**Marcelo Giovani Mota Souza**

# Paralelismo computacional de processamento digital de imagens aplicado à detecção de MARFEs no JET

Dissertação de Pós-Graduação *Stricto Sensu* apresentada à Coordenação de Formação Científica do Centro Brasileiro de Pesquisas Físicas - CBPF do Ministério da Ciência, Tecnologia e Inovação para obtenção do título de Mestre no programa Mestrado Profissional em Física com ênfase em Instrumentação Científica.

Orientador:
D.Sc. Nilton Alves Jr.

Orientador:
D.Sc. Márcio Portes de Albuquerque

Rio de Janeiro - RJ / Brasil
2013

**Marcelo Giovani Mota Souza**

# Paralelismo computacional de processamento digital de imagens aplicado à detecção de MARFEs no JET

Dissertação de Pós-Graduação *Stricto Sensu* apresentada à Coordenação de Formação Científica do Centro Brasileiro de Pesquisas Físicas - CBPF do Ministério da Ciência, Tecnologia e Inovação para obtenção do título de Mestre no programa Mestrado Profissional em Física com ênfase em Instrumentação Científica.

Aprovada em *30 de outubro de 2013*

---

D.Sc. Ricardo Magnus Osório Galvão (USP/SBF - RNF)

---

D.Sc. Joaquim Teixeira de Assis (IPRJ/UERJ)

---

D.Sc. Andrea Murari (Consorzio RFX - JET/EFDA/EURATOM)

---

D.Sc. Nilton Alves Jr.
(Orientador)

---

D.Sc. Márcio Portes de Albuquerque
(Orientador)

Rio de Janeiro - RJ / Brasil
2013

Este documento foi produzido com o suporte do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), pela Financiadora de Estudos e Projetos (FINEP) do Ministério da Ciência, Tecnologia e Inovação (MCTI). Este trabalho foi realizado no âmbito do Acordo Europeu para o Desenvolvimento da Fusão (*European Fusion Development Agreement*) dentro do acordo do Brasil com a EURATOM para pesquisas em fusão, coordenado pela Rede de Fusão do Brasil pela Comissão Nacional de Energia Nuclear (MCTI). As opiniões aqui expressas não refletem necessariamente as da Comissão Europeia.

---

This document was produced with the support from the National Council for Scientific and Technological Development (CNPq) and the Brazilian Research and Projects Financing (FINEP) of the Brazilian Ministry of Science, Technology and Inovation. This work has been carried out under the European Fusion Development Agreement under the Brazil - EURATOM collaboration agreement on fusion research, coordinated by the Brazilian Fusion Network and by the Brazilian Nuclear Energy Commission. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

*Dedico esta dissertação ao professor Ademarlaudo França Barbosa (Laudo), um exemplo profissional, pessoal, fundador e incentivador do mestrado em Instrumentação Científica, conselheiro e amigo.*

# Agradecimentos

Agradeço inicialmente à minha esposa, aos meus amigos de trabalho e aos companheiros de copo.

Agradeço aos meus colegas de mestrado e aos professores responsáveis por esta jornada.

Agradeço ao amigo Ednardo Miranda pelo apoio e confiança no decorrer deste trabalho e também a amiga Fernanda Dutra Moraes pelos desenvolvimentos conjunto dos códigos para avaliação de desempenho.

Agradeço ao amigo Germano T. Chacon e parabenizo-lhe pelo sucesso da primeira etapa deste projeto. Da mesma forma, exprimo meus agradecimentos aos orientadores Nilton Alves Jr. e Márcio Portes de Albuquerque.

Agradeço ao Prof. Ricardo Magnus Osório Galvão, pelo apoio e viabilização inicial do projeto e ao Dr. Andrea Murari (Consorzio RFX, Itália) do Laboratório JET/EFDA, pela proposta de pesquisa e suporte ao projeto no Centro de Ciências de Culham em Abingdon/Inglaterra.

Agradeço ao amigo Jaime Paixão Fernandes Jr., por permitir o desenvolvimento de minhas atividades profissionais em sua sala e compartilhando sua amizade e companhia por todo esse tempo.

Agradeço ao CBPF/MCTI (Ministério da Ciência, Tecnologia e Inovação), às suas coordenações, pesquisadores e tecnologistas que me apoiaram e incentivaram durante esses anos na arte da física e de sua instrumentação.

Agradeço a colaboração entre o CBPF e o Laboratório JET na Inglaterra que cedeu as imagens para o desenvolvimento deste trabalho dentro do convênio de cooperação da Rede Nacional de Fusão (RNF/CNEN).

Agradeço ao professor José Abdala Helayël Neto, por ter aberto as primeiras portas no mundo da física e depositado sua confiança em meu crescimento profissional.

Agradeço aos idealizadores de outros projetos que deram suporte a esse trabalho, dentre eles: Richard Stallman, Linus Torvalds, Brian Kernighan, Ken Thompson, Dennis Ritchie, Donald Kunuth, Leslie Lamport, Brian Fox e especialmente à Patrick Volkerding.

Agradeço aos amigos "*slackers*", parceiros e guerreiros.

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Resumo

**Título: Paralelismo computacional de processamento digital de imagens aplicado à detecção de MARFEs no JET**

O JET (*Joint European Torus*) é um dos principais laboratórios do mundo dedicado à pesquisa em fusão nuclear e suas possíveis viabilidades de uso. O laboratório opera na fronteira das aplicações científicas, onde é necessário desenvolver dispositivos e tecnologias para compreensão do comportamento do plasma nas condições e dimensões de um reator de fusão. O tokamak do JET é um reator na forma toroidal no qual o campo magnético é utilizado para realizar o confinamento do plasma. Uma das tecnologias desenvolvidas no laboratório está ligada à análise, detecção e o controle de um fenômeno denominado de MARFE (*Multifaceted Asymmetric Radiation From The Edge*), que pode ocasionar a instabilidade no plasma e causar a interrupção no processo de confinamento do campo magnético. Com o auxílio de câmeras de vídeo de altíssima velocidade instaladas no reator e o uso de processamento digital de imagens o fenômeno pode ser detectado, tornando possível o estudo de suas correlações com a instabilidade do plasma e atuação na prevenção. O desenvolvimento de algoritmos otimizados de processamento de imagens para análises dos vídeos de alta velocidade no JET, em tempo real, é ainda um importante desafio tecnológico. Esta tese aborda a segunda etapa do desenvolvimento do projeto de colaboração com o laboratório JET, fazendo uso de tecnologias de paralelismo computacional para elevar a taxa média de processamento de imagem obtida anteriormente (650,29 $\pm$ 47,92 frames por segundo). Considerando o desempenho do sistema de classificação de padrões desenvolvidos anteriormente, o compromisso apresentando neste trabalho é o tratamento de pelo menos 10 imagens dentro do intervalo de ocorrência de MARFE, tornado possível alcançar uma taxa de mais de 10 mil imagens processadas por segundo. O documento também expõe as dificuldades encontradas e as soluções propostas para os módulos de processamento de imagens desenvolvidos inicialmente.

**Palavras-Chave:** processamento paralelo; fusão nuclear; fork; thread; speedup; processamento de imagens; instrumentação científica.

# Abstract

**Title: Image processing parallel algorithm applied to MARFE detection on JET**

The Joint European Torus (JET) is one of the leading laboratories in the world dedicated to the investigation of the potential of nuclear fusion power as a safe, clean, and virtually limitless energy source. JET tokamak is a ring-shaped vacuum vessel in which very hot plasma is confined using magnetic fields. The laboratory was explicitly designed to study plasma behavior in conditions and dimensions approaching those required in a fusion reactor, where it is necessary to develop technologies and devices for their purposes. MARFE is a fast phenomenon of instability that appears in tokamaks as a toroidal ring of increased radiation. Since MARFEs cause a significant growth in impurity radiation, they leave a quite clear signature in frames and can be taken by visible spectrum high-speed cameras used in JET. The videos of these cameras provide essential information for the experiment control and for the physical interpretation of the experiments. The development of optimized image processing algorithms for analyze JET high-speed videos in real time is still a technological challenge. In this thesis we present the second stage of the project development in collaboration with the JET laboratory, by making use of parallel computing technologies to increase the image processing average rate obtained before (650.29 $\pm$47,92 frames per second). Considering the performance of the classifier (i.e. more than 9 hits in 10, on average), a good compromise presented in this document is an analysis of at least ten frames within a MARFE period. This would lead to a significant acquisition and processing rate of more than 10 thousand frames per second. The document also outlines the difficulties encountered and the proposed solutions for the image processing modules developed before.

**Key Words:** parallel processing; nuclear fusion ; fork; thread; speedup; image processing; scientific instrumentation.

# Capítulo 1

# Introdução

Os avanços tecnológicos das últimas décadas aliados aos problemas inerentes ao uso de combustíveis fósseis têm estimulado cada vez mais as pesquisas por fontes energéticas alternativas nas quais ainda há muito interesse na energia nuclear. A geração de energia tendo como base a fissão nuclear é uma alternativa usada por muitos países, porém produz resíduos radioativos que não possuem aproveitamento comercial nem tratamento. Esses resíduos são acomodados em áreas específicas para aguardar sua degradação radioativa, o que além de ser um risco à região é extremamente demorado. Nesse cenário, a fusão nuclear se desponta como uma alternativa eficiente e "ecológica" na geração de energia, restando ser tecnologicamente viável. Como nas estrelas, a fusão nuclear de átomos leves é capaz de gerar grandes quantidades de energia a um baixo nível de resíduos e de toxidade dos mesmos.

No final de 2009, um acordo de cooperação foi assinado entre a Comunidade Europeia de Energia Atômica (EURATOM) e o governo brasileiro, integrando os estudos entre os cientistas e disponibilizando acesso aos laboratórios europeus pertencentes ao grupo. O laboratório JET (*Joint European Torus*) desenvolve estudos na área de fusão nuclear usando confinamento de plasma por campos magnéticos em um grande tokamak (acrônimo de origem russa para "câmara toroidal magnética"), buscando a geração de energia de forma limpa e segura [Wes06]. O JET tem como

um de seus principais objetivos servir como base para o estudo da viabilidade da fusão nuclear para um reator de maior porte, no projeto ITER (*International Thermonuclear Experimental Reactor*), que está sendo construído na cidade de *Cadarache*, no sul da França. Mais informações sobre o laboratório JET e o projeto ITER poderão ser encontradas em capítulos posteriores deste documento.

Em 2010 o Centro Brasileiro de Pesquisas Físicas - CBPF formalizou a colaboração com o laboratório JET e iniciou um estudo para a detecção de um fenômeno denominado MARFE (*Multifaceted Asymmetric Radiation From The Edge*) por meio de algoritmos e técnicas de processamento digital de imagens. Uma sequência de imagens do MARFE pode ser vista na figura 1.1. Com o auxílio de câmeras ultra rápidas instaladas no tokamak e técnicas de processamento das imagens obtidas, iniciou-se uma primeira etapa da pesquisa na qual se desenvolveu e se avaliou algoritmos para a detecção do fenômeno. Com o poder computacional disponível e as técnicas de processamento de imagens propostas foi possível atingir a taxa média de tratamento de 650,29 imagens por segundo, conforme foi descrito em [Cha12].

Este trabalho pretende ser uma continuação do sistema desenvolvido anteriormente, se tornando uma segunda etapa com a capacidade de ampliar a taxa média de tratamento de imagens por segundo. Para isso foram incluídas técnicas de processamento paralelo como *fork* e *threads*, memória compartilhada e comunicação entre processos, que serão descritas com mais detalhes neste documento.

## 1.1 Objetivo

A fusão nuclear de átomos leves através do uso de plasma exige a superação de inúmeros desafios, dentre eles a compreensão do fenômeno denominado de MARFE. O MARFE é uma instabilidade no plasma que ocasiona a não sustentabilidade das reações nucleares, podendo levar ao término do processo de geração de energia. A ocorrência do MARFE, aparentemente, não possui prenúncios e sua duração não ultrapassa alguns poucos milésimos de segundo.

Em trabalhos anteriores foram usadas câmeras de altíssimas velocidades acopladas ao tokamak com o objetivo de realizar diversas análises do plasma, dentre elas também estudos sobre o MARFE [Mur10, LCH+09, LW02]. Na primeira etapa do projeto com o CBPF foram desenvolvidas técnicas e algoritmos de processamento digital de imagens também para este fim. Ao final da primeira etapa foi possível realizar análises e classificações das imagens a uma taxa média de aproximadamente 650 imagens por segundo (fps). Deve ser ressaltado que com essa taxa de análise, a velocidade média de processamento é de aproximadamente 1,5 ms (1/650). Considerando que o tempo médio de duração de um MARFE é também de 1,5 ms, é possível perceber que no melhor caso serão analisadas 2 imagens dentro do intervalo de tempo de sua ocorrência. Porém, a situação mais provável será mesmo a análise de apena uma imagem por ocorrência de MARFE. E, finalmente, ainda há a possibilidade de não se analisar nenhuma imagem para algumas ocorrências do fenômeno. Cabe lembrar que ainda que se tenha 100% de acerto na detecção do fenômeno, nesta taxa pode não sobrar um intervalo de tempo suficiente para uma possível atuação externa (e.g. acionamento de unidade de controle do campo de confinamento magnético do plasma).

Um sistema de detecção de MARFEs deve ser analisado em seu desempenho de classificação para que não se comporte indevidamente, i.e. classificando padrões como MARFEs onde eles não deveriam ser detectados. Tendo como base de comparação o banco de dados/imagens catalogadas previamente no JET e comparando com os resultados dos algoritmos desenvolvidos é possível fazer uma previsão do comportamento do sistema como um todo, incluindo o classificador. A porcentagem de falsos positivos detectados pelos algoritmos na primeira etapa do projeto foi de 0,98%. Para os casos dos falsos negativos, onde houve MARFE e o mesmo não foi detectado, a razão atingiu 3,01%. Analisando o processamento das imagens em que se sabia previamente da existência de MARFEs, a taxa de classificações coincidente com o banco de dados do JET foi de 97,9% [AAC+12].

O controle do MARFE ou o conhecimento de suas origens não fazem parte do escopo desse documento, mais informações sobre estes assuntos podem ser encon-

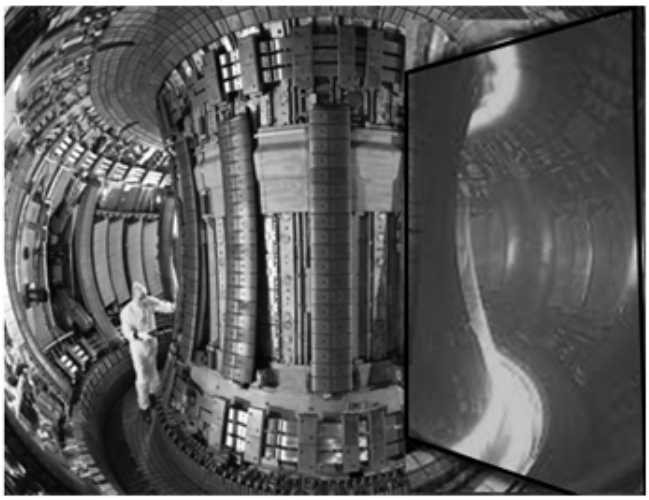
tradas em [LLM+84]. O objetivo desse projeto é promover o desenvolvimento de técnicas de instrumentação científica que aumentem a probabilidade de detecção de MARFEs por meio de um aumento substancial na taxa média de análise e classificação de imagens em comparação com a obtida anteriormente. A ferramenta de base para conseguir o aumento de desempenho é o emprego de técnicas de paralelismo computacional. Com a possibilidade de análise de mais imagens de uma mesma ocorrência haverá maior probabilidade de acerto na detecção do fenômeno, além do maior tempo hábil para uma atuação uma vez que o MARFE poderá ser detectado nas suas fases mais iniciais. Um compromisso colocado pelo projeto é garantir a análise e o processamento de pelo menos 10 imagens de uma mesma ocorrência (dez análises distribuídas ao longo do intervalo de ocorrência de um MARFE, como pode ser visto na figura 1.1(c)). Isso implicaria em um processamento e classificação a cada 130,72 $\mu$s, o que estipula um objetivo mínimo para este projeto em um processamento de pelo menos 7.650 fps.

Considerando que a precisão do algoritmo desenvolvido anteriormente foi bastante satisfatória para um passo inicial do projeto, esse trabalho tem como objetivo também a sustentação daqueles resultados, principalmente com relação à taxa de acertos dos padrões observados. Para isso foram utilizadas as taxas de acertos nas classificações do MARFE, anteriormente obtidas, como um parâmetro de validação do trabalho aqui desenvolvido. Aprimoramentos adicionais nas proporções de acertos das classificações são destinados às etapas futuras.

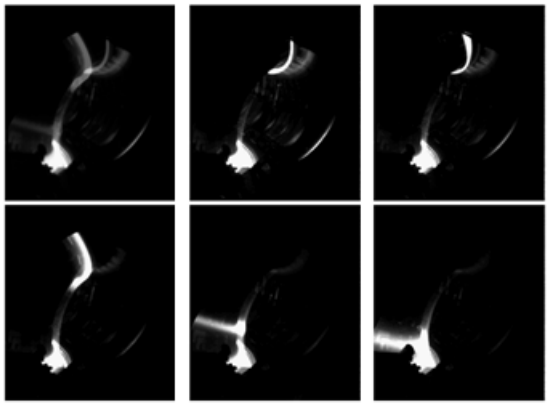
## 1.2 Estrutura do documento

Este documento evolui sob os seguintes temas:

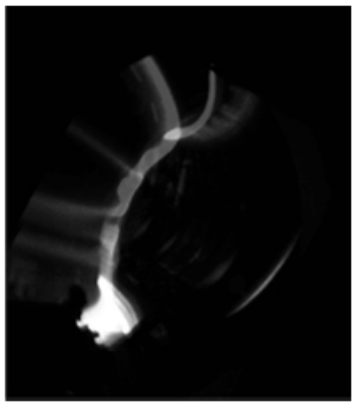
Inicialmente são apresentadas, no capítulo 2, uma introdução sobre o tema da fusão nuclear. As informações apresentadas englobam um histórico sobre o assunto; tais como os primeiros experimentos, datas e os principais cientistas envolvidos. Um embasamento teórico sobre a fusão nuclear e informações sobre a fissão nuclear também estão presentes nesta etapa. Dando continuidade, são também apresenta-



(a) Vista do interior do tolamak do JET. Em descaque a região capturada pela câmera de alta velocidade.



