas na Figura 34 foram 6 acessos de parâmetros. Cabe observar que o mesmo acontece na solução de diversos blocos, em que é necessário o acesso a parâmetros durante a solução. Um exemplo desse caso é o bloco GANHO, um dos blocos mais usados no SIN, cuja solução é definida conforme a Equação (25).

$$V_{SAI}\left(t\right) = P_1 \times V_{ENT}\left(t\right) \tag{25}$$

Por outro lado, não se pode adotar a mesma estratégia com as constantes. Conforme a Figura 34, percebe-se que as constantes dependem do passo integração *step*. Embora a possibilidade de variação de passo ainda não esteja implementada, isso acontecerá no futuro e o código já se encontra preparado para isso. Portanto, é necessário chamar a função de cálculo das constantes toda vez que seja alterado o passo de integração. Dessa forma, a substituição por meio da diretiva do pré-processador não pode ser realizada.

A quinta diferença é mascarada pela estrutura simplificada da Figura 60. Na realidade, os acessos não acontecem de forma direta, sendo necessária a passagem por um vetor intermediário. Dessa forma, o bloco LEDLAG destacado na figura não sabe diretamente que sua entrada é o segundo elemento do vetor de variáveis. É necessário que o bloco acesse um vetor intermediário que lhe informe qual a posição no vetor de variáveis que corresponde a sua entrada. No caso do UDCC o acesso existe, porém se dá de modo direto. Ou seja, o Anatem precisa de dois acessos para resgatar um valor, enquanto o UDCC precisa apenas de um.

A sexta diferença é melhor compreendida retomando-se a Figura 58. Nela o compilador percebe que *xmm0* usado na linha 5 ainda está associado ao endereço de x[1] quando é usado na instrução seguinte. Com isso, elimina-se a cópia do endereço de x[1] para *xmm0*. Entretanto, essa otimização, realizada pelo compilador Clang integrado ao UDCC, está fora do alcance do Anatem. Isso acontece porque, no Anatem, entre a chamada para solução de um bloco e de outro existem diversas instruções intermediárias. Tomando-se por base a Figura 61. Após a solução de um bloco, o Anatem precisa incrementar o loop DO e na sequência determinar o próximo bloco a ser resolvido acessando a função *blockType*. Por fim, ainda é necessário determinar o tipo do bloco a partir da instrução *select case*. Apenas depois a função de solução apropriada é de fato chamada. O resultado é que o valor contido no registrador da saída do bloco anterior registrador foi sobrescrito pelas instruções intermediárias e, com isso, precisa ser importado novamente da memória RAM, implicando um aumento de tempo significativo, o que pode ser deduzido a partir da Tabela 1.

Portanto, otimizações interblocos, como descritas no parágrafo anterior, não são realizadas pelo Anatem. Isso é resultado direto da abordagem interpretada utilizada, em que as instruções de solução de blocos não são sequenciais e é necessário determinar qual o bloco seguinte a ser solucionado. Destaca-se, entretanto, que otimizações intrablocos são realizadas, o que se deve ao fato do código fonte do Anatem ser compilado. Desse modo, todo o código específico de cada bloco é otimizado. Ou seja, as equações de cálculo de

constantes e as de cálculo da solução de cada bloco (determinação da saída a partir da entrada, das constantes e possivelmente do termo histórico[32]) são otimizadas.

Com essas considerações, retoma-se o tópico 2.5, em que foram elencadas as diferenças entre linguagens compiladas e interpretadas. Neste ponto, é possível aplicar essas diferenças ao caso específico do UDCC e do Anatem. Primeiramente, implementações compiladas, como o UDCC, tendem a ser mais velozes. De modo geral, em linguagens interpretadas, o *script* é lido, traduzido e executado em *runtime* pelo interpretador. No caso do Anatem, o CDU é equivalente ao *script* e o interpretador é o próprio Anatem. Da mesma forma que o *script*, o processamento do CDU se dá em *runtime*.

3.6 Integração ao AnaHVDC

O UDCC tem como objetivo gerar um código C de inicialização e de solução de CDUs e compilá-lo em memória gerando uma DLL em código de máquina. Essa DLL no tópico 3.2.2 foi usada pelo UDCrun, que é um protótipo de simulador bastante simples para executar apenas a solução dos CDUs isoladamente. Entretanto, quando a solução do CDU precisa ser utilizada por programas de simulação mais completos, é necessário um trabalho de integração. Dessa forma, é tratada neste tópico a integração do UDCC ao AnaHVDC.

No UDCrun há dois níveis de loop na solução, conforme a Figura 65.

```
1 // Loop do Tempo
2 for (int current = 0; i <= total; ++current){
3   // Loop da Convergência
4   while (!converged){
5     converged = solCdu(...);
6   }
7 }
```

Figura 65 – Loop de Solução do UDCrun

No loop do tempo é mantido o controle do ponto corrente da simulação até que se alcance o valor máximo, dado pela variável *total*, e a solução termine. A variável *total* é igual ao tempo de simulação dividido pelo passo de integração. Desse modo, se o usuário configura um tempo de simulação de 10 s e um passo de integração de 0,001, isso significa

[32] Apenas os blocos dinâmicos apresentam termo histórico.

que serão plotados 10000 pontos considerando-se também uma frequência de plotagem igual a 1, conforme visto no tópico 3.2.1.

O segundo nível é o loop da convergência. Nesse caso, é chamada a função *solcdu*, que percorre os diversos blocos do CDU (conforme ilustrado na Figura 62). A *solcdu* é chamada o número de vezes necessário para que aconteça a convergência. Caso o sistema não possua retroalimentação, é realizada apenas uma iteração. Nesse contexto, a convergência deve ser entendida como o valor da saída de cada bloco das malhas retroalimentadas subtraído do valor de saída do respectivo bloco na iteração anterior ser inferior a um valor de tolerância.

No caso do AnaHVDC, a solução se encontra esquematizada na Figura 66.

```
1  // Loop do Tempo
2  for (int current = 0; current <= total; ++current){
3
4    // Loop Modelo-Rede
5    while (!converged){
6
7      // Loop dos Equipamentos
8      while (!convergedCdu){
9        convergedCdu = solCdu(...)
10     }
11     while (!convergedGenerator){
12       convergedGenerator = solGenerators(...)
13     }
14     while (!converged SVC){
15       convergedSVC = solSvc(...)
16     }
17     // loops para outros equipamentos
18   }
19 }
```

Figura 66 – Loop de Solução AnaHVDC

Primeiramente, é importante notar que o loop dos equipamentos corresponde ao loop da convergência do UDCrun. Porém, abarcados neste loop estão todos os equipamentos, ao invés de apenas o CDU. Portanto, depois de todos os equipamentos convergirem é necessário verificar se a rede convergiu. Nesse caso, a convergência deve ser entendida como o valor da tensão de cada barramento subtraído do valor de tensão do respectivo barramento na iteração anterior ser inferior a um valor de tolerância. Após, se obter a convergência do Modelo-Rede avança-se para o próximo instante de tempo.

Percebe-se que de fato quem gerencia a chamada da solCdu é o próprio AnaHVDC e não o UDCC. Dessa forma, o gerenciamento de chamadas para cada função da DLL[33] é responsabilidade do AnaHVDC bem como o armazenamento dos resultados retornados por cada função.

Para comparativo com o Anatem, como descrito, os blocos IMPORT e EXPORT foram removidos dos CDUs do SIN e convertidos, respectivamente, para blocos ENTRAD e SAIDA, nos quais foram aplicados valores de inicialização. A partir deste ponto esses blocos possuem os valores fixos de inicialização, exceto se for aplicado um evento neles. No AnaHVDC os CDUs estão interligados à rede e os blocos IMPORT e EXPORT não podem ser tratados como blocos de valor constante. Durante a solução, esses blocos interagem com a rede, de modo que suas saídas podem vir a se alterar a cada nova iteração.

Para analisar essa questão considere-se como exemplo o recorte de CDU da Figura 67.

```
01 (nb) (tipo) (stip)s(vent) (vsai) ( p1 )( p2 )( p3 )( p4 )
02 (-------------------------------------------------------
03 0001 IMPORT CCNV          IDC
04 0252 LEDLAG         IDC   IDCM      1.0          1.0#Tmc
05 0252 GANHO          IDC   IDCM      1.0
06 0253 EXPORT CDU     IDCM
07 0002 IMPORT CDU           I0*       1002
08 0004 SOMA           IDCM  ERRC
09                    -I0*   ERRC
10 0005 GANHO          ERRC  UCP     #KPCCA
```

Figura 67 – Sinais EXPORT e IMPORT

Pode-se notar é necessário nos blocos IMPORT e EXPORT que seja preenchido o subtipo, que se refere à espécie do sinal importado ou exportado. No CDU da Figura 67 é importado pelo bloco #0001 a corrente de um conversor, o que é indicado pelo subtipo CCNV. Por outro lado, o subtipo do bloco #0253 indica que a sua variável pode ser exportada para um outro CDU. Uma relação com todos os subtipos existentes pode ser encontrada em [19].

No caso da Figura 67, além do subtipo, o segundo bloco IMPORT também possui o parâmetro P1 preenchido. Esse valor, no caso particular dos blocos IMPORT, é conhecido por DLOC e definido como “um número de identificação de uma localização remota de sinal” [19]. A Figura 68 mostra um outro recorte do mesmo CDU.

[33] Reforça-se que não apenas a chamada da função referente à solução e sim a chamada de todas.

```
DLOC
(Lc)    (Tipo)( El )( Pa)Nc( Ex)Gr(Bl)
1002    CDU     9001                 50
1011    CDU     9001                101
1043    CDU     9001                250
```

Figura 68 – DLOC do tipo CDU

Dessa forma, o bloco IMPORT #0002 da Figura 67 possui como DLOC o valor 1002. Consultando-se o campo de DLOCs do CDU, conforme a Figura 68, nota-se que o sinal é importado do bloco de número 50 do elemento 9001, que é do tipo CDU. Caso o DLOC não esteja preenchido, subentende-se que o sinal é importado do equipamento associado ao controlador. Esse é o caso do bloco IMPORT #0001 em que a variável é importada do conversor associado ao CDU.

Considere-se como outro exemplo o recorte de CDU da Figura 69.

```
DEFPAR #LocM                193

(-------------------------------------------------------
(nb) (tipo) (stip)s(vent) (vsai) ( p1 )( p2 )( p3 )( p4 )
(-------------------------------------------------------
0001 IMPORT DWMAQ          Df      #LocM
0002 GANHO          Df     DfHZ      50.0


(Lc)    (Tipo)( El )( Pa)Nc( Ex)Gr(Bl)
 193    MAQ     1100             10
```

Figura 69 – DLOC a partir de DEFPAR

Neste caso, o subtipo do IMPORT é DWMAQ, o que significa que ele é o desvio de velocidade angular da máquina em relação à velocidade síncrona em pu [19]. Entretanto, diferente do exemplo anterior, na Figura 69 o DLOC está associado a um DEFPAR, que por sua vez corresponde ao valor numérico 193. Consultando-se a régua dos DLOCs, conclui-se que o DLOC 193 é do tipo gerador, que está conectado na barra CA de número 1100 e pertence ao grupo de equipamentos identificado por 10. Nota-se que enquanto no exemplo da Figura 68 o elemento do DLOC era uma identificação do próprio equipamento, no caso da Figura 69 o elemento do DLOC se refere ao gerador da barra e do grupo indicados.

Todas essas características descritas foram consideradas na integração do UDCC ao AnaHVDC. É intuitivo que a DLL deve conter os números dos DLOCs do CDU para que os

blocos IMPORT e EXPORT funcionem. Portanto, além das funções explicadas ao longo desta dissertação[34], a DLL também possui uma função específica chamada *inivar* para tratamento desses blocos. Nessa função, há a lista de todos os blocos IMPORT, EXPORT e DEFVAL do CDU bem como as informações associadas a cada um deles, as quais são mostradas na Figura 68. Por sua vez, cabe ao AnaHVDC processar essas informações. Por exemplo, a DLL gerada atribui o valor 0 para o DLOC caso o P1 do bloco esteja em branco. O AnaHVDC a partir dessa leitura é programado para buscar o valor do bloco a partir do equipamento a ele associado. Essa questão obrigatoriamente deve ser responsabilidade do AnaHVDC uma vez que é ele que possui conhecimento de toda a rede, incluindo seus equipamentos.

3.7 Processamento Paralelo

Na solução isolada dos CDUs, utilizando o programa UDCrun com o UDCC, foi desenvolvida a solução por processamento paralelo utilizando *threads* criadas na programação. Nesta solução adotada, o sistema operacional gerencia a distribuição da execução das *threads* criadas de forma automática e com variação da frequência dos processadores otimizadas dinamicamente em função do monitoramento pela CPU do estado de seus processadores. No Capítulo 4 são apresentados resultados deste processamento paralelo por meio do qual se obteve ganhos de desempenho computacional significativos.

Na solução do UDCC integrado ao AnaHVDC, o processamento paralelo não se mostrou tão eficiente, embora ainda assim tenha proporcionado ganhos. Testaram-se três possibilidades. A primeira, de mais simples programação, foi a de criação das threads com a paralelização das chamada da solução dos CDUs dentro do laço iterativo de solução, em que a demora de criação e destruição das *threads* múltiplas vezes deteriorava muito o desempenho a ponto de ficar muito pior do que a simulação original sem processamento paralelo. A segunda tentativa foi a de criação das *threads* de solução dos CDUs fora do laço iterativo de solução, que ficavam armazenadas de forma estática em memória, fazendo-se as suas múltiplas chamadas sem destruí-las. O desempenho melhorou, mas continuou sendo desvantajoso. A atual solução consiste em uma adaptação da segunda tentativa, utilizando-se da biblioteca OpenMP, adotando-se o recurso de hibernação das *threads* enquanto o

[34] As outras funções da DLL são as de inicialização, cálculo de constantes (Figura 34), cálculo do termo histórico (Figura 35) e cálculo da solução (Figura 36).

programa não estiver solucionando os CDUs. Com essa solução, já é possível obter ganhos a partir do uso de processamento paralelo. Acredita-se que uma estratégia de criação de processos com trocas de dados e controle de acessos concorrentes de leitura e escrita podem melhorar a eficiência do processamento paralelo. No entanto, por limitação de tempo de dissertação, esta pesquisa foi deixada como um trabalho futuro.

Capítulo 4 - Resultados

Os resultados se concentraram em comprovar a eficiência computacional do UDCC e o sucesso de sua integração ao AnaHVDC. Dessa forma, os resultados são baseados em três casos descritos nos tópicos a seguir. As simulações foram realizadas em quatro computadores de diferentes desempenhos:

1) Intel Core i5-3230 (3ª Geração, 4 núcleos lógicos e CPU@2,60GHz/3,20GHz). 8 GB de memória RAM. Cache: L1=128kB, L2=512kB e L3=3MB.
2) Intel Core i7-6700 (6ª Geração, 8 núcleos lógicos, e Cpu@3,40GHz/4,00GHz). 16 GB de memória RAM. Cache: L1=256kB, L2=1MB e L3=6MB.
3) Intel Core i7-9750H (9ª geração, 12 núcleos lógicos e CPU@2.6GHz/4.50GHz). 16 GB de memória RAM. Cache: L1=384kB, L2=1,5MB e L3=12MB.
4) Intel Core i9-9900K (9ª Geração, 16 núcleos lógicos e CPU@3,60GHz / 5.00GHz). 64 GB de memória RAM. Cache: L1=512kB, L2=2MB e L3=16MB.

Para facilidade de referência os computadores foram nomeados pelo tipo da CPU em maiúsculas, seguido da quantidade de núcleos lógicos em algarismo romanos, ou seja, I5IV, I7VIII, I7XII e I9XVI.

4.1 Caso 1: Simulação do SIN no Anatem e no UDCC

Neste caso, o objetivo foi realizar o comparativo do tempo de solução dos CDUs do SIN de forma isolada no UDCC (utilizando o programa UDCrun) e no Anatem. Para isso, o arquivo com os CDUs do SIN modificados para validação, que foi descrita no tópico 3.2.2, foi usado também para realizar o comparativo de tempo.

Entretanto, algumas diferenças merecem destaque. No caso da marcação de tempo foram usadas como opções do UDCrun -a-0.1 -n -cX AnaHVDC[35]. Portanto, nesse caso, o degrau também é ajustado para uma amplitude negativa de 0,1 pu (-a-0.1). Entretanto, conforme indicado pela segunda instrução (-n), não é plotada nenhuma variável. Na sequência, -cX se refere a quantas threads adicionais devem ser usadas na solução dos CDUs. X é um valor numérico que foi variado de modo a se medir o tempo considerando diferentes números de *threads*. As outras instruções que se encontram implícitas foram descritas no

[35] No caso da validação as opções foram: -a-0.1 AnaHVDC.

tópico 3.2.2. Além disso, foi estabelecido um tempo de espera de 5 s entre uma simulação e outra, de modo a evitar que o aquecimento da CPU prejudicasse a aferição de tempo. Convém ressaltar que se verificou que os 5 s da espera eram suficientes para volta da temperatura da CPU para o estado de repouso. Dessa forma, aproxima-se do caso real em que não se executa em sequência dezenas de simulações.

A seguir, são apresentados os desempenhos computacionais considerando uma versão inicial não otimizada do UDCC (item 4.1.1) e, posteriormente, uma versão modificada se valendo de um melhor uso da memória cache dos computadores (item 4.1.2). Ao todo foram simulados e validados 414 CDUs de reguladores de tensão e de velocidade do SIN.

4.1.1 Sem Otimizações de Uso de Cache

Nesse primeiro caso, o comparativo foi realizado entre:

- Anatem;
- UDCC com compilação da DLL via TCC;
- UDCC com compilação da DLL via Clang.

A seguir são apresentados os resultados para cada computador considerando os tempos médios das simulações. A lista completa dos tempos consta no Apêndice C e uma breve revisão das ferramentas estatísticas utilizadas na apresentação de dados deste apêndice pode ser encontrada no Apêndice B.

A partir desse ponto são expostos os resultados obtidos. Para o computador I5IV os resultados estão apresentados na Figura 71. Primeiramente, o eixo x se refere ao número de threads explicitamente criadas por ordem do código do desenvolvedor. Em qualquer caso o programa possui ao menos uma thread, que é criada automaticamente no início da execução do programa. Portanto, 0 *thread* no gráfico se refere ao caso em que nenhuma thread foi criada explicitamente. Nesse mesmo sentido, 1 *thread* indica que, além da *thread* lançada automaticamente, foi criada posteriormente mais uma, por ordem do código do desenvolvedor. A lógica é a mesma para os outros casos. O Anatem, por não haver processamento paralelo, foi enquadrado no caso de 0 *thread*.

De acordo com o gráfico percebe-se que, em relação ao Clang, o caso ótimo acontece para 24 threads, enquanto para o TCC se dá para 64 *threads*. Esse resultado é inusitado pelo menos por dois motivos. Conforme descrito, o computador I5IV possui 4 núcleos lógicos,

de modo que o caso ótimo, em condições teóricas ideais, aconteceria para 4 threads. Na verdade, para 4 *threads*, o resultado é significativamente pior. O segundo motivo é o fato do caso ótimo se dar para um número diferentes de threads dependendo do compilador utilizado. Isto demonstra a importância da aferição de tempo para cada caso específico, pois o aproveitamento do processamento do computador é bem mais complexo, dependendo de diversos outros fatores como a curva de aquecimento da CPU durante uma simulação que depende não só do código de máquina executado, como também das condições de umidade e temperatura ambiente. Como na tecnologia da Intel as CPUs se monitoram e controlam dinamicamente a sua frequência de operação e o compartilhamento de tarefas entre núcleos lógicos, o desempenho de diferentes programas com diferentes threads passa a variar de forma pseudoaleatória.

Entretanto, a tendência geral é clara: o aumento no número de *threads* tende a reduzir o tempo de solução até se atingir o caso ótimo. A partir desse ponto, o tempo volta a crescer. Isso acontece porque o custo computacional para criação da *thread* não compensa o ganho de desempenho. Na verdade, dependendo do caso, nem sequer ganho computacional existe. Destaca-se também a superioridade do Clang em relação ao TCC, o que já era de se esperar, uma vez que o código de máquina gerado pelo Clang é altamente otimizado, ao contrário daquele gerado pelo TCC. Uma exposição qualitativa sobre isto foi realizada no tópico 3.3.

Outro fato importante a ser notado é que, sem processamento paralelo, o Clang é mais lento do que o Anatem neste caso. A princípio isto não faria sentido, uma vez que o Anatem utiliza uma metodologia interpretada, enquanto que o UDCC utilizaria a metodologia compilada. A explicação então encontrada para esta inconsistência foi a grande diferença de utilização das memórias caches. Como o Anatem utiliza vetores de dados para os diversos blocos e os COMMON do Fortran concentram os vetores mais utilizados na solução em áreas de memórias mais concentradas, há um grande ganho de desempenho do Anatem, a ponto de sua metodologia interpretada superar a metodologia compilada do UDCC. No próximo item serão apresentados resultados de uma nova versão do UDCC em que o código C gerado faz um melhor aproveitamento da memória cache. Com isso, a metodologia compilada se revela, de fato, superior à interpretada, mesmo sem utilizar threads explícitas para o processamento paralelo.

Outro fator que prejudicou o UDCC é que os resultados de tempos foram aferidos quando ainda havia uma subotimização quanto à verificação de convergência. Todos os blocos de um CDU estavam sendo verificados nesse processo, enquanto isso precisa ser

realizado apenas para os blocos em malhas de realimentação. Entretanto, ainda não foram levantados os novos tempos considerando essa nova modificação.

A Tabela 2 apresenta em termos relativos os tempos apresentados na Figura 71.

Tabela 2 – Tempos Comparativos (Computador i5IV)

	Anatem	TCC	Clang	Speedup TCC	Speedup Clang
0 thread	17,85 s	36,38 s	19,52 s	0,49	0,91
Caso ótimo	17,85 s	16,17 s	7,26 s	1,10	2,46

O TCC, que realiza apenas otimizações rudimentares no código, realizou a solução dos CDUs aproximadamente no dobro de tempo do Anatem para o caso sem processamento paralelo. Por sua vez, o tempo do Clang, também sem processamento paralelo, foi apenas 9% superior ao do Anatem. Para o caso ótimo tanto o TCC como o Clang são solucionaram o caso mais rápido do que o Anatem.

Na Figura 70 é mostrado um caso exemplo de solução do SIN para o TCC com 17 threads totais, ou seja, uma thread lançada automaticamente no início da execução do programa e 16 threads lançadas por instrução explícita do desenvolvedor. A partir do gerenciador de tarefas atesta-se a afirmativa de que o processamento paralelo do UDCC funciona conforme esperado.

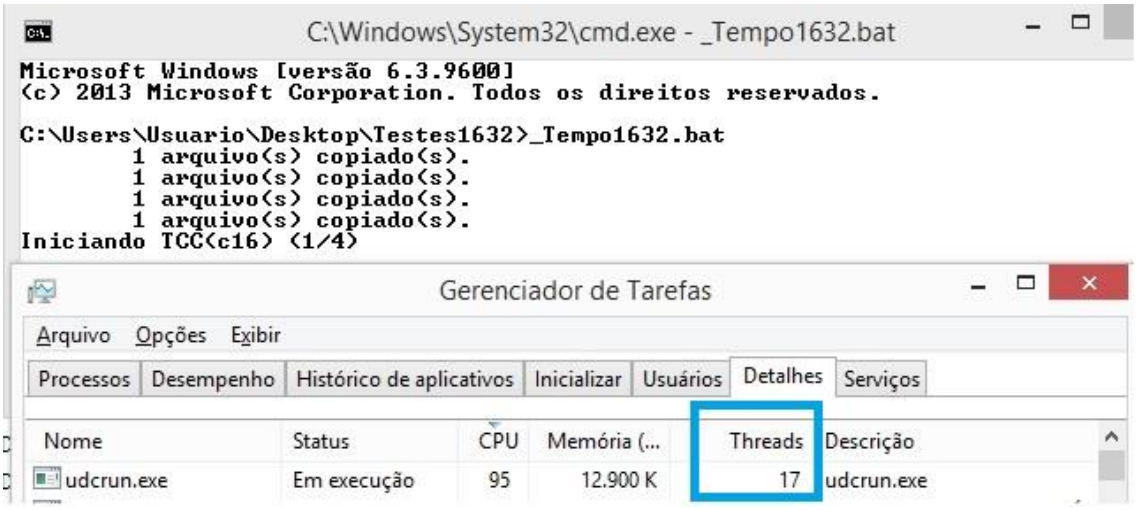


Figura 70 – Número de Threads em Execução

Na sequência, são analisados os resultados obtidos a partir do computador I7VIII. Os tempos obtidos são apresentados na Figura 73. A diferença para o computador I5IV é sensível. Já com 0 *thread* o Clang é capaz de ser significativamente mais rápido do que o Anatem. Como nesse caso não há processamento paralelo, demonstra-se a eficiência da abordagem compilada. Destaca-se que, para o computador I5IV, o Anatem foi mais rápido, de forma que, de algum modo, essa eficiência está atrelada à CPU e à memória do computador e, conforme citado, depende do aproveitamento da memória cache, que no I5IV é mais reduzida. Esse é um ótimo fato: com o rápido desenvolvimento tecnológico na área de computação a abordagem compilada do UDCC tende gradativamente a alargar sua vantagem em relação à abordagem interpretada do Anatem, principalmente em sua nova versão com melhor aproveitamento da memória cache.

Percebe-se que o tempo ótimo no caso do TCC ocorre com 96 *threads*, enquanto para o Clang acontece com 4 *threads*. Essa disparidade é um fato a ser investigado em trabalhos futuros. Além disso, também é interessante o fato de um computador com 8 núcleos lógicos atingir seu caso ótimo para apenas 4 *threads*.

Os tempos comparativos são apresentados na Tabela 3.

Tabela 3 – Tempos Comparativos (Computador I7VIII)

	Anatem	TCC	Clang	Speedup TCC	Speedup Clang
0 thread	13,78 s	29,13 s	5,97 s	0,47	2,30
Caso ótimo	13,78 s	7,18 s	3,21 s	1,92	4,29

Sem processamento paralelo, o TCC é mais lento do que o Anatem, porém o Clang já é aproximadamente 2,3 vezes mais rápido. Para o caso ótimo, tanto o TCC como o Clang solucionaram o caso mais rápido do que o Anatem. Destaca-se que o *speedup* do Clang foi de 4,29, o que indica que a vantagem do UDCC em relação ao Anatem se alarga em relação ao computador I5IV.

Dessa forma, passa-se a análise do terceiro computador de testes, o i7XII, cujos tempos de simulação estão apresentados na Figura 73. No computador i7XII usado para os testes a superioridade do UDCC é ainda maior, o que reforça a afirmação anterior de a eficiência da abordagem compilada estar atrelada à sofisticação do computador. Mesmo no caso com 0 *thread* o tempo de solução do Clang é consideravelmente menor do que o do Anatem. Observa-se também que o caso ótimo acontece para o Clang com 10 *threads*. Para

o TCC o caso ótimo aconteceu com 64 *threads*, quatro vezes o número de núcleos lógicos, o que também constitui um resultado inesperado. Nesse ponto uma tendência também se evidencia: os casos ótimos do TCC acontecem para um número consideravelmente maior de *threads*.

A Tabela 4 apresenta em termos relativos os tempos apresentados na Figura 73.

Tabela 4 – Tempos Comparativos (Computador I7XII)

	Anatem	TCC	Clang	Speedup TCC	Speedup Clang
0 thread	13,23 s	24,30 s	4,83 s	0,54	2,74
Caso ótimo	13,23 s	2,71 s	1,11 s	4,88	11,92

Sem processamento paralelo, o TCC é mais lento do que o Anatem, porém o Clang é aproximadamente 2,7 vezes mais rápido. Para o caso ótimo tanto o TCC como o Clang solucionaram o caso mais rápido do que o Anatem. Destaca-se que o *speedup* do Clang foi de 11,92, o que indica que a vantagem do UDCC em relação ao Anatem se alargou ainda mais.

O Clang com 0 thread é 2,7 vezes mais rápido do que o Anatem e para o caso ótimo praticamente 12 vezes mais rápido. Portanto, a vantagem do UDCC em relação ao Anatem permanece se alargando proporcionalmente à CPU/Memória, porém em menor escala.

Por fim, a Figura 74 apresenta o tempo se solução para o caso do computador I9XVI. Como esperado, o Clang mesmo com 0 thread obtém um tempo melhor do que o do Anatem. O caso ótimo para o TCC foi obtido com 64 threads, enquanto para o Clang foi com 24 threads.

Os tempos comparativos são apresentados na Tabela 5.

Tabela 5 - Tempos Comparativos (Computador I9XVI)

	Anatem	TCC	Clang	Speedup TCC	Speedup Clang
0 thread	10,29 s	17,94 s	4,12 s	0,57	2,50
Caso ótimo	10,29 s	1,82 s	0,82 s	5,44	12,55

Sem processamento paralelo, o TCC demonstra-se ser mais lento do que o Anatem para todos os computadores testados. O Clang, porém, é aproximadamente 2,5 vezes mais

rápido no computador I9XVI. Destaca-se que o *speedup* do Clang, para o caso ótimo com processamento paralelo, foi de 12,55.

Na Tabela 2, Tabela 3, Tabela 4 e na Tabela 5 descritas anteriormente foram apresentados os desempenhos de cada computador em relação ao Anatem, ao TCC e ao Clang. Na sequência, apresenta-se o comparativo entre os computadores, em relação ao TCC (Figura 75) e em relação ao Clang (Figura 76).