(b) Da esquerda para adireita, de cima para baixo, a sequência de imagem obtidas de uma ocorrência de um MARFE.



(c) Superposição da sequência de imagens do MARFE. A assinatura do fenômeno pode ser observada na região esquerda da imagem.

**Figura 1.1:** Descrição visual do tokamak e do MARFE. Fonte: Laboratório JET.

das informações sobre o plasma e seu confinamento magnético. Posteriormente são expostas informações básicas sobre o laboratório JET, sua história e dados técnicos sobre sua atuação e instrumentação.

As tecnologias computacionais referentes ao tratamento e processamento das imagens também são fundamentais para este projeto. Um histórico sobre o paralelismo computacional e as descrições sobre as principais características das tecnologias empregadas são expostas no capítulo 3. Informações sobre *thread*, *fork*, memória compartilhada, *race condition*, *starvation*, *deadlock*, zonas críticas, granularidade,

lei de Amdhal e um histórico da evolução do paralelismo computacional estão contidos nesse capítulo.

Como a proposta do projeto é dar continuidade a um trabalho desenvolvido anteriormente, é imprescindível uma exposição das etapas anteriores. O capítulo 4 apresenta informações sobre os tratamentos desenvolvidos e os resultados obtidos na versão serial do algoritmo de processamento das imagens e detecção do MARFE. O capítulo também aborda algumas melhorias feitas ainda na versão serial do algoritmo.

Na sequência, o capítulo 5 expõe os estudos desenvolvidos neste projeto com o objetivo de se atingir os patamar de desempenho proposto (ter pelo menos dez imagens processadas com uma alta taxa de acerto dentro do intervalo de tempo de ocorrência de um MARFE). Uma abordagem da metodologia para medições de tempo em computadores se mostrou importante, além das informações sobre problemas oriundos da mesma metodologia no caso utilizada nos desenvolvimentos anteriores para o processamento em única CPU. Comparativos de *speedups* entre *forks* e *threads*, técnicas de medidas e suas validações podem ser encontradas nesse capítulo.

Por fim, os resultados obtidos com o algoritmo de execução paralela desenvolvido são expostos e discutidos no capítulo 6. Análises e considerações sobre os resultados encontrados, assim como conclusões e propostas futuras são encontradas e avaliadas no capítulo 7.

Os resultados das pesquisas que originaram este documento também foram publicados na revista científica *IEEE Transaction on Plasma Science* na edição de fevereiro de 2013 [AAS+13a] e internamente como nota técnica do laboratório JET [AAS+13b]. Este trabalho encontra-se no Apêndice A deste documento.

# Capítulo 2

# Características, conceitos e história da fusão nuclear

Neste capítulo exibe-se o cenário onde se desenvolveu os estudos, seus embasamentos teóricos necessários ao entendimento da pesquisa e da importância do laboratório JET[1]. Os conceitos físicos e uma breve síntese das instalações físicas e instrumentos são também expostos aqui, além de expor sucintamente a evolução histórica científica da fusão nuclear.

## 2.1 Linha do tempo

Em 27 de setembro de 1905 o físico alemão Albert Einstein publicou um artigo intitulado *"Does the inertia of a body depend upon its energy-content?"* (ou "A inércia de um corpo depende da sua quantidade de energia?")[2], em que propunha a "equivalência massa-energia" como um princípio geral que é consequência das simetrias do espaço e tempo [Ein05]. Nessa publicação ele quantifica a relação en-

[1] JET-EFDA Contributors: See the Appendix of F. Romanelli et al., Proceedings of the 24th IAEA Fusion Energy Conference 2012, San Diego, USA.

[2] O artigo original foi publicado em alemão com o título *"Ist Die Trägheit Eines Körpers Von Seinem Energieinhalt Abhängig?"* na revista *Annalen der Physik*, em setembro de 1905.

tre massa e energia a uma razão proporcional à velocidade da luz ao quadrado que, por ter uma ordem de grandeza elevada, implica em uma conversão de pequenas quantidades de massa em valores energéticos extremamente altos. Esta relação foi descrita pela famosa equação 2.1.

$$E = m.c^2 \tag{2.1}$$

As publicações de Einstein incentivaram a corrida pelo domínio e controle das reações de fissão nuclear e ao avanço da física nuclear de uma forma geral. A figura 2.2 ilustra o processo de fissão nuclear. O conhecimento e controle deste levou a humanidade às bombas atômicas e às usinas nucleares conhecidas e existentes até hoje.

No ano de 1920 o britânico Francis William Aston descobriu através de medições que a massa do átomo de hélio é ligeiramente menor do que quatro vezes a massa do átomo de hidrogênio. Logo, em 1926, o astrofísico e também britânico, Arthur Stanley Eddington sugeriu em discursos que a fonte de energia das estrelas provinha da fusão nuclear do hidrogênio em hélio [Ili07].

Baseado nas sugestões de Arthur S. Eddington e com o surgimento e suporte da física quântica, o também britânico físico e astrônomo Robert d'Escourt Atkinson propôs em 1929 em parceria com outro físico nuclear, nascido na Prússia, Fritz G. Houtermans, as explicações pelas quais as incríveis temperaturas e pressões no interior do sol poderiam fundir núcleos de hidrogênio gerando hélio e liberando energia em forma de radiação [AH29]. Essas reações seriam o combustível das estrelas e posteriormente, em 1938, Hans Albrecht Bethe e Carl Friedrich von Weizsäcker detalharam o processo de fusão.

A busca pelo controle e geração de energia através da transmutação atômica e não mais por meio de reações químicas se inicia. Muitos estudos foram alavancados nesta época onde primeiramente o domínio da fissão nuclear atingiu sua "maturidade", proporcionando o controle e geração de energia por meio da quebra de núcleos atômicos em uma reação exotérmica nuclear em cadeia.

Em 1950, com a participação do físico e professor Andrei Dmitrievich Sakharov (prêmio Nobel em 1975) integrante da equipe de desenvolvimento de armas termonucleares soviética, a antiga URSS intensificou os estudos sobre a fusão nuclear. Os cientistas Igor E. Tamm, Andrei D. Sakharov e colaboradores formularam os princípios para o confinamento magnético de plasma, que após um protótipo feito com vidro, porcelana, metal e isolantes em 1951, resultou na construção do primeiro tokamak do mundo, chamado de T-1, no instituto Kurchatov em 1958 [Smi09], visto na figura 2.1.

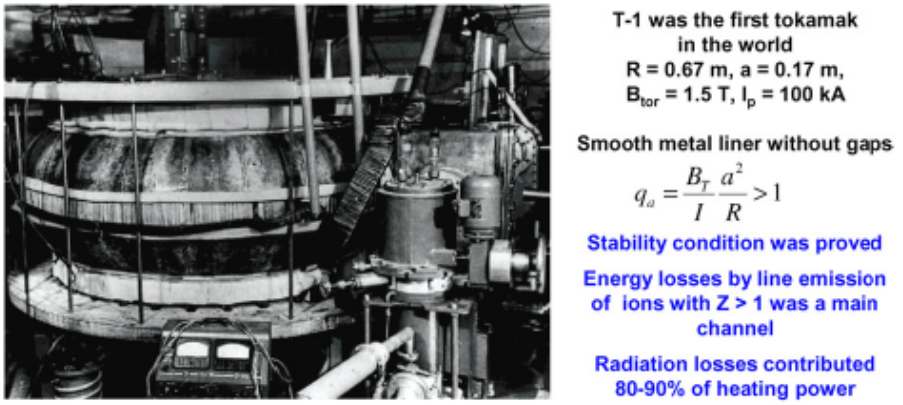



**Figura 2.1:** T-1, o primeiro tokamak construído no mundo, em 1958, conforme informações apresentadas por V. P. Smirnov, [Smi09] na comemoração dos 50 anos de seu desenvolvimento pela antiga União Soviética.

Durante a segunda grande guerra as pesquisas se intensificaram pela produção de energia através de reações termonucleares. Com o desenvolvimento da física do plasma e o surgimento de novas tecnologias a possibilidade de sustentação das reações de fusão nuclear passou a ser considerada.

Ainda em 1958, em Genebra, foi realizada a conferência "Átomos para a Paz". Nessa conferência ficou clara a necessidade de troca de informações para se evoluir na compreensão da física do plasma e consequentemente na fusão nuclear.

Nascida em 25 de março 1957 e sediada no Reino Unido, a Comunidade Europeia de Energia Atômica (EURATOM) foi instituída através do "Tratado de Roma". A EURATOM passou a controlar os programas de pesquisas em fusão termonuclear da Europa.

Em 1978 iniciou-se a construção de um projeto de um Toroide Europeu em conjunto com vários países, denominado JET *"Joint Europen Torus"*, próximo a pequena cidade de Abingdon, no Centro de Ciência de *Culham*, a aproximadamente 80 quilômetros de Londres. No ano de 1983 foi produzido o primeiro plasma no JET e em 09 de abril de 1984 este foi inaugurado oficialmente, atingindo sua estabilidade de funcionamento somente em 1991.

Em 02 de julho de 1999 a EURATOM assinou um acordo com representantes de países da comunidade europeia para fortalecer as pesquisas e colaborações em fusão nuclear, esse acordo foi chamado de EFDA - *Europen Fusion Development Agreement*. Dentre outras definições, este estipula o uso coletivo das instalações dos laboratórios do então EFDA-JET.

Recentemente, através de um acordo assinado entre o governo brasileiro e a EURATOM, em 27 de novembro de 2009, pesquisadores do Brasil passaram a fazer parte do projeto de pesquisa, colaborando com o desenvolvimento de tecnologias no laboratório JET. Umas das ramificações dos projetos de pesquisa realizados no JET possui o CBPF como colaborador por meio das coordenações de Física Aplicada e de Atividades Técnicas (instrumentação científica). Uma das etapas objetivou o estudo de tecnologias e algoritmos de processamento digital de imagens no auxílio à compreensão do MARFE.

A figura 2.3 ilustra de forma básica o processo de fusão nuclear. O uso da fusão nuclear na geração de energia por meio de uma fonte limpa e segura é um dos principais objetivos das pesquisas recentes, mas, finalidades militares e outras possíveis aplicações também são de interesses da comunidade científica mundial.

## 2.2 Conceitos básicos

### 2.2.1 Fissão nuclear

O processo de fissão nuclear foi desenvolvido e aperfeiçoado pela comunidade científica anteriormente àquele da fusão nuclear. Nesse processo os núcleos de átomos pesados são bombardeados, normalmente por nêutrons, causando sua separação em dois ou mais núcleos menores. Geralmente é usado Urânio como combustível nuclear e, após bombardeado, a maioria das vezes o resultado é a geração de átomos de Bário, Criptônio, três nêutrons e liberação de energia. A soma das energias dos novos núcleos mais a energia liberada para o ambiente em forma fotons, mais as energias cinéticas dos produtos de fissão e dos nêutrons liberados devem ser igual a energia total do núcleo original após a absorção da energia do nêutron bombardeado.

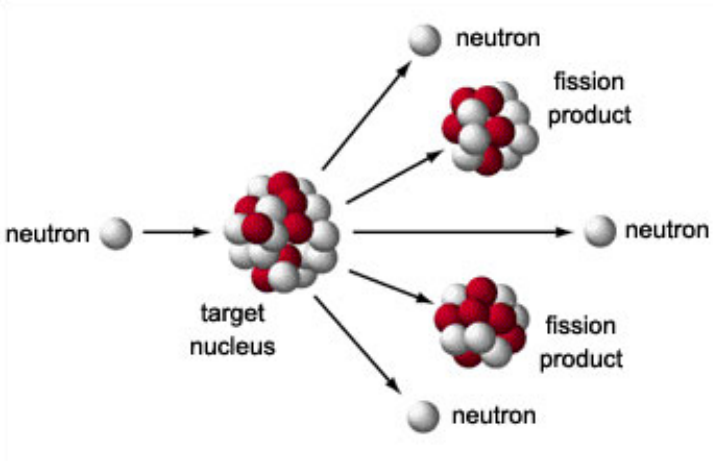



**Figura 2.2:** Fissão nuclear, onde o núcleo de um átomo pesado é atingido por um nêutron com energia cinética suficiente para parti-lo, gerando dois núcleos menores, nêutrons excedentes e liberando energia. Fonte: atomicarchive.com.

Os resíduos do processo de fissão nuclear são de alta toxidade e altamente radioativos. Estes resíduos não são destinados a nenhum tipo de aplicação, não existe tratamento conhecido, exceto, a estocagem em territórios isolados por longos períodos, aguardando o decaimento radioativo natural dos elementos, que pode demorar centena ou milhares de anos.

### 2.2.2 Fusão nuclear

A fusão nuclear é um fenômeno conhecido de longa data. Na verdade, a liberação de energia por meio de reações termonucleares de fusão é o que nos mantém vivos. Devido as altas temperaturas e pressão gerada pela gravidade, o processo de fusão nuclear ocorre espontaneamente no interior das estrelas, liberando grandes quantidades de energia e nos mantendo aquecidos pela nossa estrela mais próxima, o Sol.

Um estudo do físico e químico neozelandês Ernest Rutherford provou que os núcleos atômicos são extremamente pequenos, ainda assim os núcleos precisam ter mais que 83 prótons para serem instáveis. A conclusão de Rutherford reside na existência de uma forma de energia bastante forte que atua apenas à pequenas distâncias e mantém os prótons unidos nos núcleos de pequenos volumes denominada "Energia de Empacotamento". A magnitude dessa energia é centenas de milhões de vezes maior do que a energia de uma ligação química forte (iônica).

Em 1919 o físico e químico Francis William Aston realiza medidas precisas das massas atômicas de quase todos os isótopos que ocorrem espontaneamente na natureza. Seus estudos mostraram que os núcleos de dimensão média, próximas as do ferro, são os mais empacotados. A energia de empacotamento seria liberada se esses novos núcleos fossem produzidos a partir de reações envolvendo núcleos extremamente leves ou extremamente pesados. Na realidade, seus estudos já apontavam para o fato de que fusão nuclear (núcleos leves) produziria muito mais energia por unidade de massa do que a fissão de núcleos pesados.

Com o favorecimento de elementos leves na geração de energia, o hidrogênio é o elemento usado como combustível para os reatores de fusão, mais especificamente os hidrogênios pesados Deutério e o Trítio. No decorrer das reações tem-se um ambiente com frações misturadas de Deutério ($D^2$), Trítio ($T^3$) e Hélio ($He^3$ e $H^4$), cujas proporções e o tamanho de suas seções de choque caracterizam probabilidades diferente para cada tipo possível de colisão reativa.

A colisão entre dois Deutérios pode resultar em dois cenários diferentes:

- A criação de um Hélio $He^3$ mais um nêutron e liberação de energia;
- A criação de um Trítio $T^3$ mais um próton e liberação de energia.

A colisão entre um Deutérios e um Trítio pode resultar em:

- A criação de um Hélio $He^4$ mais um nêutron e uma enorme liberação de energia.

A colisão entre um Deutérios e um Hélio $He^3$ pode resultar em:

- A criação de um Hélio $He^4$ mais um Hélio $He^3$ e uma enorme liberação de energia.

Devido a sua grande liberação de energia, de uma forma geral o cenário ilustrado na figura 2.3 é o desejado.

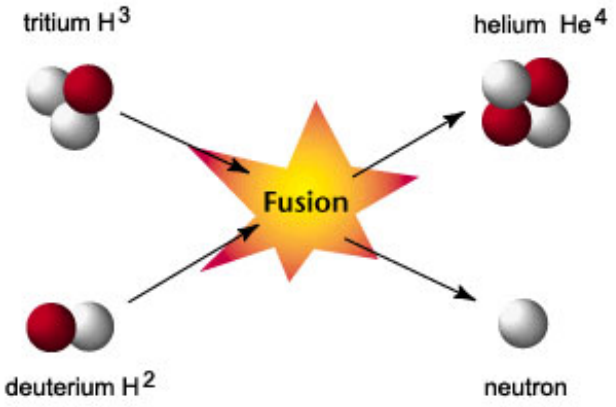



**Figura 2.3:** Fusão nuclear, onde dois núcleos de átomos leves, o Trítio $H^3$ e o Deutério $H^2$, são acelerados e colididos, gerando um átomo de Hélio $He^4$ e um nêutron, liberando energia. Fonte: geoinfo.nmt.edu.

Dentre as vantagens da fusão nuclear, destacam-se [3]:

[3] O texto que se segue foi extraído e adaptado do site do Laboratório Associado de Plasma do Instituto Nacional de Pesquisas Espaciais do Ministério da Ciência, Tecnologia e Inovação (INPE/MCTI): www.plasma.inpe.br.

• A fusão nuclear produz uma quantidade de energia por unidade de massa muito maior que a fissão nuclear.

• Maior segurança. Reações ou liberação de energia fora de controle não são possíveis. No caso de uma falha, o plasma atinge as paredes do reator e esfria imediatamente.

• Não há produtos de combustão química em uma reação de fusão e portanto não há poluição do ar ou da água.

• Não há produção de resíduos radioativos de longa duração. Há radioatividade produzida pelos nêutrons liberados nas reações de fusão quando interagem com a estrutura do reator, mas, decai rapidamente com a escolha apropriada de materiais de baixa ativação para construção do reator.

• Suprimento de eletricidade, diretamente ou através de uma cadeia baseada na utilização de hidrogênio produzido por eletrólise e células de combustível.

• O deutério, usado como combustível nuclear, é extraído da água sendo praticamente inesgotável e estando disponível em todo o mundo. O trítio precisa ser produzido a partir do lítio introduzido no reator. O lítio se encontra distribuído na crosta terrestre e as reservas existentes podem durar por milhares de anos, mesmo que toda a eletricidade seja gerada por fusão. Além disso, com o desenvolvimento de reatores avançados de fusão, operando em temperaturas mais elevadas, o deutério puro poderia ser usado como reagente no futuro e o combustível disponível na Terra duraria por bilhões de anos.

## 2.3 O plasma e confinamento magnético

O plasma é considerado um estado distinto da matéria, em que as composições conhecidas não seriam nem sólidas, nem líquidas e nem gasosas. A ideia básica no plasma do JET centra-se no aquecimento elevado de um gás pode provocando o rompimento de suas ligações moleculares, o que converte sua matéria em seus átomos constituintes. O aquecimento extremo também ocasiona ionização dos átomos suspensos, gerando um conjunto de propriedades não encontradas em materiais em seus estados naturais à temperatura ambiente.

Apesar do plasma imprimir certas complexidades em seu entendimento e de ser um estado da matéria percebido como tal de uma forma relativamente nova, alguns autores afirmam que mais de 99% da matéria visível de nosso universo se encontra neste estado [SFH05, GB05]. O espaço, ou meio interestrelar, é preenchido por um tipo de plasma, as estrelas são constituídas basicamente por plasma e existe plasma em inúmeras manifestações na terra.

A transição de um gás para plasma não é definida em termos de uma temperatura exata. Os valores dos plasmas usados nos laboratórios do JET encontram-se nesse patamar. As características de plasma podem ser encontradas em gases a poucos milhares de graus Celsius, por exemplo: a 2000°C as moléculas de gás se tornam atômicas e ionizadas; a 3000°C o gás assume uma viscosidade similar à um líquido a pressão atmosférica e as cargas elétricas livres conferem condutividades elétricas relativamente altas que podem se aproximar das apresentadas por metais [GRC+08].

A proporção de elétrons livres e portadoras positivas no plasma ionizado o torna um condutor elétrico, consequentemente respondendo a campos elétricos e magnéticos. Essa característica é explorada para promover seu confinamento, que permite sua manipulação. Sem suas características magnéticas a sustentação do plasma se tornaria impraticável, uma vez que sua temperatura pode superar os 200 milhões de graus Celsius (de 10 a 30 vezes maior que a temperatura estimada para o interior do nosso sol) e poderia pulverizar outro material em contato com ele.

O confinamento do plasma dentro dos tokamaks se dá por meio de intensos campos magnéticos e sofisticados sistemas de controle desses. Tanto nos tokamaks toroidais quanto nos esféricos são necessários poderosos eletroímãs e sofisticados sistemas de controle para se manter o plasma confinado.

Usualmente, os raios dos tokamaks variam entre 0,3m (para os pequenos) e 3m, sendo valores típicos de campos magnéticos praticados atualmente entre 1T e 5T. Um outro importante parâmetro para o confinamento magnético é a razão de aspecto do tokamak. A razão do reator do JET é de aproximadamente 2,5 [Ama11]. Razões de aspecto inferiores a 2 proporcionam melhor estabilidade do plasma e facilitam as atuações e controle dos fenômenos envolvidos durante o processo de reação nuclear. O futuro reator do projeto ITER possui uma razão próxima à 1,5.

## 2.4 O laboratório JET

No ano de 1970 o "Conselho da União Europeia" decidiu a favor de um programa de fusão robusto e providenciou o quadro jurídico necessário para o desenvolvimento de um dispositivo de fusão nuclear. Três anos depois, começou o trabalho de design para o JET. Em 1977, iniciou-se o projeto para sua construção e, no final do mesmo ano, foi decidido que o Centro de Ciência de *Culham*, em Abingdon na Inglaterra, sediaria o laboratório.

Em 1978, o "*JET Joint Undertaking*" foi criado como uma entidade jurídica. Somente cinco anos depois, em 1983, a construção foi concluída. Em 25 de junho de 1983 o primeiro plasma foi alcançado e em 09 de abril de 1984 sua Majestade a Rainha Elizabeth II inaugurou oficialmente o principal experimento de fusão europeu.

O ano de 1991 foi um marco na história da fusão nuclear e no dia 09 de novembro foi realizada a primeira reação de fusão nuclear controlada de Trítio. Seis anos mais tarde, em 1997, um segundo marco foi alcançado produzindo 16 megawatts

de energia de fusão a partir de 24 megawatts de potência total inserida no reator, representando um aproveitamento de 66%.

Em 1998, os físicos e engenheiros do JET desenvolveram um sistema de manipulação à distância (*Remote Handling*) de certos componentes utilizados no reator, com a qual pela primeira vez foi possível a utilização de apenas mãos artificiais para este fim[4].

Em 1999, o Acordo de Desenvolvimento da Fusão Nuclear (EFDA) foi criado com a responsabilidade do uso futuro do laboratório JET em colaborações. A partir de então, o programa científico do JET passa a ser determinado pelo EFDA. No ano de 2000, deu-se início o uso coletivo das instalações do JET, conforme resultado do EFDA.

Em 2006 o JET começou a operar com as configurações magnéticas ajustadas para o projeto ITER. Em 2011 a primeira parede da câmara de vácuo foi alterada, fazendo uso de compostos como o berílio e o tungstênio, servindo também como base para o projeto ITER.

### 2.4.1 Principais características técnicas

O equipamento central do projeto de fusão nuclear do JET é um grande reator (tokamak) que promove, a partir das reações de fusão nuclear em seu interior, estudos das propriedades físicas para aprimoramentos tecnológicos futuros.

Construído na Inglaterra, o núcleo do tokamak é uma câmara de vácuo onde o plasma de fusão está confinado por meio de fortes campos magnéticos e correntes de plasma (até 4 tesla e 5 mega amperes respectivamente). A configuração de raios (maior e menor) do plasma no interior de toroide são de 3 metros e 0,9 metros respectivamente. O volume total do plasma é de aproximadamente 200 metros cúbicos. O equipamento contém um diversor na parte inferior da câmara de

[4]Com o tempo, os nêutrons de alta energia tornam radiativos todos os componentes e estruturas de apoio do reator. Além disso, as paredes do reator são cobertas de berílio, o qual, pode apresentar um perigo adicional para qualquer pessoa que trabalhe no interior do reator.

vácuo, que permite coletar o calor e gases de escape de forma controlada. Mais informações sobre um diversor podem ser encontradas em [LMMS+98].

Outras características importantes do JET são [5]:

- Injetor de feixes de átomos em plasma com capacidade de até 34 megawatts;
- *Ion Cyclotron Resonance Heating* com capacidade de até 10 megawatts;
- *Lower Hybrid Current Drive* com capacidade de 7 megawatts;
- Um amplo conjunto de diagnóstico com cerca de 100 instrumentos individuais capaz de capturar até 18 gigabytes de dados brutos por pulso de plasma;
- Injector de alta frequência para reabastecimento de plasma e para estudos de estimulação ELM;
- Uma válvula de injeção de gás para estudos de estabilidade do plasma;
- Capacidades únicas de operar com Trítio como combustível com plasma envolto em componentes de Berílio;
- Dispositivos de manipulação remota que permitem que o trabalho de engenharia possa ser realizado no interior da câmara de vácuo, sem a necessidade de acesso tripulado.

A figura 2.4 exibe uma vista aérea de parte das instalações do JET, próximo a pequena cidade de Abingdon, no Centro de Ciência de *Culham*, a aproximadamente 80 quilômetros de Londres, na Inglaterra.

Em novembro de 1985, durante a Cúpula das Superpotências em Genebra, o então presidente da França François Maurice Adrien Marie Mitterrand, a primeiro-

[5] Texto retirado e adaptado de www.efda.org/jet/jet's-main-features

**Figura 2.4:** Vista aérea de parte do laboratório JET (*Joint European Torus*), próximo à pequena cidade de *Abingdon*, a 80 quilômetros de Londres, na Inglaterra. Fonte: www.efda.org/jet.

ministro do Reino Unido Margaret Hilda Thatcher e o secretário geral da antiga União Soviética Mikhail Serguéievich Gorbachev propuseram ao então presidente dos Estados Unidos da América Ronald Wilson Reagan um projeto internacional que visava o desenvolvimento da energia de fusão para fins pacíficos. Nascia então o projeto ITER (*International Thermonuclear Experimental Reactor*). Atualmente, um dos objetivos do JET é servir como base de estudo da viabilidade da fusão nuclear para um reator de maior porte no projeto ITER. Com suas obras em andamento na cidade de *Cadarache*, no sul da França, o ITER receberá aprimoramentos científicos e tecnológicos fruto dos estudos realizados nos laboratórios do JET. A figura 2.5 mostra o pátio de obras do projeto ITER.

**Figura 2.5:** Vista aérea do pátio de obras do projeto ITER (*International Thermonuclear Experimental Reactor*), sendo construído no sul da França, na cidade de *Cadarache*. Fevereiro de 2013. Fonte: www.iter.org.

As figuras 2.6 e 2.7 expõem a magnitude física dos reatores destes projetos.

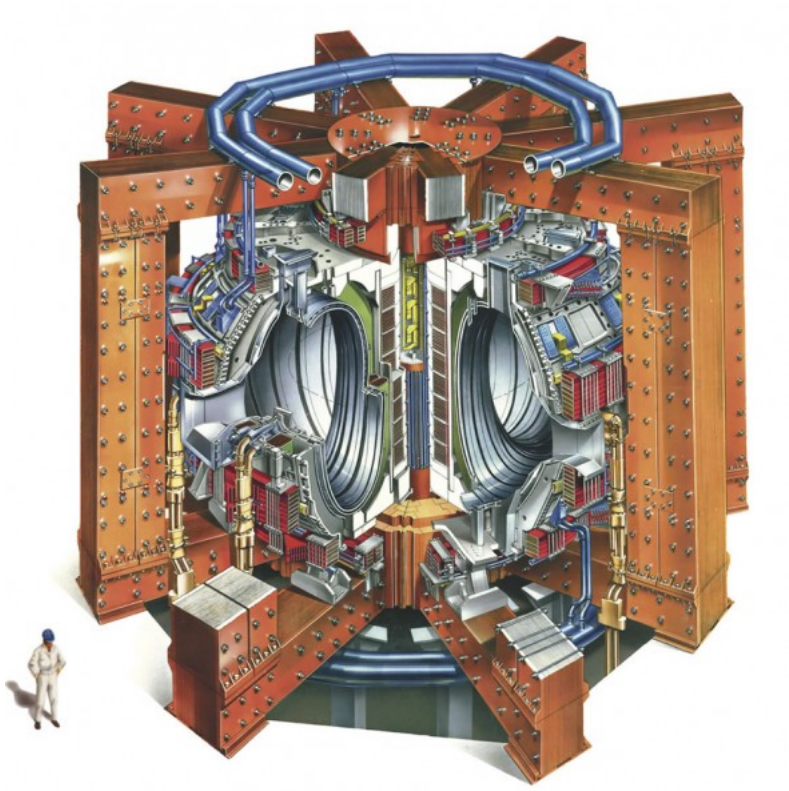


**Figura 2.6:** Dimensões do tokamak do laboratório JET em relação a um homem adulto. Fonte: www.efda.org/jet.

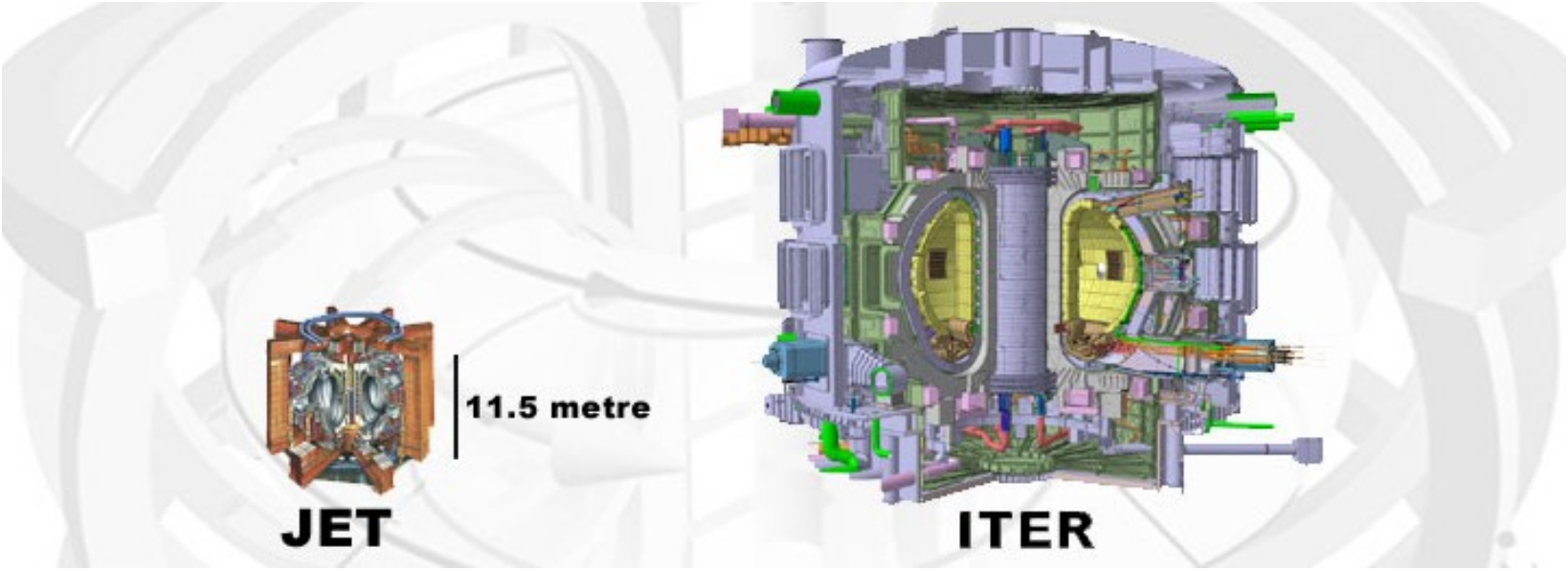



**Figura 2.7:** Dimensões do atual tokamak do laboratório JET em relação ao futuro tokamak do projeto ITER. Fonte: www.iter.org .

# Capítulo 3

# Paralelismo computacional

Este capítulo apresenta e evolução tecnológica na busca por execução simultânea de tarefas. O detalhamento de algumas destas tecnologias, que foram usadas no projeto com o laboratório JET, é exposto nesta etapa assim como alguns conceitos teóricos e complicações a serem observadas.

## 3.1 Evolução do paralelismo computacional

A construção de rotinas e algoritmos destinados à solução de problemas é tipicamente e historicamente um procedimento de escrita serial. Os passos necessários são organizados sequencialmente em forma de rotinas e chamados de sistema para que se atinja o objetivo desejado quando executados por um processador. A velocidade de execução da tarefa é proporcional à velocidade de processamento ou poder computacional do mesmo. Uma limitação na evolução do poder computacional das unidade processadoras implicaria diretamente na limitação temporal de execução de tarefas, exigindo a criação de alternativas para solucionar problemas mais complexos.

Após décadas de um acentuado crescimento no desempenho dos processadores, tem-se observado um travamento nessa evolução. Para sustentar o crescimento no poder de processar dados ao longo da história um grande aparato tecnológico foi exigido e pode ser considerado como revoluções tecnológicas. Nestas revoluções, tecnologias foram extintas dando lugar a novas formas mais avançadas de trabalhar o bit de informação. Inicialmente a tecnologia do cartão furado deu lugar aos "velozes" relés que por sua vez foram substituídos pelas válvulas e sucessivamente pelos transistores discretos e circuitos integrados posteriormente.

A lei de Moore, que originalmente referia-se ao número de transistores por volume financeiro [Moo65], tornou-se um parâmetro não oficial para a estimativa de desempenho sustentando-se desde 1965 devido às revoluções tecnológicas. Observa-se que quando uma tecnologia atinge um fator limitante uma outra tecnologia surge e mantem válida a estimativa de Moore.

A característica que recebia a principal atenção no aumento do poder computacional dos processadores é seu *clock*. As mudanças de paradigmas que proporcionaram às tecnologias a manutenção da taxa de crescimento da capacidade computacional tiveram como principal objetivo o aumento na frequência interna (*clock*) dos processadores, o que proporcionou a capacidade de executar um maior número de instruções por segundo.

A redução do tamanho físico dos componentes eletrônicos foi, durante muito tempo, uma das principais estratégias para o aumento do *clock* interno dos processadores. Essa miniaturização dos componentes, tendo como alvo principal o tamanho dos transistores internos, encontrou uma grande barreira ao aproximar-se ao nível do átomo. Fenômenos quânticos estão assumindo proporções suficientemente grandes para disputarem espaço com os fenômenos elétricos conhecidos e usados no tratamento do bit dentro de um processador, os quais limitam o crescimento de sua frequência interna. A limitação no aumento do poder computacional pode ser percebida claramente, próximo e a partir do ano de 2005, nas pesquisas da "Standard Performance Evaluation Corporations" com medições para cálculos sob pontos flutuantes e inteiros, respectivamente nas figuras 3.1 e 3.2. Alguns especialistas acre-

ditam que é necessário outra revolução tecnológica para dar continuidade à taxa de crescimento do poder computacional observada até os dias de hoje [FM11].

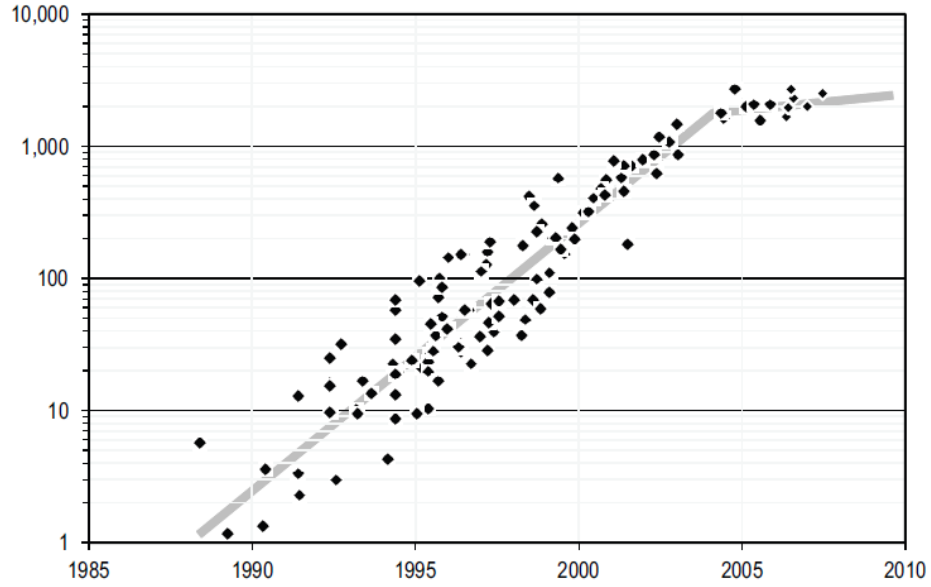



**Figura 3.1:** Gráfico linear/log da evolução do poder computacional ao longo dos anos para cálculos com **ponto flutuante**, normalizado pela capacidade computacional de um computador do ano de 1985. Fonte: www.spec.org.