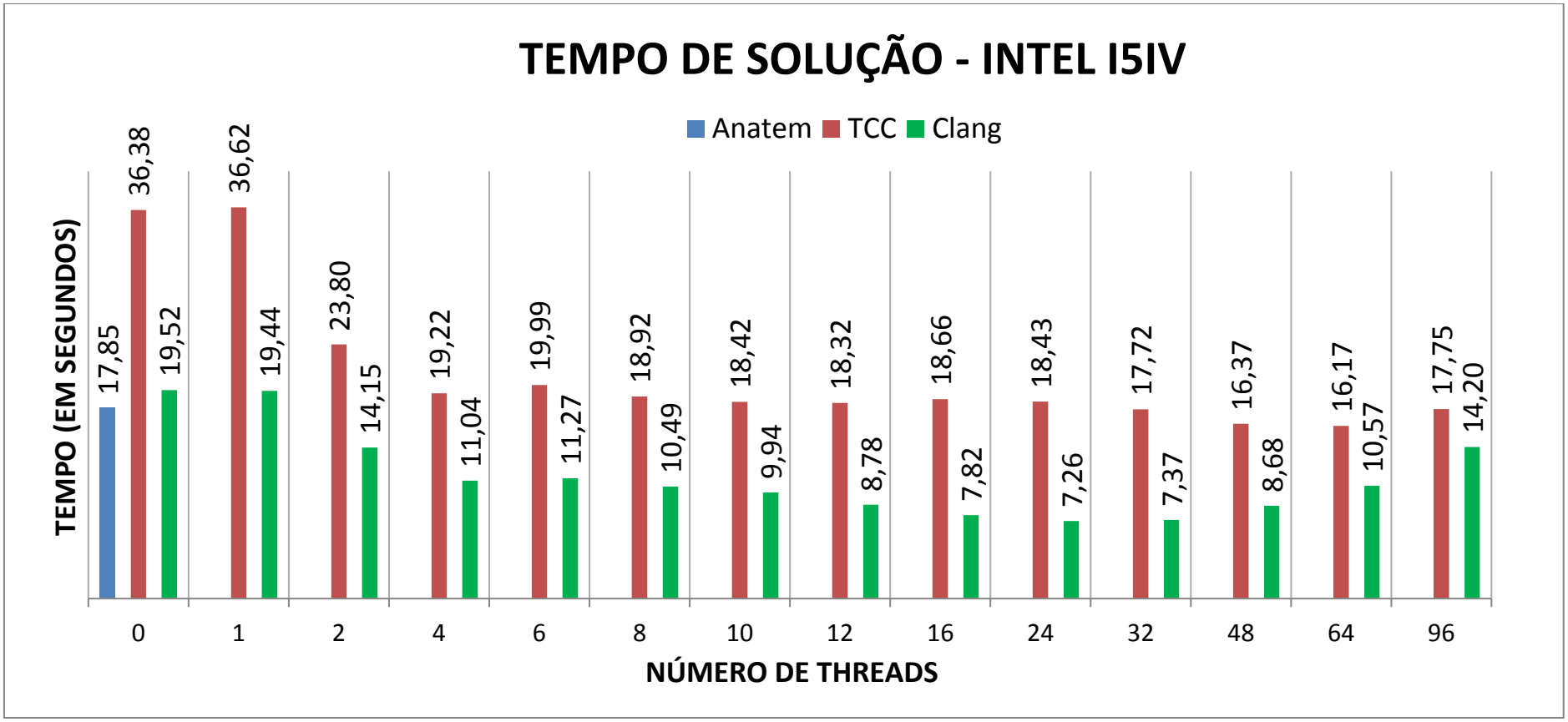


Figura 71 – Tempo de Solução (Computador I5IV)

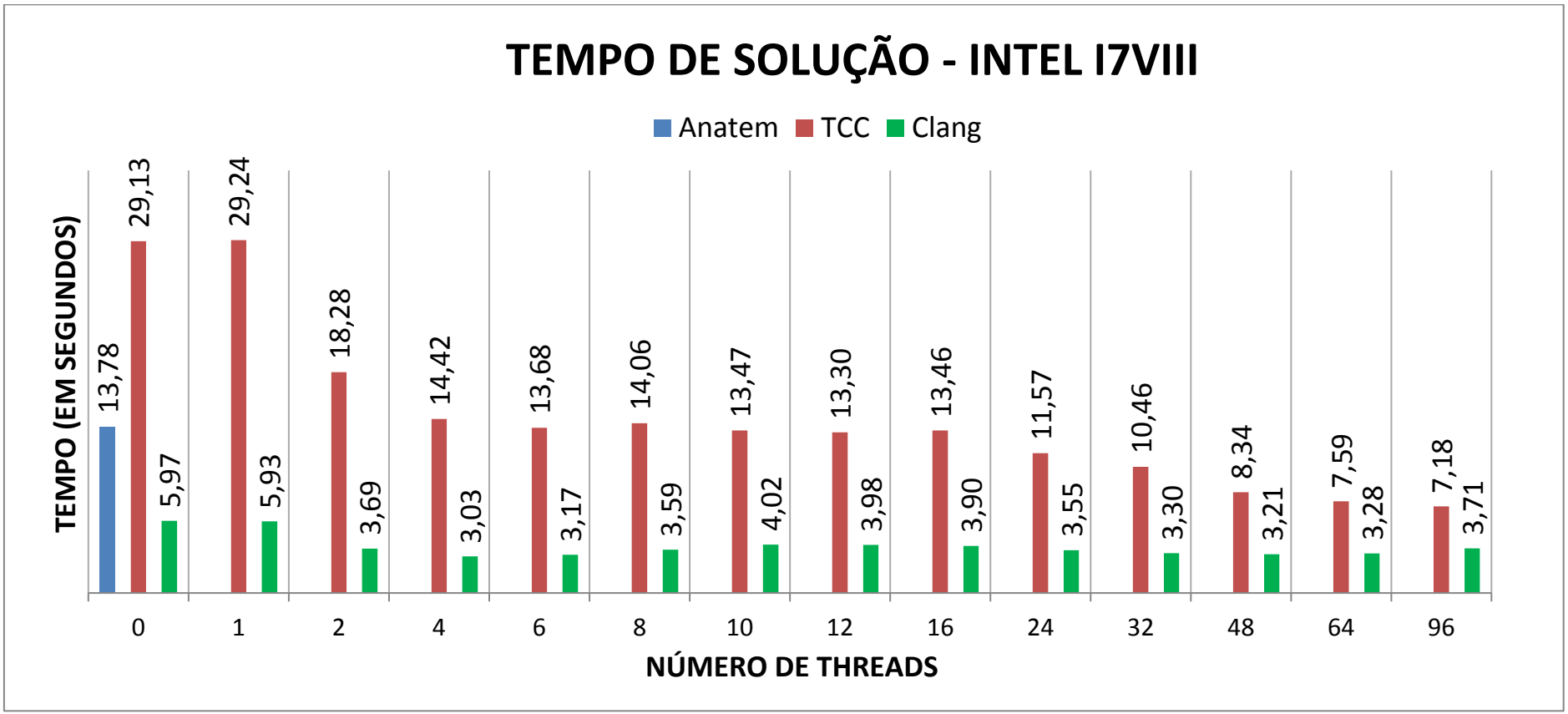


Figura 72 – Tempo de Solução (Computador I7VIII)

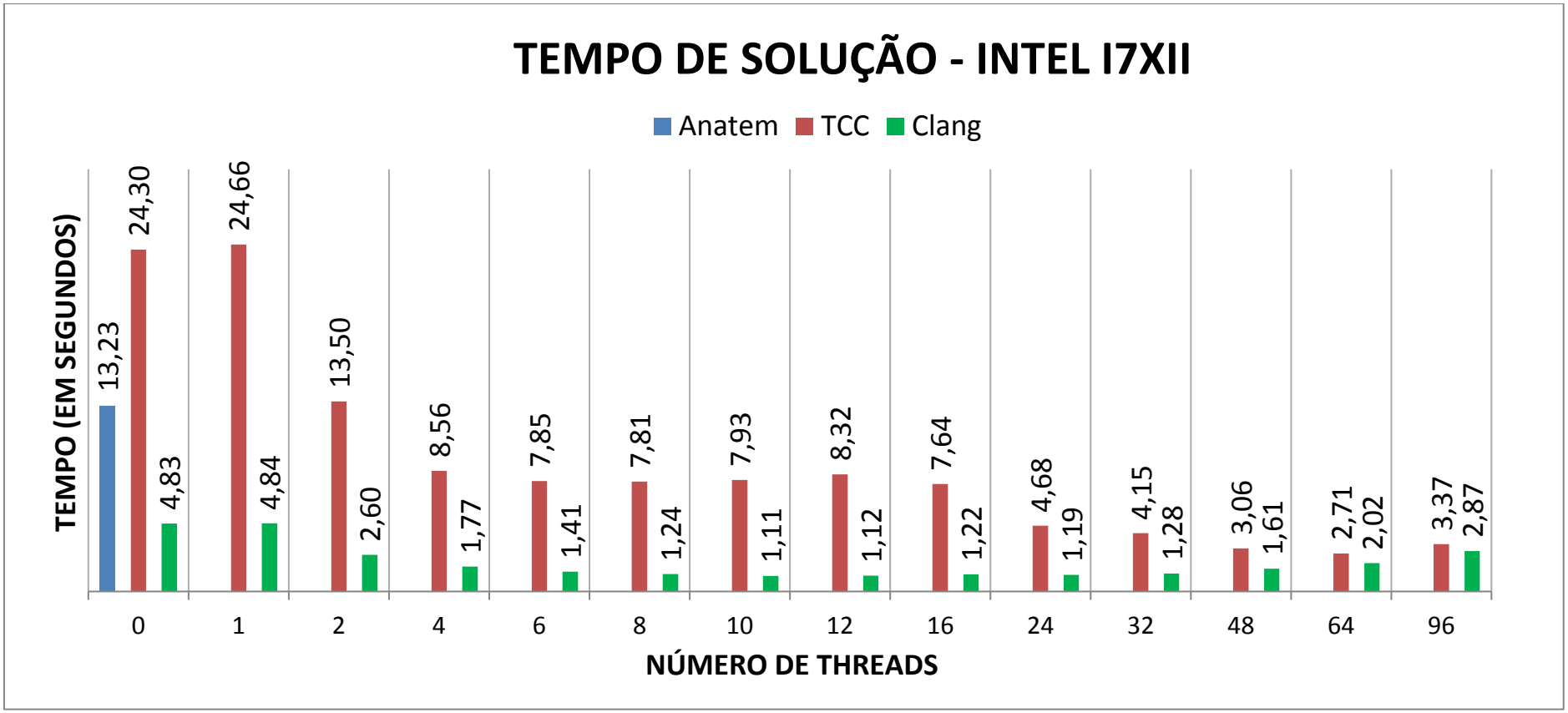


Figura 73 – Tempo de Solução (Intel I7XII)

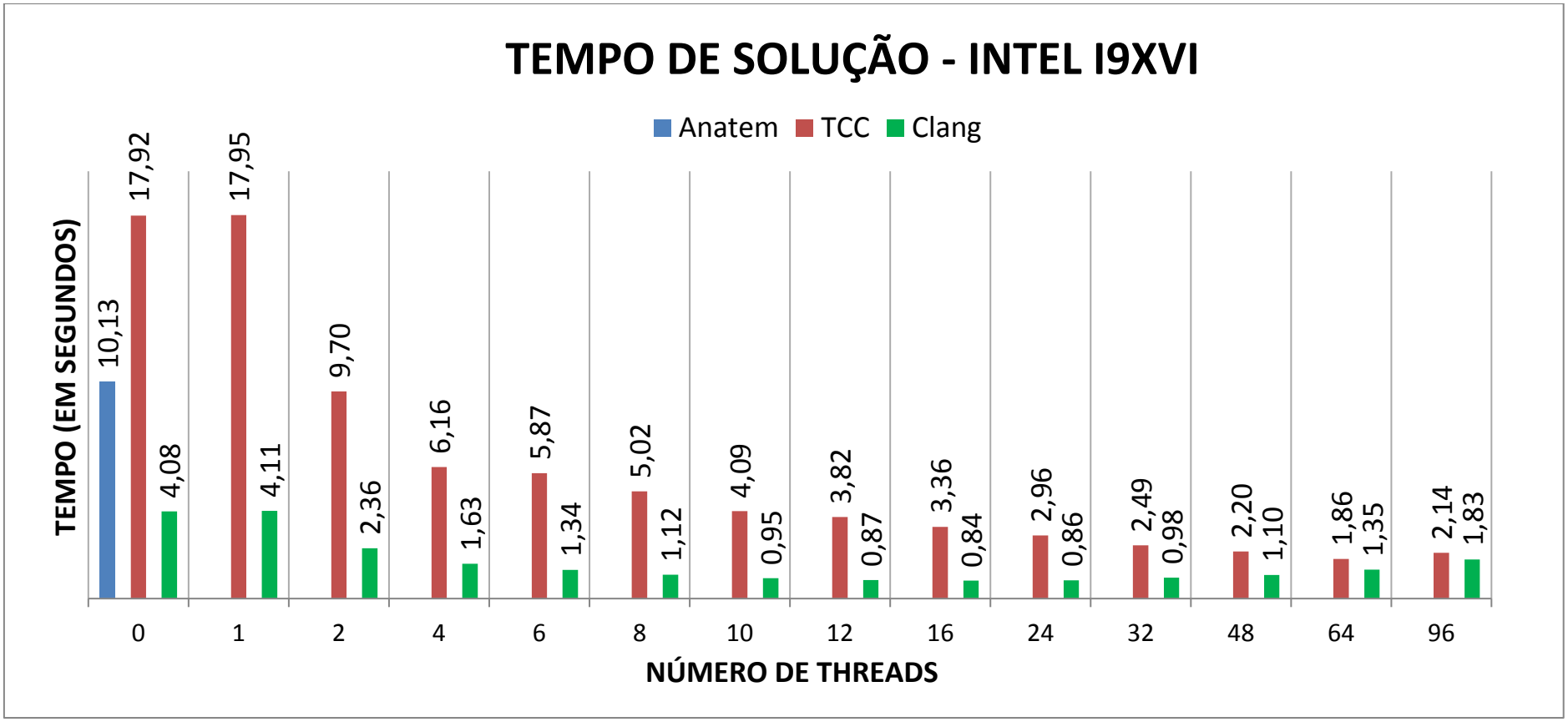


Figura 74 – Tempo de Solução (Computador I9XVI)

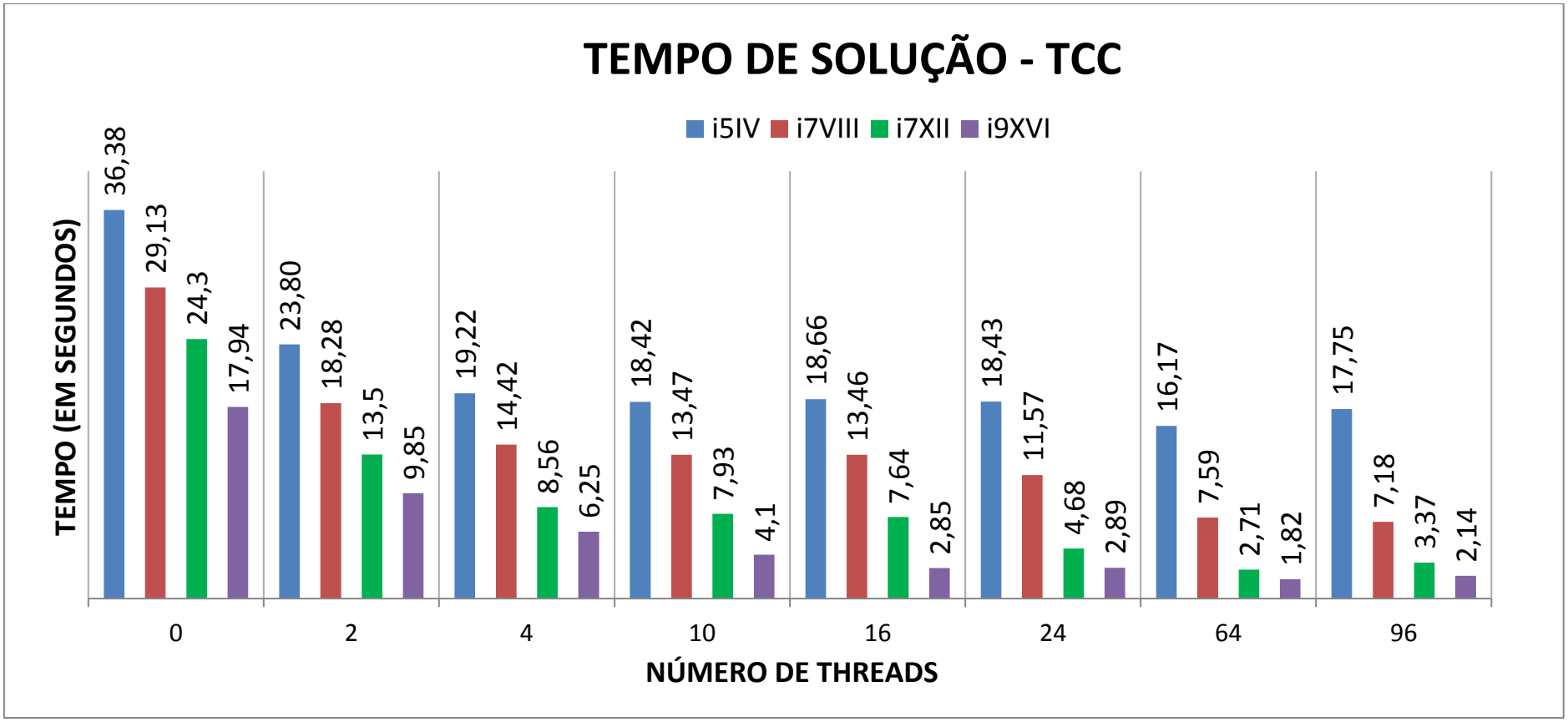


Figura 75 – Tempo de Solução (TCC)

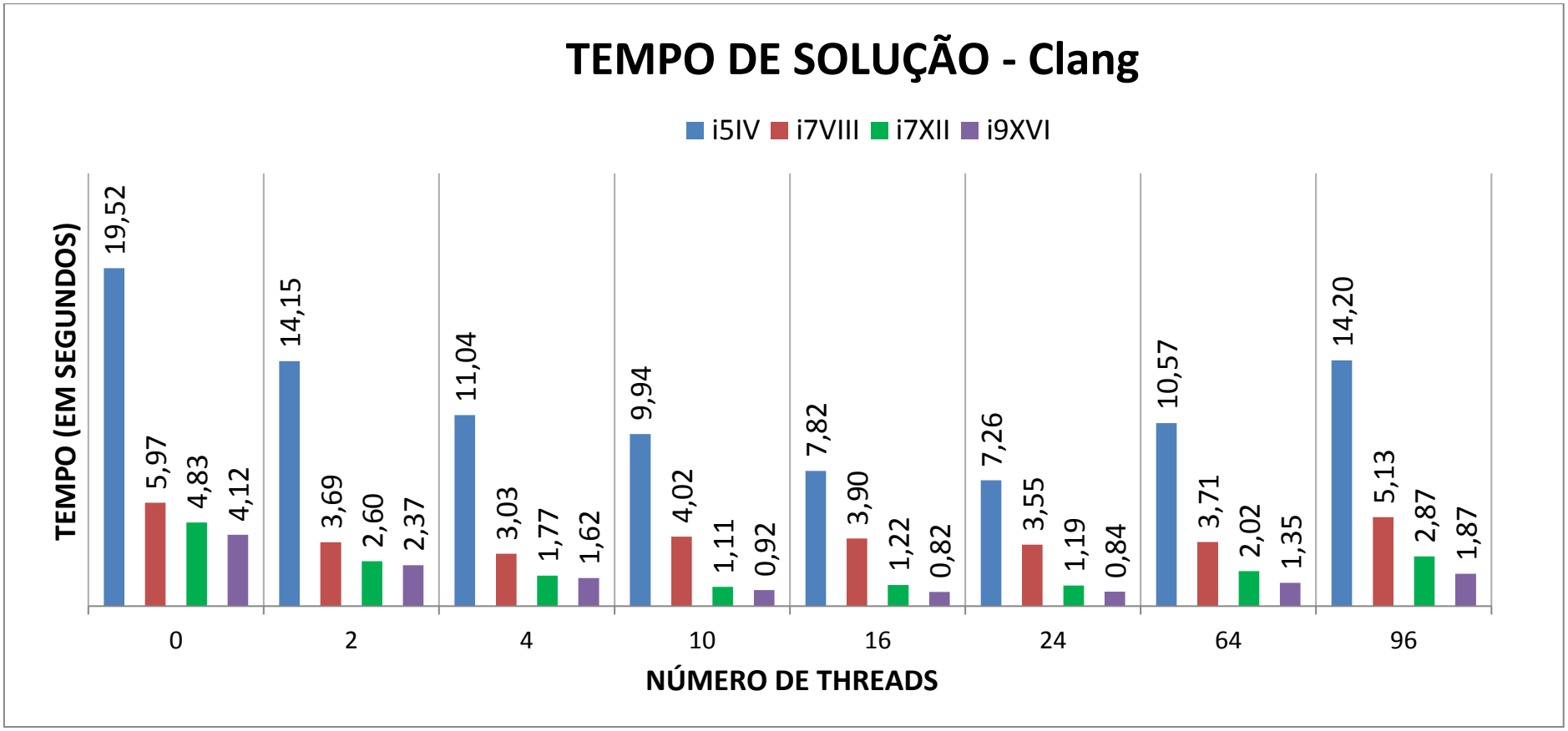


Figura 76 – Tempo de Solução (Clang)

4.1.2 Com Otimizações de Uso de Cache

O código do UDCC foi alterado visando a implementar otimizações de uso de cache seguindo as ideias gerais expostas no tópico 2.7. Dessa forma, foi possível obter um segundo degrau de redução de tempo. O caso é exatamente igual ao simulado no item anterior (4.1.1) e, por isso, os resultados serão expostos de modo mais sucinto. Os resultados com os tempos detalhadas encontram-se no Apêndice D. Além disso, como o Anatem não sofreu alteração, os tempos deste programa foram repetidos do tópico anterior.

Entre o espaço de tempo da simulação sem otimização de cache (item 4.1.1) e aquela com otimização, houve problemas de travamento do computador I7XII, que para ser resolvido, o Turbo Boost, que é o recurso de aumento individualizado de frequência de processadores ao máximo quando há condições adequadas de temperatura, precisou ser desligado. A consequência é que o desempenho do I7XII se revelou pior do que o I7VIII nas simulações com otimização de cache e por ser um problema atípico de computadores deste tipo, optou-se por eliminar o computador I7XII deste comparativo.

A Tabela 6 apresenta em termos relativos os tempos apresentados na Figura 77.

Tabela 6 - Tempos Comparativos (Computador I5IV) (OC)[36]

	Anatem	TCC	Clang	Speedup TCC	Speedup Clang
0 thread	17,85 s	19,38 s	6,21 s	0,92	2,87
Caso ótimo	17,85 s	8,34 s	3,32 s	2,14	8,34

O mau desempenho do TCC com 0 threads em relação à metodologia interpretada do Anatem se deve, em parte, à otimização muito rudimentar do código de máquina gerado pelo TCC, conforme já mencionado anteriormente. Entretanto, a verificação ineficiente de convegência explicada no item 4.1.1 também contribui significativamente.

Por sua vez, a Tabela 7 apresenta em termos relativos os tempos apresentados na Figura 78.

[36] Indica que para o caso foram consideradas as Otimizações de Cache (OC).

Tabela 7 – Tempos Comparativos (Computador I7VIII) (OC)

	Anatem	TCC	Clang	Speedup TCC	Speedup Clang
0 thread	13,78 s	14,07 s	4,37 s	0,98	3,15
Caso ótimo	13,78 s	3,62 s	1,27	3,81	10,85

Percebe-se para este caso que a vantagem do UDCC em relação ao Anatem se alarga em relação ao computador I5IV. Isso foi exatamente o que aconteceu com a simulação sem as otimizações de cache.

Dessa forma, passa-se a análise do quarto computador de testes, o I9XVI, cujos tempos de simulação estão apresentados na Figura 79 (lembrando novamente que o terceiro computador, I7VIII não foi considerado) e condensados comparativamente na Tabela 8.

Tabela 8 - Tempos Comparativos (Computador I9XVI) (OC)

	Anatem	TCC	Clang	Speedup TCC	Speedup Clang
0 thread	10,13 s	8,46 s	3,24 s	1,20	3,12
Caso ótimo	10,13 s	1,06 s	0,53 s	9,55	19,11

No computador I9XVI usado para os testes a superioridade do UDCC é ainda maior, o que reforça a afirmação anterior de a eficiência da abordagem compilada estar atrelada à sofisticação do computador. Mesmo no caso com 0 *thread* o tempo de solução do TCC é menor do que o do Anatem. O Clang com 0 *thread* é 2,5 vezes mais rápido do que o Anatem e para o caso ótimo 19 vezes mais rápido.

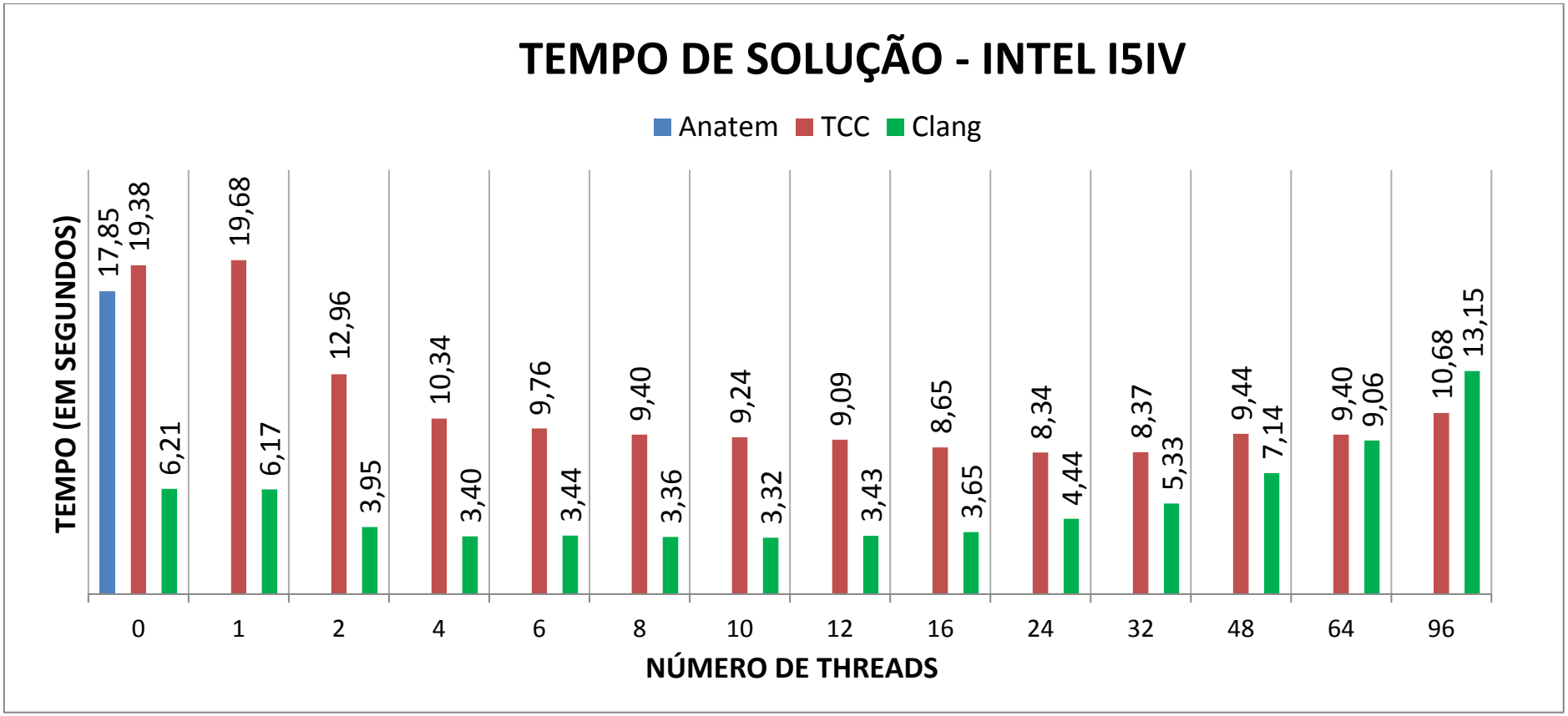


Figura 77 – Tempo de Solução (Computador i5IV) (OC)

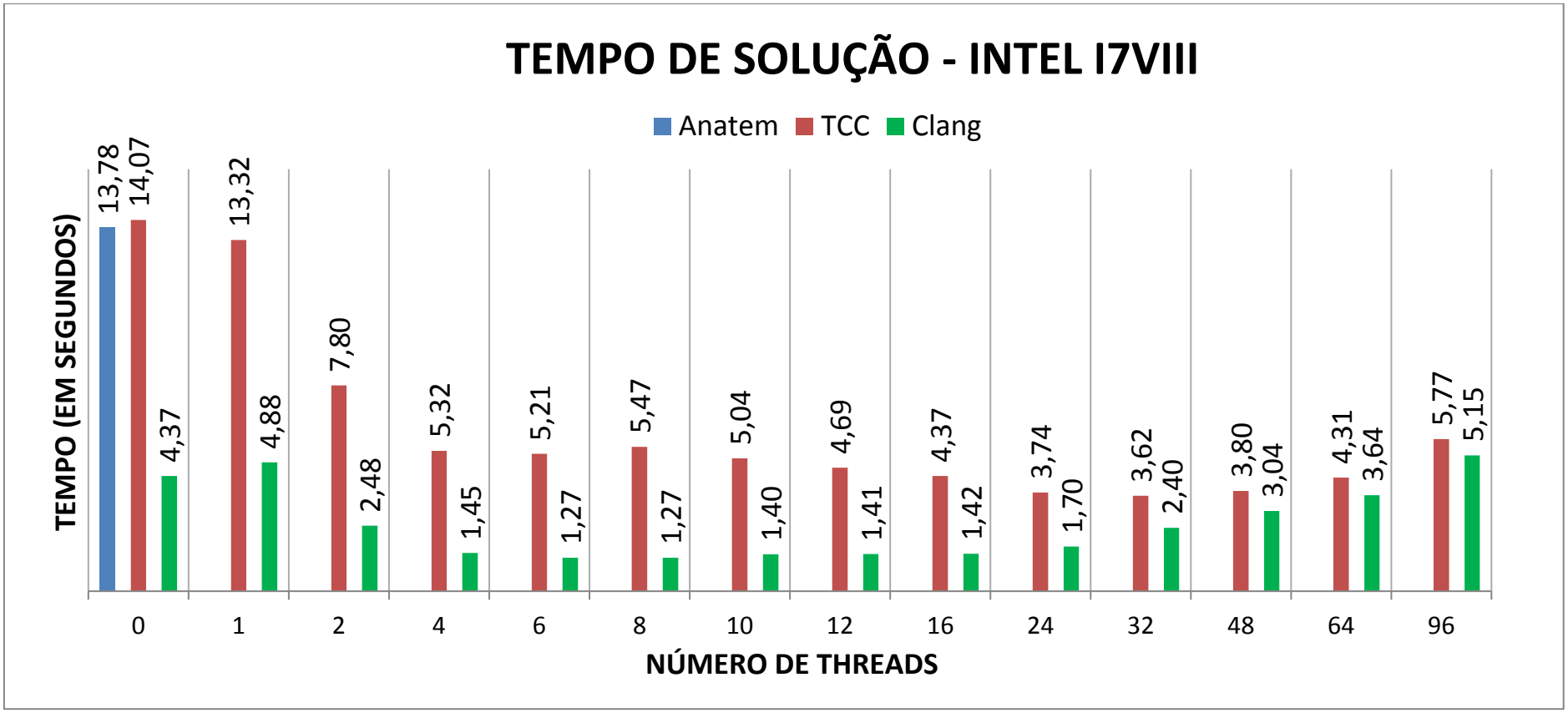


Figura 78 – Tempo de Solução (Computador i7VIII) (OC)

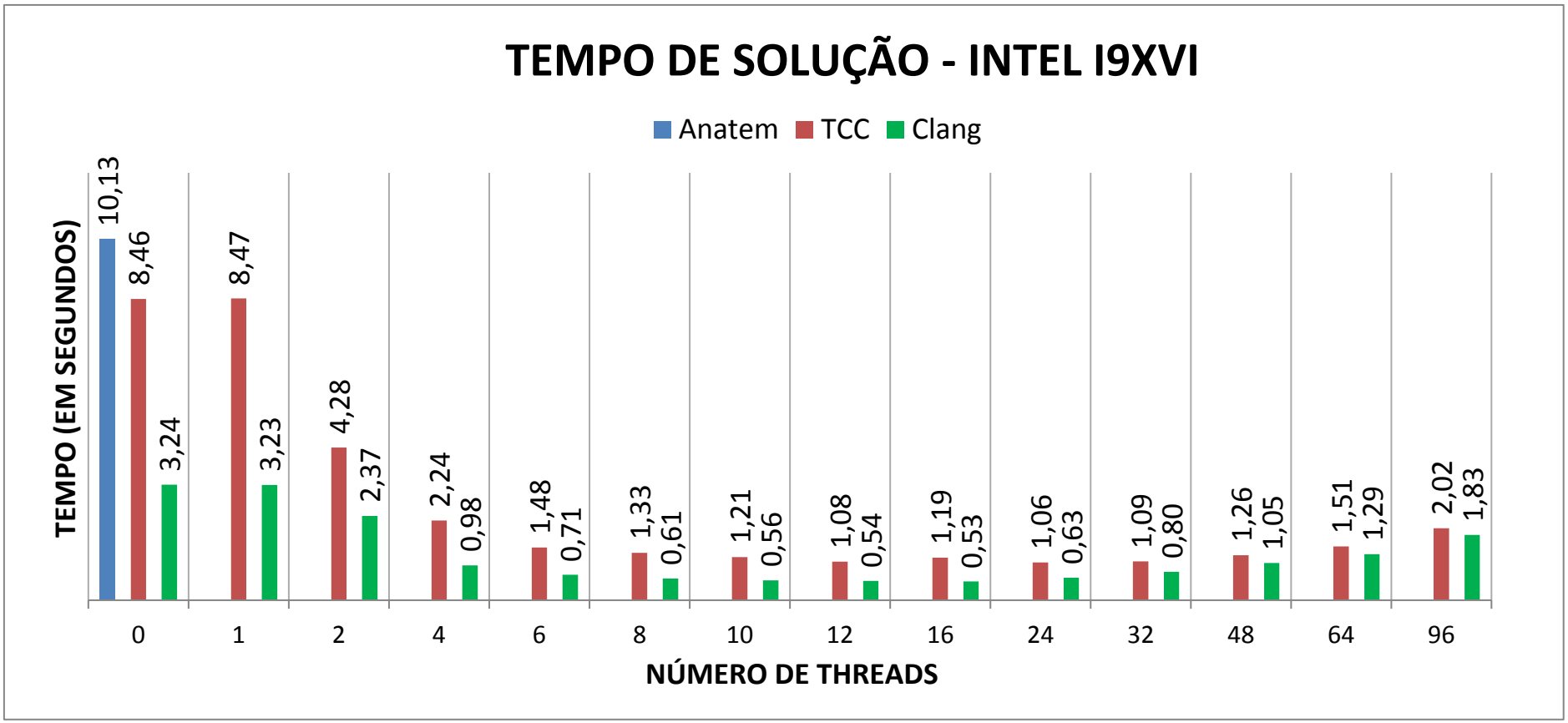


Figura 79 – Tempo de Solução (Computador i9XVI) (OC)

4.2 Caso 2: Simulação do Caso BenchCA no AnaHVDC

O primeiro caso simulado no AnaHVDC foi o BenchCA, adaptado de [66] pela exclusão do elo de corrente contínua, cuja estrutura é mostrada na Figura 80.