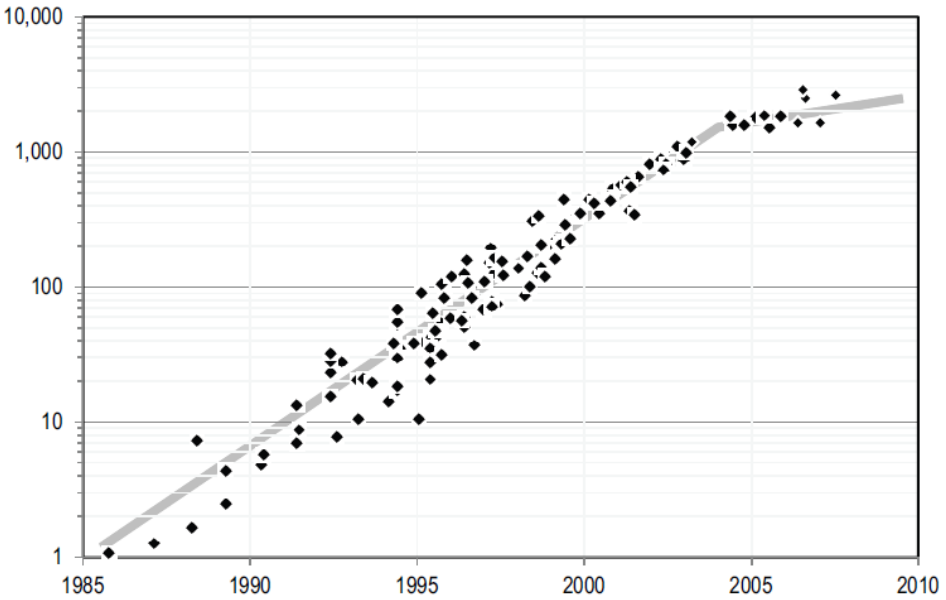



**Figura 3.2:** Gráfico linear/log da evolução do poder computacional ao longo dos anos para cálculos de **inteiros**, normalizado pelo poder computacional de um computador de 1985. Fonte: www.spec.org.

Há décadas estuda-se a computação paralela e seus aprimoramentos. Com a estagnação do aumento do *clock* dos processadores, próximo a 2005, como visto na

figura 3.3 a computação paralela assumiu um papel de destaque no cenário do aumento do poder computacional. A partir desse momento a corrida pelo aumento de frequência dos processadores se transformou em uma corrida pelo aumento do número de núcleos de processamento em um mesmo circuito integrado [ABD+09] interligados por uma rede "intra-chip" nomeada chips *many-core* [SCS+08]. Esse cenário trás para dentro dos computadores pessoais as tecnologias empregadas em *clusters* de computadores, otimizando o uso dos ambiente *multi-core*.

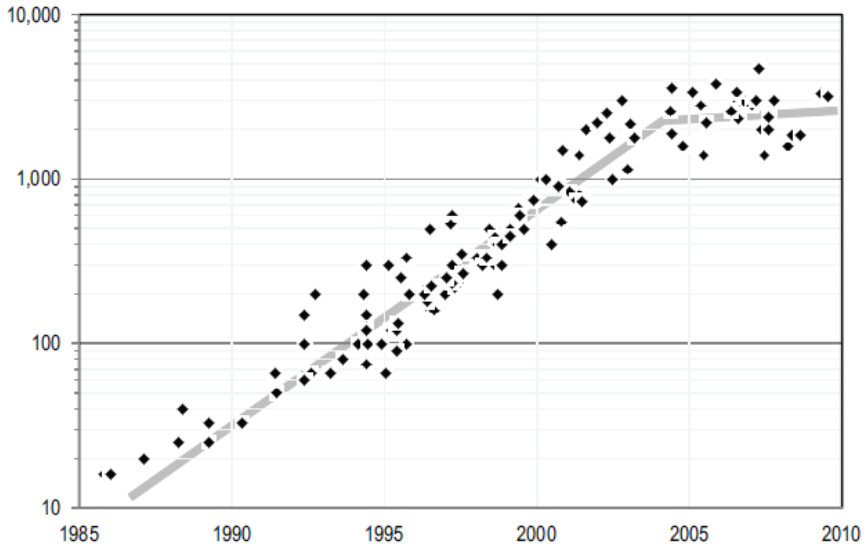



**Figura 3.3:** Gráfico linear/log ao longo dos anos da evolução do ***clock*** interno de processadores em **MHz** (milhões de ciclos por segundo). Fonte: www.spec.org.

Várias técnicas podem ser usadas para promover algum nível de paralelismo na resolução de problemas, uma delas é o *pipeline*. Possivelmente o *pipeline* é uma das técnicas mais antigas na busca pelo paralelismo em execução de instruções. "Projetistas de computadores têm explorado formas de se sobrepor operações desde os anos cinquenta." [RL77]

Ainda na década de 60, o *pipeline* foi introduzido nas arquiteturas de processadores e posteriormente a Intel utilizou esse recurso para melhorar o desempenho do 386, criando o 486. Essa tecnologia consiste basicamente em dividir a execução de uma instrução em vários estágios e cada um deles processa uma etapa da instrução por ciclo de *clock*. Dessa forma várias instruções podem ser executadas simultaneamente, desde que cada instrução esteja em etapas diferentes.

Algumas instruções podem possuir algum tipo de dependência, obrigando ao processador a fazer uma pausa na execução daquela instrução (*pipeline stall*). Essas pausas na execução também são conhecidas como bolhas e, apesar de depreciar o desempenho teórico final, ainda permitem que essa tecnologia atue como otimizador de desempenho.

O uso de *pipeline* permite mais de uma instrução possa ser trabalhada ao mesmo tempo pelo processador e, ao final de uma sequência de instruções (tamanho do *pipeline*), outras tarefas já se encontram preprocessadas e em fase final de tratamento. O uso de *pipelines* maiores permitiu o aumento do *clock* de alguns famílias de processadores, uma vez que as instruções internas eram mais simples e mais rápidas de serem executadas em um único ciclo de *clock*. Um exemplo do uso do aumento do tamanho do *pipeline* para fins comerciais, aumentando a frequência interna dos processadores sem promover ganho de desempenho proporcional, foi a família Pentium4 da Intel que possuía *pipelines* de até 31 estágios (11 estágios para os Pentium3).

A tecnologia de paralelização *pipeline* é implementada em *hardware* e em nível de sistema operacional, exigindo muito pouco ou nenhum conhecimento para usufruir da mesma por parte do programador de aplicativos.

Outra forma de paralelismos é obtida através do escalonamento feito por Sistemas Multitarefas (escalonamento entre processos). Estes fornecem recursos de mudança de contexto que podem promover um sentimento de paralelismo ao oferecer uma fatia de tempo a vários aplicativos, quase que simultaneamente, mesmo em *hardware*s dotados de apenas uma unidade processadora. O processo de interrupção de uma execução e retomada de outra é conhecido como troca de contexto e é gerenciada pelo escalonador do sistema operacional. Em arquiteturas dotadas de mais de uma unidade processadora um real paralelismo é promovido, além do fato de naturalmente reduzir os tempos ociosos da CPU, o que reflete significativamente no desempenho final do sistema.

Os Sistemas Multitarefas são na verdade uma implementação de paralelismo temporal, semelhante ao *pipeline*, porém via *software*. Esses sistemas impõem uma complexidade maior aos usuários mas possuem poucas dependências de *hardware*, o que permite serem aprimorados quase sem a necessidade de troca do pátio de *hardware* ou de aguardar o lançamento de uma nova família de CPUs.

Para os casos de equipamentos dotados de mais de uma unidade de processamento, além do *pipeline* implementado no *hardware* e do escalonamento de tarefas, outras técnicas podem ser empregadas objetivando o real aproveitamento de múltiplas CPUs. A divisão do fluxo de dados em blocos de programa independente pode promover um paralelismo mais efetivo (paralelismo de tarefas). Essa técnica está sujeita a diversos prerrequisitos como a existência de etapas independentes (o que nem sempre é possível) e uma especial atenção no tratamento de informações compartilhadas entre as diferentes etapas em execução. As figuras 3.4 e 3.5 ilustram as duas principais técnicas de paralelismos via *software* citadas (escalonamento de processos e paralelismo de tarefas).

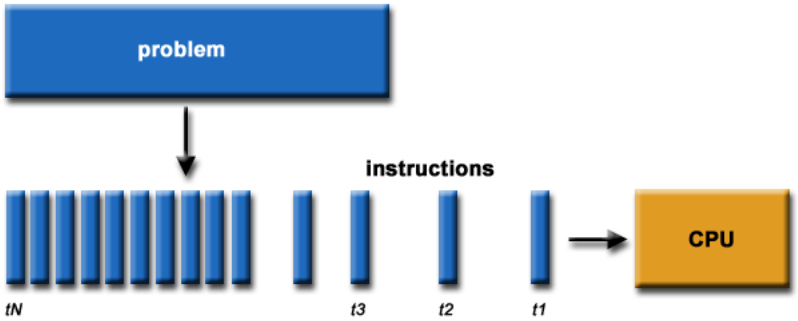



**Figura 3.4:** Exemplo de divisão de uma tarefa em fatias de tempo, permitindo o escalonamento pelo sistema operacional com outras tarefas. Fonte: computing.llnl.gov.

Muitos problemas/tarefas não permitem sua divisão em partes independentes, um exemplo disso é o cálculo da série de Fibonacci em que o elemento seguinte depende do conhecimento dos dois elementos imediatamente anteriores a ele.

O cenário encontrado mais comumente quando se trata de computação paralela são problemas parcialmente paralelizáveis. Nesses casos, as partes independentes são entregues a unidades processadoras diferentes e há necessidade de sincronismo entre as etapas, forçando-as a aplicar o correto tratamento à correta fonte de dados.

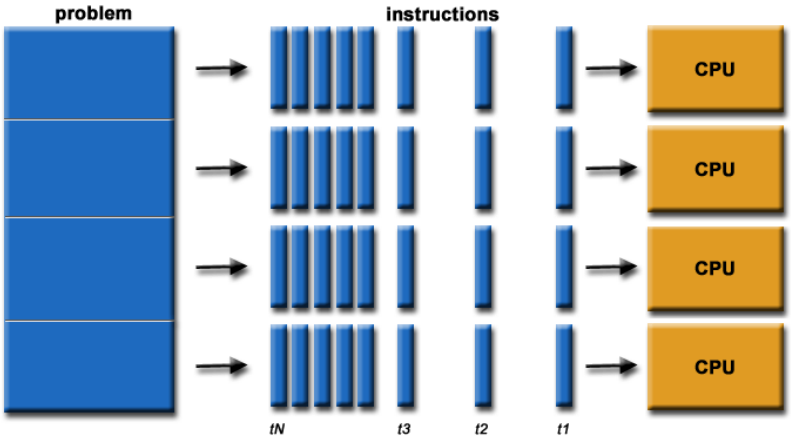



**Figura 3.5:** Exemplo de divisão da tarefa em partes independentes (permitindo o paralelismo de execução), subdivididas no tempo para permitir o escalonamento pelo sistema operacional. Fonte: computing.llnl.gov

A figura 3.6 ilustra um problema parcialmente paralelizável, onde algumas etapas necessitam que outras tenham cumprido certa parte da tarefa previamente.

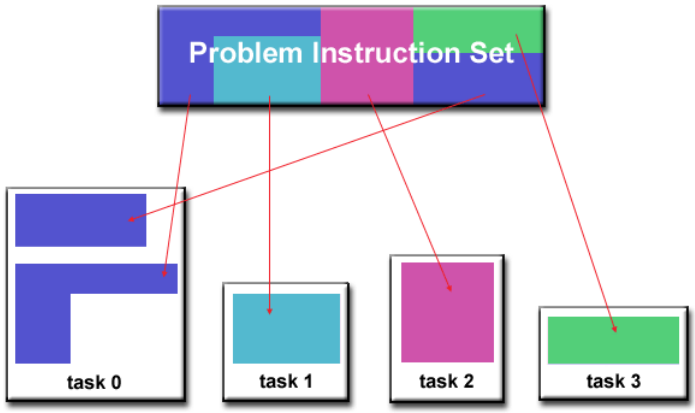



**Figura 3.6:** Exemplo de problema **parcialmente paralelizável**. Divisão do problema em partes **semi-independentes** (permitindo o paralelismo de execução), subdivididas no tempo para permitir o escalonamento pelo sistema operacional. Fonte: computing.llnl.gov.

Alguns problemas decorrentes de dependências no ciclo de execução (decaimento do *speedup*, Amdhal) e compartilhamento de informações (zona crítica) serão abordados em capítulos posteriores, bem como propostas de soluções e prerrequisitos das mesmas.

Muitas tecnologias podem ser empregadas para dividir e enviar o fluxo de execução de um problema para o processamento paralelo de suas etapas, inclusive fazendo uso de CPUs externas ao processamento central, conectadas via rede de compu-

tadores. O uso de unidades de processamento externas ao processamento central pode ser classificado como uma arquitetura MIMD (*Multiple Instruction Multiple Data*) de Flynn com memória distribuída, cada unidade com sua própria memória (node), também conhecida como "multicomputadores" [Pit08].

A aplicação alvo dos estudos aqui presentes é o aprimoramento de um específico programa de processamento de imagens rápidas em arquitetura de memória local compartilhada, multiprocessadores, tornando-o mais ágil com o uso de programação paralela para processar um maior número de imagens por segundo. Como os fenômenos envolvidos e os cálculos usados no tratamento das imagens são extremamente rápidos, as técnicas de programação paralela que envolvem comunicação via redes de computadores foram desconsideradas devido aos altos valores de latências envolvidas nesse processo.

Esse documento abordará as duas principais técnicas de paralelização em arquitetura de memória compartilhada, a tecnologia de multiprocesso *fork* e de processos leves *thread* .

## 3.2 *Fork*

Em geral, os computadores que possuem sistemas operacionais multitarefa disponibilizam um conjunto de funções para a divisão e compartilhamento do(s) processador(es) e da memória. Estes sistemas costumam disponibilizar chamadas ao *kernel* que possibilitam a criação de múltiplos processos. Se a máquina tem mais de um processador, o sistema operacional distribui os processos pelos processadores.

Em sistemas GNU/Linux e nas variantes do Unix, há a possibilidade de um processo ser clonado, com a função *fork*. A comunicação entre os processos pode feita de forma simplificada com o uso de *pipes* [HH97]. Outras tecnologias também podem ser citadas no suporte a troca de informações entre processos, tais como: arquivos (*files*), *shared memory* (shm) [Jon10], *inter process communication* (ipc)

[IBM06], semáforos (sem) [Dij68], passagem de sinais (*signal*) [Gai84], *sockets* [Jon06] dentre outros, mas não serão aprofundados nesse documento.

Basicamente, uma chamada à instrução *fork* dentro de um programa cria uma cópia exata a partir do ponto em que a instrução foi inserida e dois processos idênticos são executados simultaneamente. O processo principal, o qual chamou a instrução *fork*, é denominado processo pai e os processos que são gerados pelo pai são chamados de processos filhos. Os processos filhos ao serem criados ganham um novo espaço de memória onde terão variáveis globais diferentes das variáveis do processo pai, ainda que possuam os mesmos nomes na programação. Além disso, todos os processos podem ter acesso a uma região memória compartilhada especialmente criada onde poderão compartilhar (ler, escrever e editar) informações.

## 3.3 Memória compartilhada

A necessidade de segurança dos dados processados por sistemas multitarefas evoluiu em gerenciamentos complexos de memória com rígidas regras de acesso às regiões de dados e de programa, restringindo o acesso a informações internas de um processo para outros processos distintos.

Em computação paralela a memória compartilhada pode ser definida como um método de comunicação entre processos (*inter-process communication* - IPC). Um dos processos criará uma área em memória na qual outro processo poderá acessá-la, figura 3.7. Uma vez que ambos processos possuem acesso à região de memória compartilhada, como uma memória de uso regular, é criada uma ponte de comunicação extremamente rápida entre as partes (em comparação com métodos como *pipes*, *unix sockets*, etc). No entanto, a necessidade do acesso físico às memórias força a execução destes processos na mesma máquina (não usando a rede), tornando-o um formato de comunicação entre aplicações mais restrito.