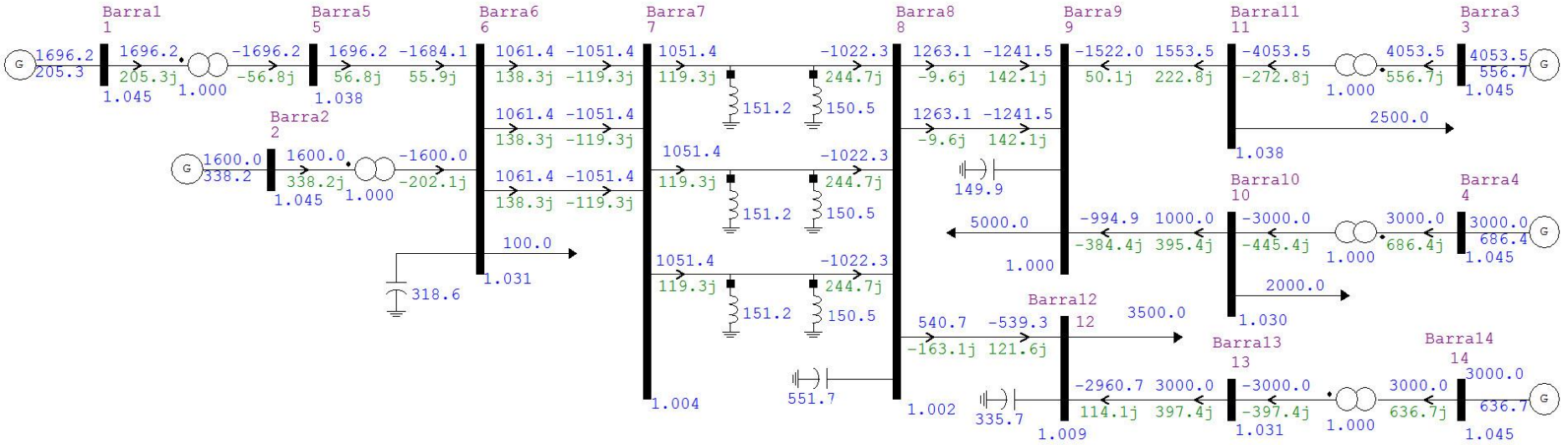


Figura 80 – Circuito do Caso BenchCA

Tendo-se como base o BenchCA foram simulados dois subcasos. Ambos consistiram em um curto na barra 7, sendo um deles um curto monofásico e outro um curto trifásico. Os detalhes sobre os casos, incluindo os arquivos de dados, podem ser encontrados no Apêndice D. O comparativo foi realizado considerando-se o AnaHVDC, o PSCAD e o Anatem.

4.2.1 Curto Balanceado

O curto franco, neste caso trifásico, é aplicado em 0,2 s na barra 7 e representado por uma resistência de 10^{-6} pu Na sequência, em 0,3 s, ocorre a abertura total do circuito do lado da barra 7. Por fim, em 0,32 s, é realizada a abertura total do circuito do lado da barra 8. Neste caso considerou-se uma abertura instantânea dos polos do disjuntor, de forma a evitar o surgimento de desbalanços no circuito. O objetivo é verificar didaticamente se o Anatem, que possui modelagem apenas da sequência positiva equivalente, aproxima-se do AnaHVDC e PSCAD, mesmo que seja uma premissa menos próxima da realidade.

Esse é um caso instável e o Anatem interrompe a simulação logo após 2 s devido à perda de sincronismo de uma das máquinas[37]. Dessa forma, todas as simulações deste subcaso foram restritas a dois segundos.

Da Figura 81 à Figura 118 são mostrados os resultados obtidos. Deve-se entender "Corrente de Campo Gerador 1" como "Corrente de campo do gerador conectado à barra 1". Além disso, "Ângulo Gerador 1" é o "Ângulo absoluto em radianos do eixo q do gerador conectado à barra 1"

Verifica-se neste caso uma ótima concordância entre o AnaHVDC e o PSCAD, onde as diferenças nas grandezas fasoriais explicam-se pela medição diferenciada desses valores nos programas. No PSCAD utiliza-se um medidor RMS digital e, no AnaHVDC, se faz a raiz da soma quarática dos valores instantâneos das três fases.

[37] Mensagem exata transcrita do aquivo OUT do Anatem: "T = 2.0080s - Ângulo da máquina 10 da barra 1 Barra1 ultrapassou o limite de 1000.0 graus. O caso será encerrado. Ângulo em relação ao centro de massa : 1001.2 graus."

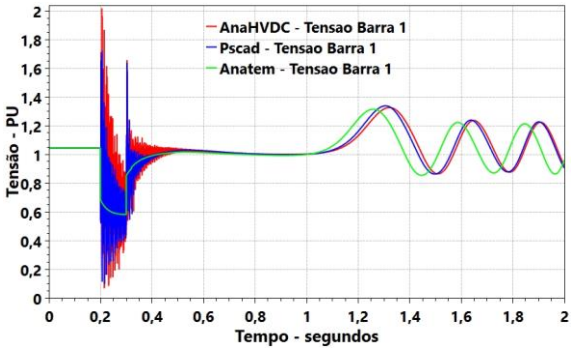


Figura 81 - Tensão Barra 1

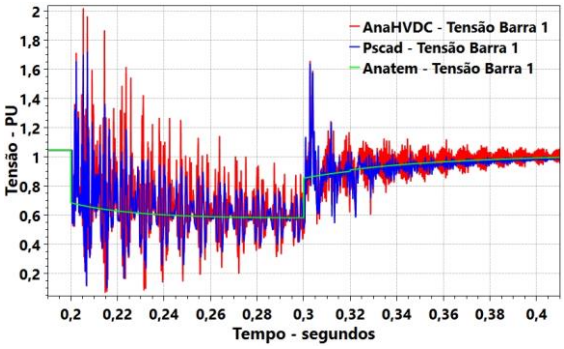


Figura 82 - Tensão Barra 1 (zoom evento)

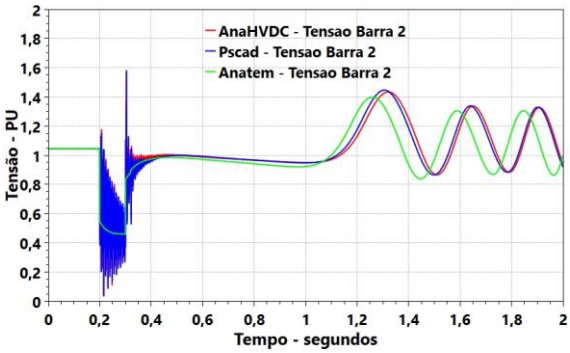


Figura 83 - Tensão Barra 2

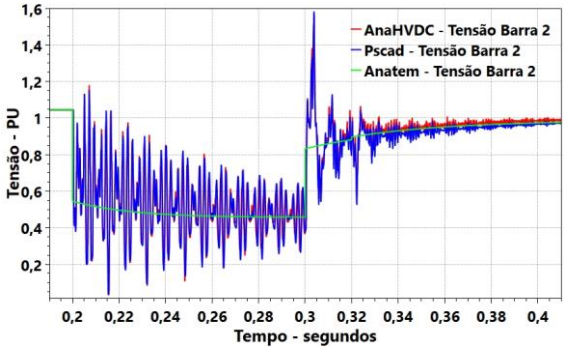


Figura 84 - Tensão Barra 2 (zoom evento)

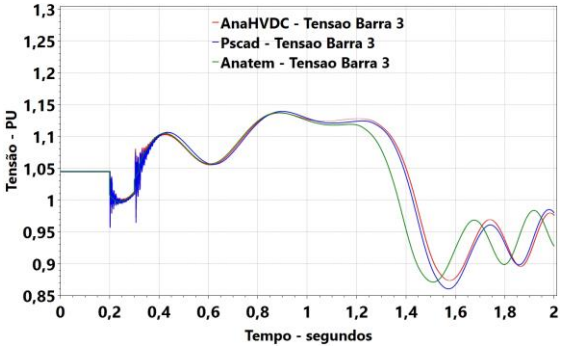


Figura 85 - Tensão Barra 3

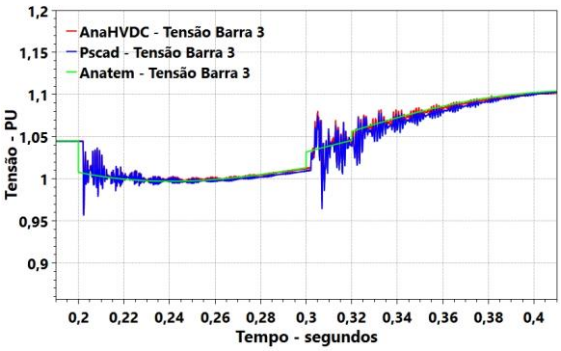


Figura 86 - Tensão Barra 3 (zoom evento)

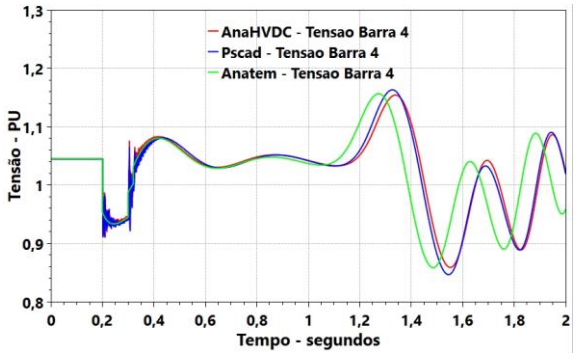


Figura 87 - Tensão Barra 4

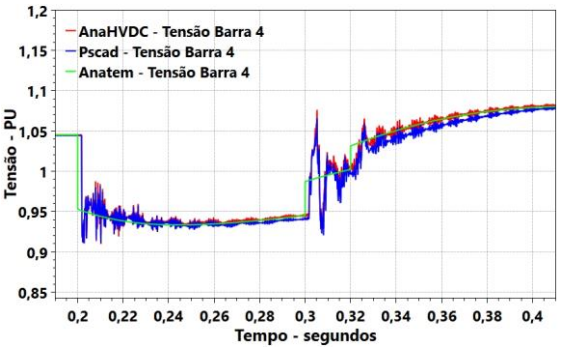


Figura 88 - Tensão Barra 4 (zoom evento)

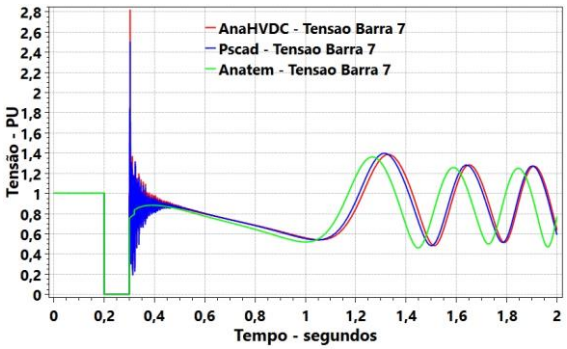


Figura 89 - Tensão na Barra 7

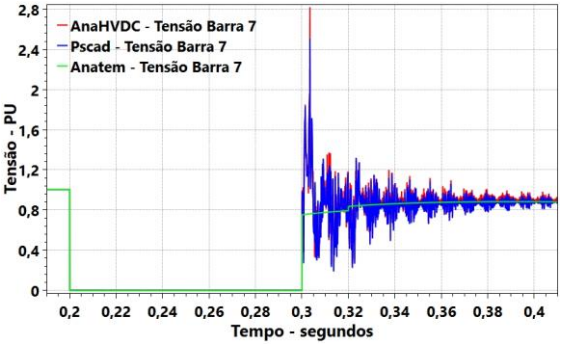


Figura 90 - Tensão Barra 7 (zoom evento)

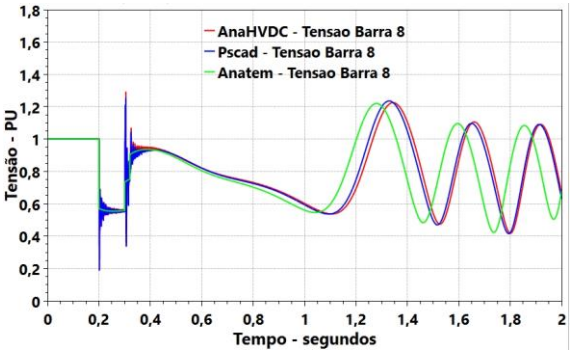


Figura 91 - Tensão na Barra 8

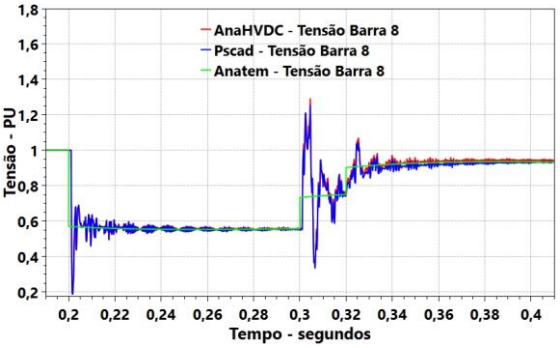


Figura 92 - Tensão Barra 8 (zoom evento)

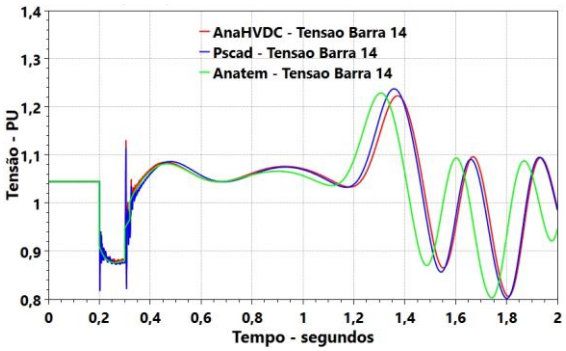


Figura 93 - Tensão Barra 14

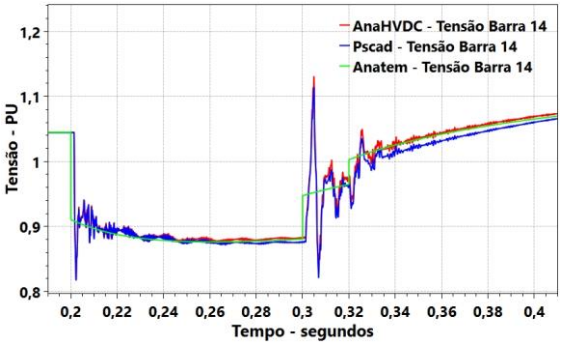


Figura 94 - Tensão Barra 14 (zoom evento)

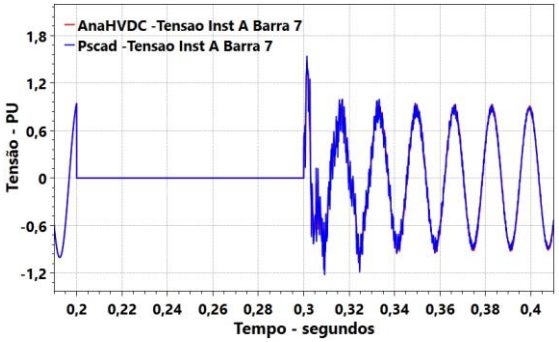


Figura 95 - Tensão Instantânea na Fase A na Barra 7

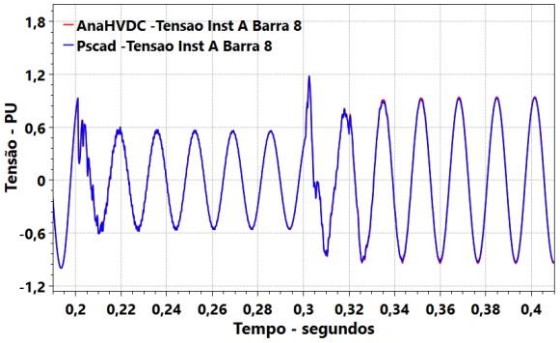


Figura 96 - Tensão Instantânea na Fase A na Barra 8

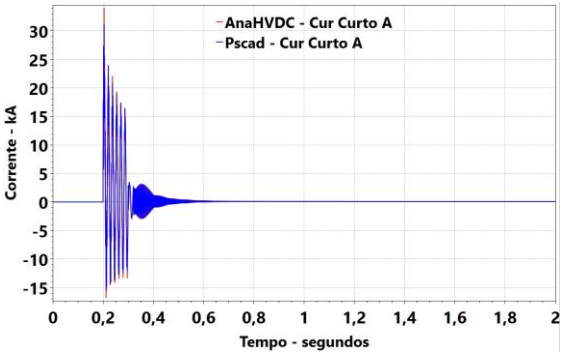


Figura 97 - Corrente Instantânea de Curto na Fase A

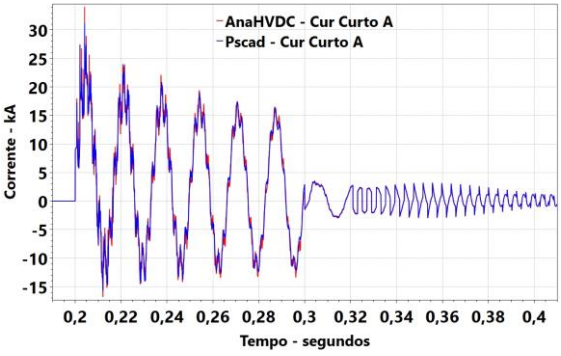


Figura 98 - Corrente Instantânea de Curto na Fase A (zoom no evento)

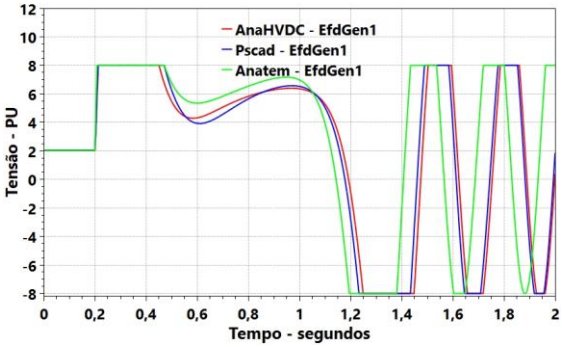


Figura 99 - Corrente de Campo Gerador 1

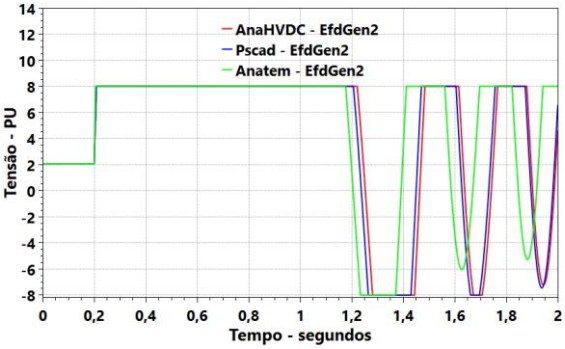


Figura 100 - Corrente de Campo Gerador 2

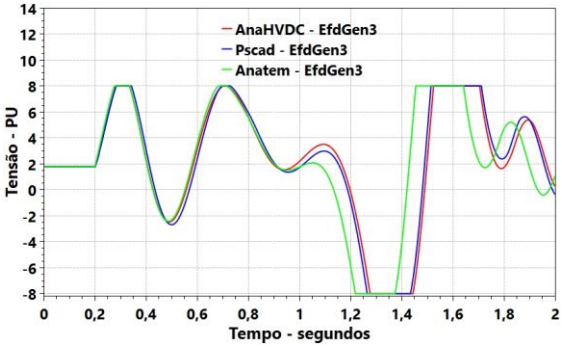


Figura 101 - Corrente de Campo Gerador 3

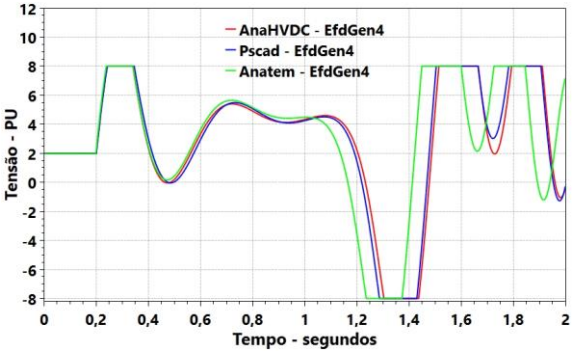


Figura 102 - Corrente de Campo Gerador 4

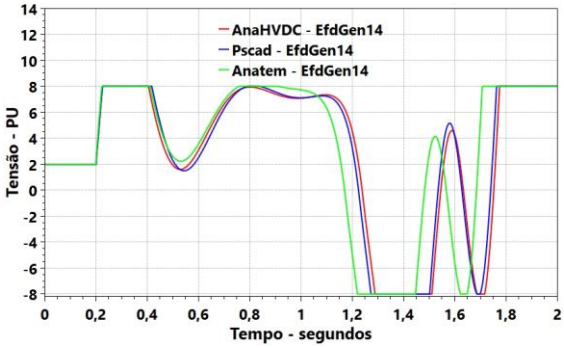


Figura 103 - Corrente de Campo Gerador 14

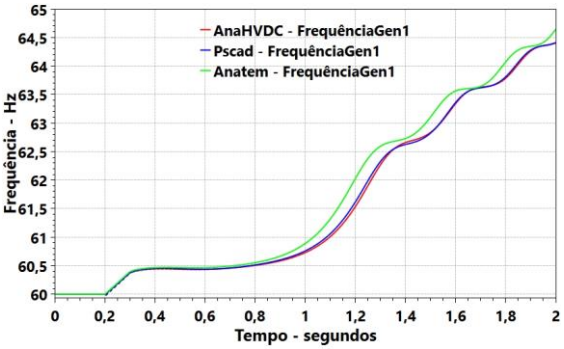


Figura 104 - Frequência Gerador 1

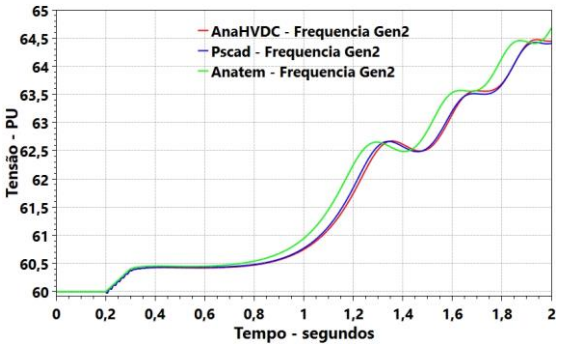


Figura 105 - Frequência Gerador 2

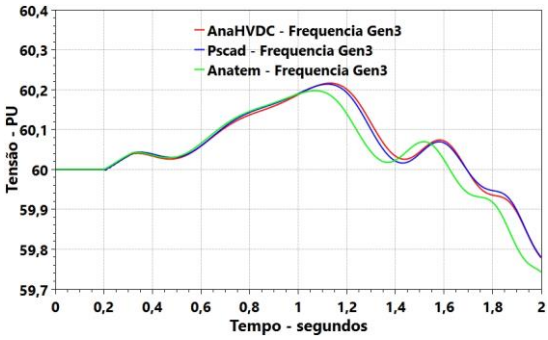


Figura 106 - Frequência Gerador 3

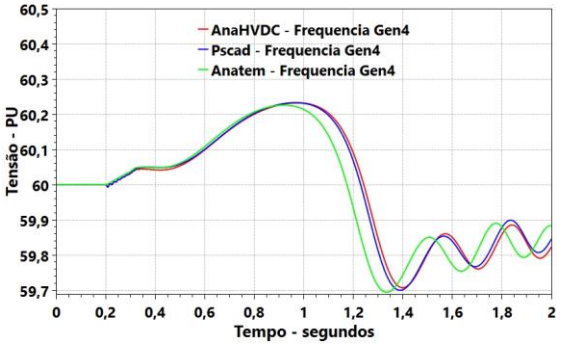


Figura 107 - Frequência Gerador 4

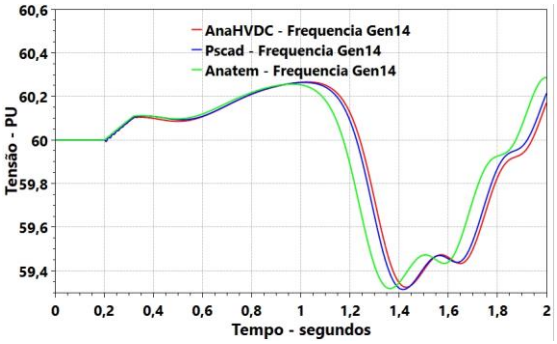


Figura 108 - Frequência Gerador 14

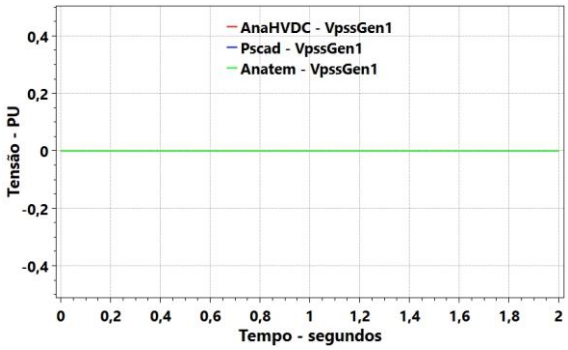


Figura 109 - Tensão Estabilizador 1

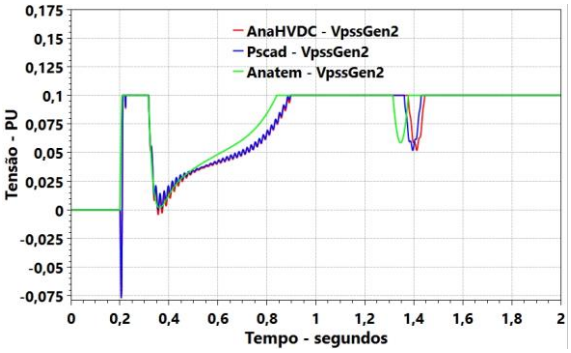


Figura 110 - Tensão Estabilizador 2

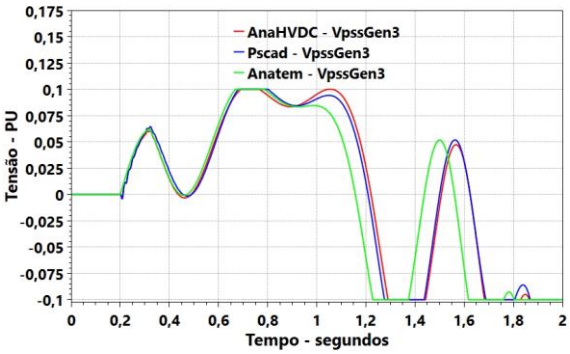


Figura 111 - Tensão Estabilizador 3

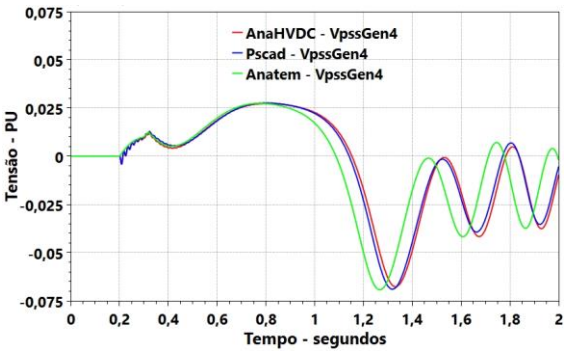


Figura 112 - Tensão Estabilizador 4

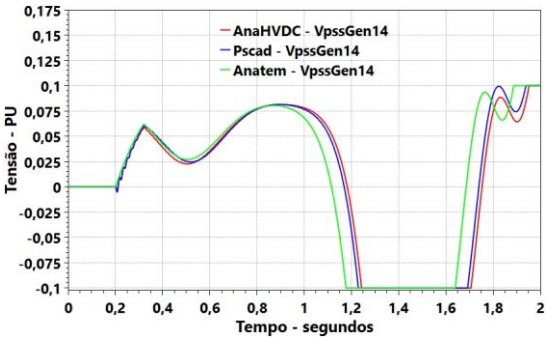


Figura 113 - Tensão Estabilizador 14

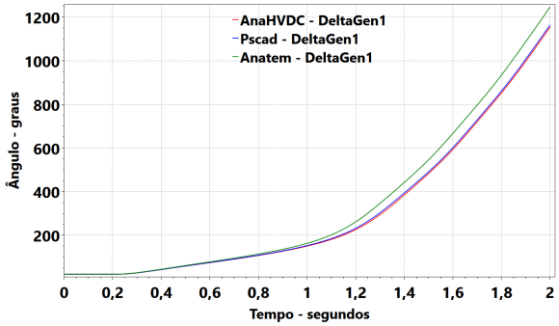


Figura 114 - Ângulo Gerador 1

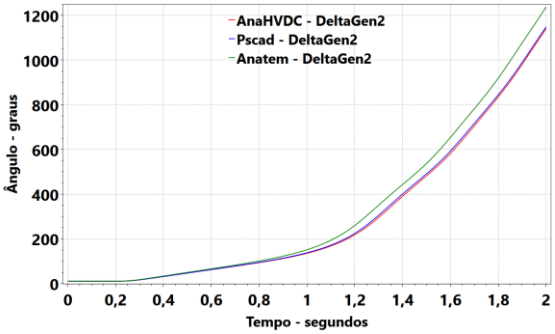


Figura 115 - Ângulo Gerador 2

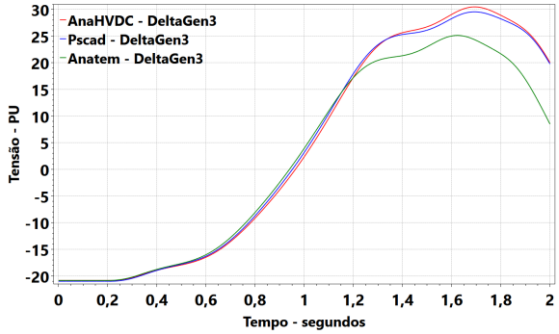


Figura 116 - Ângulo Gerador 3

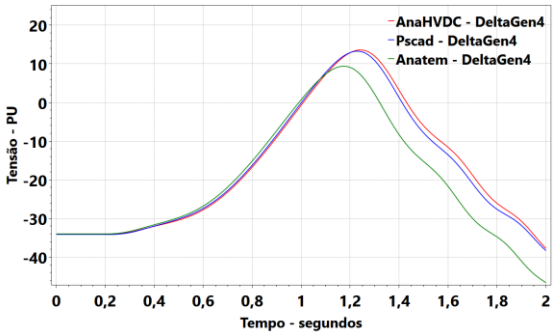


Figura 117 - Ângulo Gerador 4

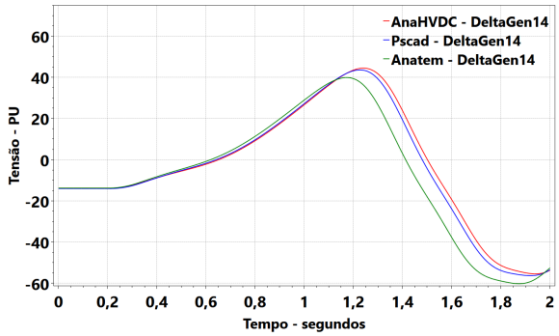


Figura 118 - Ângulo Gerador 14

Os resultados da simulação se apresentaram coerentes. As curvas do AnaHVDC e do Pscad são muito próximas. Por sua vez, a ausência da dinâmica de transitórios magnéticos no Anatem fez com que o resultado do programa apresentasse uma diferença significativa. Entretanto, nota-se que para os três programas os resultados não divergem entre si e ainda apresentam o mesmo formato.