As bibliotecas dinâmicas são normalmente utilizadas e mapeadas para os múltiplos processos. A norma POSIX do UNIX padronizou uma API para uso da memória

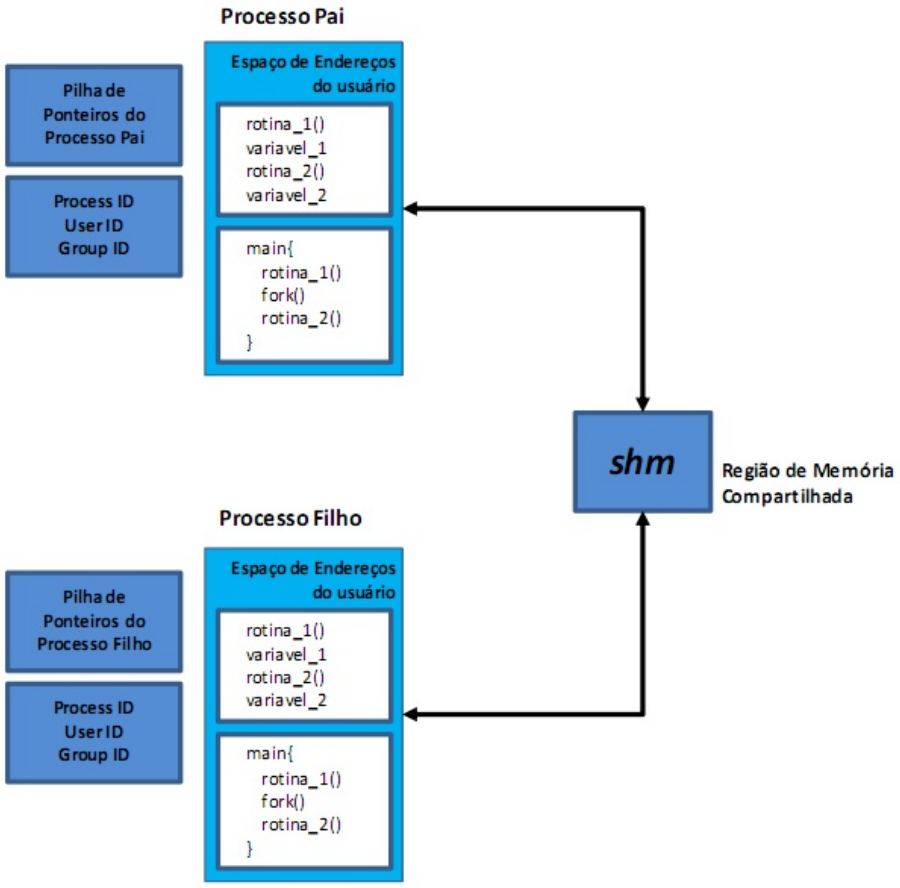



**Figura 3.7:** Comunicação entre processos distintos por memória compartilhada (shm).

compartilhada (POSIX *Shared Memory*). Esta usa as funções shm_open (sys/mman.h). As comunicações entre processos na arquitetura POSIX (POSIX:XSI Extension) inclui funções para manuseio de memória compartilhada como: shmat, shmctl, shmdt e shmget.

Em 1986, Kai Li e P. Hudak propuseram um modelo de memória compartilhada distribuída (DSM) entre computadores conectados pela rede [LH89]. Esse modelo estende as funcionalidades do uso de memória compartilhada entre processos de forma a possibilitar a inclusão de um número maior de CPUs envolvidas por meio de comunicação via rede, figura 3.8. O uso de comunicação de rede na troca de informações entre os processos promove uma grande latência na execução dos mesmos. Em virtude das características dos fenômenos físicos envolvidos na aplicação a qual se objetiva esse documento, a tecnologia de DSM é apenas citada mas não será abordada.

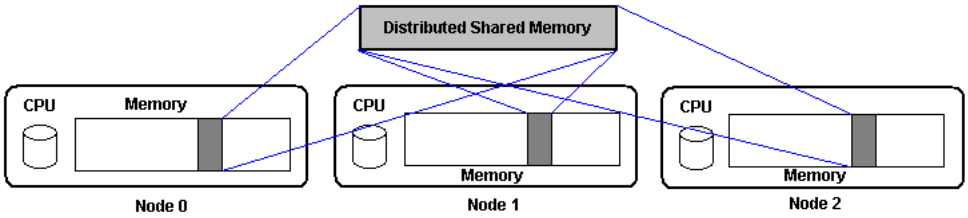



**Figura 3.8:** Ilustração de uma região de Memória Compartilhada Distribuída, onde mais de um *host* compartilham uma mesma região de memória de programa pela rede.

## 3.4 *Thread*

*Threads*, ou processos leves, é a tecnologia que permite que tarefas possam ser executadas simultaneamente, em geral, em arquiteturas multiprocessadas. Em arquitetura de memória compartilhada, *threads* pode ser uma forma simplificada de se implementar um paralelismo, dividindo um processo em fluxos de execuções internos distintos.

O suporte a *threads* deve ser oferecido pelo sistema operacional, e as aplicações devem ser implementadas fazendo uso de alguma biblioteca para este fim. Historicamente os fabricantes dos *hardwares* ofereciam suas próprias implementações de *threads* e estas se diferenciavam substancialmente umas das outras, dificultando o desenvolvimento por parte dos programadores. Na tentativa de minimizar as variações existentes nas diferentes implementações de *threads*, foi desenvolvida uma padronização em 1995. Neste trabalho foi usada a implementação da biblioteca “pthread”, padronizada pelo padrão POSIX do IEEE (IEEE POSIX 1003.1c standard) [IEE04] , que possivelmente é a biblioteca mais difundida para esse fim [Bar12].

Ao se dividir um processo em fluxos distintos de execução, criando *threads* distintas, as *threads* passam a usufruir do compartilhamento de memória de programa e dados do processo originário. Como consequência, cada *thread* pode acessar qualquer posição de memória dentro do espaço de endereçamento do processo, figura 3.9. Essa característica permite a uma *thread* ler, escrever ou até apagar informações usadas por outras *threads*, exigindo um maior cuidado por parte do programador no manuseio e tratamento dos dados e tarefas do programa.

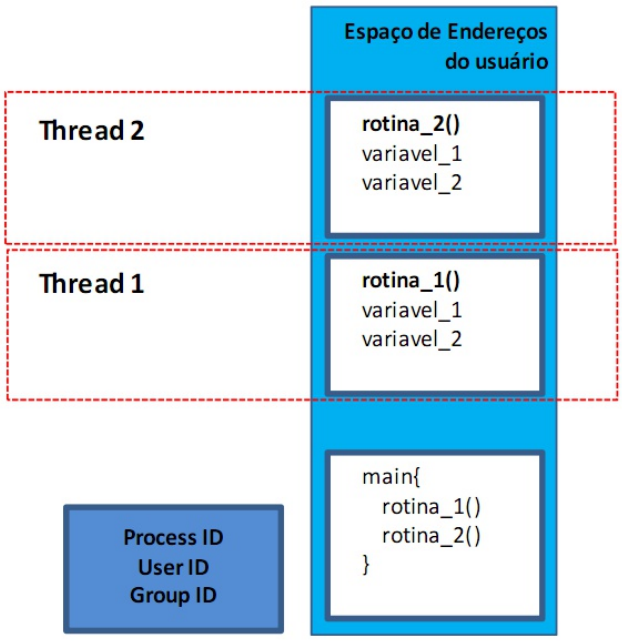



**Figura 3.9:** Ilustração de endereçamento de variáveis globais e locais em *threads*, compartilhadando a mesma região de memória de programa.

O compartilhamento do endereçamento em memória torna as *threads* mais ágeis no processo de troca de contexto pelo escalonador do sistema operacional em comparação com processos independentes (*forks*), gerando uma taxa reduzida de *overhead* e facilitando todo processo de gerenciamento por parte do núcleo do sistema. Essa característica pode impactar diretamente no desempenho das aplicações, uma vez que o sistema pode executar a troca de contexto dezenas de vezes por segundo ou até mesmo centenas de vezes por segundo.

## 3.5 Problemas relativos a sincronismo de dados e de tarefas

Ao se dividir uma tarefa em pedaços menores, inúmeros problemas poderão surgir, dependências entre as partes devem ser tratadas, conflitos e concorrências a dados devem ser previstas e cautelosamente gerenciadas. O acesso aos dados corretos, no momento exato é vital para a devida resolução da tarefa, exigindo o uso de técnicas e tecnologias apropriada para estes fins por parte do programador.

Exceto nas raras e improváveis situações onde um problema maior pode ser dividido em etapas menores de forma completamente independente, será necessário a implementação de algum tipo de sincronismo e troca de informações entre as tarefas menores. Tal comunicação poderá ocorrer de forma indireta, em que um outro processo (gerente) ou tarefa se encarrega de transmitir e receber as informações entre os envolvidos, ou diretamente entre as subtarefas.

Em uma implementação de *software* inadequada os três cenários expostos a seguir, *Race Condition*, *Starvation* e *Deadlock*, são os principais a serem observados e provavelmente os de maiores ocorrências.

### 3.5.1 Condição de corrida

Devido a natureza dos problemas (muitos deles) uma ordem exata de acontecimentos deve ser seguida para o sucesso da resolução final. Uma característica especial e indesejada ocorre quando, em uma troca de informação por mais de uma ação a ordem desejada dos acontecimentos não ocorre adequadamente. Esse cenário é conhecido como *race condition* ou condição de corrida e é causado principalmente pelas trocas de contexto executadas por sistemas operacionais "multitarefa".

O ato aparentemente simples de se incrementar uma variável implica em "uma" ação do processador composta por várias etapas internas (acessos aos registradores, decodificação de endereços, operações matemáticas). Durante a escrita na memória o sistema operacional pode escalonar o processo no meio da ação, entregando uma fatia de tempo de CPU a outro processo que, em caso de acesso à mesma variável em questão, irá receber um dado errado pois a informação contida ainda não foi atualizada pelo processo anterior (mas deveria).

A figura 3.10 exemplifica uma atribuição errônea para a variável "Var_A" causada pelas mudanças de contexto executadas pelo escalonador do sistema em um algoritmo sem o devido tratamento. Neste exemplo, a falta de um mecanismo de sincronização entre as ações acarretará em um resultado errôneo da execução do

programa, que pode gerar até mesmo a interrupção forçada antes da execução de todas as suas etapas esperadas.

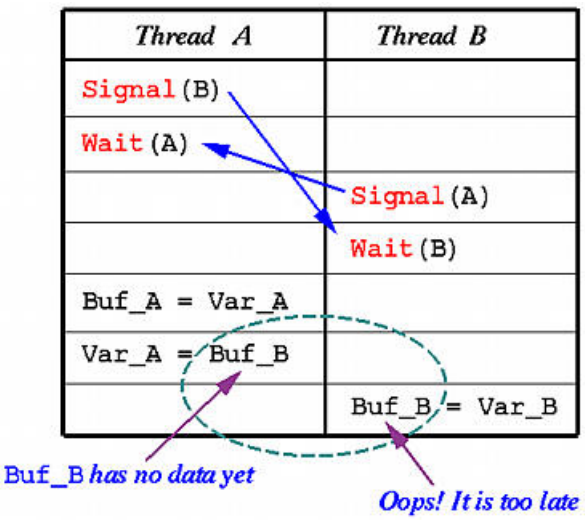



**Figura 3.10:** ***Race Condition***. Erro na atribuição da variável "Var_A" devido a uma condição de corrida não atendida, onde o valor de "Buf_B" deveria ter sido armazenado anteriormente à atribuição de Var_A. Fonte: *Department of Computer Science - Michigan Technological University*; www.mtu.edu.

### 3.5.2 *Starvation*

Outros problemas decorrentes da ausência de um mecanismo de sincronismo apropriado ou causado por concorrência de dados por algoritmos paralelos também devem ser considerados.

Em algumas situações, uma espera (*wait*) de um processo por uma determinada condição pode se estender ininterruptamente. Inúmeros são os motivos que podem ocasionar uma deficiência de comunicação entre um processo em estado de *wait* e sua condição que o tiraria deste estado. Um processo poderia estar aguardando um sinal, uma resposta de um outro processo que por algum motivo já se encerrou. Este processo em *wait* não sairia deste estado, possivelmente ele interromperia a linha de execução de todos os demais processos subsequentes. A este cenário de estagnação da execução dá-se o nome de *Starvation*.

O exemplo de execução visto na figura 3.10 somente **não** ocasiona um *starvation* na execução de *"A"* devido ao fato de que estas mensagens trocadas entre as *threads* , na verdade, são enviadas à uma terceira "entidade" não ilustrada (uma variável de memória compartilhada, por exemplo), que faz o papel de intermediador nas comunicações. Observa-se que ao enviar um sinal à *"A"*, a thread *"B"* encontra *"A"* em aguarda por sua mensagem, que por sua vez dá procedimento à suas tarefas. O retorno (resposta de *"A"*) não ocorreria nunca, uma vez que *"B"* se colocou em estado de espera posteriormente ao envio de mensagem por *"A"*. Esta figura 3.10 ilustra um erro de Condição de Corrida (*Race Condition*) até sua penúltima linha, e um possível *Starvation* que causaria a não execução da última linha.

A figura 3.11 exemplifica de forma mais clara um caso particular de *starvation*, onde duas *threads* aguardam simultaneamente um sinal, uma da outra, para prosseguir com suas respectivas execuções. Como ambas entraram em estado de espera (*wait*), a resposta da outra e consequentemente a liberação para processamento, o qual nunca irá ocorrer.

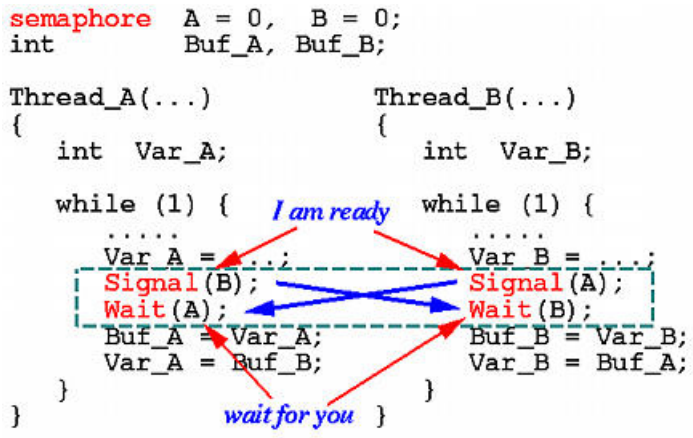



**Figura 3.11:** ***Starvation***. Estagnação da *thread B* devido ao fato da *thread A* ter enviado a comunicação de liberação anterioemnte à *"B"* entrar em seu estado de espera pelo sinal. Como *"A"* não enviará outro sinal, *"B"* permanecerá em *wait* indeterminadamente. Fonte: *Department of Computer Science - Michigan Technological University*; www.mtu.edu.

### 3.5.3 *Deadlock*

Segundo a definição do professor Andrew Stuart Tanenbaum, "Um conjunto de processos estará em situação de *deadlock* se todo processo pertencente ao conjunto estiver esperando por um evento que somente um outro processo desse mesmo conjunto poderá fazer acontecer" [Tan03].

Problemas semelhantes, decorrente da ausência de um mecanismo de sincronismo adequado ou de concorrência de dados por algoritmos paralelos são discutidos mais profundamente em [Hoa78], onde são apresentados e debatidos diferentes exemplos de *deadlock*.

Como visto na seção 3.5.2, a espera infinita de um processo aguardando uma determinada condição para prosseguir com sua tarefa é uma deficiência conhecida como *starvation*. O *deadlock* pode ser considerado um tipo particular de *starvation* e seu conceito pode se estender para outros cenários não computacionais, como humanos, sociais, etc. A figura 3.12 exemplifica uma ocorrência de *deadlock* onde analogamente quatro processos entram em estagnação por aguardar que outro prossiga antes de continuar com suas próprias tarefas. Esse encadeamento de processos impedirá que o fluxo de execução ocorra, mantendo-os bloqueados indeterminadamente ou até uma intervenção externa. Todos esses cenários devem

obrigatoriamente ser tratados pelo programador no desenvolvimento de algoritmos paralelos.

**Figura 3.12:** Ilustração de uma situação de *deadlock* no cotidiano de uma cidade, onde todos os processos pertencentes ao conjunto estão esperando por um evento que somente um outro processo desse mesmo conjunto poderá fazer acontecer. O local corresponde ao cruzamento das Avenidas Brigadeiro Faria Lima com Juscelino Kubischek, em São Paulo/SP.

### 3.5.4 Zonas críticas

Em programação paralela, para evitar o resgate de um dado "desatualizado" ou impedir grande parte das situações causadoras de *starvation*, faz-se necessário algum tipo de comunicação entre os processos envolvidos para negociar e sincronizar suas ações.

Provavelmente, a técnica de programação mais comumente utilizada para promover uma comunicação adequada entre processos é a de criação de regiões críticas de códigos, também chamada de zona crítica, que garante (ou deveria) o acesso a um dado exclusivamente a um único processo ou *thread*.

Um dos grandes causadores de problemas à sincronização de processos é justamente um recurso do sistema operacional muito desejado, que promove um tipo de paralelismo, a capacidade de escalonamento e troca de contexto entre processos. Tal recurso, além de promover o gerenciamento de fatias de tempo de processamento a múltiplas execuções simultâneas, promove dificuldades ao tentar garantir a integridade de uma informação quando esta é manipulada por mais de um processo. As figuras 3.10 e 3.11 ilustram falhas geradas pelo processo de escalonamento e pela falta de tratamento adequado à comunicação entre processos.

A inibição do processo de escalonamento, feito pelo núcleo do sistema, poderia fornecer uma solução à necessidade de regiões críticas em sistemas monoprocessados, porém, o controle das interrupções do sistema entregue a processos de usuários poderia colapsar com todo o mecanismo de gerência de processos, prioridades e paralelismo, que inviabilizaria todo o sistema. Por outro lado, o uso de variáveis compartilhadas e passagem de mensagens para sinalização de exclusão mútua e inicializar uma zona crítica não garante uma comunicação apropriada pois o sistema operacional pode escalonar o processo no exato momento da comunicação (atribuição). Na prática, para uma implementação segura de regiões críticas o *hardware* e o sistema operacional devem fornecer algumas instruções que não permitem troca de contexto antes do término de sua execução. Estas instruções são conhecidas como instruções atômicas, devem ser suportadas tanto pelo sistema operacional

quanto pelo *hardware* de execução e somente com seu uso pode-se garantir uma zona crítica adequada.

Em geral, a região de um *software* que faz acesso a uma variável compartilhada deve garantir que somente esta fará o acesso ao devido recurso, inserindo toda manipulação dentro de uma zona crítica. A zona crítica criada deve fazer uso de instruções atômicas ao ser iniciada e finalizada, caso contrário seria necessário a criação de outra zona crítica para iniciar aquela em questão, ocasionando um processo em cadeia.

## 3.6 Lei de Amdhal

A forma possivelmente mais utilizada para medir o desempenho de um algoritmo paralelo é comparando o tempo gasto em sua execução de forma serial com o tempo gasto da execução paralelizada. A essa comparação dá-se o nome de *speedup*.

O *Speedup* quantifica o fator de redução do tempo de execução de um programa paralelizado em um número $P$ de processadores comparado à sua forma não paralelizada. É obtido pelo cálculo da razão do intervalo de tempo de processamento do programa em um único processador denominado $T_s(s = serial)$ pelo intervalo de tempo de execução em um computador utilizando $P$ processadores em paralelo, denominado $T_p$, esta relação é expressa na equação 3.1.

$$Speedup = \frac{T_S}{T_P} \tag{3.1}$$

Existem dois tipos de *Speedup*, o dito relativo que é obtido quando $T_s$ é igual ao tempo de execução do programa paralelo executado em um único processador e o dito absoluto quando $T_s$ que é igual ao tempo de execução do mais otimizado programa sequencial.