Analisando-se individualmente os resultados apresentado é possível traçar algumas conclusões. Primeiramente, a instabilidade do caso é notada claramente na Figura 104, em que a frequência cresce sem apresentar qualquer tendência de estabilização. Essa mesma ideia pode ser percebida a partir da análise da Figura 114, que mostra a completa perda de

sincronismo do gerador. É importante pontuar que se está simulando a defasagem em relação à referência síncrona.

Por sua vez, é possível perceber que o afundamento da tensão da Barra 1 (Figura 81) é significativamente maior do que aquele verificado, por exemplo, na Barra 3 (Figura 85). O resultado gráfico corrobora a intuição de que o impacto sobre a tensão está relacionada com a distância em relação ao curto. Nota-se também que a tensão da Barra 7 (Figura 89), após a aplicação do curto, crava em zero. Isso era era de se esperar uma vez que o curto foi aplicado justamente nessa barra. Com a abertura do disjuntor em 0,3 s a tensão aumenta novamente. Por sua vez, na barra 8 (Figura 91), embora seja fortemente afetada, não se verifica a tensão tendendo a zero. Isso se justifica pela presença da linha de transmissão entre a barra e o curto, o que ameniza o impacto.

4.2.2 Curto Fase-Terra

Neste caso, um curto circuito franco é aplicado na fase a em 0,2 s na barra 7 e é representado por uma resistência de 10^{-6} pu. Na sequência, a partir de 0,3 s, inicia-se o processo de abertura do circuito do lado da barra 7. Por fim, em 0,32 s, é iniciado o processo de abertura do circuito do lado da barra 8. Em ambas as aberturas, considerou-se que cada polo do disjuntor efetivamente abria na passagem pelo zero da corrente, muito próximo de como ocorre em um sistema real de alta tensão, aumentando os desbalanços da simulação.

Da Figura 119 à Figura 144 são mostrados os resultados obtidos. Nas figuras em que não é possível perceber a linha do AnaHVDC é porque ele está encoberto pelo Pscad, o que demonstra a proximidade dos resultados. Ao contrário do caso apresentado no tópico anterior, este é um caso estável e, portanto, foi possível realizar uma simulação de 5 s.

Neste caso de evento desbalanceado, ocorre o que era esperado, o Anatem possui diferenças mais expressivas de resultados por desprezar os transitórios eletromagnéticos e só possuir modelagem de sequência positiva, não permitindo reproduzir os desbalanços das fases. Para considerar o curto circuito na Anatem, adotou-se uma reatância de curto circuito de 0,2%, que neste caso produz uma queda da tensão da barra do curto para um valor entre 0,6 pu e 0,7 pu, conforme premissa comum adotada nos estudos práticos de simulações do SIN pelo setor elétrico. Por outro lado, os resultados entre AnaHVDC e PSCAD estão bem próximos, validando o UDCC que foi utilizado na simulação dos controladores. As diferenças dos fasores de tensão encontradas explicam-se pelo fato que estes fasores no

PSCAD foram medidos por um medidor RMS digital, enquanto que no AnaHVDC, foi utilizada a raiz quadrada da soma quadrática das tensões nas fases.

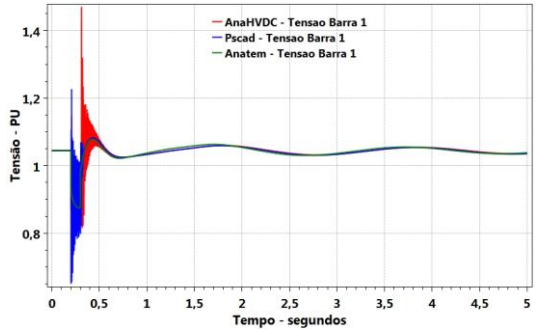


Figura 119 - Tensão Barra 1

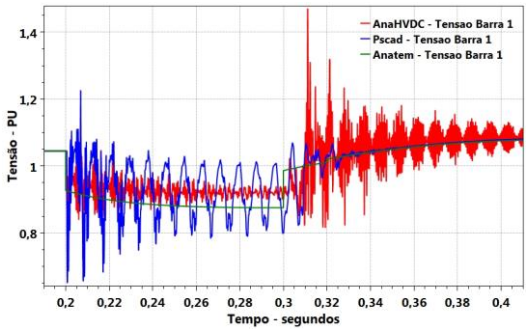


Figura 120 - Tensão Barra 1 (zoom evento)

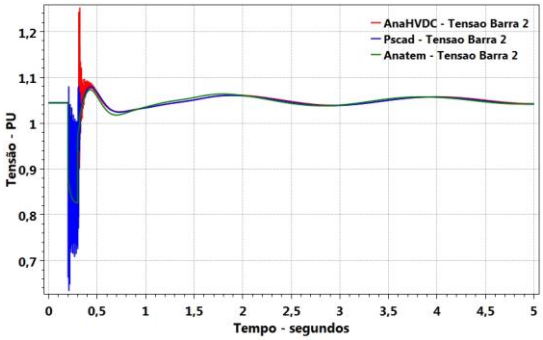


Figura 121 - Tensão Barra 2

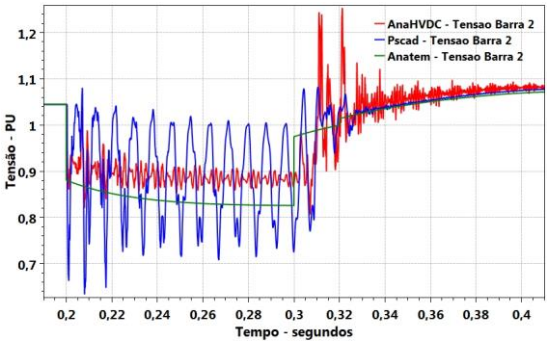


Figura 122 - Tensão Barra 2 (zoom evento)

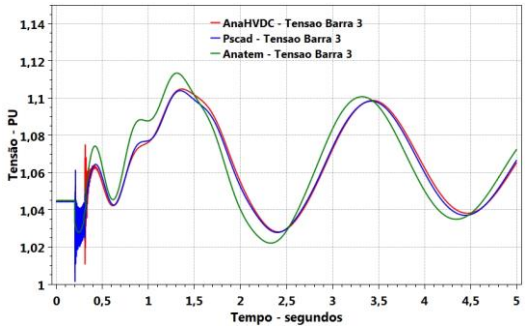


Figura 123 - Tensão Barra 3

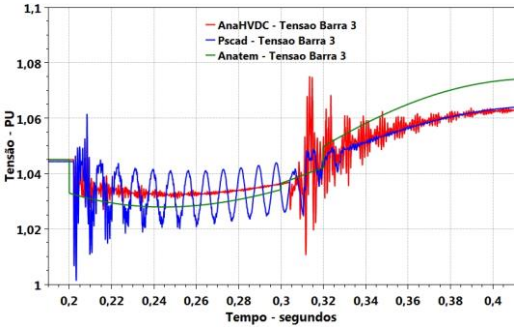


Figura 124 - Tensão Barra 3 (zoom evento)

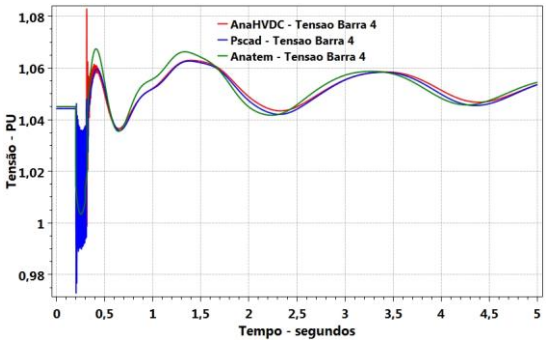


Figura 125 - Tensão Barra 4

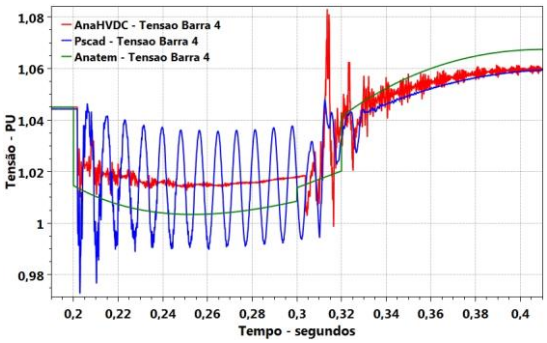


Figura 126 - Tensão Barra 4 (zoom evento)

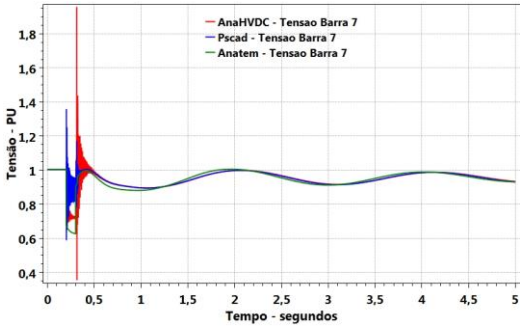


Figura 127 - Tensão Barra 7

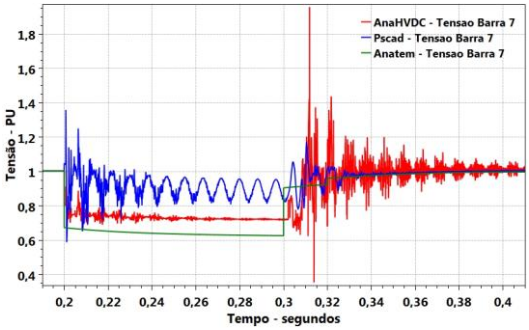


Figura 128 - Tensão Barra 7 (zoom evento)

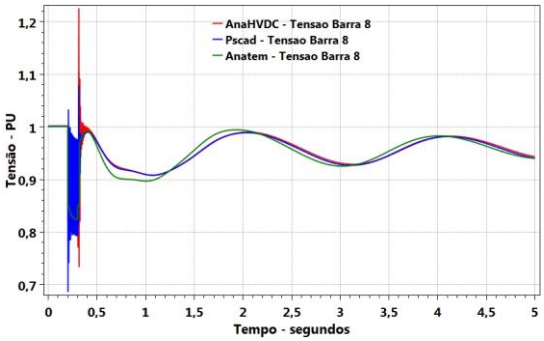


Figura 129 - Tensão Barra 8

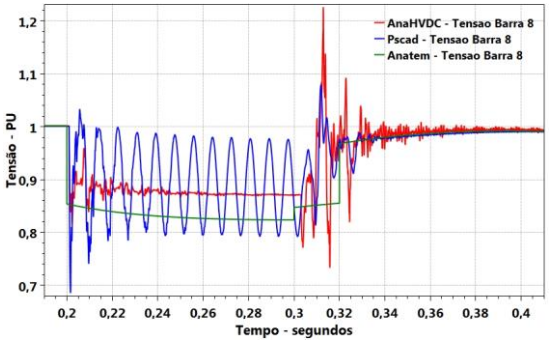


Figura 130 - Tensão Barra 8 (zoom evento)

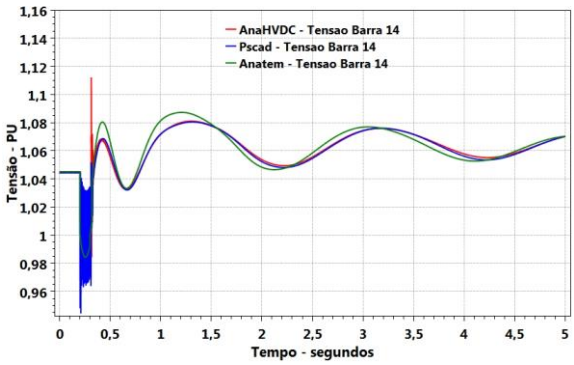


Figura 131 - Tensão Barra 14

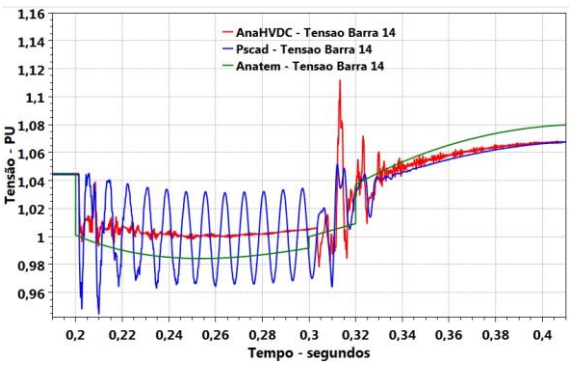


Figura 132 - Tensão Barra 14 (zoom evento)

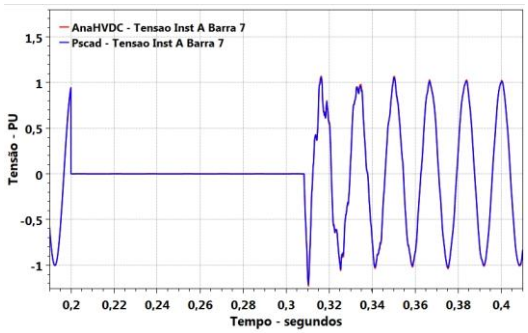


Figura 133 - Tensão Instantânea da Fase A na Barra 7

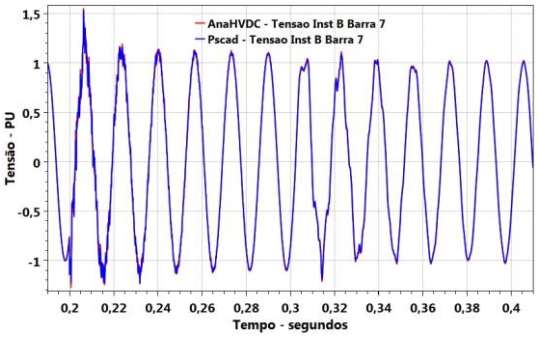


Figura 134 - Tensão Instantânea da Fase B na Barra 7

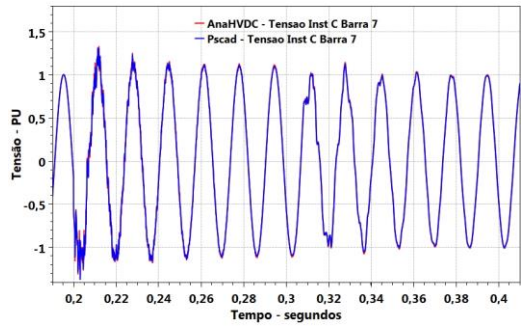


Figura 135 - Tensão Instantânea Fase C Barra 7

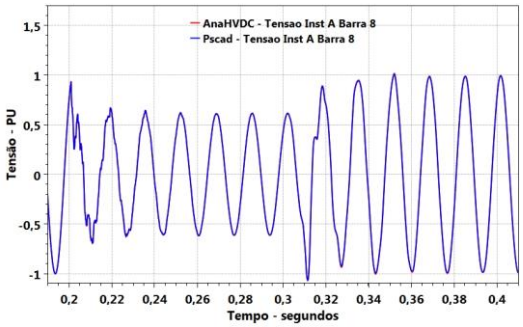


Figura 136 - Tensão Instantânea Fase A Barra 8

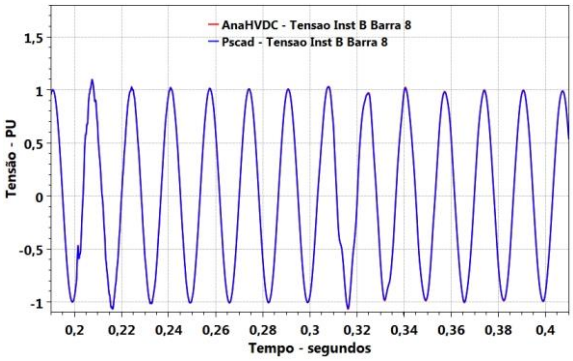


Figura 137 - Tensão Instantânea Fase B Barra 8

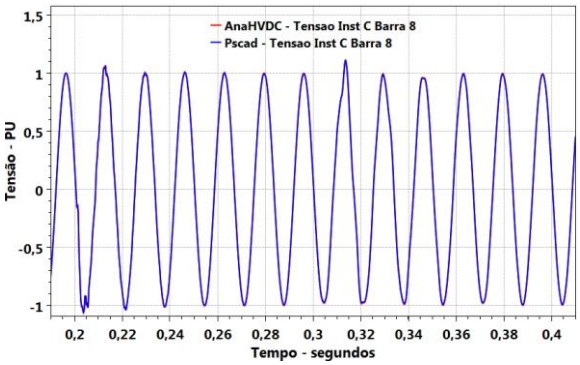


Figura 138 - Tensão Instantânea Fase C Barra 8

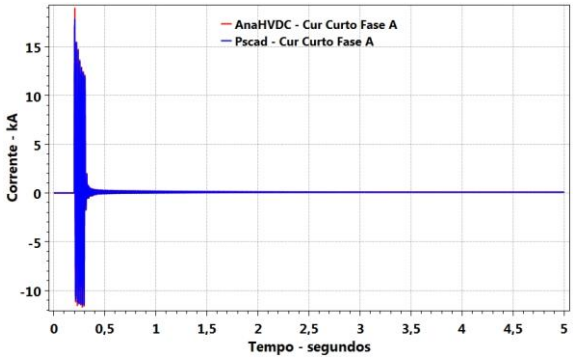


Figura 139 - Corrente Instantânea Curto Fase A

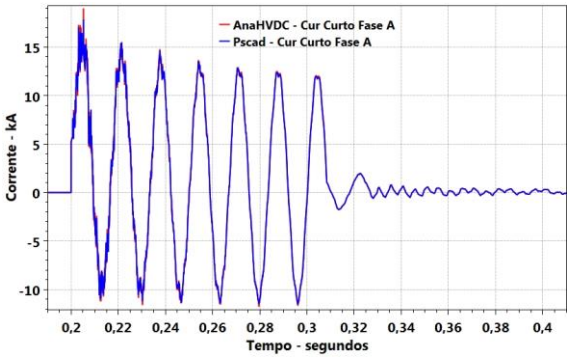


Figura 140 - Corrente Instantânea de Curto na Fase A (zoom no evento)

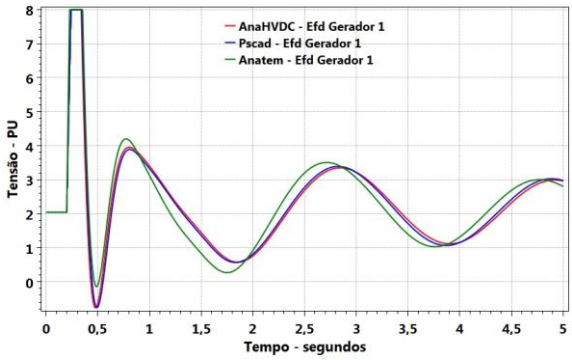


Figura 141 - Tensão de Campo Gerador 1

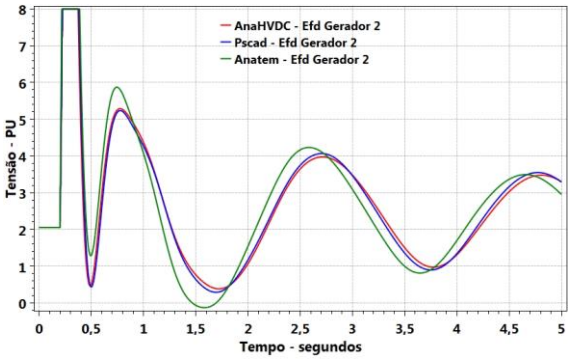


Figura 142 - Tensão de Campo Gerador 2

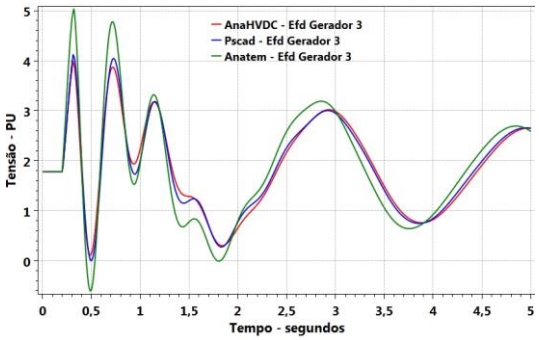


Figura 143 - Tensão de Campo Gerador 3

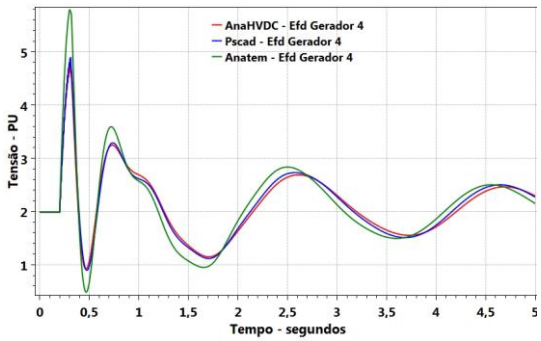


Figura 144 - Tensão de Campo Gerador 4

Novamente, os resultados da simulação se apresentaram coerentes. As curvas do AnaHVDC e do Pscad são muito próximas. Por sua vez, a ausência da dinâmica de transitórios magnéticos no Anatem fez com que o resultado do programa apresentasse uma diferença significativa. Entretanto, nota-se que para os três programas os resultados não divergem entre si e ainda apresentam o mesmo formato.

Comparando-se a Figura 119 com a Figura 81 nota-se que a severidade de um curto trifásico balanceado é maior do que a de um curto fase-terra no sentido de que o afundamento das tensões para o caso fase-terra foi significativamente menor. Entretanto, a tendência geral se verifica: as barras mais afastadas do curto tendem a sofrer um impacto menor. Analisando-se o caso particular da tensão fasorial da barra 7 (Figura 125) percebe-se que ela não tende a zero, o que contrasta com o que aconteceu no caso trifásico balanceado (Figura 89). Entretanto, a tensão instantânea da Fase A na barra 7 (Figura 133) de fato se anula, como esperado. Por sua vez as tensões da Fase B (Figura 134) e da Fase C (Figura 135) também na barra 7 sofrem o impacto do curto, mas de modo pouco expressivo.

4.3 Caso 3: Simulação do Caso PAR2023 no AnaHVDC

4.3.1 Pequena Perturbação

Como descrito no descrito no tópico 3.6, no AnaHVDC os CDUs estão conectados a outros equipamentos e acontece a importação e exportação de sinais. Além disso, como mencionado no tópico 1.2, não foi realizado um comparativo de tempo com o Anatem, uma vez que ele se encontra no domínio eletromecânico, enquanto o AnaHVDC também abrange o domínio eletromagnético, de modo que as equações no AnaHVDC são mais precisas e, portanto, o cálculo é computacionalmente mais pesado. Dessa forma, um comparativo de tempo não seria justo.

Na Figura 145 é mostrado o arquivo de entrada do AnaHVDC do exemplo simulado.

```
( Caso SIN sem HVDC sem SVC
DHIS
'PAR2023.SAV' 4
DSTB
'bd1218.blt'
'bd1218.cdu'
'dmaq_ssr.dat'
FIM
DDLL
'bd1218.dll' 'bd1218.cdu'
FIM
DLOA
(label) (nb1) (type)
 ALL            0
FIM
DLTA
(label) (from) (toid) (ncir) (length) (nPi) (minTal)
 ALL                                    1
FIM
DSIM
(step) (duration) (period) (tolerr) (maxiter)
 20e-6      5.0      1       1e-10     50
FIM
DEVT
(tp) (  time  ) (bus)(value)
ADDR       0.02   105  100.
(STEP_VSRC  0.02     10   -0.1
FIM
DPLT
('        label      ') (mode) (device) (variable) (id1) (id2) (id3)
(Seq) ( factor ) ( T )
'Volt#Bus105abs'         ABS     BUS         VOLT   105
'Volt#Bus106abs'         ABS     BUS         VOLT   106
'Volt#Bus10abs'          ABS     BUS         VOLT    10
'Volt#Bus1100abs'        ABS     BUS         VOLT  1100
'Volt#Bus1107abs'        ABS     BUS         VOLT  1107
'Volt#Bus502abs'         ABS     BUS         VOLT   502
'Tensao Inst 105'       INST     BUS         VOLT   105
'Tensao Inst 106'       INST     BUS         VOLT   106
'WMAQ'                  INST     GEN           WG     10  10
'PE'                    INST     GEN           PE     10  10
'DELTA'                 INST     GEN        DELTA     10  10
'EFD'                   INST     GEN          EFD     10  10
'PM'                    INST     GEN           PM     10  10
FIM
```

Figura 145 - Arquivo PAR2023.ahv

Dessa forma, inicia-se uma breve descrição dos códigos de execução do arquivo de entrada.

DHIS, indica o arquivo de fluxo de potência do Anarede a ser importado. Dessa forma, o AnaHVDC pode inicializar automaticamente suas variáveis.

DSTB contém a identificação de três arquivos associados. O CDU é usado para realizar a importação dos DLOCs, explicados no tópico 3.6. O arquivo bd1218.blt descreve os dados dos modelos *built-in* das máquinas. Um recorte do arquivo deste exemplo se encontra na Figura 146 e um detalhamento de cada campo pode ser encontrado em [19] no tópico "Código de Execução DMDG" e no tópico "Código de Execução DCST". Por sua vez, o arquivo dmaq_ssr.dat realiza a associação entre os modelos das máquinas descritas pelo arquivo bd1218.blt com a rede. Dessa forma, no arquivo bd1281.blt está descrito a qual barra cada modelo está conectado, além de outras informações que podem ser consultadas também em [19], no tópico "Código de Execução DMAQ". A Figura 147 ilustra um recorte desse arquivo.

```
( Modelos de máquinas c/ polos lisos (Usinas Térmicas)
(
DMDG MD03
(
(....... UNE Angra I
(No)    (CS)  (Xd )(Xq )(X'd)(X'q)(X"d)(Xl )(T'd)(T'q)(T"d)(T"q)
0100    0100 172.0167.948.8080.0033.70 26.6 5.300.6250.0480.066
(No)    (Ra )( H )( D )(MVA)Fr C
0100         3.859     760.0   N
```

Figura 146 - Arquivo bd1218.blt associado ao PAR2023

```
DMAQ
( Nb)    Gr (P) (Q) Und ( Mg ) ( Mt )u( Mv )u( Me )u(Xvd)(Nbc) (Mrot)
   10    10 100 100   1    100    100u   140u   170u        10          ANGRA-1--1G
   12    10 100 100   6    103    103u   143u   173u        12          LCBARRET-6G
   14    10 100 100   3    105    105u   145u               14          FUNIL----3G
```

Figura 147 - Arquivo bd1218.dat associado ao PAR2023

Dá-se continuidade à descrição dos códigos de execução da Figura 145. DDLL se refere a DLL gerada pelo UDCC do arquivo CDU associado ao caso. Nota-se que o próprio arquivo CDU (bd1218.cdu) também é informado. Dessa forma, caso não haja a DLL, o próprio AnaHVDC assume a responsabilidade de chamar o UDCC para realizar sua geração.

Em DSIM são detalhados os dados de simulação, como o passo (*step*) e a duração da simulação (*duration*).

DEVT é o evento aplicado. No caso é o chaveamento de uma resistência de 100 pu entre a barra 105 e a terra, o que ocorre no instante 0,02 s.

Por fim, DPLT são as variáveis plotadas. A primeira coluna indica o nome da variável a ser plotada e, na sequência das colunas, o tipo de plotagem da variável (apenas parte real da variável (REAL), apenas parte imaginária (IMAG) ou seu valor absoluto (ABS)) , o tipo de equipamento (barra (BUS) ou linha de transmissão (LINE)) e o tipo da própria variável (tensão (VOLTAGE) ou corrente (CURRENT)). As colunas *nb1*, *nb2* e *nb3* são os identificadores do equipamento. No exemplo da Figura 145, é usado apenas o primeiro campo, *nb1*, que, nesse caso, se refere ao número da barra. Por fim, a última coluna se refere à fase que deve ser plotada.

O caso simulado, baseado no PAR2023, possui 6972 barras CA, 10088 linhas de transmissão CA, 739 barras infinitas, 265 máquinas síncronas, 173 PSSs e 659 CDUs. Da Figura 148 à Figura 158 são plotadas as variáveis na ordem em que aparecem no campo DPLT da Figura 145. Portanto, primeiramente é plotada a tensão fasorial da barra 105, que foi escolhida porque é nela em que acontece o evento. Também foi plotada a tensão na barra 106, próxima ao evento e também na barra 10, também próxima ao evento e que contém um gerador. Além disso, foram plotadas diversas variáveis desse gerador. Em vermelho é o resultado do AnaHVDC e em azul o do PacDyn. Destaca-se que o PacDyn é justamente voltado para pequenas perturbações e permite a simulação tanto de transitórios eletromecânicos como eletromagnéticos. Uma vez que ele se baseia em um modelo linearizado, as diferenças verificadas podem ser atribuídas a essas características. De forma geral, os sinais são muito próximos, mesmo para um sistema com alto grau de complexidade.