A lei de Amdhal [Amd67], vista na equação 3.2 na forma parametrizada, detalha o *speedup* para programas totalmente ou parcialmente paralelizáveis. Nesta forma

a lei pode ser obtida pela relação entre o tempo total de execução de suas partes $(T_s + T_p)$ em uma única unidade de processamento sob a soma do tempo de sua parte serial $(T_s)$ com o tempo total das partes paralelas $(T_p)$ dividido pelo número de CPUs usadas nelas $(n)$, onde 1 é a forma parametrizada da soma total de tempo de execução de forma serial $(T_s + T_p)$.

$$Speedup = \frac{1}{T_S + \frac{T_P}{n}} \tag{3.2}$$

Considerando uma tarefa totalmente paralelizável (100% paralela, $T_s = 0$) e considerando que os processos responsáveis para promover o paralelismo não consumissem tempo (custo computacional para promover o paralelismo igual à zero), a lei de Amdhal pode ser expressa como:

$$\underset{T_S \to 0}{Speedup} = \underset{100\%\ parallel}{Speedup} = \frac{T_S + T_P}{T_S + \frac{T_P}{n}}$$

$$\underset{100\%\ parallel}{Speedup} = \frac{T_P}{T_P/n}$$

$$\underset{100\%\ parallel}{Speedup} = \frac{n.T_P}{T_P}$$

$$Speedup = n \tag{3.3}$$

O comportamento teórico esperado no gráfico de *Speedup* para execução de uma tarefa 100% paralela que possua segmentos exatamente iguais no tempo deveria ser uma reta inclinada. Caso a tecnologia necessária para promover o paralelismo não possuísse custos computacionais adicionais, os gráficos de *Speedup* dessa tarefa hipotética deveriam ser uma função linear crescente diretamente proporcional

ao número de CPUs usadas nos cálculos, com coeficiente angular igual a 1 e coeficiente linear igual a 0, conforme apresentado na equação 3.3.

A figura 3.13 exibe o comportamento real de um *speedup* obtido por meio de multiplicação de matrizes, comparando-o com o *speedup* ideal desejado. O processo de multiplicação de matrizes é extremamente paralelizável, proporcionando um alto grau de paralelismo (independência entre tarefas).

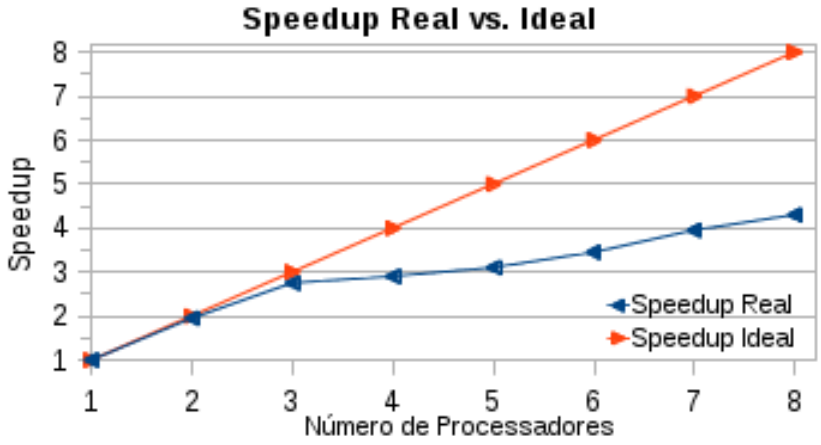



**Figura 3.13:** Superposição de um *speedup* real obtido por meio de produtos matriciais e um *speedup* teórico ideal. Dados obtidos nos laboratórios do CBPF, usando tecnologia *fork* e *shm* para promover o paralelismo.

É de fácil percepção que mesmo para tarefas com alto índice de paralelização, como o produto matricial, o gráfico do *speedup* não exibe uma linearidade e sua evolução com o aumento de CPUs é notoriamente depreciada. Aos causadores desse fenômeno de depreciação da evolução do *speedup*, em tarefas com um paralelismo teórico ideal, dá-se o nome de *parallel overhead*.

Outros fenômenos também possuem grande importância no estudo do paralelismo e desempenho computacional, como por exemplo a granularidade.

## 3.7 Granularidade

Em programação paralela, o conceito de granularidade está ligado à relação entre a carga de trabalho ("tamanho") das frações paralelas de tarefas em relação à carga de trabalho para a realização total da tarefa.

À medida em que o tamanho das frações paralelas de trabalho aproximam-se do tamanho total da carga de trabalho da tarefa inteira, consequentemente diminuindo a quantidade de subtarefas, diz-se que a granularidade é grossa. À medida que o tamanho das frações paralela de trabalho diminui em relação ao tamanho total da carga de trabalho da tarefa inteira, consequentemente aumentando a quantidade de subtarefas, diz-se que a granularidade é fina.

O aumento da divisão de um problema em subtarefas menores pode permitir um maior grau de paralelização, o que permite a entrega de subtarefas para uma maior quantidade de unidades processadoras e/ou reduzindo os tempos ociosos de espera pelo término de outra subtarefa. Por outro lado, como visto na seção 3.6, a promoção das condições necessárias ao paralelismo de tarefas possui um custo computacional conhecido por *parallel overhead*. O aumento da granularidade em partes mais finas promove uma maior taxa de comunicação entre processos e custo de gerenciamento, aumentando o *parallel overhead*.

Como pode ser observado no gráfico da figura 3.13, o aumento no uso do número de CPUs faz com que se distancie cada vez mais o desempenho prático obtido do desempenho teórico esperado. Para se fazer uso de uma maior quantidade de unidades processadoras em um problema, é necessário fragmentar a tarefa total em uma maior quantidade de partes menores, o que torna a granularidade mais fina. O aumento da granularidade, tornando-a mais fina, acarreta em um aumento no *parallel overhead*. O aumento indiscriminado da granularidade e do *parallel overhead* pode acarretar em uma taxa de trabalho extra ao sistema de maneira intensa ao ponto de ocasionar um uma depreciação no desempenho final da tarefa.

# Capítulo 4

# Algoritmo de detecção serial

Este capítulo expõe algumas etapas enfrentadas no desenvolvimento da versão serial do algoritmo para tratamento, classificação de padrões e detecção de MARFEs. Também são apresentadas as otimizações de desempenho implementadas após a validação da taxa de acerto do módulo de classificação do algoritmo. Mais informações sobre esta etapa do projeto podem ser obtidas em [AAC+12].

## 4.1 Breve histórico

O uso de técnicas de processamento de imagens tornou-se uma ferramenta importante em várias áreas da instrumentação científica. No entanto, o uso destas técnicas pode ser relativamente complexo e é frequentemente custosa computacionalmente. Em alguns casos tais técnicas têm sido implementadas em dispositivos eletrônicos, como as FPGAs (*Field-Programmable Gate Array*), os DSPs (*Digital Signal Processor*) ou ainda através dos sistemas de computação de alto desempenho baseado em processadores com *multi-cores*. Os processadores *multi-cores* tornaram-se bastante interessantes devido a quantidade de bibliotecas em *software*s disponíveis, que permitem o desenvolvimento de sistemas complexos em *software*. O uso de técnicas de programação paralela está, certamente, associado ao tipo de problema

a ser resolvido. Não obstante, o baixo custo relativo dos processadores *multi-cores*, que contam hoje com diversos núcleos integrados e permitem o uso de várias bibliotecas de programação paralela, chamaram a atenção para esta tecnologia. Por outro lado, os dispositivos FPGAs são ainda muito interessantes devido a capacidade de processamento no nível de portas lógicas eletrônicas, que é sempre muito rápida. A principal dificuldade nas FPGAs é a necessidade de desenvolvimento de uma grande quantidade de bibliotecas de alto nível e consequentemente o consumo das unidades básicas de portas lógicas nos dispositivos.

A fusão nuclear baseada em confinamento magnético (MCNF - *Magnetic Confinement Nuclear Fusion*) é um dos campos de pesquisa que se interessou pelo uso das técnicas de processamento de imagens como um ferramental fundamental para a instrumentação científica. Atualmente a técnica de processamento de imagens não é só importante para a interpretação dos experimentos, mas também para o uso em reconhecimento de padrões em grandes bases de imagens, como em [AAC+12] e em [AAS+13a]. No JET foram instaladas recentemente 30 novas câmeras para o experimento de caracterização da nova parede metálica do reator. Um dos desafios associados a estas novas câmeras como instrumentos científicos é a imensa quantidade de dados que elas produzem. Por exemplo, nos 90 Tera bytes de dados globais dos do JET, mais de 50% correspondem a imagens.

Em trabalhos anteriores, a análise e detecção do MARFE constituía-se em desenvolver e aplicar um algoritmo, tendo como base um banco de dados e imagens previamente construído por meio de classificações manuais no JET. A posse de um banco de imagens adequado é fundamental para a caracterização de qualquer sistema de processamento de imagens. O algoritmo de detecção de MARFE deveria ser avaliado em função do seu desempenho em classificar corretamente os padrões detectados dentro de um determinado intervalo de tempo. O tempo de execução do algoritmo é um fator primordial, pois, a velocidade na detecção poderia levar o algoritmo a fornecer informações para um módulo de controle externo, com o objetivo final de atuar na prevenção da ocorrência do MARFE ou mesmo na prevenção da interrupção do processo de geração de energia.

Após desenvolvidas algumas versões, a técnica de processamento e detecção conseguiu alcançar índices de acertos considerados muito bons, tendo uma baixa taxa de falsos negativos e baixíssimas taxas de falsos positivos. De forma simplificada, o algoritmo desenvolvido consiste em módulos que representam etapas no processamento das informações contidas nas imagens visando a detecção do MARFE. Uma breve explicação da atuação de cada módulo pode ser encontrada na Tabela 4.1.

**Tabela 4.1:** Descrição dos módulos do algoritmo de detecção de MARFE. Uma ilustração pode ser vista na Figura 4.1.

| Modulo | Descrição |
|---|---|
| Open Image | "Abre", inicia um arquivo de imagem |
| Background Average Subtraction | Subtrai a imagem atual de um conjunto pré-tratado pela média das anteriores |
| Image Binarization | Processo de segmentação pela binarização da imagem obtida após a subtração |
| Feature Extraction | Extração de características dos objetos presentes na imagem binária |
| SVM | Módulo classificador para cada região, tendo como base as caracterísicas extraídas |

A Figura 4.1 ilustra os módulos desenvolvidos para a detecção do MARFE em ordem de atuação (transformação da informação). A importância da distinção das ações em blocos de códigos distintos, ou módulos, foi fundamental para determinar os principais pontos de ação a sofrerem otimizações do código, visando a redução do tempo de execução. Aperfeiçoamentos na taxa de acertos dos padrões obtidos nas imagens estão mais ligados à técnica de processamento empregada que na alteração do código em si.

Uma medição dos tempos individuais de execução de cada módulo de uma versão serial do algoritmo pode ser vista na Figura 4.2. Esta figura expõe de forma clara os pontos onde deve-se concentrar as atuações para a redução do tempo total de execução. O detalhamento de sua legenda é exposto na tabela 4.2.

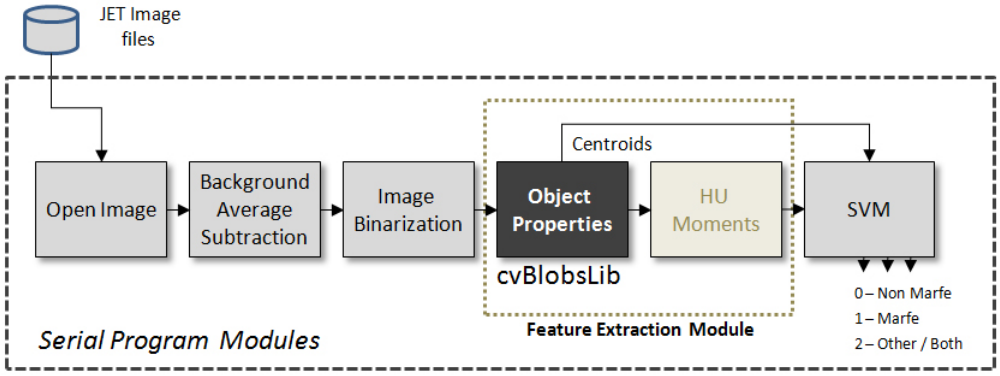



**Figura 4.1:** Diagrama de blocos dos módulos de uma versão serial do algoritmo de detecção de MARFE.

**Tabela 4.2:** Descrição das etapas mensuradas individualmente do algoritmo serial de detecção de MARFE, que pode ser visto na Figura 4.2.

| | |
|---|---|
| TOp: | Time for Open Image Module (Intervalo de tempo para o módulo de abertura da imagem) |
| TSAv: | Time for Subtrat Average Module (Intervalo de tempo para o módulo de Subtração e cálculo da imagem de fundo) |
| TBin: | Time for Binarization Module (Intervalo de tempo para o módulo de Binarização da imagem) |
| TFHu: | Time for Feature Extration Module (Intervalo de tempo para o módulo de Extração de Características) |
| TCls: | Time for SVM Classification Module (Intervalo de tempo para o módulo de Classificação) |
| TRls: | Time for Releasing Image Memory (Intervalo de tempo para o módulo de Controle de Memória) |

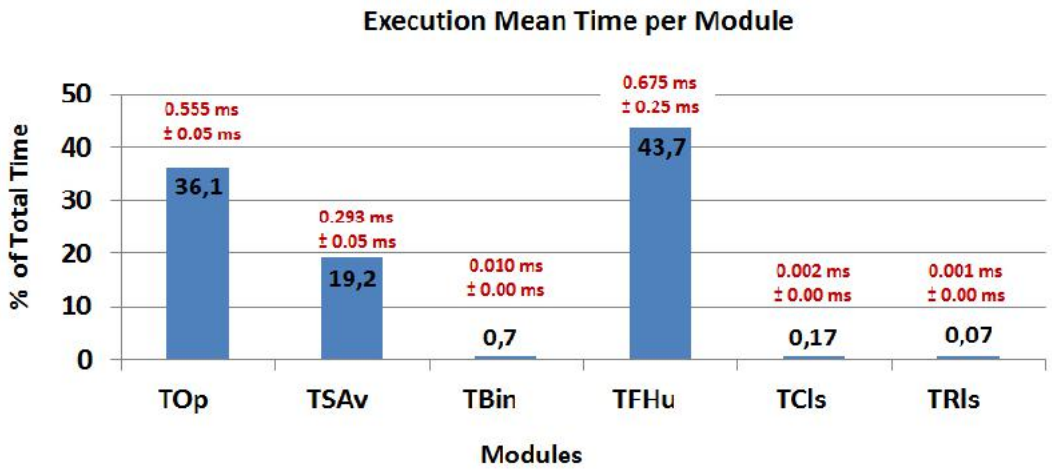



**Figura 4.2:** Tempos individuais de execução dos módulos da versão serial do algoritmo de detecção de MARFE. Uma explicação do significado destes módulos é apresentada na Tabela 4.2.

## 4.2 Otimizações no algoritmo serial

Ainda em trabalhos anteriores, uma atuação significante na performance do algoritmo deu-se através do emprego da biblioteca *cvBlob*[1], em substituição à *cvBlobsLib* que é parte integrante da própria biblioteca *OpenCV*[2], como ilustrado na Figura 4.3. Esse procedimento permitiu otimizar o tempo de execução do algoritmo de extração de características e consequentemente de todo o sistema de detecção.

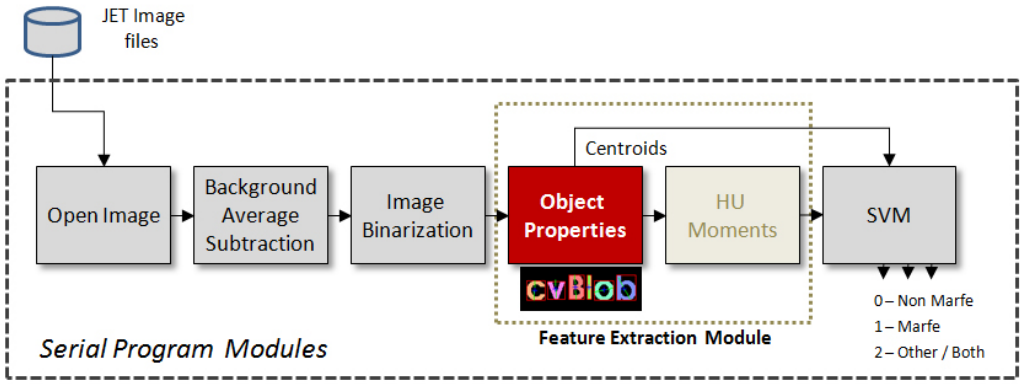



**Figura 4.3:** Diagrama de blocos dos módulos de uma versão serial do algoritmo de detecção de MARFE, fazendo uso da biblioteca *cvBlob*.

O ganho de desempenho é claramente notado pelas medições apresentada na Figura 4.4, em comparação com a figura 4.2, com a redução do tempo gasto pelo módulo de extração de características, de 43,7% do tempo total de execução para apenas 10,2%. Obviamente a porcentagem referente aos demais módulos foi acrescida proporcionalmente, acréscimos estes que não refletem no tempo absoluto de cada módulo, somente no tempo do módulo relativo ao tempo total de execução (exceto no módulo SVM). O tempo de execução do módulo SVM, referenciado pelo tempo "TCls", correspondia inicialmente à 0,17% do tempo total de execução. Com o uso da biblioteca *cvBlob* o tempo deste módulo passou a representar 0,77% do tempo total de execução. Acredita-se que essa diferença se deu devido ao uso do compilador GCC, uma vez que o compilador usado anteriormente (ICC) não possuía suporte à *cvBlob*.