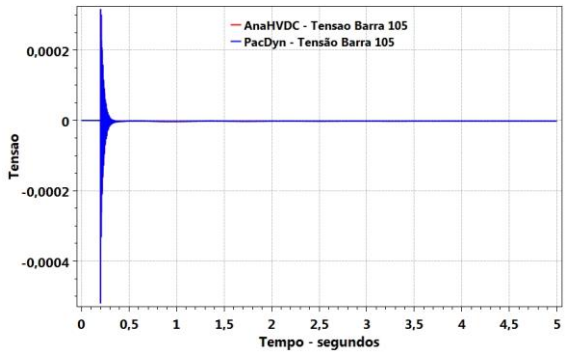


Figura 148 - Tensão Barra 105

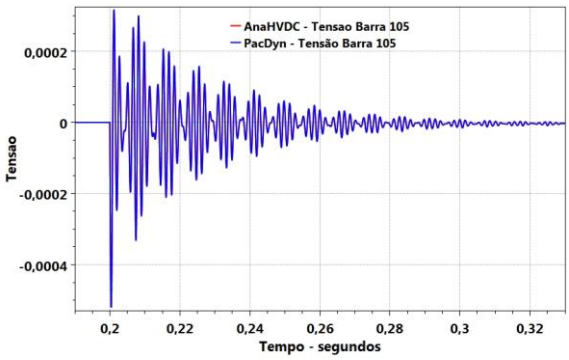


Figura 149 - Zoom na Tensão anterior

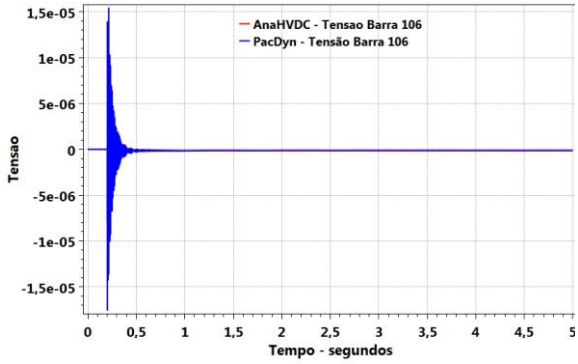


Figura 150 - Tensão Barra 106

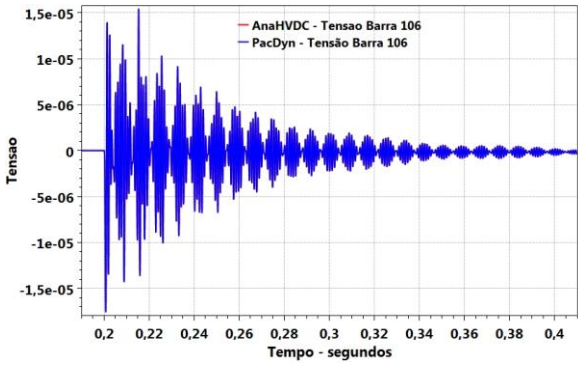


Figura 151 - Zoom na Tensão Anterior

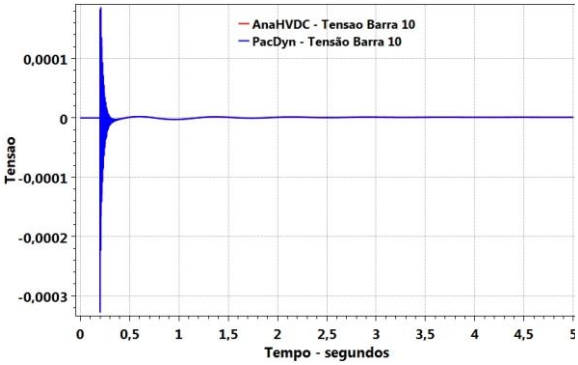


Figura 152 - Tensão Barra 10

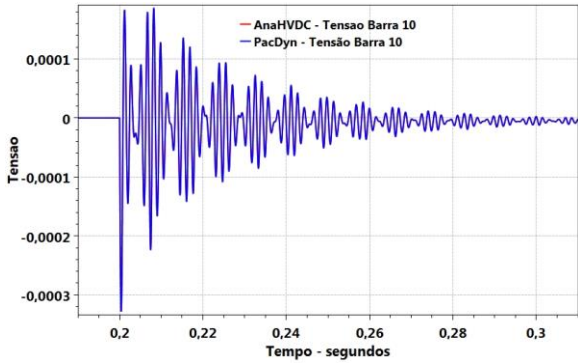


Figura 153 - Zoom na Tensão Anterior

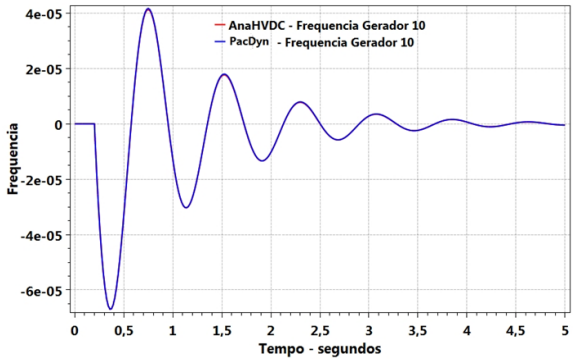


Figura 154 - Frequência Gerador 10

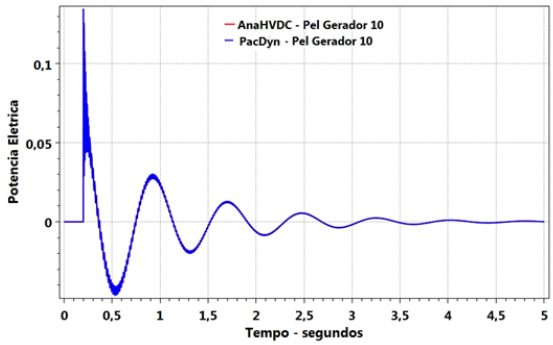


Figura 155 - Potência Elétrica Gerador 10

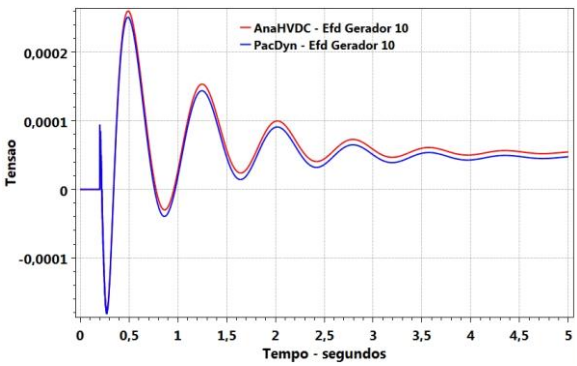


Figura 156 - Tensão de Campo Gerador 10

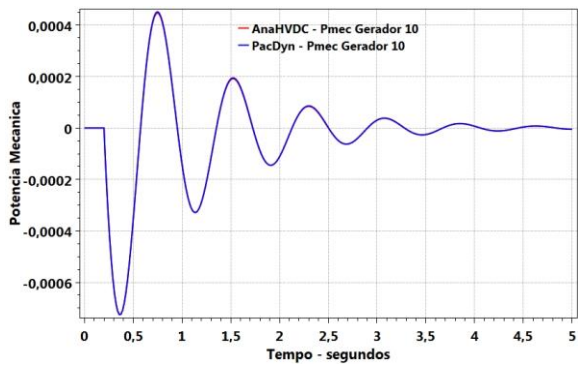


Figura 157 - Potência Mecânica Gerador 10

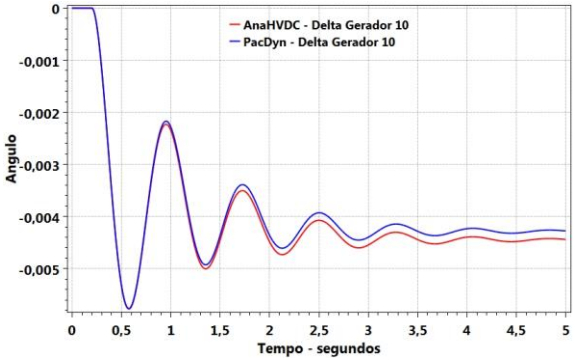


Figura 158 - Ângulo Gerador 10

Novamente, os resultados da simulação se apresentaram coerentes. As curvas do AnaHVDC e do PacDyn são muito próximas. Importante notar que o eixo y das curvas plotadas indicam a diferença para o valor inicial (t = 0). Esse modo de plotagem é o padrão do PacDyn e, portanto, o AnaHVDC foi adaptado para gerar resultado considerando a mesma lógica. Como o curto é representado por uma resistência de 100 pu, de fato o impacto no sistema é pequeno, como se pode comprovar pelos resultados obtidos. Também conforme esperado o resultado é estável, o que pode ser notado pela Figura 158.

4.3.2 Grande Perturbação

Uma grande perturbação foi simulada tendo base o caso PAR2023 usado no tópico anterior. Para isso, foram realizadas duas alterações em relação ao caso anterior. A primeira foi a redução da resistência de curto de 100 pu para 0,0001 pu. Além disso, foi considerado que o curto se extinguia em 0,2 s[38].

Primeiramente foi escolhida a tensão na barra 105, uma vez que foi nela que aconteceu o curto, além da barra 106, próxima ao curto. Por outro lado, foram escolhidas duas barras distantes do curto: 1100 e 1107. A barra 10 também foi escolhida porque é nela que acontece o curto. Além disso, foram plotadas diversas variáveis do gerador mencionado.

[38] No caso da pequena perturbação o chaveamento da resistência perdurou por toda a simulação.

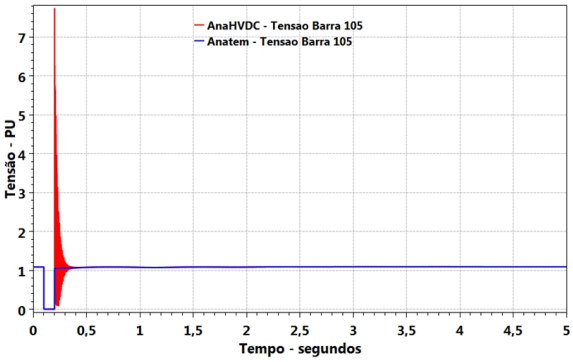


Figura 159 - Tensão Fasorial Barra 105

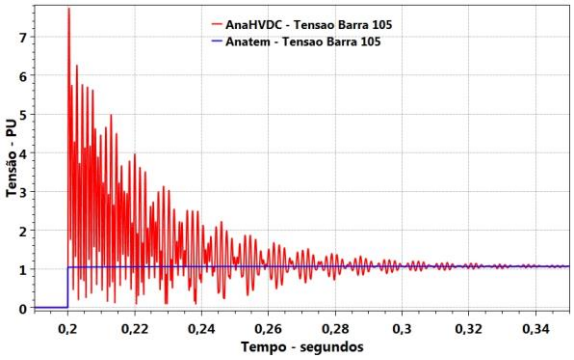


Figura 160 - Tensão anterior (zoom evento)

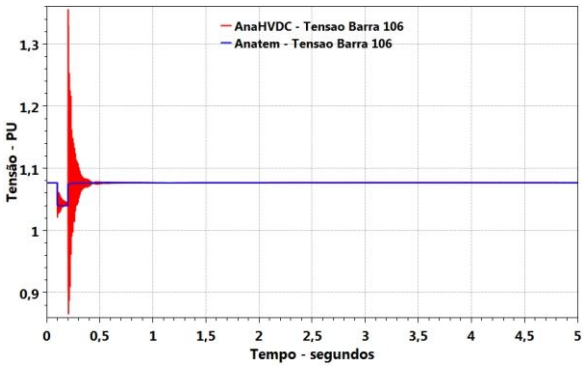


Figura 161 - Tensão Fasorial Barra 106

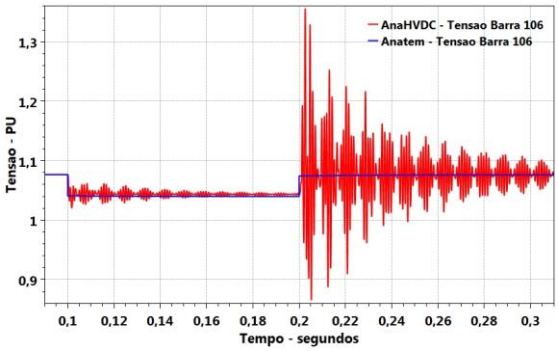


Figura 162 - Tensão anterior (zoom evento)

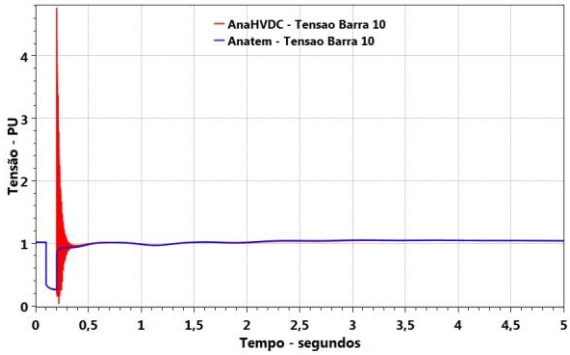


Figura 163 - Tensão Fasorial Barra 10

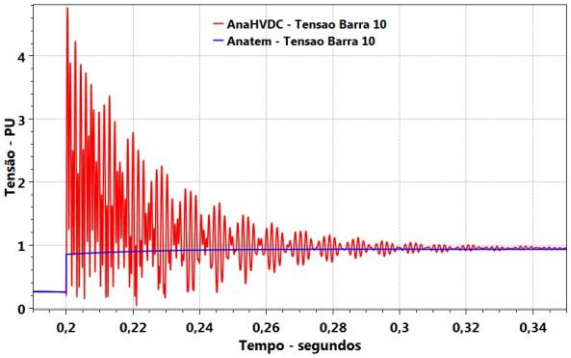


Figura 164 - Tensão anterior (zoom evento)

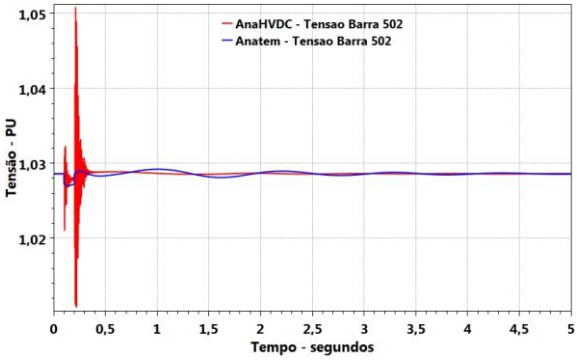


Figura 165 - Tensão Fasorial Barra 502

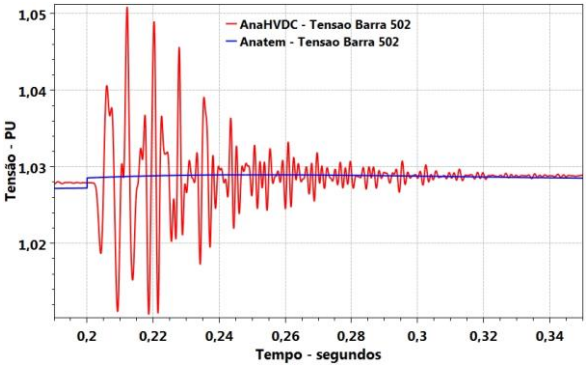


Figura 166 - Tensão anterior (zoom evento)

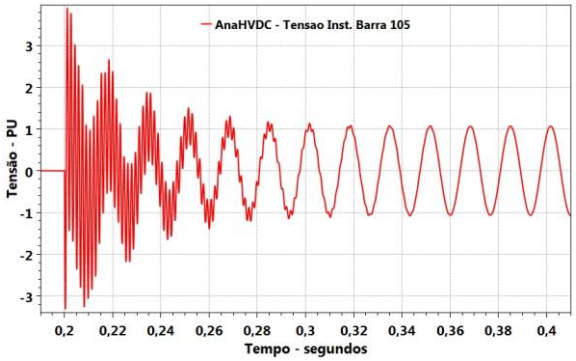


Figura 167 –-Tensão Instantânea Barra 105

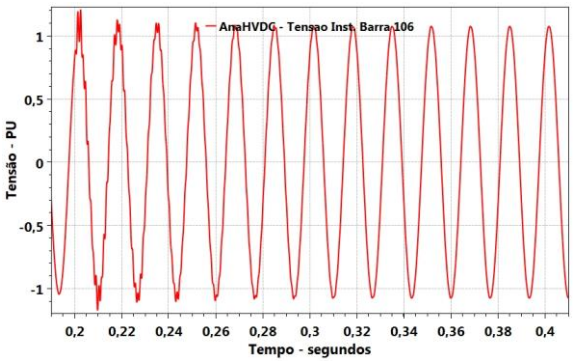


Figura 168 - Tensão Instantânea Barra 106

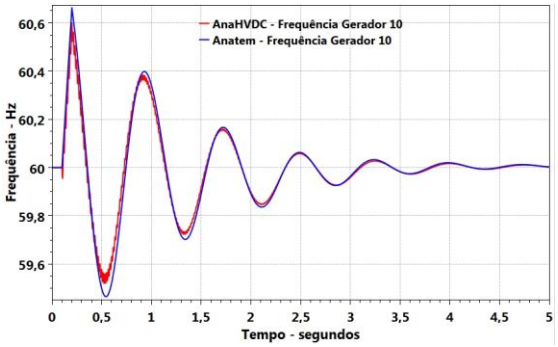


Figura 169 - Frequência Gerador 10

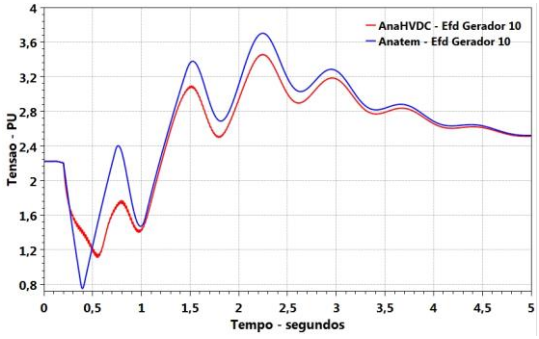


Figura 170 - Tensão de Campo Gerador 10

O Anatem não possui modelagem eletromagnética e, portanto, é incapaz de reproduzir o transitório verificado nas curvas de resultados do AnaHVDC. De todo modo, os resultados se mostram coerentes no sentido de que ambos convergem para um mesmo ponto de operação para todos resultados mostrados.

Merece destaque a Figura 159, que indica a tensão fasorial na barra 105, em que o AnaHVDC indica que se atinge valores próximos a 8 pu. Quando acontece o curto, em 0,1 s a corrente tende a subir significativamente enquanto a tensão tende a zero. Na sequência, em 0,2 s o curto é extinto e a corrente tende a diminuir. Entretanto, a rede, sendo indutiva, tenta preservar o valor da corrente. O resultado é que a tensão fasorial cresce subitamente atingindo o pico de quase 8 pu. Entretanto, as tensões de fase sofrem aumentos com amplitudes significativamente menores, como pode ser notado na Figura 167. Portanto, pode-se afirmar que o valor da tensão fasorial atingido é resultado do modelo teórico empregado pelo AnaHVDC. A tensão que de fato pode ser medida, que são as tensões de fase, atingem valores dentro da faixa usual.

Capítulo 5 - Conclusão

Na introdução foi exposto o duplo objetivo deste trabalho: o desenvolvimento de um compilador otimizado de CDUs e a sua integração ao AnaHVDC. Isso foi motivado pelo fato da abordagem tradicional usada na solução de CDUs ser interpretada, resultando um significativo gargalo de tempo na solução de CDUs. Desse modo, se desenvolveu uma abordagem compilada que, aliada ao processamento paralelo, permite uma otimização desse processo.

Primeiramente, no comparativo de simulações com controladores isolados do programa desenvolvido utilizando a metodologia proposta com o programa Anatem largamente utilizado nos estudos dinâmicos do setor elétrico, foi possível obter reduções significativas de tempo na solução do arquivo de CDUs do SIN. Processamento paralelo e utilização otimizada da memória cache são também fatores significantes que propiciam ainda maiores ganhos de desempenho. Percebe-se ainda que o ganho de desempenho é tão melhor quanto mais sofisticado é o computador. Esse é um ponto importante. O veloz desenvolvimento tecnológico na área de computação conduz a uma melhoria substancial dos processadores ao longo do tempo. Desse modo, a vantagem verificada no UDCC tende a aumentar ainda mais futuramente. Destaca-se que no computador i9 utilizado para os testes a redução de tempo foi de até 20 vezes. Em termos de confiabilidade, destaca-se que o UDCC se encontra plenamente validado em relação a todos os CDUs da base do SIN de 2016.

Por sua vez, a segunda parte dos resultados demonstra o sucesso da integração do UDCC ao AnaHVDC. O programa AnaHVDC, único no mundo, em corrente desenvolvimento no Cepel, possibilitará uma mudança de paradigma nos estudos dinâmicos envolvendo elos de corrente contínua, por possibilitar a modelagem do sistema completo, com um detalhamento dos modelos dos equipamentos, incluindo os transitórios eletromagnéticos do sistema e as formas de ondas em válvulas de pontes conversoras para identificação de problemas de falha de comutação, prevendo possíveis situações críticas de operação e permitindo evitar blecautes de grandes proporções. Graças ao trabalho desenvolvido nesta dissertação, o programa AnaHVDC passou a ter o recurso de simulações de controladores de alto grau de detalhamento e complexidade.

Os resultados da integração do UDCC ao AnaHVDC foram inicialmente validados comparando-os com o programa PacDyn a partir de pequenas perturbações equilibradas no SIN. Com isso, se confirmou o correto desenvolvimento da comunicação entre os CDUs e

os demais equipamentos do AnaHVDC, o que foi providenciado pelos blocos IMPORT e EXPORT e também dos DEFVALs. Destaca-se também que o UDCC permite a geração da DLL tanto por meio do TCC como por meio do Clang. A vantagem do TCC é a rapidez de compilação, e a do Clang é a grande eficiência do código de máquina gerado. Dessa forma, atende-se ao caso em que o desempenho computacional na solução é o foco principal, a exemplo de CDUs extensos, e também ao caso em que o mais importante é a geração rápida da DLL, como em CDUs curtos. Portanto, demonstra-se que os dois objetivos propostos foram alcançados.

Como discutido no capítulo referente aos resultados, a integração ao AnaHVDC prejudicou substancialmente o tempo de solução dos CDUs, o que já era esperado. Entretanto, duas ressalvas são importantes. A primeira é o fato de que mesmo sem processamento paralelo os ganhos ainda são significativos. Isso pode ser comprovado pelo fato de a solução do caso do SIN com 0 thread a partir do Clang ainda ser capaz de reduzir em até três vezes o tempo de solução. Embora esse valor quantitativo de redução seja para o UDCC operando-se isoladamente, essa vantagem é preservada na integração com o AnaHVDC. Além disso, como discutido, alternativas estão sendo pesquisadas para aumentar o ganho referente ao processamento paralelo. Até o momento, já foi possível uma melhora considerável em comparação à primeira versão do AnaHVDC após a integração com o UDCC. Em síntese, apesar desse revés, a abordagem compilada aplicada ao AnaHVDC, por si só, já representa um ganho de desempenho significativo, e que tende a aumentar com as recentes melhorias no paralelismo do AnaHVDC.

De forma geral, os programas computacionais desenvolvidos funcionaram de forma robusta, eficiente e precisa, assim como foram apresentados resultados que demonstram a aplicação prática dos desenvolvimentos em estudos de simulação de sistemas de potência.

5.1 Trabalhos Futuros

O trabalho mais imediato, e que já se encontra em andamento, é a otimização de desempenho do AnaHVDC, em relação ao processamento paralelo nele empregado. Como descrito, o impacto da criação das *threads* atualmente é muito alto, de modo que o ganho oferecido pelo processamento paralelo é insignificante.

Além disso, como mencionado no Capitulo 4, algumas máquinas do caso PAR2023 não foram consideradas no exemplo simulado, uma vez que elas causavam a divergência do

resultado. A causa é provavelmente algum erro no UDCC, o que também já está sendo investigado.

Após se determinar o melhor modo de realizar o processamento paralelo no contexto do AnaHVDC, solucionando-se o problema referente ao impacto de criação das threads, acredita-se que seja possível avançar para a integração do UDCC ao Anatem. Sendo o Anatem um programa largamente usado, essa seria uma contribuição bastante útil na área.

Por fim, também cabe a implementação do bloco atraso, que atualmente é representado por um bloco LEDLAG.

Referências Bibliográficas

[1] *Encyclopaedia Britannica, Chicago (US), 2020.*

[2] ONS *Mapa do SIN. Brasil, 2020.*

[3] EPE. *Plano Decenal de Expansão de Energia 2029. EPE, Rio de Janeiro, 2019.*

[4] NORTE ENERGIA. *A História de Belo Monte - Cronologia.*

[5] GRAHAM J.F.; FISCHER T.; HOLMGREN P.; SHORE N. L. *The Rio Madeira HVDC System – Design aspects of Bipole 1 and the connector to Acre-Rondônia. Cigré, p. 26-31, 2012.*

[6] BMTE. *Relatório de Impacto Ambiental Belo Monte Transmissora de Nergia SPE S.A. Alto da Boa Vista, 2014.*

[7] OKETCH I. *Commutation Failure Prevention for HVDC. Charlmers University of Technology, Gothenburg (SE), 2016.*

[8] YANG H.; ZEXIANG C.; XIAOHUA C., YU L. *Assessment of commutation failure in HVDC systems considering spatial-temporal discreteness of AC system faults. Clean Energy, v. 6, p. 1055-1065, 2018.*

[9] RAHIMI E.; GOLE A. M.; DAVIES J. B.; FERNANDO I. T.; KENT K. L. *Commutation Failure Analysis in Multi-Infeed HVDC Systems. IEEE Transactions on Power Delivery, V.26(1), P. 378 - 384, 2011.*

[10] DANIEL L. O.; GOMES JR S.; WATANABE E. H. *Utilização de Fasores Dinâmicos para Modelagem de Transitórios. XVII ERIAC-Decimoséptimo Encuentro Regional Iberoamericano de Cigré, Ciudad del Este (PY), 2017.*

[11] GOMES, S.; ALMEIDA, L. P.; LIRIO, F. L.; PARREIRAS, T. J. M. A.; DANIEL, L. O.; AMARAL, T. S.; GODIM, R. A.; ROCHA, T. J. B. *O novo Programa Computacional ANAHVDC para Simulação dos Múltiplos Elos HVDC do SIN considerando Transitórios Eletromecânicos e Eletromagnéticos. XXV SNPTEE - Seminário Nacional de Produção e Transmissão de Energia Elétrica. Belo Horizonte, 2019.*

[12] DANIEL L. O. *Simulador de Transitórios Eletromagnéticos utilizando Fasores Dinâmicos para Análise Não-linear de Redes Elétricas com Equipamentos FACTS. Tese de Doutorado, Universidade Federal do Rio de Janeiro (COPPE), Rio de Janeiro, 2018.*

[13] Cepel. *Anarede - Versão 11.3.2. Rio de Janeiro, 2020.*

[14] Cepel. *Anatem - Versão 11.9.3. Rio de Janeiro, 2020.*

[15] CEPEL. *Curso Básico de Utilização do Programa Anatem. Rio de Janeiro, 2017.*

[16] ROCHA T. J. B; AMARAL T. S.; GOMES JR S.; L. P.; OLIVEIRA ALMEIDA L. A. A. *Simulação eficiente de controladores definidos pelo Usuário utilizando compilação em tempo real. XXV SNPTEE - Seminário Nacional de Produção e Transmissão de Energia Elétrica. Belo Horizonte, 2019.*

[17] KHUNDUR P. *Power System Stability and Control. 1ª ed. Nova Iorque (EUA), McGraw-Hill Professional Publishing, 1994.*

[18] MACHOWSKI J.; BIALEK J. W.; BUMBY J. *Power System Dynamics: Stability and Control. 2ª ed. Hoboken (EUA), John Wiley & Sons LTD, 2008.*

[19] CEPEL. *Anatem - Analysis of Electromechanical Transients - Version 11.4. Rio de Janeiro, 2019.*

[20] CATTAN F. C. *Análise Linear de Oscilações Subsíncronas em Sistemas Elétricos de Potência. Universidade Federal do Rio de Janeiro (COPPE), Rio de Janeiro, 2007.*

[21] IEEE. *Recommended Practice for Excitation System Models for Power System Stability Studies. Nova Iorque (EUA), IEEE, 2016.*

[22] ANEEL. *Procedimentos de Distribuição de Energia Elétrica no Sistema Elétrico Nacional – PRODIST (Módulo 8: Qualidade da Energia Elétrica). Brasília, 2018.*

[23] VIEIRA FILHO X. *Operação de Sistemas de Potência com Controle Automático de Geração. 1ª ed. Rio de Janeiro, Campus, 1984.*

[24] WOODWARD. *Governing Fundamentals - Reference Manual. 1ª ed. Fort Collins (EUA), Woodward, 2004.*

[25] UCTE. *Load-Frequency Control. 1ª ed. UCTE, União Européia, 2004.*

[26] SINGH B.; SAHA R.; CHANDRA A.; AL-HADDAD K. *Static synchronous compensators (STATCOM): a review. IET Power Electronics, v. 2, p. 297-324, 2009.*

[27] TOCCI R.; N. WIDMER; G. MOSS *Digital Systems. 12ª ed. Londres,, Pearson, 2016.*

[28] STOKES J. *Inside the Machine: An Illustrated Introduction to Microprocessors and Computer Architecture. 1ª ed. São Francisco (EUA), No Starch Press, 2006.*

[29] GODBOLT M. *Compiler Explorer. Disponível em: https://godbolt.org/.*

[30] INTEL. *Intel® 64 and IA-32 Architectures. 1ª ed. Santa Clara (US), Intel, 2016.*

[31] CARTER P. *PC Assembly Language. 4ª ed. Oklahoma (US), University of Central Oklahoma, 2004.*

[32] CURCIC M. *Modern Fortran. 1ª ed. Shelter Island (US), Manning Publications, 2020.*

[33] STROUSTRUP B. *The C++ Programming Language. 4 ed. Boston (US), Addison-Wesley Professional, 2013.*

[34] DECYK V. K.; NORTON C. D.; GARDNER H. J. *Why Fortran. Computing in Science & Engineering, v.9, p. 68-71, 2007.*

[35] STALMAN R. M. *Using the GNU Compiler Collection. 1ª ed. Boston (US), GNU Press, 2019.*

[36] BELLARD F. *Tiny C Compiler Reference Documentation. Grenoble (FR), Fabrice Bellard, 2017.*

[37] PEREIRA R.; COUTO M.; RIBEIRO F.; RUA J., CUNHA R.; FERNANDES J. P.; SARAIVA J. *Energy efficiency across programming languages: how do energy, time, and memory relate? The 10th ACM SIGPLAN International Conference, Association for Computing Machinery, 2017.*

[38] Google. *Git repositories on chromium.*

[39] CHEN R. *One Dev Question with Raymond Chen - What Programming Language is Windows Written In? Channel 9, 2019.*

[40] FITZGERALD B.; MOCKUS A.; ZHOU M. *Towards Engineering Free/Libre Open Source Software (FLOSS) Ecosystems for Impact and Sustainability: Communications of NII Shonan Meetings. 1ª ed. Nova Iorque (US), Springer, 2019.*

[41] PETZOLD C. *Code: The Hidden Language of Computer Hardware and Software. 1ª ed. Redmond (EUA), Microsoft Press, 2000.*

[42] Jeff Duntemann *Assembly Language Step-by-Step: Programming with Linux*, 3rd ed. Hoboken (US): John Wiley & Sons, 2011.

[43] JAMES, T. T. STREIB *Guide to Assembly Language: A Concise Introduction*. Nova Iorque (EUA): Springer, 2014.

[44] GONZALEZ T.; DIAZ-HERRERA J.; TUCKER A. *Computing Handbook: Computer Science and Software Engineering, Volume 1. 1ª ed. Londres (UK), Chapman and Hall/CRC, 2014*.

[45] PRATA, S. *C++ Primer Plus. 6ª ed. Boston (US), Addison-Wesley Professional, 2011*.

[46] LITTON W. W. *From Computer to Brain: Foundations of Computational Neuroscience. 1ª ed. Nova Iorque (EUA), Springer, 2002.*

[47] OAOKS S. *Java Performance: The Definitive Guide. 1ª ed. Sebastopol (US), O'Reilly Media, 2014.*

[48] GODBOLT M. *Optimizations in C++ Compilers. Queue, 69-100, 2019.*

[49] BELLARD F. *Tiny C Compiler (TCC) - Documentation. Grenoble (FR), 2018.*

[50] CLEMENTES A. *Computer Organization & Architecture: Themes and Variations. 1ª ed. Boston (US), Cengage Learning, 2014.*

[51] Drepper U. *What Every Programmer Should Know About Memory. 1ª ed. Red Hat, Raleigh (US), 2007.*

[52] JACOB B.; WANG D.; NG S. *Memory Systems: Cache, DRAM, Disk. 1ª ed. Morgan Kaufmann, Burlington (US), 2008.*

[53] WILLIAMS A. *C++ Concurrency in Action: Practical Multithreading. 2ª ed. Nova York (US), Manning Publications, 2017.*

[54] CHAPMAN S. J. *Fundamentos de Máquinas Elétricas. 5ª ed. Rio de Janeiro, 2013.*

[55] LU H.J.; MATZ M.; M., GIRKAR ; HUBI J. *System V Application Binary Interface AMD64 Architecture Processor Supplement (With LP64 and ILP32 Programming Models). 1ª ed. Mountain View (US), INTEL, 2018.*

[56] WOLFRAM. *WolframAlpha. Disponível em: https://www.wolframalpha.com/.*

[57] SHAMPINE L. F. *Numerical Solution of Ordinary Differential Equations. 1ª ed. Londres (UK), Chapman and Hall/CRC, 1994.*

[58] TRIOLA M. F. *Elementary Statistics. 13ª ed. Londres (UK), Pearson, 2017.*

[59] ABELL M. L. L.; BRASELTON J. P.; RAFTER J. A. *Statistics with Mathematica. 1ª ed. Cambridge (US), Academic Press, 1998.*

[60] *ISO/IEC 14882:2017. 5ª ed, 2017.*

[61] TSELIKAS. G. S.; TSELIKAS N. D. *C: From Theory to Practice. 2 ed. Ohio (EUA), CRC Press, 2017.*

[62] ALAM M. A.; SIDDIQUI T.; SEEJA K. R. *Recent Developmnets in Computing and its Applications. 1ª ed. New Delhi (IN), IK International Publishing House, 2009.*

[63] WOLFRAM. *Elementary Function. Wolfram, 2020. Disponível em: https://mathworld.wolfram.com/ElementaryFunction.html*.