[1] http://code.google.com/p/cvblob
[2] http://opencv.org

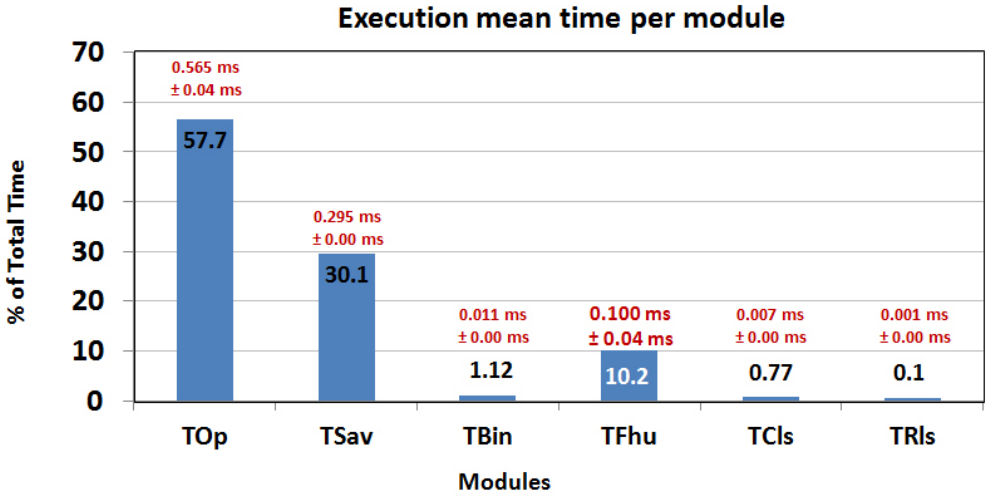



**Figura 4.4:** Tempos individuais de execução dos módulos de uma versão serial do algoritmo de detecção de MARFE, fazendo uso da biblioteca *cvBlob*.

Outras otimizações menores também foram implementadas nas últimas versões do algoritmo serial, algumas delas citadas a diante:

• **Image Crop:** O total de pixels adquiridos pela câmera KL8 do JET ocupa uma dimensão de 224x256 pixels (57.344 pixels ou bytes)[3]. A área de ocorrência de MARFEs se localiza em uma sub-região de 160x186 pixels (29.760 pixels ou bytes). Esse corte cria uma região de interesse para o processamento, inserindo uma ação a mais no algoritmo dedicado ao corte da imagem, mas reduzindo a área de análise em aproximadamente 48%. A Figura 4.5 exibe a área total da imagem e subregião de corte.

• **ICC vs GCC + *cvBlob*:** O processo de compilação dos cógigos em C/C++ pode influenciar bastante o desempenho final do código executável. Os compiladores modernos permitem diversos ajustes de sintonia fina para o processador efetivo no qual o código será executado.

[3] A camera KL8 do JET é opera na região visivel do espectro eletromagnético, com a capacidade de ter sua velocidade ajustada de 60 fps com a resolução de 1024x1024 pixels até 250.000 fps na resolução de 128 x 16 pixels.

A quantidade de "*flags*[4]" para estes ajustes é imensa e específica de cada compilador. Inicialmente foi utilizado o compilador C/C++ do projeto GNU, o GCC. No entanto, foram realizados testes de desempenho e otimização e o projeto foi migrado para uso com o compilador C/C++ da Intel (ICC), que apresentou melhores resultados [AAC+12], uma vez que o computador continha processadores da linha Intel. A substituição da biblioteca *cvBlobsLib* pela *cvBlob* forçou o retorno ao compilador GCC por motivo de compatibilidade, apresentando resultados finais expressivamente superiores com a combinação "*cvBlob* + GCC" do que "*cvBlobsLib* + ICC" [AAS+13a]. Esse conjunto de compilador e biblioteca se manteve por toda etapa paralela do projeto. Os códigos gerados pelos compiladores utilizados foram de 64 bits para todo o projeto.

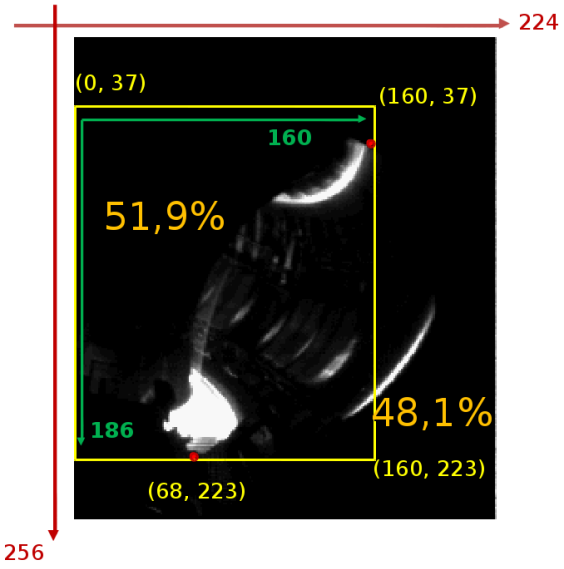



**Figura 4.5:** Representação da área total capturada pela câmera no tokamak e a área efetiva de análise de MARFEs. A área analisada representa cerca de 52% da área total da imagem.

Objetivando o reconhecimento comportamental do MARFE para uma aceleração de detecções futuras ou tentativas de prever o fenômeno, o módulo de extração de características executava uma análise adicional baseada em algoritmos de inteligência artificial. Apesar de a análise IA não impactar severamente no desempenho

[4]Pode-se encontrar várias "*flags*" de otimização para modelos específicos de *hardware* na documentação do compilador, para o caso do GCC o endereço eletrônico é: http://gcc.gnu.org

final do algoritmo seu aproveitamento era inferior ao seu retorno, o que ocasionou em sua retirada e mais um pequeno ganho em performance.

## 4.3 Método de análise de resultados

O método utilizado, proposto na etapa anterior, para analisar os resultados de classificação obtidos foi a *Matriz Confusão Estendida*. A Matriz de Confusão Estendida tem como base a Matriz de Confusão simples, sendo adaptada para o caso de falha nos mecanismos de detecção pelos algoritmos utilizados.

Um matriz confusão contém informações sobre as características de um sistema contraposto às classificações obtidas por algum mecanismo destinado a esse fim. A performance de um sistema de classificação é comumente avaliada por meio de informações extraídas de uma matriz confusão [Con91].

A disposição das informações em uma matriz confusão faz com que sua diagonal principal represente o número de classificações corretas para cada característica avaliada. As demais informações representam falhas no mecanismo de classificação. Os dados dispostos à esquerda e abaixo da diagonal principal representam classificações erroneamente positivas encontradas pelo mecanismo de classificação, os dados à direita e acima representam as classificações erroneamente negativas.

A tabela 4.3 exemplifica uma matriz confusão que analisa a presença ou não de um fenômeno MARFE. As equações 4.1, 4.2, 4.3 e 4.4 complementam as informações sobre os significados dos dados da matriz.

**Tabela 4.3:** Ilustração de uma matriz confusão com finalidade de exemplificar o posicionamento e a interpretação dos dados.

| | | **Pré-conhecido** | | |
|---|---|---|---|---|
| | | N.M. | MARFE | Outros |
| **Predição** | N. M. | a | b | c |
| | MARFE | d | e | f |
| | Outros | g | h | i |

• A extração da acurácia de um sistema pela análise dos dados de uma matriz confusão se dá pela razão do número total de classificações corretas pelo universo amostral total (verdadeiro positivo):

$$Verdadeiro\ Positivo = \frac{a+e+i}{a+b+c+d+e+f+g+h+i} \tag{4.1}$$

• A taxa de geral de erros nas classificações é a razão entre as classificações incorretas pela quantidade total universo amostral total:

$$Taxa\ de\ Erro\ Total = \frac{b+c+d+f+g+h}{a+b+c+d+e+f+g+h+i} \tag{4.2}$$

• Classificações positivas errôneas, em que se esperava uma classificação negativa de uma característica, foi considerada nesse projeto como sendo a razão **(falso positivo)**:

$$Falso\ Positivo = \frac{b+c+f}{a+b+c+d+e+f+g+h+i} \tag{4.3}$$

• Classificações negativas errôneas, em que se esperava um classificação positiva de uma característica, foi considerada nesse projeto como sendo a razão **(falso negativo)**:

$$Falso\ Negativo = \frac{d+g+h}{a+b+c+d+e+f+g+h+i} \tag{4.4}$$

Algumas tabelas neste documento, referentes a matriz confusão, podem apresentar uma coluna e uma linha extra. Estas informações são dados adicionais de imagens, existentes no banco da dados, classificadas como "não importante" ou "não classificada" pelo algoritmo. Por esse motivo estas tabelas são consideradas como *Matriz de Confusão Estendida*. Para mais detalhes sobre a Matriz de Confusão ver [Cha12, AAC+12]. Esta metodologia de análise foi mantida neste trabalho.

## 4.4 Resultados do algoritmo serial

### 4.4.1 Precisão das classificações

A precisão final do algoritmo, levando em consideração todos os seus módulos foi extremamente satisfatória. A taxa de acertos supera 96%, detectando corretamente a presença ou ausência de MARFE em 1466 objetos de um universo amostral de 1527. A taxa de falsos positivos ficou abaixo de 1% e a taxa de falsos negativos ficou em torno de 3%. A Tabela 4.4 apresenta o desempenho do algoritmo com mais detalhes.

**Tabela 4.4:** Desempenho do algoritmo serial na classificação da ocorrência de MARFE ou ausência do mesmo, baseada no banco de dados de imagens pré-classificadas do JET.

| | **NM** | **Mf** | **Oth** | **NoBD** |
|---|---|---|---|---|
| **NM** | 1175 | 3 | 39 | 434 |
| **Mf** | 10 | 284 | 5 | 16 |
| **Oth** | 3 | 1 | 7 | 6 |
| **NoIP** | 180 | 2 | 4 | |

A Tabela 4.5 resume a precisão do algoritmo exibindo a porcentagem dos erros e acerto obtidos.

**Tabela 4.5:** Resumo do desempenho do algoritmo serial na classificação da ocorrência de MARFE ou ausência do mesmo, baseada em um banco de dados de imagens pré-classificadas do JET.

| Total | 1527 objetos |
|---|---|
| **TP** | **96,01%** |
| FP | 0,98% |
| FN | 3,01% |

Várias situações devem ser consideradas ao se avaliar a precisão dos resultados finais. O banco de dados de imagens pré-catalogadas do JET apresenta uma distinção em três classificações: presença do MARFE, ausência do MARFE e uma terceira classificação em que não se pode afirmar com certeza sobre a presença ou não do MARFE, o padrão poderia ser classificado como MARFE ou Não-MARFE para este caso. Sob essas classificações o algoritmo tenta atingir o maior número possível de acertos em MARFE ou Não-MARFE, com o objetivo final sendo principalmente a presença dele.

A análise da precisão dos resultados é feita por meio de uma matriz que interliga as possíveis classificações feitas pelo algoritmo com as classificações preestabelecidas. Sob essa análise, uma ilustração da proporção de acerto do algoritmo serial é apresentada na Figura 4.6.

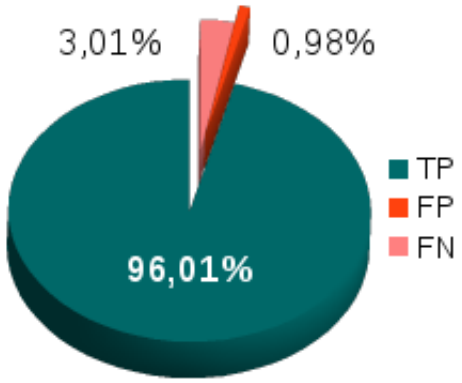



**Figura 4.6:** Ilustração da proporção de acertos do algoritmo serial na classificação de MARFEs e Não-MARFEs baseada no banco de dados pré-classificados do JET.

Caso seja restringido ao universo das ocorrências de MARFEs pré-classificadas no banco de dados do JET, o algoritmo de processamento, classificação e detecção apresenta uma precisão ainda melhor, acertando praticamente **98%** das ocorrências. Das 290 ocorrências de MARFEs presentes no banco de dados do JET (coluna **Mf** da tabela 4.4), 284 foram detectadas corretamente, três (3) ocorrências de MARFE foram classificadas como Não-MARFE, dois (2) objetos não foram classificadas e apenas um (1) foi desconsiderado. A Figura 4.7 ilustra a proporção de acerto do algoritmo serial para os objetos do banco de dados classificadas como ocorrência de MARFE. Um resumo é exibido na tabela 4.6.

**Tabela 4.6:** Resumo da precisão do algoritmo serial na classificação da ocorrência de MARFE, baseada no banco de dados pré-classificados do JET, para um universo restrito aos objetos com a presença de MARFE.

| Total | 290 objetos |
|---|---|
| **Correct Detection** | **97,9%** |
| Misclassified | 0,69% |
| Not found | 1,38% |

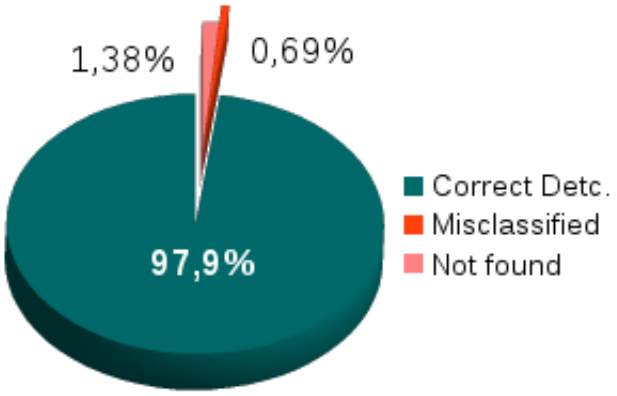



**Figura 4.7:** Ilustração da proporção de acertos do algoritmo serial na classificação de MARFEs e Não-MARFEs baseada no banco de dados pré-classificados do JET, considerando apenas as ocorrências de MARFEs previamente conhecidas.

### 4.4.2 Performance

Em um *hardware* padrão[5] utilizado (que pode ser visto na Figura 5.1) foi possível atingir uma taxa de processamento de 650,29 $\pm$ 47,92 imagens por segundo (fps). As otimizações citadas aplicadas ainda no algoritmo serial proporcionaram resultados mais expressivos. A capacidade de análise e classificação desse algoritmo "otimizado" se elevou para 937,98 $\pm$ 49,82 fps. Caso fosse possível tratar as imagens imediatamente após a sua captura, por exemplo ainda na memória da câmera (ou de um *hardware* de aquisição), em um dispositivo desenvolvido especialmente para esse fim, seria possível desconsiderar o módulo *Open Image*. Para esta situação a capacidade de processamento passaria para 2427,05 $\pm$ 206,81 fps.

As estimativas apresentadas da capacidade de processamento de imagens por segundo são baseadas na relação entre o tempo total de execução do algoritmo, em um *hardware* "padrão", e a quantidade total de imagens processadas. De fato, como visto na Figura 4.8, as ocorrências de MARFEs são detectadas em um tempo substancialmente inferior à média de tempo do processo de tratamento das imagens. Ao se explorar essa particularidade pode-se melhorar o desempenho do algoritmo serial ainda mais, superando os valores apresentados.

[5]Intel Xeon E5550, 8GB de RAM-ECC

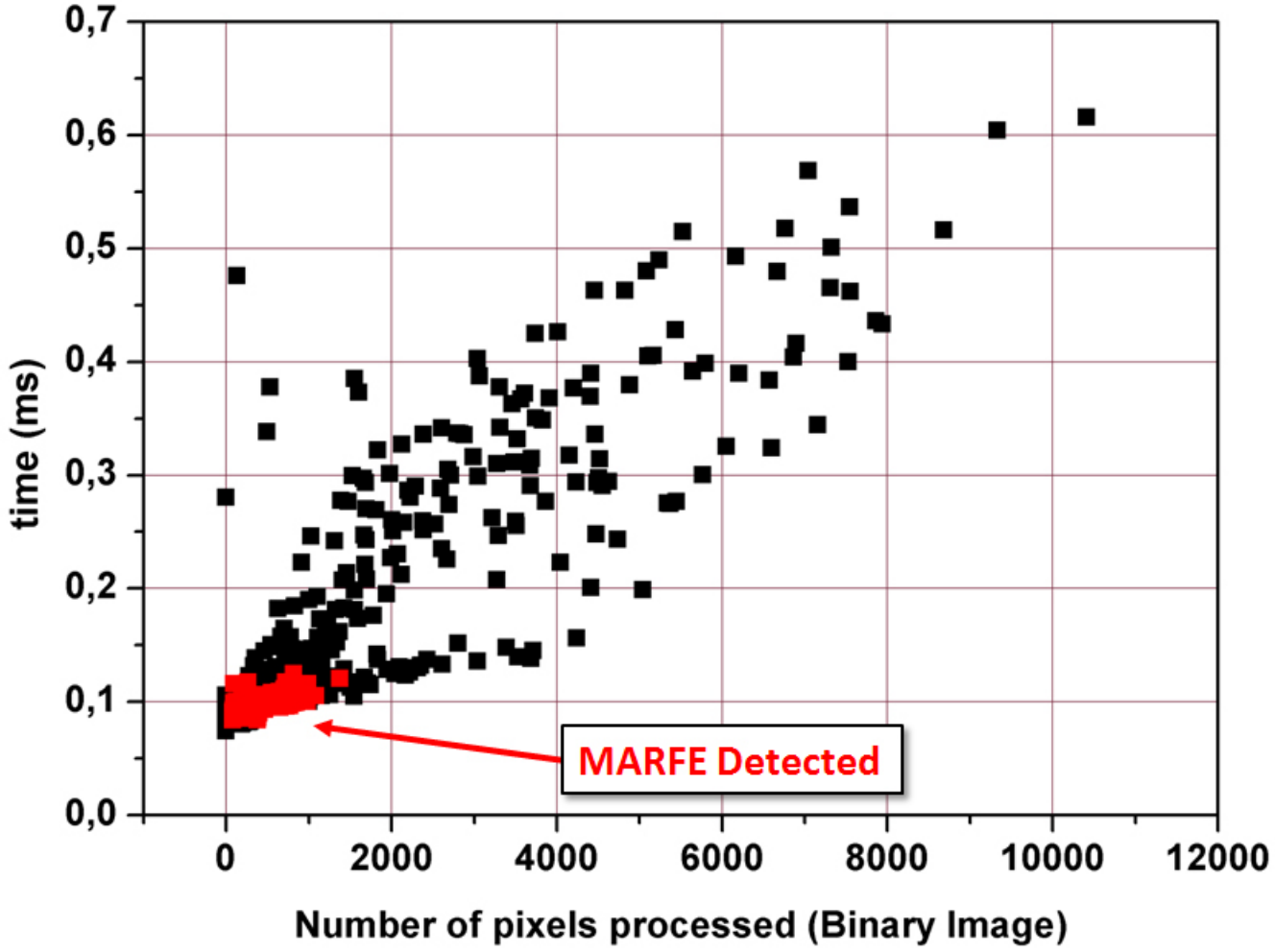



**Figura 4.8:** Sobreposição dos tempos de processamento de 710 imagens (1527 objetos) com os os respectivos tempos onde houveram detecção de MARFEs (destaque).