[64] Cepel. *PacDyn - Versão 9.9.10. Rio de Janeiro, 2020.*

[65] P. Kundur *Power System Stability and Control. 1ª ed. Nova Iorque (US), McGraw-Hill Education, 1994.*

[66] ALMEIDA, L. P.; LIRIO, F. L.; GOMES JUNIOR, Sergio; LUZ, G. S. *Sistema Benchmark no PSCAD e ATP contendo Elo de Corrente Contínua e Máquinas In: XIII SEPOPE - Symposium Of Specialists In Electric Operational And Expansion Planning, 2014, Foz do Iguaçu.*

Apêndice A – Revisão do Método Trapezoidal

A relação entre a entrada e a saída de um LEDLAG pode ser modelada conforme a Equação (26), que é repetida abaixo.

$$y + T\ddot{y} = u \tag{26}$$

Entretanto, é importante evidenciar que tanto a entrada como a saída são dependentes do tempo, de modo que a Equação (26) pode ser reescrita conforme a Equação (27) de modo a evidenciar essa dependência.

$$y(t) + T\ddot{y}(t) = u(t) \tag{27}$$

Por sua vez, a Equação (28) representa a solução da EDO da Equação (27) [56].

$$y(t) = c_1 e^{-t/T} + e^{-t/T} \underbrace{\int_1^t \frac{e^{\xi/T} u(\xi)}{T} d\xi}_{\text{solução não elementar}} \tag{28}$$

A constante c_1 poderia, em tese, ser determinada a partir da inicialização do bloco. Entretanto, na Equação (28) há uma integral cuja solução é uma função não elementar, ou seja, não possui forma fechada e, portanto, não pode ser escrita em termos de um número finito de funções elementares[39]. Analisando-se a Equação (27) percebe-se que o problema decorre do fato de haver duas variáveis (entrada e saída) dependentes do tempo, porém ambas com equações desconhecidas. Se fosse possível desenvolver a entrada[40] a EDO poderia ter sido resolvida como uma Equação Diferencial Linear de 1ª Ordem[41].

Dessa forma, é necessário recorrer a métodos de integração numérica para se encontrar a solução da Equação (28). No Anatem, no UDCC e no AnaHVDC é usado o Método Trapezoidal, ilustrado na Figura 171.

[39] Funções elementares são aquelas compostas apenas por funções constantes, algébricas, exponenciais e logarítmicas e sob as operações padrões de adição, multiplicação, divisão e extração de raiz [63].
[40] Desenvolver a entrada seria escrevê-la, por exemplo, como $t^2 + 2t$ ao invés da forma genérica x(t).
[41] Caso a Equação (27) fosse $y(t) + T\ddot{y}(t) = t^2 + 2t$ a solução seria $y(t) = c_1 e^{-t/T} + t^2 - 2tT + 2T^2 + 3$. Portanto, demonstra-se que uma solução em forma de uma função elementar pode ser determinada. Como esse não é o caso, recorre-se ao método trapezoidal de integração numérica.

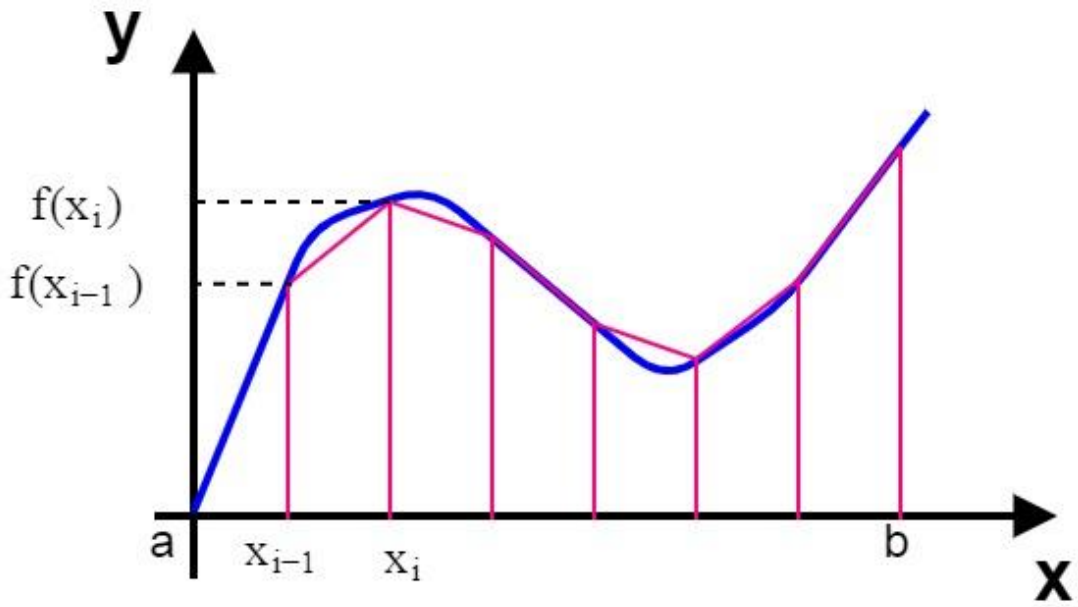


Figura 171 - Método Trapezoidal

Portanto, a integral é substituída pelo somatório da Equação (32) [57].

$$\int_a^b f(x)\,dx = \frac{1}{2}\sum_{k=1}^{N}\left[f(x_{k-1}) + f(x_k)\right]\Delta T_k \tag{29}$$

Em que "N" seja tal que "k" varra todo o intervalo compreendido entre "a" e "b". Por sua vez, ΔT é o intervalo entre x_{i+1} e x_i, podendo assumir diferentes valores dependendo do intervalo considerado.

No caso específico do método trapezoidal aplicado à resolução de blocos dinâmicos de CDU, a complexidade aumenta uma vez que há duas variáveis dependentes do tempo[42], a entrada e a saída, o que difere do caso da Figura 171, em que existe apenas uma variável independente (x) e apenas uma variável dependente (y). Dessa forma, retoma-se a Equação , repetida abaixo.

$$y(t) + T\dot{y}(t) = u(t) \tag{30}$$

$$\dot{y}(t) = \frac{u(t) - y(t)}{T} \tag{31}$$

$$\int_{t-\Delta T}^{t} \dot{y}(t) = \int_{t-\Delta T}^{t} \frac{u(t) - y(t)}{T} \tag{32}$$

[42] Para o caso específico dos controladores, o tempo "t" é a variável independente.

$$y\big|_{t-\Delta T}^{t} = \frac{1}{T}\left(\underbrace{\int_{t-\Delta T}^{t} u(t)\,dt}_{\text{solução não elementar}} - \underbrace{\int_{t-\Delta T}^{t} y(t)\,dt}_{\text{solução não elementar}} \right) \tag{33}$$

Como esperado, as integrais na Equação (33) não apresentam solução em termos de funções elementares. Portanto, cada integral é desenvolvida pelo método trapezoidal, o que resulta na Equação (34).

$$y(t) - y(t-\Delta T) = \frac{1}{T}\left(\frac{1}{2}\left[u(t)+u(t-\Delta t)\right]\times\Delta t - \frac{1}{2}\left[y(t)+y(t-\Delta t)\right]\times\Delta t \right) \tag{34}$$

$$T\,\frac{y(t)-y(t-\Delta T)}{\Delta T} = \frac{u(t)+u(t-\Delta t)}{2} - \frac{y(t)+y(t-\Delta t)}{2} \tag{35}$$

$$\frac{y(t)+y(t-\Delta t)}{2} + T\,\frac{y(t)-y(t-\Delta t)}{\Delta t} = \frac{u(t)+u(t-\Delta t)}{2} \tag{36}$$

Comparando-se a Equação (33) com a Equação (36) percebe-se uma regra prática. As variáveis (como a entrada u e a saída y) podem ser substituídas conforme a Equação (37) em que “MT” indica a transformação pela aplicação do Método Trapezoidal.

$$u \overset{MT}{\rightarrow} \frac{u(t)+u(t-\Delta t)}{2} \tag{37}$$

Por sua vez, as derivadas (como $\dot{y}$) podem ser substituídas conforme a Equação (38)

$$\dot{y} \overset{MT}{\rightarrow} \frac{y(t)-y(t-\Delta t)}{\Delta t} \tag{38}$$

Com isso, demonstra-se a passagem da Equação (13) para a Equação (14) no tópico 3.1.

Apêndice B – Ferramentas Estatísticas

O desvio padrão populacional é definido conforme a Equação (39) [58][43].

$$\sigma = \frac{\sum_{i=1}^{N}\left(x_i - \overline{x}\right)}{N} \tag{39}$$

Em que "x_i" é o elemento i, $\overline{x}$ é a média aritmética dos dados e "N" é o número de dados.

O desvio padrão fornece uma medida para a dispersão dos dados. Isso é importante, uma vez que, para o caso desta Dissertação, os dados precisam apresentar baixa dispersão. Se durante uma das simulações os dados apresentarem alta dispersão isso é um indicativo de que fatores externos estão influenciando a execução da solução, o que não pode acontecer.

Entretanto, é possível notar que o desvio padrão não é uma ferramenta boa para comparativo de dispersão entre amostras com médias diferentes. Nesses casos é usual se usar o coeficiente de variação, definido como o desvio padrão dividido pela média aritmética da população conforme a Equação (40) [58].

$$CV = \frac{\sigma}{\overline{x}} \tag{40}$$

[43] O desvio padrão populacional pressupõe que foi possível obter acesso e considerar todos os elementos da população. Este é exatamente o caso em questão. Isso se conctrapõe ao desvio padrão amostral, que considera que uma parcela da população não está sendo considerada nos cálculos [59].

Apêndice C – Lista de Tempos de Solução (Antes do uso otimizado de Cache)

Neste apêndice são apresentados os tempos de solução do caso adaptado do SIN para cada simulação realizada antes da alteração de código visando ao uso otimizado da memoria cache expostas no tópico 2.7. O comparativo se deu entre o Anatem, o UDCC via TCC e o UDCC via Clang[44]. As simulações foram executadas nos quatro computadores citados no Capítulo 4.

C.1 Anatem

A Tabela 9, a Tabela 10, a Tabela 11 e a Tabela 12 mostram o tempo de solução do SIN pelo Anatem em cada um dos quatro computadores usados para os testes. Nas tabelas citadas consta o tempo de cada simulação realizada, que totalizam cinco, além da média aritmética, do desvio padrão populacional (DP) e o do coeficiente de variação (CV) desses valores.

Tabela 9 – Tempos de Solução do SIN pelo Anatem (Computador i5IV)

Tempo de cada simulação - Anatem [s]				
17,81	18,87	18,19	17,34	17,05

Média	DP	CV
17,85	0,642	3,59%

Tabela 10 – Tempos de Solução do SIN pelo Anatem (Computador i7VIII)

Tempo de cada Simulação - Anatem [s]				
14,26	13,47	13,50	13,68	14,00

Média	DP	CV
13,78	0,304	2,21%

Tabela 11 – Tempos de Solução do SIN pelo Anatem (Computador i7XII)

Tempo de cada Simulação - Anatem [s]				
13,27	13,2	13,24	13,19	13,25

Média	DP	CV
13,23	0,030	0,23%

[44] UDCC via Clang significa que o CDU do SIN foi lido pelo UDCC e o código C resultante foi compilado pelo Clang e gerada uma DLL. Posteriormente, o UDCC realizou a solução a partir da DLL gerada pelo Clang.

Tabela 12 – Tempos de Solução do SIN pelo Anatem (Computador i9XVI)

Tempo de cada Simulação - Anatem [s]				
10,38	10,29	10,19	10,3	10,27

Média	DP	CV
10,29	0,061	0,59%

Portanto, para os três casos, o desvio padrão populacional e o coeficiente de variação são baixos, o que é um indicativo da baixa influência externa, de modo que a média de fato traduz um tempo fiel de solução[45].

C.2 TCC

Neste caso, foi realizada a simulação do caso do SIN por meio do UCCC via TCC. Para este caso, as simulações foram executadas cinco vezes para cada número de threads consideradas. O número de threads consideradas se encontra na primeira coluna (Thd). A Tabela 13, a Tabela 14, a Tabela 15 e a Tabela 16 condensam os resultados obtidos.

Tabela 13 – Tempos de Solução do SIN pelo UDCC via TCC (Computador i5IV)

Thd	Tempo de cada simulação [s]					Média	DP	CV
0	36,39	36,28	36,27	36,57	36,39	36,38	0,11	0,30%
1	36,55	36,44	36,42	36,62	37,08	36,62	0,24	0,66%
2	23,67	23,64	23,65	24,40	23,64	23,80	0,30	1,27%
4	19,28	19,22	19,22	19,19	19,19	19,22	0,03	0,17%
6	18,94	18,97	22,84	18,89	20,31	19,99	1,52	7,61%
8	18,77	18,97	18,81	19,34	18,70	18,92	0,23	1,22%
10	18,35	18,41	18,41	18,35	18,57	18,42	0,08	0,45%
12	18,33	18,33	18,29	18,41	18,25	18,32	0,05	0,29%
16	18,75	18,72	18,76	18,54	18,51	18,66	0,11	0,59%
20	18,48	18,44	18,44	18,40	18,46	18,44	0,03	0,16%
24	18,39	18,38	18,50	18,49	18,40	18,43	0,05	0,29%
32	17,71	17,74	17,89	17,77	17,51	17,72	0,12	0,69%
40	16,60	16,81	16,82	16,76	16,77	16,75	0,08	0,48%
48	16,26	16,06	16,54	16,56	16,43	16,37	0,19	1,14%
64	16,27	16,52	15,95	16,02	16,08	16,17	0,21	1,27%
96	17,64	17,84	17,76	17,80	17,71	17,75	0,07	0,39%
100	18,21	18,25	18,25	18,31	18,37	18,28	0,06	0,31%

[45] A execução concomitante ao UDCC de um programa com uso constante da CPU/memória também resultaria em dados com baixo coeficiente de variação. Entretanto, isso seria um cenário bastante não usual.

Tabela 14 – Tempos de Solução do SIN pelo UDCC via TCC (Computador i7VIII)

Thd	Tempo de cada simulação [s]					Média	DP	CV
0	29,15	29,07	29,03	29,35	29,08	29,13	0,11	0,39%
1	29,51	29,48	29,19	29,09	28,90	29,24	0,23	0,79%
2	18,34	18,16	18,30	18,31	18,30	18,28	0,06	0,34%
4	14,32	14,32	14,37	14,71	14,36	14,42	0,15	1,02%
6	13,69	13,70	13,66	13,70	13,65	13,68	0,02	0,15%
8	13,97	13,95	14,21	14,24	13,94	14,06	0,13	0,94%
10	13,45	13,45	13,48	13,47	13,50	13,47	0,02	0,14%
12	13,73	13,11	13,1468	13,22	13,30	13,30	0,22	1,67%
16	13,54	13,52	13,38	13,45	13,42	13,46	0,06	0,45%
20	12,85	12,93	12,79	12,84	12,96	12,87	0,06	0,48%
24	11,76	11,58	11,52	11,56	11,43	11,57	0,11	0,93%
32	10,39	10,41	10,51	10,37	10,62	10,46	0,09	0,87%
40	9,00	9,05	9,09	9,16	9,25	9,11	0,09	0,98%
48	8,32	8,39	8,29	8,41	8,32	8,34	0,04	0,54%
64	7,53	7,71	7,67	7,58	7,49	7,59	0,08	1,09%
96	7,11	7,14	7,29	7,22	7,15	7,18	0,07	0,93%
100	7,24	7,22	7,25	7,27	7,34	7,27	0,04	0,60%

Tabela 15 – Tempos de Solução do SIN pelo UDCC via TCC (Computador i7XII)

Thd	Tempo de cada simulação [s]					Média	DP	CV
0	24,24	24,26	24,29	24,32	24,38	24,30	0,05	0,20%
1	24,66	24,37	24,40	25,57	24,29	24,66	0,47	1,92%
2	13,61	13,61	13,55	13,29	13,45	13,50	0,12	0,91%
4	8,42	8,53	8,42	8,57	8,87	8,56	0,17	1,93%
6	7,83	8,13	7,71	7,80	7,79	7,85	0,14	1,84%
8	7,80	7,82	7,75	7,98	7,70	7,81	0,10	1,22%
10	7,90	7,90	7,91	7,98	7,96	7,93	0,03	0,43%
12	8,41	8,28	8,31	8,28	8,31	8,32	0,05	0,59%
16	7,68	7,42	7,61	7,60	7,90	7,64	0,16	2,05%
20	5,96	6,08	6,08	5,92	5,62	5,93	0,17	2,87%
24	4,97	4,67	4,65	4,62	4,50	4,68	0,16	3,33%
32	4,05	4,07	4,26	4,18	4,17	4,15	0,08	1,86%
40	3,49	3,52	3,47	3,48	3,44	3,48	0,03	0,75%
48	3,07	3,08	3,05	3,05	3,05	3,06	0,02	0,51%
64	2,67	2,73	2,71	2,71	2,72	2,71	0,02	0,67%
96	3,40	3,33	3,38	3,40	3,36	3,37	0,03	0,75%
100	3,53	3,55	3,50	3,49	3,46	3,51	0,03	0,94%

Tabela 16 – Tempos de Solução do SIN pelo UDCC via TCC (Computador i9XVI)

Thd	Tempo de cada simulação [s]					Média	DP	CV
0	17,89	17,94	18,03	17,93	17,89	17,94	0,03	0,17%
1	18,10	18,11	18,09	18,05	18,07	18,08	0,02	0,13%
2	9,77	9,89	9,87	9,92	9,78	9,85	0,06	0,61%
4	6,29	6,23	6,26	6,20	6,26	6,25	0,03	0,50%
6	5,74	5,70	5,89	5,80	5,81	5,79	0,06	1,11%
8	4,89	4,97	4,83	4,74	5,03	4,89	0,10	2,13%
10	4,00	4,03	4,20	4,19	4,06	4,10	0,08	2,04%
12	3,99	3,84	3,85	3,91	3,79	3,88	0,07	1,82%
16	2,79	2,91	2,82	2,81	2,92	2,85	0,06	1,97%
20	3,23	3,16	3,38	3,19	3,16	3,22	0,08	2,57%
24	2,88	2,97	2,87	2,80	2,95	2,89	0,06	2,10%
32	4,44	4,38	4,36	4,18	4,32	4,34	0,09	0,02
40	2,41	2,41	2,44	2,46	2,42	2,43	0,02	0,86%
48	2,14	2,13	2,09	2,14	2,08	2,11	0,03	1,26%
64	1,84	1,81	1,81	1,82	1,84	1,82	0,01	0,68%
96	2,13	2,14	2,13	2,13	2,14	2,14	0,01	0,24%
100	2,20	2,16	2,18	2,20	2,24	2,20	0,03	1,17%

C.3 Clang

O SIN também foi simulado por meio do UDCC via Clang. Para este caso, a semelhança do TCC, as simulações foram executadas cinco vezes para cada número de threads consideradas, que foram as mesmas do caso do TCC. A Tabela 17, a Tabela 18, a Tabela 19 e a Tabela 20 condensam os resultados obtidos.

Tabela 17 – Tempos de Solução do SIN pelo UDCC via Clang (Computador i5IV)

Thd	Tempo de cada simulação [s]					Média	DP	CV
0	19,39	19,53	19,44	19,49	19,63	19,49	0,08	0,42%
1	19,56	19,35	19,34	19,52	19,41	19,44	0,09	0,47%
2	14,23	14,26	14,05	14,10	14,14	14,15	0,08	0,56%
4	11,04	11,15	10,99	10,97	11,05	11,04	0,06	0,56%
6	10,68	11,82	11,60	11,01	11,24	11,27	0,41	3,61%
8	10,94	10,52	10,29	10,39	10,31	10,49	0,24	2,28%
10	9,87	9,96	9,94	10,04	9,90	9,94	0,06	0,57%
12	8,58	8,53	8,98	8,80	9,00	8,78	0,19	2,21%
16	7,68	7,81	7,89	7,84	7,86	7,82	0,07	0,91%
20	7,58	7,69	7,35	7,53	7,33	7,50	0,14	1,83%
24	7,31	7,19	7,33	7,02	7,47	7,26	0,15	2,09%
32	7,28	7,35	7,37	7,39	7,45	7,37	0,06	0,75%
40	7,95	7,98	8,00	7,97	7,99	7,98	0,02	0,22%
48	8,69	8,64	8,68	8,62	8,78	8,68	0,05	0,63%
64	10,62	10,48	10,54	10,59	10,62	10,57	0,05	0,50%
96	14,17	14,15	14,29	14,11	14,28	14,20	0,07	0,52%
100	14,69	14,67	14,82	14,62	14,61	14,68	0,07	0,51%

Tabela 18 – Tempos de Solução do SIN pelo UDCC via Clang (Computador i7VIII)

Thd	Tempo de cada simulação [s]					Média	DP	CV
0	5,78	6,28	5,92	6,07	5,81	5,97	0,18	3,10%
1	5,89	5,83	5,79	5,83	6,30	5,93	0,19	3,18%
2	3,70	3,71	3,69	3,71	3,62	3,69	0,03	0,93%
4	3,07	2,94	3,00	2,96	3,17	3,03	0,09	2,83%
6	3,28	3,16	3,15	3,15	3,10	3,17	0,06	1,90%
8	3,59	3,58	3,57	3,61	3,60	3,59	0,02	0,44%
10	4,01	4,00	3,98	4,08	4,02	4,02	0,03	0,84%
12	3,99	3,98	3,92	4,06	3,95	3,98	0,05	1,14%
16	4,10	3,80	4,08	3,76	3,75	3,90	0,16	4,05%
20	3,85	3,81	3,82	3,79	3,77	3,81	0,03	0,75%
24	3,50	3,57	3,65	3,51	3,50	3,55	0,06	1,57%
32	3,28	3,25	3,36	3,39	3,24	3,30	0,06	1,81%
40	3,17	3,33	3,13	3,28	3,16	3,21	0,08	2,43%
48	3,25	3,26	3,31	3,25	3,31	3,28	0,03	0,88%
64	3,72	3,71	3,68	3,69	3,74	3,71	0,02	0,57%
96	5,17	5,12	5,13	5,12	5,13	5,13	0,02	0,40%
100	5,32	5,41	5,40	5,39	5,26	5,35	0,06	1,08%

Tabela 19 – Tempos de Solução do SIN pelo UDCC via Clang (Computador i7XII)

Thd	Tempo de cada simulação [s]					Média	DP	CV
0	4,86	4,83	4,83	4,84	4,78	4,83	0,03	0,59%
1	4,84	4,85	4,83	4,86	4,83	4,84	0,01	0,24%
2	2,58	2,58	2,58	2,61	2,65	2,60	0,03	1,02%
4	1,76	1,80	1,79	1,72	1,78	1,77	0,03	1,67%
6	1,42	1,38	1,40	1,40	1,43	1,41	0,02	1,14%
8	1,23	1,24	1,22	1,26	1,23	1,24	0,01	1,15%
10	1,10	1,13	1,08	1,13	1,10	1,11	0,02	1,73%
12	1,12	1,13	1,11	1,10	1,12	1,12	0,01	0,97%
16	1,20	1,23	1,22	1,20	1,25	1,22	0,02	1,58%
20	1,22	1,22	1,21	1,17	1,20	1,20	0,02	1,70%
24	1,19	1,18	1,18	1,19	1,20	1,19	0,01	0,75%
32	1,27	1,28	1,29	1,26	1,30	1,28	0,01	1,15%
40	1,42	1,44	1,55	1,41	1,41	1,45	0,05	3,74%
48	1,61	1,57	1,60	1,67	1,58	1,61	0,04	2,22%
64	2,00	2,09	1,98	2,00	2,00	2,02	0,04	1,95%
96	2,84	2,88	2,90	2,86	2,86	2,87	0,02	0,70%
100	3,01	3,05	2,99	3,00	2,99	3,01	0,02	0,72%

Tabela 20 – Tempos de Solução do SIN pelo UDCC via Clang (Computador i9XVI)

Thd	Tempo de cada simulação [s]					Média	DP	CV
0	4,10	4,14	4,11	4,13	4,12	4,12	0,01	0,33%
1	4,14	4,12	4,15	4,14	4,12	4,13	0,01	0,34%
2	2,35	2,40	2,37	2,35	2,37	2,37	0,02	0,74%
4	1,61	1,63	1,61	1,64	1,60	1,62	0,01	0,79%
6	1,31	1,30	1,38	1,32	1,39	1,34	0,04	2,80%
8	1,13	1,12	1,08	1,10	1,07	1,10	0,02	1,93%
10	0,93	0,91	0,94	0,89	0,94	0,92	0,02	2,00%
12	0,86	0,89	0,86	0,86	0,85	0,87	0,01	1,53%
16	0,83	0,84	0,80	0,81	0,84	0,82	0,01	1,57%
20	0,87	0,88	0,86	0,85	0,88	0,87	0,01	1,44%
24	0,85	0,84	0,84	0,85	0,83	0,84	0,01	0,67%
32	0,93	0,95	0,88	0,94	0,93	0,92	0,03	2,77%
40	1,04	1,02	1,05	0,99	0,97	1,01	0,03	3,06%
48	1,14	1,14	1,15	1,15	1,04	1,12	0,04	3,67%
64	1,38	1,36	1,35	1,28	1,36	1,35	0,04	2,63%
96	1,87	1,88	1,79	1,89	1,89	1,87	0,04	2,01%
100	1,96	1,94	1,93	1,91	1,93	1,93	0,02	0,83%

Apêndice D – Lista de Tempos de Solução (Após otimizações de Cache)

Neste apêndice são apresentados os tempos de solução do caso adaptado do SIN para cada simulação realizada após serem implementadas as otimizações de cache expostas no tópico 2.7. O comparativo se deu entre o Anatem, o UDCC via TCC e o UDCC via Clang. As simulações foram executadas nos quatro computadores citados no Capítulo 4. Uma vez que os tempos de simulação não foram afetados, eles não foram medidos novamente.

Conforme mencionado anteriormente, entre o espaço de tempo da simulação sem otimização de cache e aquela com otimização, houve problema de superaquecimento no computador i7XII. Com isso, o turbo boost precisou ser desligado. A consequência é que o desempenho do i7XII se revelou pior do que o i7VIII. Por isso, optou-se por eliminar o computador i7XII deste comparativo.

D.1 TCC

Neste caso, foi realizada a simulação do caso do SIN por meio do UCCC via TCC. Para este caso, as simulações foram executadas cinco vezes para cada número de threads consideradas. O número de threads considerado se encontra na primeira coluna (Thd). A Tabela 21, a Tabela 22 e a Tabela 23 condensam os resultados obtidos.

Tabela 21 – Tempos de Solução do SIN pelo UDCC via TCC (Computador i5IV) (OC)[46]

Thd	Tempo de cada simulação [s]					Média	DP	CV
0	19,60	19,23	19,04	19,45	19,60	19,38	0,22	1,13%
1	19,51	19,04	18,98	21,64	19,24	19,68	1,00	5,06%
2	13,44	13,00	12,75	12,63	12,98	12,96	0,28	2,14%
4	10,24	10,57	10,13	10,20	10,57	10,34	0,19	1,83%
6	9,80	9,79	9,74	9,70	9,76	9,76	0,04	0,37%
8	9,34	3,41	9,48	9,40	9,35	8,20	2,39	29,20%
10	9,23	9,15	9,28	9,31	9,21	9,24	0,06	0,60%
12	9,22	9,07	8,99	9,16	9,03	9,09	0,08	0,93%
16	8,62	8,57	8,68	8,71	8,65	8,65	0,05	0,56%
20	8,36	8,30	8,29	8,23	8,23	8,28	0,05	0,59%
24	8,30	8,99	8,07	8,18	8,17	8,34	0,33	3,98%
32	8,44	8,40	8,42	8,31	8,27	8,37	0,07	0,79%
40	8,85	8,79	8,87	8,80	8,82	8,83	0,03	0,34%
48	9,54	9,42	9,39	9,43	9,44	9,44	0,05	0,54%
64	9,45	9,35	9,44	9,40	9,38	9,40	0,04	0,40%
96	10,69	10,59	10,78	10,72	10,63	10,68	0,07	0,62%
100	11,67	11,58	11,77	11,54	11,61	11,63	0,08	0,69%

Tabela 22 – Tempos de Solução do SIN pelo UDCC via TCC (Computador i7VIII) (OC)

Thd	Tempo de cada simulação [s]					Média	DP	CV
0	14,49	13,95	14,28	14,35	13,29	14,07	0,43	3,05%
1	13,86	13,12	13,26	13,24	13,11	13,32	0,28	2,09%
2	7,61	8,35	7,59	7,74	7,73	7,80	0,28	3,58%
4	5,36	5,29	5,32	5,32	5,29	5,32	0,03	0,51%
6	5,16	5,07	5,65	5,16	5,02	5,21	0,23	4,33%
8	5,48	5,47	5,54	5,46	5,41	5,47	0,04	0,76%
10	5,06	5,01	5,00	5,09	5,02	5,04	0,03	0,67%
12	4,69	4,69	4,72	4,72	4,61	4,69	0,04	0,86%
16	4,28	4,64	4,31	4,34	4,30	4,37	0,13	3,07%
20	4,00	4,05	3,94	3,94	3,92	3,97	0,05	1,21%
24	3,85	3,70	3,75	3,68	3,71	3,74	0,06	1,62%
32	3,62	3,65	3,60	3,64	3,61	3,62	0,02	0,51%
40	3,65	3,64	3,69	3,65	3,68	3,66	0,02	0,53%
48	3,90	3,70	3,79	3,84	3,76	3,80	0,07	1,80%
64	4,35	4,28	4,28	4,30	4,32	4,31	0,03	0,62%
96	5,66	6,24	5,66	5,64	5,63	5,77	0,24	4,12%
100	5,84	5,86	5,86	5,89	5,88	5,87	0,02	0,30%

[46] Para deferenciar as tabelas deste Apêndice em relação às do Apêndice C escreve-se (OC) indicando que os resultados das tabelas deste Apêndice se referem aos tempos medidos após as otimizações de cache (OC).

Tabela 23 – Tempos de Solução do SIN pelo UDCC via TCC (Computador i9XVI) (OC)

Thd	Tempo de cada simulação [s]					Média	DP	CV
0	8,51	8,44	8,50	8,39	8,44	8,46	0,04	0,52%
1	8,47	8,49	8,53	8,60	8,27	8,47	0,11	1,30%
2	4,27	4,30	4,35	4,27	4,23	4,28	0,04	0,93%
4	2,21	2,24	2,25	2,27	2,24	2,24	0,02	0,86%
6	1,49	1,45	1,54	1,45	1,48	1,48	0,03	2,23%
8	1,26	1,27	1,29	1,38	1,45	1,33	0,07	5,53%
10	1,36	1,12	1,29	1,15	1,15	1,21	0,09	7,74%
12	1,09	1,09	1,11	1,06	1,07	1,08	0,02	1,61%
16	1,29	1,17	1,17	1,15	1,18	1,19	0,05	4,19%
20	1,12	1,11	1,10	1,07	1,10	1,10	0,02	1,52%
24	1,04	1,06	1,05	1,09	1,06	1,06	0,02	1,58%
32	1,09	1,09	1,09	1,12	1,06	1,09	0,02	1,74%
40	1,20	1,16	1,17	1,16	1,19	1,18	0,02	1,38%
48	1,22	1,32	1,23	1,23	1,31	1,26	0,04	3,45%
64	1,53	1,51	1,55	1,51	1,47	1,51	0,03	1,75%
96	2,03	2,01	1,98	2,03	2,04	2,02	0,02	1,06%
100	2,07	2,04	2,04	2,03	2,07	2,05	0,02	0,80%

D.2 Clang

O SIN também foi simulado por meio do UDCC via Clang. Para este caso, a semelhança do TCC, as simulações foram executadas cinco vezes para cada número de threads consideradas, que foram as mesmas do caso do TCC. A Tabela 17, a Tabela 18, a Tabela 19 e a Tabela 20 condensam os resultados obtidos.

Tabela 24 – Tempos de Solução do SIN pelo UDCC via Clang (Computador i5IV) (OC)

Thd	Tempo de cada simulação [s]					Média	DP	CV
0	6,26	6,29	6,16	6,19	6,13	6,21	0,06	0,97%
1	6,14	6,25	6,16	6,16	6,13	6,17	0,04	0,69%
2	3,91	3,93	3,96	3,99	3,96	3,95	0,03	0,70%
4	3,40	3,40	3,39	3,39	3,41	3,40	0,01	0,22%
6	3,42	3,58	3,36	3,47	3,38	3,44	0,08	2,28%
8	3,34	3,33	3,38	3,37	3,40	3,36	0,03	0,77%
10	3,28	3,38	3,32	3,33	3,30	3,32	0,03	1,01%
12	3,44	3,46	3,44	3,44	3,37	3,43	0,03	0,90%
16	3,65	3,67	3,65	3,64	3,64	3,65	0,01	0,30%
20	4,02	4,04	4,03	4,01	4,01	4,02	0,01	0,29%
24	4,47	4,42	4,43	4,43	4,44	4,44	0,02	0,39%
32	5,35	5,32	5,33	5,34	5,32	5,33	0,01	0,22%
40	6,70	6,24	6,22	6,21	6,23	6,23	0,02	0,33%
48	7,14	7,12	7,12	7,11	7,20	7,14	0,03	0,46%
64	9,09	9,05	9,05	9,05	9,07	9,06	0,02	0,18%
96	13,04	13,22	13,30	13,08	13,12	13,15	0,09	0,72%
100	15,23	15,40	15,34	15,48	15,24	15,34	0,10	0,62%

Tabela 25 – Tempos de Solução do SIN pelo UDCC via Clang (Computador i7VIII) (OC)

Thd	Tempo de cada simulação [s]					Média	DP	CV
0	4,36	4,45	4,35	4,36	4,35	4,37	0,04	0,87%
1	4,41	4,78	4,73	5,21	5,29	4,88	0,33	6,67%
2	2,50	2,50	2,46	2,46	2,50	2,48	0,02	0,79%
4	1,41	1,46	1,40	1,49	1,47	1,45	0,03	2,42%
6	1,27	1,29	1,30	1,26	1,23	1,27	0,02	1,93%
8	1,44	1,23	1,20	1,22	1,26	1,27	0,09	6,86%
10	1,32	1,26	1,46	1,47	1,48	1,40	0,09	6,47%
12	1,56	1,48	1,34	1,35	1,34	1,41	0,09	6,38%
16	1,42	1,42	1,42	1,42	1,40	1,42	0,01	0,56%
20	1,56	1,55	1,58	1,56	1,55	1,56	0,01	0,70%
24	1,70	1,69	1,71	1,69	1,70	1,70	0,01	0,44%
32	2,35	2,45	2,38	2,43	2,40	2,40	0,04	1,48%
40	2,42	2,40	2,40	2,77	2,70	2,54	0,16	6,40%
48	3,11	3,00	2,95	3,08	3,07	3,04	0,06	1,92%
64	3,91	3,58	3,51	3,60	3,60	3,64	0,14	3,82%
96	5,15	5,15	5,08	5,23	5,14	5,15	0,05	0,93%
100	5,36	5,33	5,36	5,34	5,33	5,34	0,01	0,25%

Tabela 26 – Tempos de Solução do SIN pelo UDCC via Clang (Computador i9XVI) (OC)

Thd	Tempo de cada simulação [s]					Média	DP	CV
0	3,22	3,26	3,23	3,25	3,24	3,24	0,01	0,44%
1	3,23	3,25	3,23	3,23	3,23	3,23	0,01	0,25%
2	2,35	2,40	2,37	2,35	2,37	2,37	0,02	0,74%
4	1,03	0,97	0,95	0,97	0,97	0,98	0,03	2,77%
6	0,71	0,66	0,70	0,74	0,75	0,71	0,03	4,48%
8	0,58	0,59	0,65	0,61	0,61	0,61	0,02	3,95%
10	0,59	0,53	0,54	0,56	0,56	0,56	0,02	3,70%
12	0,54	0,53	0,53	0,55	0,55	0,54	0,01	1,66%
16	0,51	0,61	0,51	0,51	0,50	0,53	0,04	7,80%
20	0,58	0,58	0,58	0,56	0,58	0,58	0,01	1,39%
24	0,63	0,63	0,62	0,67	0,62	0,63	0,02	2,93%
32	0,81	0,82	0,83	0,81	0,73	0,80	0,04	4,47%
40	0,85	0,91	0,92	0,88	0,84	0,88	0,03	3,59%
48	0,98	1,06	1,04	1,07	1,08	1,05	0,04	3,40%
64	1,33	1,24	1,30	1,29	1,30	1,29	0,03	2,26%
96	1,83	1,85	1,82	1,81	1,85	1,83	0,02	0,87%
100	1,83	1,87	1,90	1,83	1,88	1,86	0,03	1,50%

Apêndice E – Caso BenchCA

E.1 Arquivo Anarede

A seguir é apresentado o arquivo de dados de fluxo de potência do caso em formato cartão, com os dados da topologia da rede elétrica e do ponto de operação considerado. Este arquivo foi utilizado como dado de entrada para convergência da solução de fluxo de potência e convertido para o formato binário BenchCA.sav, utilizado pelo AnaHVDC, pelo processo de gravação de caso convergido.

```
TITU
BenchCA

DOPC IMPR
(Op) E (Op) E (Op) E (Op) E (Op) E (Op) E (Op) E (Op) E (Op) E (Op) E
QLIM L NEWT L RCVG L RMON L MFCT L
99999

DCTE
(Mn) ( Val) (Mn) ( Val) (Mn) ( Val) (Mn) ( Val) (Mn) ( Val) (Mn) ( Val)
TEPA  .1E-8 TEPR  .1E-8 TLVC  .1E-7
99999

DBAR
(Num)OETGb(    nome    )Gl( V)( A)( Pg)( Qg)( Qn)( Qm)(Bc  )( Pl)( Ql)( Sh)Are(Vf)
    1 L2 NBarra1       51045  0.          -999999999                        11000
    2 L1 NBarra2       51045    1600.     -999999999                        11000
    3 L2 NBarra3       51045-36           -999999999                        21000
    4 L1 NBarra4       51045    3000.     -999999999                        21000
    5 L  FBarra5       2                                                    11000
    6 L  FBarra6       2                                  100.  50. 300.    11000
    7 L  FBarra7       2                                                    11000
    8 L  FBarra8       2                                            550.    21000
    9 L  FBarra9       2                                 5000. 200. 150.    21000
   10 L  FBarra10      2                                 2000.  50.         21000
   11 L  FBarra11      2                                 2500.  50.         21000
   12 L  FBarra12      2                                 3500. 100. 330.    21000
   13 L  FBarra13      2                                                    21000
   14 L1 NBarra14      51045    3000.     -999999999                        21000
99999
```

```
DLIN
(De )d O d(Pa )NcEP  ( R% )( X% )(Mvar)(Tap)(Tmn)(Tmx)(Phs)(Bc  )(Cn)(Ce)Ns(Cq)(1)
    5         1 1         .55556         1.
    6         2 1         .55556         1.
   11         3 1         .18519         1.
   10         4 1         .27778         1.
   13        14 1         .27778         1.
    5         6 1   .04545.6660561.397
    6         7 1   .090541.3294122.92
    6         7 2   .090541.3294122.92
    6         7 3   .090541.3294122.92
    7         8 1   .259843.9020372.85
    7         8 2   .259843.9020372.85
    7         8 3   .259843.9020372.85
    8         9 1   .134881.9873184.70
    8         9 2   .134881.9873184.70
    8        12 1   .04545.6660561.397
    9        10 1   .04545.6660561.397
    9        11 1   .134881.9873184.70
   12        13 1   .04545.6660561.397
99999

DSHL
(De ) O  (Pa )Nc (Shde)(Shpa) ED EP
    7         8 1  -150. -150.  L  L
    7         8 2  -150. -150.  L  L
    7         8 3  -150. -150.  L  L
99999

DGER
(No ) O (Pmn ) (Pmx ) ( Fp) (FpR) (FPn) (Fa) (Fr) (Ag) ( Xq) (Sno)
    1       0. 99999.    0.    1.
    2       0. 99999.    0.    1.
99999

DGLT
(G (Vmn) (Vmx) (Vmne (Vmxe
 5    .4   1.9    .4   1.9
 2   .95  1.10    .9  1.10
99999

DARE
(Ar    (Xchg)     (      Identificacao da area       ) (Xmin) (Xmax)
  1        0.     *  AREA 1                          *
  2        0.     *  AREA 2                          *
99999

DGBT
(G ( kV)
 F  500.
 N   20.
99999

exlf newt qlim crem ctap

FIM
```

Figura 172 - Arquivo BenchCA.pwf

E.2 Arquivo Anatem

Destaca-se que o arquivo da Figura 174 se aplica tanto ao curto fase-terra bem como ao curto trifásico aplicado ao BenchCA. A única diferença é a inserção de uma reatância de 2% no caso do curto fase-terra para produzir uma queda de tensão para um valor entre 0,6 e

0,7 pu na sequência positiva, conforme prática dos estudos de simulação do setor elétrico. No caso do curto equilibrado não há reatância, considerando-se franco. Transcreve-se essa diferença na Figura 173 para maior clareza (terceira linha).

```
DEVT
(Tp)    ( Tempo)(  El  ) ( Pa) Nc  ( Ex) ( % ) (ABS ) Gr Und
APCL      0.20     7      8    1    7     0.
2.0
```

Figura 173 – Reatância do Caso Trifásico

```
(=========================================================================
=
( TITULO DO CASO
(=========================================================================
=
TITU
** Sistema Teste SCR Inv 3.1 (repres. bipolar) **
(
(=========================================================================
=
( ASSOCIACAO DE UNIDADES LOGICAS
(=========================================================================
=
( ---- arquivo Historico do fluxo de potencia ----
ULOG
2
BenchCA.SAV
(
( ---- arquivo de saida ----
ULOG
4
BenchCA.out
(
( ---- arquivo de plotagem ----
ULOG
8
BenchCA.plt
(
( ---- arquivo de plotagem ----
ULOG
9
BenchCA.log
(
(=========================================================================
( DADOS DE PADRAO PARA OPCOES DE EXECUCAO
(=========================================================================
DOPC IMPR CONT FILE
  (Op)    E  (Op)   E  (Op) E (Op) E (Op) E (Op) E (Op) E (Op) E (Op) E
IMPR         FILE
999999
(
(=========================================================================
=
( ALTERACAO DE CONSTANTES DO PROGRAMA
```

```
(=========================================================================
=
DCTE
(Ct) (Val )
(TEPQ    .01  ( tolerancia de convergencia exigida p/ fluxo de potencia
)
(TEMD  1.E-4
(TETE  1.E-4
(TABS  1.E-4  ( tolerancia absoluta )
999999
(
(=========================================================================
=
( RESTABELECIMENTO DE CASO DE FLUXO DE POTENCIA
(=========================================================================
=
ARQV REST
1
(
( ---- arquivos de modelos ----
ULOG
3
BenchCA.BLT
ARQM
(
ULOG
3
BenchCA.CDU
ARQM
(
(=========================================================================
=
( ASSOCIACAO DE MAQUINAS COM MODELOS
(=========================================================================
=
DMAQ
( Nb)   Gr (P) (Q)  Und   ( Mg ) ( Mt )u( Mv )u( Me )u(Xvd)(Nbc)
    1   10           2         1             1u
    2   10           2         1             2u   102u
    3   10           6         1             3u   103u
    4   10           4         1             4u   104u
   14   10           4         1            14u   114u
999999
(
(=========================================================================
=
( EVENTOS
(=========================================================================
=
DEVT
(Tp)    ( Tempo)(  El  ) ( Pa) Nc  ( Ex) ( % ) (ABS ) Gr Und
APCL      0.20    7        8   1     7    0.
ABCI      0.30    7        8   1     7
ABCI      0.32    7        8   1     8
999999
(
(=========================================================================
=
( VARIAVEIS DE SAIDA
(=========================================================================
=
```

```
DPLT
(Tipo)M( El ) ( Pa) Nc Gp ( Br) Gr ( Ex) (Bl) P
VOLT        1
VOLT        2
VOLT        3
VOLT        4
VOLT        5
VOLT        6
VOLT        7
VOLT        8
VOLT        9
VOLT       10
VOLT       11
VOLT       12
VOLT       13
VOLT       14
EFD         1                    10
EFD         2                    10
EFD         3                    10
EFD         4                    10
EFD        14                    10
FMAQ        1                    10
FMAQ        2                    10
FMAQ        3                    10
FMAQ        4                    10
FMAQ       14                    10
VSAD        1                    10
VSAD        2                    10
VSAD        3                    10
VSAD        4                    10
VSAD       14                    10
DELT        1                    10
DELT        2                    10
DELT        3                    10
DELT        4                    10
DELT       14                    10
999999
(
(=====================================================================
=
( DADOS DE SIMULACAO
(=====================================================================
=
DSIM
( Tmax )   (Stp) ( P ) ( I )
    10     0.001         1
(
(=====================================================================
=
( EXECUCAO DO CASO
(=====================================================================
=
EXSI
(
FIM
```

Figura 174 – Arquivo BenchCA.stb

Na Figura 174 é possível perceber no campo DMAQ que as turbinas das máquinas e seus estabilizadores são modeladas por CDUs[47]. Isso é indicado pela letra "u" ao lado do número do modelo. Portanto, na barra de número 2 encontra-se um gerador síncrono cujo regulador tensão é modelado pelo CDU de número 2 e cujo estabilizador de tensão é modelado pelo CDU de número 102.

Além disso, no arquivo STB também é possível perceber que os CDUs se encontram no arquivo BenchCA.cdu. Isso significa que é nesse arquivo em que estão definidos os CDUs que modelam os reguladores de tensão e os estabilizadores associados às máquinas.

A seguir são apresentados os arquivos BenchCA.blt e BenchCA.cdu, nas Figura 175 e na Figura 176 contendo os modelos de máquinas e CDUs.

```
(============================================================
( MODELOS DE GERADOR TIPO BARRA INFINITA
(============================================================
DMDG MD01
(No) O (L'd)(Ra )( H )( D )(MVA)Fr
0100                            60
(
999999
(
(============================================================
( MODELOS DE GERADOR ROTOR LISO
(============================================================
(
DMDG MD03
(No)    (CS) (Ld )(Lq )(L'd)(L'q)(L"d)(Ll )(T'd)(T'q)(T"d)(T"q)
0001          180  170  030  055  025  020  8.0  0.4 0.03 0.05
(No)    (Ra )( H )( D )(MVA)Fr C
0001   .25    6.5  0.0  900
(
999999
(
(============================================================
( CURVAS DE SATURACAO
(============================================================
DCST
(....... Curvas de Saturacao de Geradores
(No) O T (  Y1  ) (  Y2  ) (  X1  )
(....... Curva 1
0001   2 0.015    9.6      0.9
(
999999
(
FIM
```

Figura 175 – BenchCA.blt

[47] Com exceção do gerador conectado à barra 1, que não possui estabilizador.

```
(=====================================================================
( CONTROLADORES DEFINIDOS PELO USUARIO
(=====================================================================
DCDU IMPR
(
(
( Excitatriz Estática
(
(ncdu) ( nome cdu )
  0001 AVRMAQ1
(
(EFPAR (npar) (     valpar     )
DEFPAR #Tr                  0.05
DEFPAR #Ka                 100.0
DEFPAR #Lmin                  -8
DEFPAR #Lmax                   8
(
(nb) (tipo) (stip)s(vent) (vsai) ( p1 )( p2 )( p3 )( p4 ) (vmin) (vmax)
   1 IMPORT VOLT          ET
   2 ENTRAD               VREF
   3 IMPORT VSAD          VPSS
   4 SOMA         +VREF   X4
                  -ET     X4
                   VPSS   X4
   5 LEDLAG        X4     X5     #Ka          1.0   #Tr
   6 LIMITA        X5     EFD                              LMIN   LMAX
   7 EXPORT EFD    EFD
(DEFVA (stip) (vdef) ( d1 )
DEFVAL        LMIN   #Lmin
DEFVAL        LMAX   #Lmax
(
FIMCDU
(
(ncdu) ( nome cdu )
  0002 AVRMAQ2
(
(EFPAR (npar) (     valpar     )
DEFPAR #Tr                  0.05
DEFPAR #Ka                 100.0
DEFPAR #Lmin                  -8
DEFPAR #Lmax                   8
(
(nb) (tipo) (stip)s(vent) (vsai) ( p1 )( p2 )( p3 )( p4 ) (vmin) (vmax)
   1 IMPORT VOLT          ET
   2 ENTRAD               VREF
   3 IMPORT VSAD          VPSS
   4 SOMA         +VREF   X4
                  -ET     X4
                   VPSS   X4
   5 LEDLAG        X4     X5     #Ka          1.0   #Tr
   6 LIMITA        X5     EFD                              LMIN   LMAX
   7 EXPORT EFD    EFD
(DEFVA (stip) (vdef) ( d1 )
DEFVAL        LMIN   #Lmin
DEFVAL        LMAX   #Lmax
(
FIMCDU
(
(ncdu) ( nome cdu )
  0003 AVRMAQ3
(
```

```
(EFPAR (npar) (     valpar     )
DEFPAR #Tr                  0.05
DEFPAR #Ka                 100.0
DEFPAR #Lmin                  -8
DEFPAR #Lmax                   8
(
(nb) (tipo) (stip)s(vent) (vsai) ( p1 )( p2 )( p3 )( p4 ) (vmin) (vmax)
   1 IMPORT VOLT          ET
   2 ENTRAD               VREF
   3 IMPORT VSAD          VPSS
   4 SOMA         +VREF   X4
                  -ET     X4
                   VPSS   X4
   5 LEDLAG        X4     X5     #Ka          1.0   #Tr
   6 LIMITA        X5     EFD                                LMIN   LMAX
   7 EXPORT EFD    EFD
(DEFVA (stip) (vdef) ( d1 )
DEFVAL        LMIN   #Lmin
DEFVAL        LMAX   #Lmax
(
FIMCDU
(
(ncdu) ( nome cdu )
  0004 AVRMAQ4
(
(EFPAR (npar) (     valpar     )
DEFPAR #Tr                  0.05
DEFPAR #Ka                 100.0
DEFPAR #Lmin                  -8
DEFPAR #Lmax                   8
(
(nb) (tipo) (stip)s(vent) (vsai) ( p1 )( p2 )( p3 )( p4 ) (vmin) (vmax)
   1 IMPORT VOLT          ET
   2 ENTRAD               VREF
   3 IMPORT VSAD          VPSS
   4 SOMA         +VREF   X4
                  -ET     X4
                   VPSS   X4
   5 LEDLAG        X4     X5     #Ka          1.0   #Tr
   6 LIMITA        X5     EFD                                LMIN   LMAX
   7 EXPORT EFD    EFD
(DEFVA (stip) (vdef) ( d1 )
DEFVAL        LMIN   #Lmin
DEFVAL        LMAX   #Lmax
(
FIMCDU
(
(ncdu) ( nome cdu )
  0014 AVRMAQ14
(
(EFPAR (npar) (     valpar     )
DEFPAR #Tr                  0.05
DEFPAR #Ka                 100.0
DEFPAR #Lmin                  -8
DEFPAR #Lmax                   8
(
(nb) (tipo) (stip)s(vent) (vsai) ( p1 )( p2 )( p3 )( p4 ) (vmin) (vmax)
   1 IMPORT VOLT          ET
   2 ENTRAD               VREF
   3 IMPORT VSAD          VPSS
   4 SOMA         +VREF   X4
```

```
                  -ET     X4
                   VPSS   X4
   5 LEDLAG        X4     X5     #Ka          1.0   #Tr
   6 LIMITA        X5     EFD                              LMIN   LMAX
   7 EXPORT EFD    EFD
(DEFVA (stip) (vdef) ( d1 )
DEFVAL        LMIN   #Lmin
DEFVAL        LMAX   #Lmax
(
FIMCDU
(
(--------------------------------------------------------------------
( nc ) ( nome cdu )
   102 PSS2
(EFPAR (nome) (     valor      )
DEFPAR #TW1                    3.
DEFPAR #TN1                   .17
DEFPAR #TD1                   .03
DEFPAR #KP1                    7.
DEFPAR #LMIN                 -0.1
DEFPAR #LMAX                  0.1
(nb)i(tipo) (stip)s(vent) (vsai) ( p1 )( p2 )( p3 )( p4 ) (vmin) (vmax)
   1 IMPORT WMAQ          X1
   2 WSHOUT        X1     X2     #TW1  1     #TW1
   3 LEDLAG        X2     X3     1     #TN1  1     #TD1
   4 LEDLAG        X3     X4     1     #TN1  1     #TD1
   5 GANHO         X4     X5     #KP1
   6 LIMITA        X5     X6                               LMIN   LMAX
   7 EXPORT VSAD   X6
(EFVAL (stip) (vdef) ( d1 )
DEFVAL        LMIN   #LMIN
DEFVAL        LMAX   #LMAX
FIMCDU
(
(--------------------------------------------------------------------
(ncdu) ( nome cdu )
  0103 PSS3
(EFPAR (nome) (     valor      )
DEFPAR #TW1                    3.
DEFPAR #TN1                   .72
DEFPAR #TD1                   .08
DEFPAR #KP1                   18.
DEFPAR #LMIN                 -0.1
DEFPAR #LMAX                  0.1
(nb)i(tipo) (stip)s(vent) (vsai) ( p1 )( p2 )( p3 )( p4 ) (vmin) (vmax)
   1 IMPORT WMAQ          X1
   2 WSHOUT        X1     X2     #TW1  1     #TW1
   3 LEDLAG        X2     X3     1     #TN1  1     #TD1
   4 GANHO         X3     X4     #KP1
   5 LIMITA        X4     X5                               LMIN   LMAX
   6 EXPORT VSAD   X5
(EFVAL (stip) (vdef) ( d1 )
DEFVAL        LMIN   #LMIN
DEFVAL        LMAX   #LMAX
FIMCDU
(
(--------------------------------------------------------------------
(ncdu) ( nome cdu )
  0104 PSS4
(EFPAR (nome) (     valor      )
DEFPAR #TW1                    3.
```

```
DEFPAR #TN1                  .17
DEFPAR #TD1                  .03
DEFPAR #KP1                   7.
DEFPAR #LMIN                -0.1
DEFPAR #LMAX                 0.1
(nb)i(tipo) (stip)s(vent) (vsai) ( p1 )( p2 )( p3 )( p4 ) (vmin) (vmax)
   1 IMPORT WMAQ          X1
   2 WSHOUT        X1     X2     #TW1  1     #TW1
   3 LEDLAG        X2     X3     1     #TN1  1     #TD1
   4 GANHO         X3     X4     #KP1
   5 LIMITA        X4     X5                              LMIN   LMAX
   6 EXPORT VSAD   X5
(EFVAL (stip) (vdef) ( d1 )
DEFVAL        LMIN   #LMIN
DEFVAL        LMAX   #LMAX
FIMCDU
(
(-------------------------------------------------------------------------
(ncdu) ( nome cdu )
  0114 PSS14
(EFPAR (nome) (     valor      )
DEFPAR #TW1                   3.
DEFPAR #TN1                  .26
DEFPAR #TD1                   .1
DEFPAR #KP1                  18.
DEFPAR #LMIN                -0.1
DEFPAR #LMAX                 0.1
(nb)i(tipo) (stip)s(vent) (vsai) ( p1 )( p2 )( p3 )( p4 ) (vmin) (vmax)
   1 IMPORT WMAQ          X1
   2 WSHOUT        X1     X2     #TW1  1     #TW1
   3 LEDLAG        X2     X3     1     #TN1  1     #TD1
   4 GANHO         X3     X4     #KP1
   5 LIMITA        X4     X5                              LMIN   LMAX
   6 EXPORT VSAD   X5
(EFVAL (stip) (vdef) ( d1 )
DEFVAL        LMIN   #LMIN
DEFVAL        LMAX   #LMAX
FIMCDU
(
(ncdu) ( nome cdu )
   201 RV-MQ01
(-------------------------------------------------------------------------
(
(-------------------------------------------------------------------------
(EFPAR (npar) (     valpar     )
(
0001 IMPORT DWMAQ          DW
0002 GANHO          DW     DWN    -1
0003 GANHO          DWN    ERRP   200.
0004 PROINT         DWN    ERRI     1.     0.0  .008
0005 SOMA           ERRP   ERR
                    ERRI   ERR
0006 EXPORT PMEC    ERR
(DEFVA (stip) (vdef) ( d1 )
(
FIMCDU
(
999999
FIM
```

Figura 176 – BenchCA.cdu

E.3 Arquivo AnaHVDC (Curto ABC-Terra)

A seguir é apresentado o arquivo BenchCA.ana, em formato do programa Anafas do Cepel, contendo os dados trifásicos dos modelos dos componentes de rede, utilizado para modelagem trifásica do caso.

```
TIPO
A 1
TITU
BenchCA
DBAR
(NB  CEM      BN      VPRE ANG VBAS DISJUN          DDMMAAAAADDMMAAAA IA SA  F
(----=-= ------------ ----==== ---- ------          --------======== ---=== -
    1    Barra1       1045       20                                  1    G   V            1.045A
    2    Barra2       1045-6.3   20                                  1    G   V            1.045A          -0.1098
    3    Barra3       1045 -36   20                                  2    G   V            1.045A          -0.6283
    4    Barra4       1045 -49   20                                  2    G   V            1.045A          -0.8551
    5    Barra5       1038  -5  500                                  1        V            1.038A           -0.087
    6    Barra6       1031 -11  500                                  1        V            1.031A          -0.1924
    7    Barra7       1004 -19  500                                  1        V            1.004A          -0.3274
    8    Barra8       1002 -43  500                                  2        V            1.002A          -0.7432
    9    Barra9       1000 -57  500                                  2        V           0.9997A          -0.9954
   10    Barra10      1030 -53  500                                  2        V             1.03A          -0.9326
   11    Barra11      1038 -40  500                                  2        V            1.038A          -0.6976
   12    Barra12      1009 -45  500                                  2        V            1.009A          -0.7794
   13    Barra13      1031 -34  500                                  2        V            1.031A           -0.588
   14    Barra14      1045 -29   20                                  2    G   V            1.045A          -0.5106
99999
DCIR
(BF  CE  BT   NCT  R1    X1    R0    X0    CN  S1_PgS0_Qg TAP  TB  TCIA DEFE KM CD RNDE  XNDE CP RNPA  XNPA SA
(----=-====  --=------==========------=====-----=====-----===-====--======------==------======---
    5     1   1T      .55556      .55556                              1-30     YN            D
    6     2   1T      .55556      .55556                              1-30     YN            D
   11     3   1T      .18519      .18519                              2-30     YN            D
   10     4   1T      .27778      .27778                              2-30     YN            D
    5     6   1L.04545.66605.395673.1786       6140 3119              1
    6     7   1L.090541.3294.783496.3263      12292 6254              1
    6     7   2L.090541.3294.783496.3263      12292 6254              1
    6     7   3L.090541.3294.783496.3263      12292 6254              1
    7     8   1L.25984 3.9022.105718.003      3728519276              1
    7     8   2L.25984 3.9022.105718.003      3728519276              1
    7     8   3L.25984 3.9022.105718.003      3728519276              1
    8     9   1L.134881.98731.15579.4121      18470 9420              2
    8     9   2L.134881.98731.15579.4121      18470 9420              2
    8    12   1L.04545.66605.395673.1786       6140 3119              2
    9    10   1L.04545.66605.395673.1786       6140 3119              2
    9    11   1L.134881.98731.15579.4121      18470 9420              2
   12    13   1L.04545.66605.395673.1786       6140 3119              2
   13    14   1T       .27778       .27778                            2-30     YN            D
    0     6   1H        -3333        -3333                            1                      YN
    0     8   1H        -1818        -1818                            2                      YN
    0     9   1H        -6667        -6667                            2                      YN
    0    12   1H        -3030        -3030                            2                      YN
    0     6   1C85.03742.51885.03742.518                              1                      YN
    0     9   1C1.9968.079871.9968.07987                              2                      YN
    0    10   1C5.3012.132535.3012.13253                              2                      YN
    0    11   1C4.3081.086164.3081.08616                              2                      YN
    0    12   1C2.9064.083042.9064.08304                              2                      YN
    0     1  10G.013891.3889.013891.3889                             1                      YN
    0     2  10G.013891.3889.013891.3889                             1                      YN
    0     3  10G.00463.46296.00463.46296                             2                      YN
    0     4  10G.00694.69444.00694.69444                             2                      YN
    0    14  10G.00694.69444.00694.69444                             2                      YN
99999
DSHL
(BF  CE BT    NCTNG  Qpos  L  Rn     Xn  E Nome  NunNop              IA  SA
(====-=-=====  ==-== ====== =------======-====== ===---              === ==
    7     8   1D 1   -150                                            1
    8     7   1D 1   -150                                            2
    7     8   2D 1   -150                                            1
    8     7   2D 1   -150                                            2
    7     8   3D 1   -150                                            1
    8     7   3D 1   -150                                            2
99999
DARE
(NN  C                          NOME
(--  =            ---------------------------------------
  1               *  AREA 1                             *
  2               *  AREA 2                             *
99999
```

Figura 177 - BenchCA.ana

```
TITU
Bench CA
DHIS
('Filename' )             (Case number)
'BENCHCA.SAV'           1
DSTB
'BenchCA.BLT'
'BenchCA.stb'
FIM
DDLL
'BenchCA.DLL' 'BenchCA.CDU'
FIM
DLOA
(label) (nb1) (type)
 ALL               0
FIM
DLTA
(label) (from) (toid) (ncir) (length) (nPi) (minTal) (fParNom)
 ALL                                                    0            0
1
FIM
DANA
('Anafas file' )
'BENCHCA.ANA'
DBRK
(id) (type) (nb1) (nb2) (nb3) (gOn) (gOff) (Imargin) (State)
   1 LINE       7     8        1      1e+6   1e-6          1e10    CLOSED
   2 LINE       8     7        1      1e+6   1e-6          1e10    CLOSED
FIM
DSIM
(step) (duration) (period) (tolerr) (maxiter)
 10e-6       5            1
FIM
DEVT
        (type                                    )       (time  )    (id1)
(id2) (id3) (value)
APPLY_FAULT_LINE_ABCG   0.200000     7       8       1    1e-6
OPEN_A                                          0.300000     1
OPEN_B                                           0.300000    1
OPEN_C                                           0.300000    1
OPEN_A                                           0.320000    2
OPEN_B                                           0.320000    2
OPEN_C                                           0.320000    2
FIM
DPLT
('        label      ') (mode) (device) (variable) (id1) (id2) (id3)
'Voltage Bus 1'     ABS    BUS      VOLT        1
'Voltage Bus 2'     ABS    BUS      VOLT        2
'Voltage Bus 3'     ABS    BUS      VOLT        3
'Voltage Bus 4'     ABS    BUS      VOLT        4
'Voltage Bus 5'     ABS    BUS      VOLT        5
'Voltage Bus 6'     ABS    BUS      VOLT        6
'Voltage Bus 7'     ABS    BUS      VOLT        7
```

```
'Voltage Bus 8'      ABS     BUS       VOLT         8
'Voltage Bus 9'      ABS     BUS       VOLT         9
'Voltage Bus 10'     ABS     BUS       VOLT        10
'Voltage Bus 11'     ABS     BUS       VOLT        11
'Voltage Bus 12'     ABS     BUS       VOLT        12
'Voltage Bus 13'     ABS     BUS       VOLT        13
'Voltage Bus 14'     ABS     BUS       VOLT        14
'EfdGen1'           INST     GEN       EFD          1  10
'EfdGen2'           INST     GEN       EFD          2  10
'EfdGen3'              INST   GEN      EFD         3  10
'EfdGen4'              INST   GEN      EFD         4  10
'EfdGen14'             INST   GEN      EFD        14  10
'WgGen1'               INST   GEN      WG          1  10
'WgGen2'               INST   GEN      WG          2  10
'WgGen3'               INST   GEN      WG          3  10
'WgGen4'               INST   GEN      WG          4  10
'WgGen14'              INST   GEN      WG         14  10
'VpssGen1'             INST   GEN      VPSS        1  10
'VpssGen2'             INST   GEN      VPSS        2  10
'VpssGen3'             INST   GEN      VPSS        3  10
'VpssGen4'             INST   GEN      VPSS        4  10
'VpssGen14'            INST   GEN      VPSS       14  10
'DELTAGen1'            INST   GEN      DELTA       1  10
'DELTAGen2'            INST   GEN      DELTA       2  10
'DELTAGen3'            INST   GEN      DELTA       3  10
'DELTAGen4'            INST   GEN      DELTA       4  10
'DELTAGen14'           INST   GEN      DELTA       4  10
FIM
```

Figura 178 - Arquivo BenchCA AnaHVDC (Trifásico)

E.4 Arquivo AnaHVDC (Curto na Fase A)

O caso monofásico é muito semelhante ao monofásico da Figura 178. Por isso, na Figura 179 são destacadas apenas as diferenças.

```
DBRK
(id) (type) (nb1) (nb2) (nb3) (gOn) (gOff) (Imargin)   (State)
  1 LINE     7    8      1    1e+6  1e-6   1e10      CLOSED
  2 LINE     8    7      1    1e+6  1e-6   1e10      CLOSED
DEVT
      (type                    )     (time  )   (id1) (id2) (id3) (value)
APPLY_FAULT_LINE_A              0.200000    7     8     1    1e-6
OPEN_A                         0.300000    1
OPEN_B                         0.300000    1
OPEN_C                         0.300000    1
OPEN_A                         0.320000    2
OPEN_B                         0.320000    2
OPEN_C                         0.320000    2
```

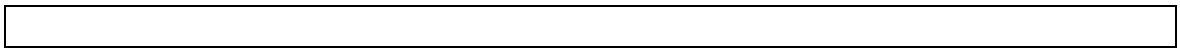


Figura 179 – Arquivo BenchCA AnaHVDC (Trifásico): diferenças em relação ao Monofásico

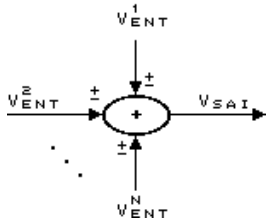
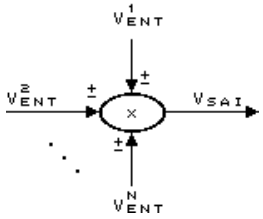
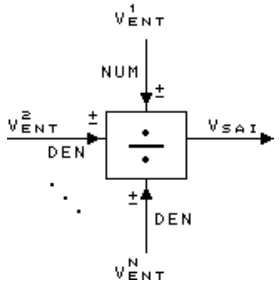
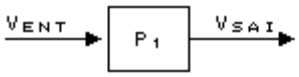
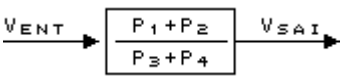
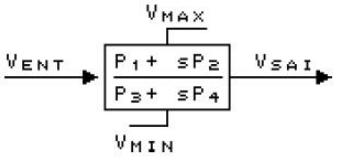
Dessa forma, percebe-se que em DEVT de fato é aplicado um curto monofásico e que ocorre na fase A (APPLY_FAULT_LINE_A). Por sua vez, na Figura 178 é aplicado um curto entre as 3 fases e a terra (APPLY_FAULT_LINE_ABCG). Além disso, em DBRK os disjuntores possuem *Imargin* igual a 10^{10}. O objetivo disto é forçar a abertura dos três polos do disjuntor no mesmo instante, sem esperar a passagem das correntes dos polos do disjuntor pelo zero, evitando surgimento de desbalanço (componentes de sequência negativa e zero), onde uma modelagem monofásica equivalente é válida.

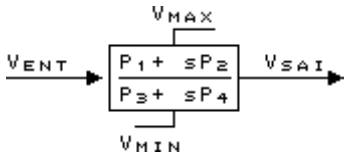
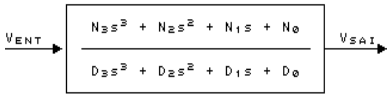
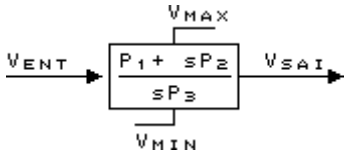
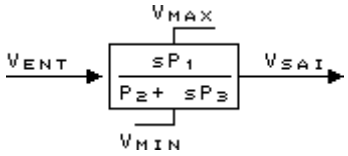
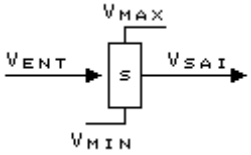
O curto circuito é aplicado em 0,2 s um curto de reatância de 2 % na linha entre as barras 7 e 8. Na sequência, em 0,3 s, ocorre a abertura do circuito do lado da barra 7. Por fim, em 0,32 s, é realizada a abertura do circuito do lado da barra 8.

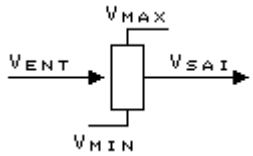
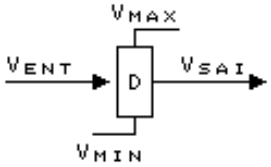
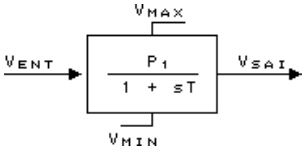
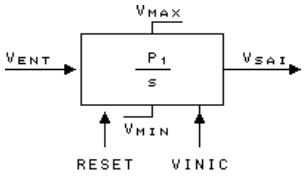
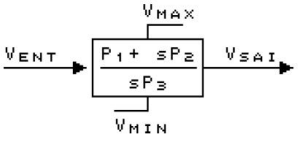
O arquivo do caso do AnaHVDC encontra-se no apêndice D2. No código de execução DEVT é possível ver os eventos aplicados, equivalentes ao do Anatem. APPLY_FAULT_LINE_A é a aplicação de um curto com resistência 10^{-6} na fase “a” e entre as barras 7 e 8. Depois são abertas as três fases no lado da barra 7 (OPEN_A, OPEN_B e OPEN_C) e depois o mesmo para as três fases do lado da barra 8.

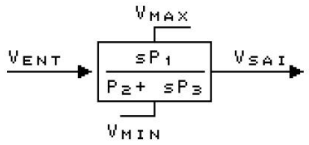
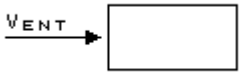
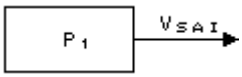
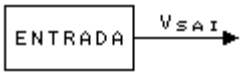
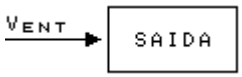
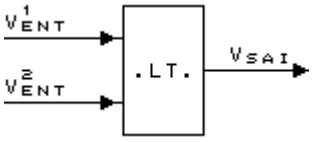
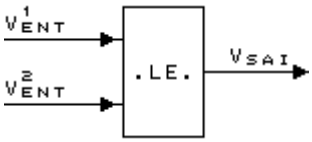
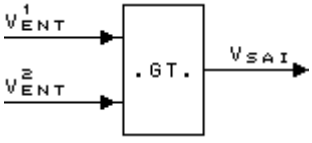
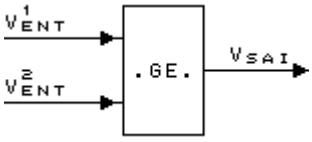
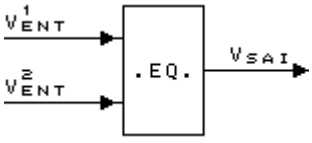
Apêndice F – Lista de Blocos Implementados

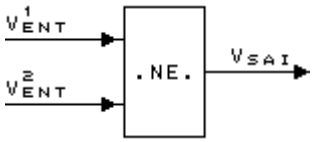
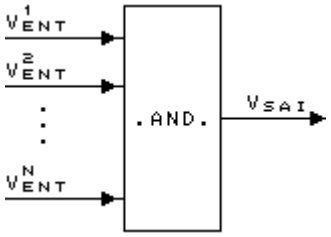
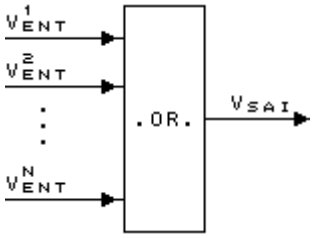
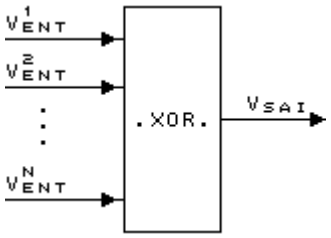
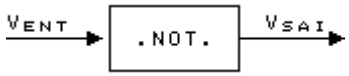
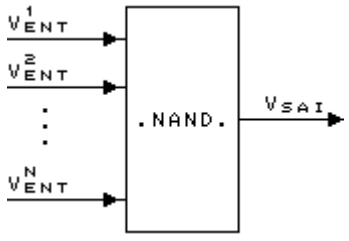
A seguir segue a lista completa com todos os blocos que foram implementados no UDCC baseada em [19].

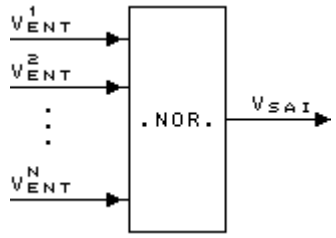
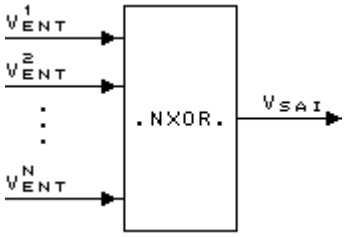
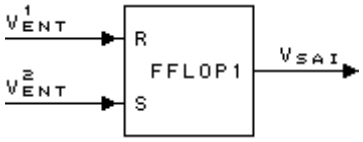
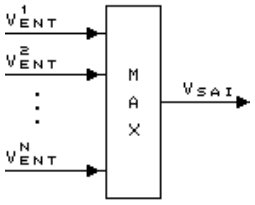
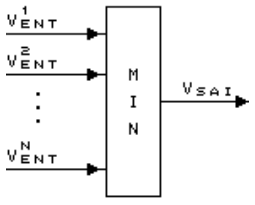
Bloco	Símbolo	Equação
SOMA		$V_{sai} = \pm V_{ent}^{1} \pm V_{ent}^{2} \pm \cdots \pm V_{ent}^{n}$
MULTPL		$V_{sai} = (\pm V_{ent}^{1}) \cdot (\pm V_{ent}^{2}) \cdot \ldots \cdot (\pm V_{ent}^{n})$
DIVSAO		$V_{sai} = (\pm V_{ent}^{1}) \div (\pm V_{ent}^{2}) \div \ldots \div (\pm V_{ent}^{n})$
GANHO		$V_{sai}(t) = P_1 \cdot V_{ent}(t)$
FRACAO		$V_{sai} = \frac{P_1 + P_2}{P_3 + P_4} V_{ent}$
ORD(1)		$Y(s) = \frac{P_1 - \frac{P_2 P_3}{P_4}}{P_3 + sP_4} V_{ent}(s)$ $Y(t) < V_{min} \Rightarrow Y(t) = V_{min}$ $Y(t) > V_{max} \Rightarrow Y(t) = V_{max}$ $V_{sai}(t) = \frac{P_2}{P_4} V_{ent}(t) + Y(t)$

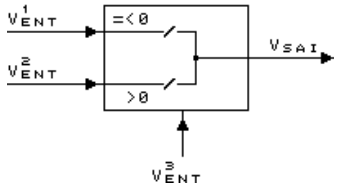
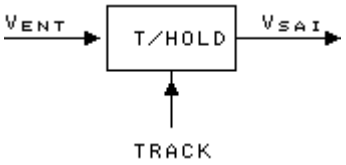
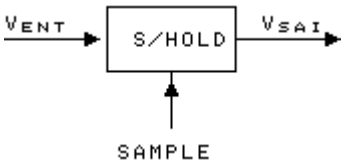
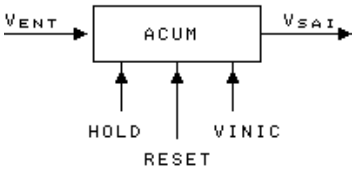
LEDLAG		$Y(s) = \frac{P_1 - \frac{P_2 P_3}{P_4}}{P_3 + sP_4} V_{ent}(s)$ $Y(t) < V_{min} \Rightarrow Y(t) = V_{min}$ $Y(t) > V_{max} \Rightarrow Y(t) = V_{max}$ $V_{sai}(t) = \frac{P_2}{P_4} V_{ent}(t) + Y(t)$
POL(S)		$A(t) = V_{ent}(t) - \frac{1}{D_k}\left(\sum_{i=0}^{k-1} D_i X_{i+1}(t)\right)$ $X_k(t) = X_k(t=0) + \int_0^t A(\xi)d\xi$ $X_i(t) = X_i(t=0) + \int_0^t X_{i+1}(\xi)d\xi \; ; i = k-1, 1$ $V_{sai}(t) = \frac{1}{D_k}\left(N_k A(t) + \sum_{i=o}^{k-1} N_i X_{i+1}(t)\right)$
PROINT		$Y(s) = \frac{P_1}{P_3} \cdot \frac{1}{s} V_{ent}(s)$ $Y(t) < V_{min} \Rightarrow Y(t) = V_{min}$ $Y(t) > V_{max} \Rightarrow Y(t) = V_{max}$ $V_{sai}(t) = \frac{P_2}{P_3} V_{ent}(t) + Y(t)$
WSHOUT		$Y(s) = \frac{-\frac{P_1 P_2}{P_3}}{P_2 + sP_3} V_{ent}(s)$ $Y(t) < V_{min} \Rightarrow Y(t) = V_{min}$ $Y(t) > V_{max} \Rightarrow Y(t) = V_{max}$ $V_{sai}(t) = \frac{P_1}{P_3} V_{ent}(t) + Y(t)$
DERIVA		$V_{sai}(t) = \begin{cases} 0, & t = 0 \\ \frac{V_{ent}(t) - V_{ent(t-\Delta t)}}{\Delta t}, & t \geq 0 \end{cases}$

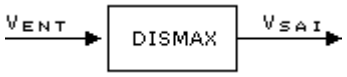
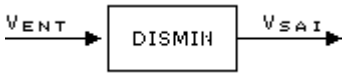
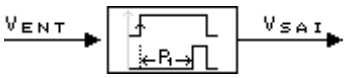
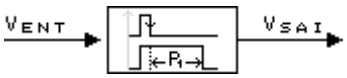
LIMITA		$V_{ent} < V_{min} \Rightarrow V_{sai} = V_{min}$ $V_{min} \leq V_{ent} \leq V_{max} \Rightarrow V_{sai} = V_{ent}$ $V_{ent} > V_{max} \Rightarrow V_{sai} = V_{max}$
RATELM		$V_{sai}(t) = \begin{cases} V_{sai}(t-\Delta t) + V_{max} \cdot \Delta t, & x(t) > V_{max} \\ V_{ent}(t), & V_{min} \leq x(t) \leq V_{max} \\ V_{sai}(t-\Delta t) + V_{min} \cdot \Delta t, & x(t) < V_{min} \end{cases}$ $Sendo, x(t) = \frac{V_{ent}(t) - V_{sai}(t-\Delta t)}{\Delta t}$
LAGNL		$V_{sai}(s) = \frac{P_1}{1+sT} V_{ent}(s)$ $Se \frac{dV_{ent}}{dt} < 0 \Rightarrow T = P_2$ $Se \frac{dV_{ent}}{dt} > 0 \Rightarrow T = P_3$ $Se \frac{dV_{ent}}{dt} = 0 \Rightarrow T = T(t-\Delta t)$ $V_{sai}(t) < V_{min} \Rightarrow V_{sai}(t) = V_{min}$ $V_{sai}(t) > V_{max} \Rightarrow V_{sai}(t) = V_{max}$
INTRES		Se RESET > 0: $V_{sai}(t) = VINIC(t)$ Senão: $V_{sai}(t) = V_{sai}(t-\Delta t) + P_1 \int_{t-\Delta t}^{t} V_{ent}(t)dt$ $V_{sai}(t) < V_{min} \Rightarrow V_{sai}(t) = V_{min}$ $V_{sai}(t) > V_{max} \Rightarrow V_{sai}(t) = V_{max}$
PROIN2		$Y(s) = \frac{-\frac{P_1 P_3}{P_2}}{P_1 + sP_2} \cdot \frac{P_2}{P_3}$ $V_{sai}(t) = \frac{P_2}{P_3} V_{ent}(t) - Y(t)$ $V_{sai}(t) < V_{min} \Rightarrow V_{sai}(t) = V_{min}$ $V_{sai}(t) > V_{max} \Rightarrow V_{sai}(t) = V_{max}$

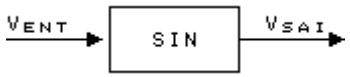
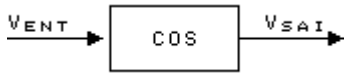
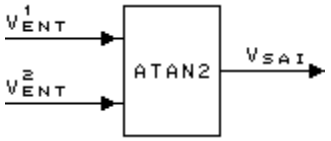
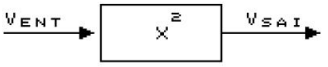
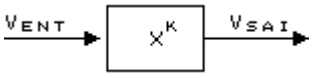
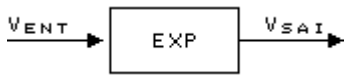
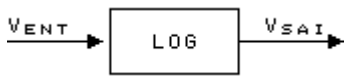
WSHOU2		$Y(s) = \frac{P_2}{sP_1} \cdot \frac{P_1}{P_3}$ $V_{sai}(t) = \frac{P_1}{P_3} V_{ent}(t) - Y(t)$ $V_{sai}(t) < V_{min} \Rightarrow V_{sai}(t) = V_{min}$ $V_{sai}(t) > V_{max} \Rightarrow V_{sai}(t) = V_{max}$
EXPORT		Exporta o valor da variável V_{ENT}.
IMPORT		Importa a variável V_{SAI}.
ENTRAD		
SAIDA		
.LT.		Se $V_{ent1} < V_{ent2}$ VERDADEIRO → $V_{sai} = 1$ FALSO → $V_{sai} = 0$
.LE.		Se $V_{ent1} \leq V_{ent2}$ VERDADEIRO → $V_{sai} = 1$ FALSO → $V_{sai} = 0$
.GT.		Se $V_{ent1} > V_{ent2}$ VERDADEIRO → $V_{sai} = 1$ FALSO → $V_{sai} = 0$
.GE.		Se $V_{ent1} \geq V_{ent2}$ VERDADEIRO → $V_{sai} = 1$ FALSO → $V_{sai} = 0$
.EQ.		Se $V_{ent1} = V_{ent2}$ VERDADEIRO → $V_{sai} = 1$ FALSO → $V_{sai} = 0$

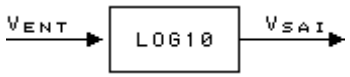
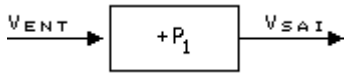
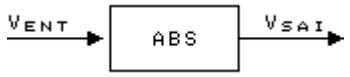
.NE.		Se $V_{ent1} \neq V_{ent2}$ VERDADEIRO → $V_{sai} = 1$ FALSO → $V_{sai} = 0$
.AND.		$V_{sai} = V_{ent1}$ .AND. V_{ent2} .AND. ... .AND. V_{entN} Os sinais de entrada são interpretados logicamente da seguinte forma: $V_{ent} \leq 0 \longrightarrow$ FALSO (0) $V_{ent} > 0 \longrightarrow$ VERDADEIRO (1)
.OR.		$V_{sai} = V_{ent1}$ .OR. V_{ent2} .OR. ... .OR. V_{entN} Os sinais de entrada são interpretados logicamente da seguinte forma: $V_{ent} \leq 0 \longrightarrow$ FALSO (0) $V_{ent} > 0 \longrightarrow$ VERDADEIRO (1)
.XOR.		$V_{sai} =$ .XOR. (V_{ent1} , V_{ent2} , ... , V_{entN}) O sinal de saída terá o valor lógico VERDADEIRO (1) quando uma e somente uma das entradas tiver valor lógico VERDADEIRO . Os sinais de entrada são interpretados logicamente da seguinte forma: $V_{ent} \leq 0 \longrightarrow$ FALSO (0) $V_{ent} > 0 \longrightarrow$ VERDADEIRO (1)
.NOT.		$V_{sai} =$ .NOT. V_{ent} Os sinais de entrada são interpretados logicamente da seguinte forma: $V_{ent} \leq 0 \longrightarrow$ FALSO (0) $V_{ent} > 0 \longrightarrow$ VERDADEIRO (1)
.NAND.		V_{sai} = .NOT.(V_{ent1} .AND. V_{ent2} .AND. ... .AND. V_{entN}) Os sinais de entrada são interpretados logicamente da seguinte forma: $V_{ent} \leq 0 \longrightarrow$ FALSO (0) $V_{ent} > 0 \longrightarrow$ VERDADEIRO (1)

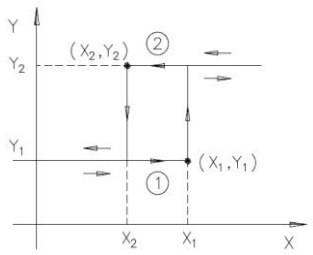
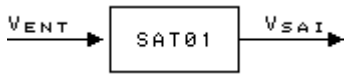
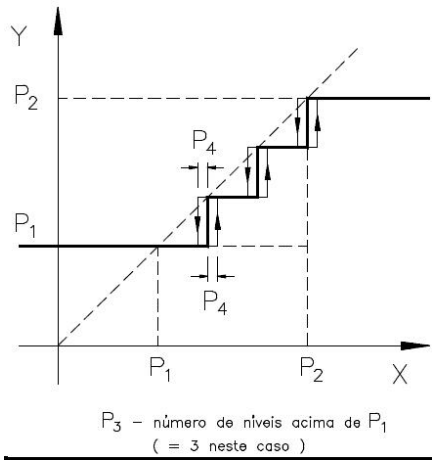
.NOR.		V_{sai} = .NOT.(V_{ent1} .OR. V_{ent2} .OR. ... .OR. V_{entN}) Os sinais de entrada são interpretados logicamente da seguinte forma: $V_{ent} \leq 0 \longrightarrow$ FALSO (0) $V_{ent} > 0 \longrightarrow$ VERDADEIRO (1)
.NXOR.		V_{sai} = .NXOR. (V_{ent1} , V_{ent2} , ... , V_{entN}) O sinal de saída terá o valor lógico VERDADEIRO (1) quando nenhuma ou mais de uma das entradas tiver valor lógico VERDADEIRO. Os sinais de entrada são interpretados logicamente da seguinte forma: $V_{ent} \leq 0 \longrightarrow$ FALSO (0) $V_{ent} > 0 \longrightarrow$ VERDADEIRO (1)
FFLOP1		Este bloco corresponde ao circuito lógico chamado FLIP-FLOP tipo SET-RESET. O valor do sinal de saída é determinado da seguinte forma: - Se apenas a primeira entrada (dita entrada de RESET) tiver valor lógico VERDADEIRO, a saída terá valor lógico FALSO (0) . - Se apenas a segunda entrada (dita entrada de SET) tiver valor lógico VERDADEIRO, a saída terá valor lógico VERDADEIRO (1) . - Se ambas as entradas tiverem o valor lógico VERDADEIRO ou FALSO, o valor lógico da saída será o mesmo que o do passo de integração anterior. Os sinais de entrada são interpretados logicamente da seguinte forma: $V_{ent} \leq 0 \longrightarrow$ FALSO (0) $V_{ent} > 0 \longrightarrow$ VERDADEIRO (1)
MAX		$V_{sai} = \text{MAX}(V_{ent}^1, V_{ent}^2, \dots, V_{ent}^n)$
MIN		$V_{sai} = \text{MIN}(V_{ent}^1, V_{ent}^2, \dots, V_{ent}^n)$

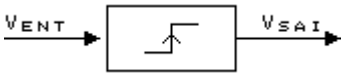
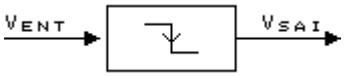
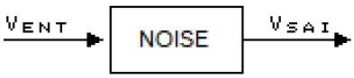
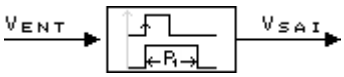
SELET2		$V_{ent3} \leq 0 \rightarrow V_{sai} = V_{ent1}$ $V_{ent3} > 0 \rightarrow V_{sai} = V_{ent2}$
DELAY		$V_{sai}(t) = V_{ent}(t - \Delta t)$ onde Δt é o passo de integração
T/HOLD		Se $TRACK(t) > 0$ então $V_{sai}(t) = V_{ent}(t)$ senão $V_{sai}(t) = V_{sai}(t - \Delta t)$ onde Δt é o passo de integração.
S/HOLD		Se $SAMPLE(t) > 0$ *e* $SAMPLE(t - \Delta t) \leq 0$ então $V_{sai}(t) = V_{ent}(t)$ senão $V_{sai}(t) = V_{sai}(t - \Delta t)$ onde Δt é o passo de integração.
ACUM		Se $RESET(t) > 0$ então $V_{sai}(t) = VINIC(t)$ senão Se $HOLD(t) > 0$ então $V_{sai}(t) = V_{sai}(t - \Delta t)$ senão $V_{sai}(t) = V_{sai}(t - \Delta t) + P_1 V_{ent}(t)$

DISMAX		Se $V_{ent}(t)$ $> P_1$ durante P_2 segundos então $V_{sai}(t) = 1$ (verdadeiro) senão Se $V_{ent}(t)$ $< P_1$ por até mais P_3 segundos então $V_{sai}(t) = 1$ (verdadeiro) senão $V_{sai}(t) = 0$ (falso)
DISMIN		Se $V_{ent}(t) < P_1$ durante P_2 segundos então $V_{sai}(t) = 1$ (verdadeiro) senão Se $V_{ent}(t) > P_1$ por até mais P_3 segundos: $V_{sai}(t) = 1$ (verdadeiro) senão $V_{sai}(t) = 0$ (falso)
DLAYON		$V_{sai}(t)$ $= \begin{cases} 1, & V_{ent}(t^*) > 0, \forall t^* \in [t - P1, t] \\ 0, & caso\ contrário \end{cases}$
DLAYOF		$V_{sai}(t)$ $= \begin{cases} 0, & V_{ent}(t^*) \leq 0, \forall t^* \in [t - P1, t] \\ 1, & caso\ contrário \end{cases}$
DEGREE		$V_{sai}(t) = \frac{180}{\pi} V_{ent}$
RADIAN		$V_{sai}(t) = \frac{\pi}{180} V_{ent}$

SIN		$V_{sai} = \sin(\mathrm{V_{ent}})$
COS		$V_{sai} = \cos(V_{ent})$
TAN		$V_{sai} = \tan(V_{ent})$
ACOS		$V_{sai} = \mathrm{acos}(V_{ent})\ ;\ 0 \le V_{sai} \le \pi$
ASIN		$V_{sai} = \mathrm{asin}(V_{ent})\ ;\ -\frac{\pi}{2} \le V_{sai} \le \frac{\pi}{2}$
ATAN		$V_{sai} = \mathrm{atan}(V_{ent})\ ;\ -\frac{\pi}{2} \le V_{sai} \le \frac{\pi}{2}$
ATAN2		$V_{sai} = \mathrm{atan2}(V_{ent1}, V_{ent2})\ ;\ -\pi \le V_{sai} \le \pi$ Se $V_{ent2} \neq 0\ \Rightarrow\ V_{sai} = \mathrm{atan}\left(\frac{V_{ent1}}{V_{ent2}}\right)\ ;\ -\pi \le V_{sai} \le \pi$ Se $V_{ent2} = 0$ e $V_{ent1} > 0\ \Rightarrow\ V_{sai} = \frac{\pi}{2}$ Se $V_{ent2} = 0$ e $V_{ent1} < 0\ \Rightarrow\ V_{sai} = -\frac{\pi}{2}$ Se $V_{ent2} = 0$ e $V_{ent1} = 0\ \Rightarrow\ V_{sai} = \mathrm{V_{sai}}(t - \Delta t)$
SQRT		$V_{sai} = \sqrt{V_{ent}}\ ;\ V_{ent} \ge 0$
X**2		$V_{sai} = (V_{ent})^2$
X**K		$V_{sai} = (V_{ent})^{\frac{P_1}{P_2}};\ V_{ent} > 0$
EXP		$V_{sai} = P_1 \cdot \exp(\ P_2(V_{ent} - P_3)\ )$
LOG		$V_{sai} = \ln(V_{ent})\ ;\ V_{ent} > 0$

LOG10		$V_{sai} = \log(V_{ent})\ ;\ V_{ent} > 0$
INVRS		$V_{sai} = \frac{1}{V_{ent}}\ ;\ V_{ent} \neq 0$
OFFSET		$V_{sai} = V_{ent} + P_1$
MENOS		$V_{sai} = -V_{ent}$
ABS		$V_{sai} = \lvert V_{ent} \rvert$
SINAL		$V_{ent} < 0 \Rightarrow V_{sai} = -1$ $V_{ent} = 0 \Rightarrow V_{sai} = 0$ $V_{ent} > 0 \Rightarrow V_{sai} = 1$
TRUNC		$V_{sai} = \text{INT}(V_{ent})$
ROUND		Se $V_{sai} > -0.0$ então $V_{sai} = \text{INT}(V_{ent} + 0.5)$ senão $V_{sai} = \text{INT}(V_{ent} - 0.5)$
PULSO		$V_{ent} \leq P_1 \Rightarrow V_{sai} = 0$ $P_1 < V_{ent} \leq P_3 \Rightarrow V_{sai} = P_2$ $V_{ent} > P_3 \Rightarrow V_{sai} = P_4$
RAMPA		$V_{ent} \leq P_1 \Rightarrow V_{sai} = P_2$ $P_1 < V_{ent} < P_3 \Rightarrow V_{sai}$ $= P_2 + \frac{P_4 - P_2}{P_3 - P_1}(V_{ent} - P_1)$ $V_{ent} \geq P_3 \Rightarrow V_{sai} = P_4$
RETA		$V_{sai} = P_1 V_{ent} + P_2$
DEADB1		$V_{ent} < P_1 \Rightarrow V_{sai} = P_3(V_{ent} - P_1)$ $P_1 \leq V_{ent} \leq P_2 \Rightarrow V_{sai} = 0$ $V_{ent} > P_2 \Rightarrow V_{sai} = P_4(V_{ent} - P_2)$

DEADB2		$V_{ent} < P_1 \Rightarrow V_{sai} = P_2$ $P_1 \leq V_{ent} \leq P_3 \Rightarrow V_{sai} = 0$ $V_{ent} > P_3 \Rightarrow V_{sai} = P_4$
HISTE1		
SAT01		$V_{ent} < -P_1 \Rightarrow V_{sai} = -P_2 + \frac{P_4 - P_2}{P_3 - P_1}(V_{ent} + P_1)$ $-P_1 \leq V_{ent} \leq P_1 \Rightarrow V_{sai} = \frac{P_2}{P_1} V_{ent}$ $V_{ent} > P_1 \Rightarrow V_{sai} = P_2 + \frac{P_4 - P_2}{P_3 - P_1}(V_{ent} - P_1)$
PONTOS		Se $X_i \leq V_{ent} < X_{i+1}$, para $i = 1, n-1$: $V_{sai} = Y_i + \frac{Y_{i+1} - Y_i}{X_{i+1} - X_i}(V_{ent} - X_i)$ $Se\ V_{ent} < X_1: V_{sai} = Y_1 + \frac{Y_2 - Y_1}{X_2 - X_1}(V_{ent} - X_1)$ $Se\ V_{ent} \geq X_n$: $V_{sai} = Y_{n-1} + \frac{Y_n - Y_{n-1}}{X_n - X_{n-1}}(V_{ent} - X_{n-1})$
STEPS		

FEX		$V_{sai} = \begin{cases} \sqrt{3} \cdot (1 - V_{ent}), & se\ 0{,}75 \leq V_{ent} \leq 1 \\ \sqrt{0{,}75 - {V_{ent}}^2}, & se\ \frac{\sqrt{3}}{4} \leq V_{ent} < 0{,}75 \\ 1 - \frac{1}{\sqrt{3}} V_{ent}, & se\ 0 \leq V_{ent} < \frac{\sqrt{3}}{4} \end{cases}$ A função é estendida para todo o domínio dos reais com a seguinte consideração: $V_{sai} = \begin{cases} 0, & se\ V_{ent} > 1 \\ 1, & se\ V_{ent} < 0 \end{cases}$
SUBIDA		$V_{sai}(t) = \begin{cases} 1, & V_{ent}(t) - V_{ent}(t - \Delta t) > P_1 \\ 0, & V_{ent}(t) - V_{ent}(t - \Delta t) \leq P_1 \end{cases}$
DESCID		$V_{sai}(t) = \begin{cases} 1, & V_{ent}(t) - V_{ent}(t - \Delta t) < P_1 \\ 0, & V_{ent}(t) - V_{ent}(t - \Delta t) \geq P_1 \end{cases}$
NOISE		Sinal ruído implementado por uma equação interna baseada em números pseudoaleatórios.
MONEST		$V_{sai}(t) = \begin{cases} 0, & \int_{t-\#P1}^{t} \uparrow f(V_{ent}(t))dt = 0 \\ 1, & caso\ contrário \end{cases}$ Sendo $\uparrow f(\cdot)$ a função "detecção de borda de subida" do sinal Vent com sensibilidade P2.