# Capítulo 5

# Paralelismo computacional aplicado na detecção de MARFE

Este capítulo descreve as etapas percorridas para a avaliação das formas de paralelismo, testes realizados para comparar as tecnologias de paralelismo computacional e suas implementações no algoritmo serial de detecção desenvolvido anteriormente, criando uma nova versão de execução paralela. Considerações e avaliações sobre medição de tempo em computadores também fazem parte desta seção.

## 5.1 Apresentação

Existem várias tecnologias capazes de promover algum nível de paralelismo computacional (e.g. *pipeline* e escalonamento ou outros como exposto no capítulo 3) suficiente para serem aplicados ao algoritmo já desenvolvido de detecção de MARFEs. As tecnologias de paralelização escolhidas, *thread* e *fork*, possivelmente são as duas mais difundidas e documentadas nos dias de hoje.

Inicialmente desenvolveu-se um estudo para avaliar o ganho de performance com o aumento do número de CPUs para as formas de paralelismo escolhidas, avaliando e

comparando os *speedups* de cada uma em executar um produto matricial paralelo. Aquela que demonstrasse um melhor desempenho receberia uma atenção especial na tentativa de introduzi-la no algoritmo de detecção do MARFE.

Tradicionalmente, no caso do uso de *fork* usa-se também outra tecnologia para promover a comunicação entre processos. Neste projeto a comunicação entre os processos *forks* foi o uso de memória compartilhada *shm*.

Simultaneamente as avaliações das técnicas de paralelização, foram desenvolvidos estudos objetivando a melhoria da precisão e validação das medidas de tempo. Os módulos do algoritmo previamente desenvolvidos são executados em intervalos de tempo extremamente curtos (poucos microssegundos) exigindo cuidado nas medições.

## 5.2 *Forks* vs *Threads*

Como visto nas seções *Forks* (3.2) e *Threads* (3.4), é conhecido que a promoção de algum nível de paralelismo computacional possui um custo, o *parallel overhead*. Para avaliar as tecnologias de paralelismo escolhidas, uma tarefa altamente paralelizável foi implementada e a partir dos resultados de execuções sucessivas pode-se obter um quantitativo de desempenho para cada caso. Por atender aos requisitos, o produto matricial foi eleito como tarefa de teste.

Em um "*hardware* padrão", visto na Figura 5.1, foram realizados e analisados cálculos de produtos matriciais em paralelo e medidos seus respectivos tempos de execuções variando o número de CPUs colaborativas. A partir destas execuções foram elaborados gráficos *speedups* que representam o ganho de desempenho relacionado ao número de CPUs usadas. Produtos de matrizes com tamanhos diferentes também foram avaliados, com o objetivo de verificar a influência do tamanho das matrizes em função das tecnologias utilizadas no tempo de cálculo.

Para a criação dos programas executáveis, o compilador utilizado, em todos os testes, foi o GNU GCC na versão 4.1.2, executado em sistema operacional Unix-Like

**Figura 5.1:** Visão do *hardware* computacional utilizado nas avaliações das tecnologias de paralelismo. A imagem mostra a visão de duas placas mães Supermicro com 2 processadores Xeon E5550 e 8GB de memória RAM-ECC DDR2-800, em cada placa mãe.

com *kernel* 2.6.18-SMP e memória física capaz de suportar (com larga margem necessária) todo conteúdo matricial de cada teste individualmente, desta forma, impedindo a interferência do lento tempo de I/O causado por *swaps* de discos.

Os códigos elaborados em linguagem C criaram duas matrizes quadradas e as povoaram de forma paralela. Após a criação, o produto matricial entre elas é calculado também paralelamente, gerando uma terceira matriz da mesma dimensão das duas anteriores.

Como as tarefas enviadas para os diferentes núcleos são extremamente semelhantes, a granulação programada foi a mais grossa (ver subcapítulo 3.7) possível. A quantidade de *forks* ou *threads* usada foi exatamente a mesma quantidade de núcleos desejada na avaliação, na tentativa de minimizar o *parallel-overhead* e maximizar o desempenho (i.e. reduzindo o tempo de execução).

As medições iniciais foram realizadas por meio de execuções sucessivas e sequenciais dos algoritmos de produto matricial paralelo no *hardware* dedicado. Cada valor apresentado nos gráficos é referente a média de sete medições. Para uma melhor visualização dos resultados as barras de erros das figuras 5.2, 5.3, 5.4 e 5.5 foram retiradas, estas podem ser vistas para dois tamanhos de matrizes apresentados na figura 5.6. Para as figuras 5.6 e 5.7 as barras de erro apresentam o maior e o menor valor obtido para cada medição.

### 5.2.1 *Speedups* para comparação de *Thread* e *Fork*

Os *speedups* foram obtidos em função do tamanho da matriz, numero de CPUs colaborativas e tecnologia empregada.

Inicialmente foram obtidos os comportamentos do tempo de execução em função do tamanho de cada matriz, como apresentado na Figura 5.2. Nesta figura o calculo refere-se ao produto matricial de matrizes quadradas com 720, 1080, 1440, 1800, 2520 e 3000 linhas, fazendo uso de 1, 2, 3, 4, 6 e 8 CPUs (representados por cores diferentes) e tecnologia de paralelização *fork*.

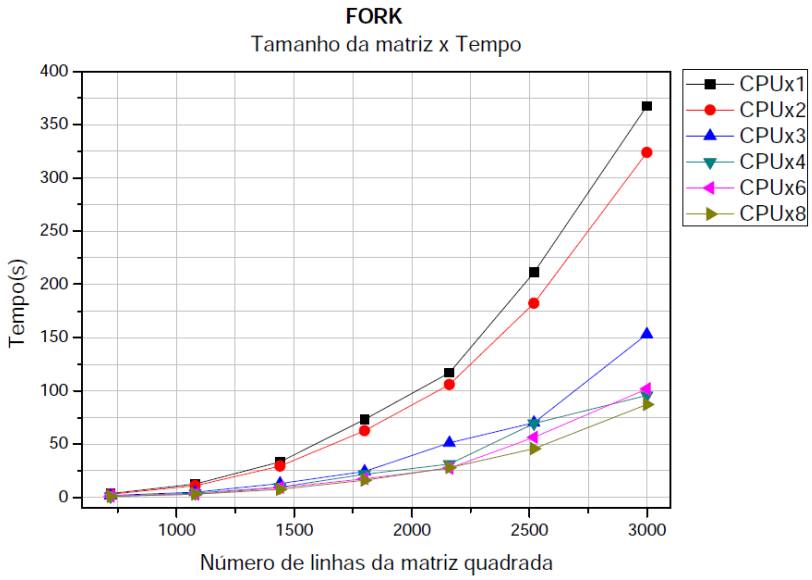



**Figura 5.2:** Comportamento do desempenho da tecnologia *fork* no cálculo de produto matricial para matrizes de tamanhos variados, para 1, 2, 3, 4, 6 e 8 núcleos. Para uma melhor visualização, as barras de erros foram retiradas, estas estão presentes nos gráficos das figuras 5.6 e 5.7.

Para o uso de *threads*, a Figura 5.3 expõe as médias de tempos encontradas na execução dos mesmos cenários.

Refazendo as medições com foco na evolução da performance em função do número de CPUs e fazendo uso da tecnologia *fork*, a figura 5.4 apresenta as médias de tempo obtidas para o cálculo dos produtos matriciais com 1, 2, 3, 4, 6 e 8 núcleos respectivamente. A legenda exibe as cores representativas do numero de elementos processados para cada matriz (linhas vezes colunas).

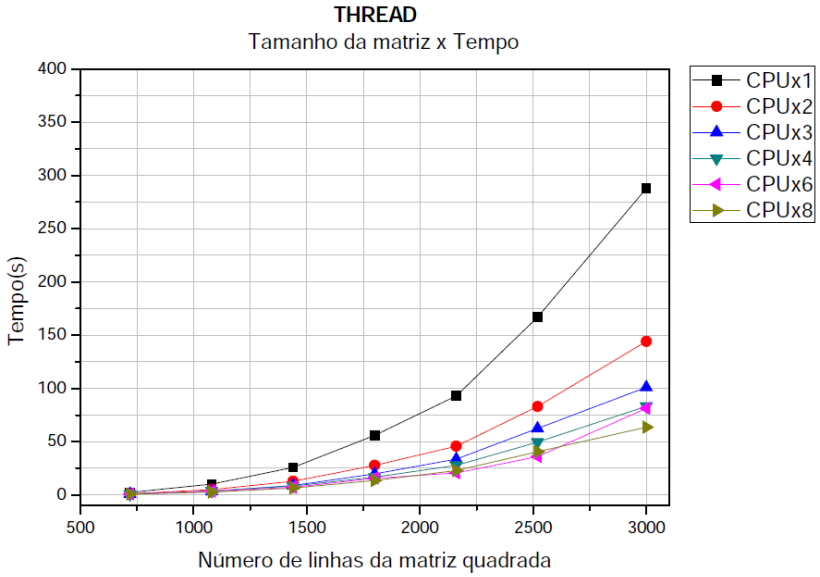



**Figura 5.3:** Comportamento do desempenho da tecnologia *thread* no cálculo de produto matricial para matrizes de tamanhos variados, para 1, 2, 3, 4, 6 e 8 núcleos. Para uma melhor visualização, as barras de erros foram retiradas, estas estão presentes nos gráficos das figuras 5.6 e 5.7.

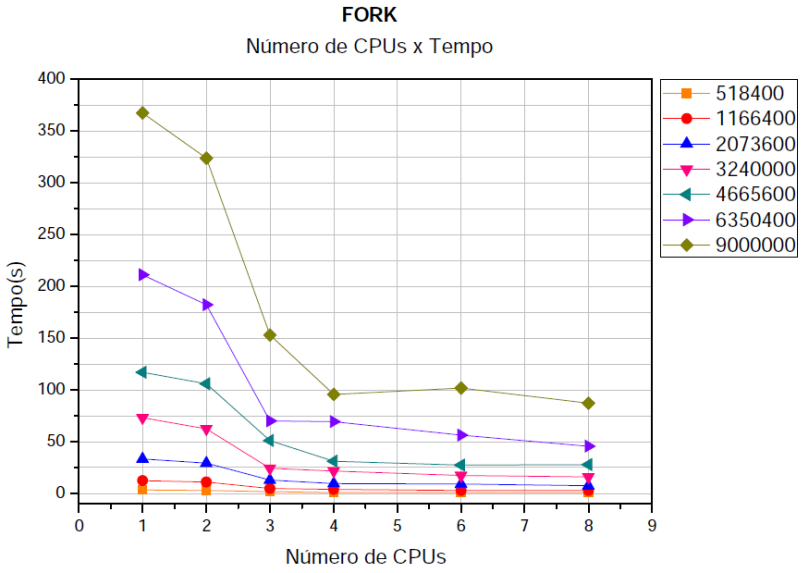



**Figura 5.4:** Comportamento do desempenho da tecnologia *fork* no cálculo de produto matricial fazendo uso de 1, 2, 3, 4, 6 e 8 núcleos respectivamente. As cores representam a quantidade total de elementos para cada matriz. Para uma melhor visualização, as barras de erros foram retiradas, estas estão presentes nos gráficos das figuras 5.6 e 5.7.

Para o uso de *threads*, a Figura 5.5 expõe as médias de tempos encontradas na execução do mesmo cenário. Tanto com o uso de *forks* quanto para *theads* o comportamento observado foi semelhante, apresentando algum grau de proporcionalidade.

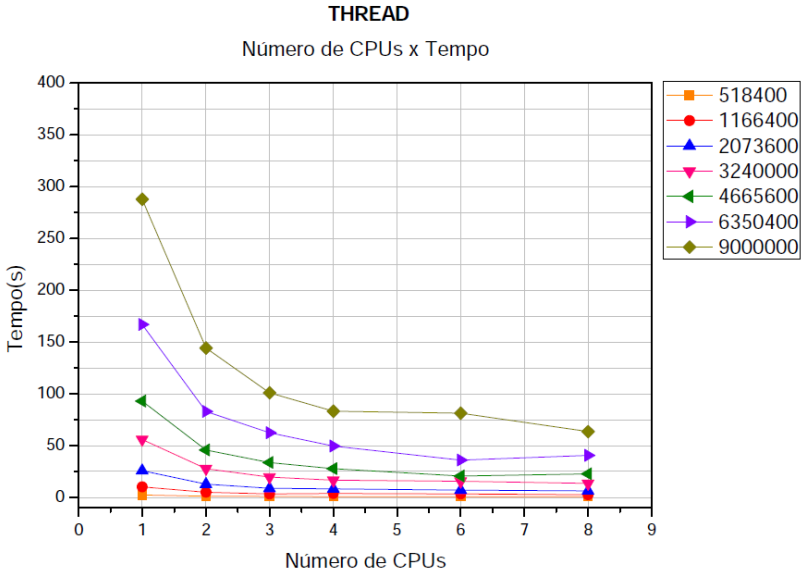



**Figura 5.5:** Comportamento do desempenho da tecnologia *fork* no cálculo de produto matricial fazendo uso de 1, 2, 3, 4, 6 e 8 núcleos respectivamente. As cores representam a quantidade total de elementos para cada matriz. Para uma melhor visualização, as barras de erros foram retiradas, estas estão presentes nos gráficos das figuras 5.6 e 5.7.

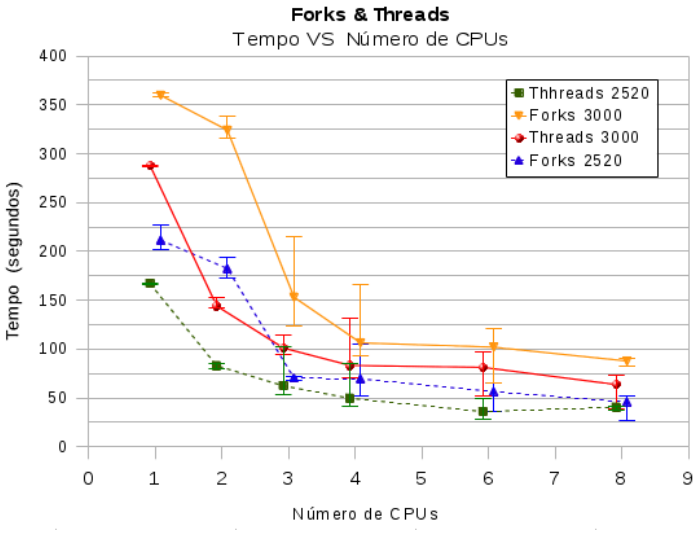



**Figura 5.6:** Comportamento do desempenho das tecnologias *fork* e *thread* no cálculo de produto matricial fazendo uso de 1, 2, 3, 4, 6 e 8 CPUs respectivamente. As cores diferenciam as tecnologias para matrizes quadradas com 2520 e 3000 linhas.

A partir destes gráficos é possível perceber que a proporcionalidade entre o tempo total de execução e o número de CPUs usadas não é linear. Esse comportamento é observado mais claramente nas figuras 5.6 e 5.7, onde o comportamento apresentado na figura 3.13 (comparativo entre um *speedup* real e um teórico) acontece também nestas avaliações. O melhor desempenho da tecnologia *thread* em relação

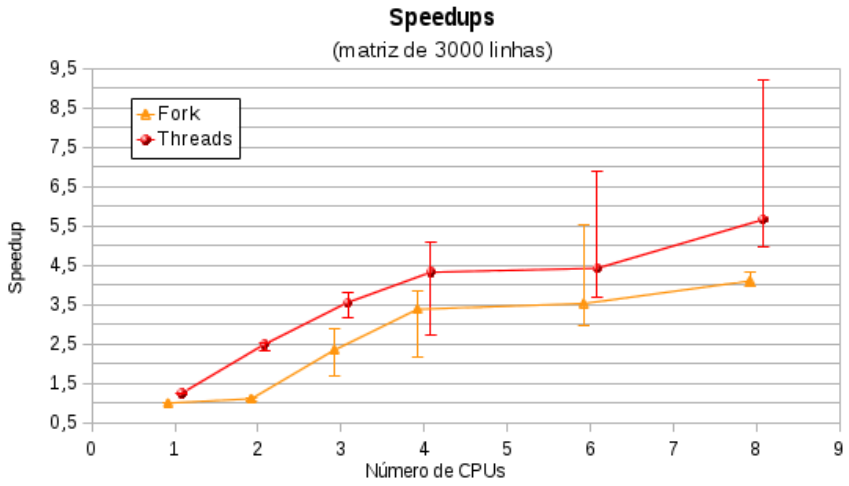



**Figura 5.7:** Gráfico comparativo do *speedups* para as tecnologias *fork* e *thread* no cálculo de produto matricial com 3000 linhas, fazendo uso de 1, 2, 3, 4, 6 e 8 CPUs respectivamente. As cores diferenciam as tecnologias. Como se trata de uma comparação, a base de cálculo foi o maior tempo gasto, matriz de 3000 linhas em 1 núcleo usando *fork*, mesmo para se calcular os *speedups* das *threads*.

à *fork* também é facilmente notado nestas figuras, onde os gráficos referentes à *trheads* apresentam sempre melhores *speedups* (ou menores tempos) que os referentes à *forks*.

## 5.3 Medições de tempo

### 5.3.1 Registrador TSC - *Time Stamp Counter Register*

Em programação, possivelmente a forma mais usada de se medir o tempo decorrente da execução de parte de um código computacional é capturar o valor do relógio do sistema imediatamente antes e depois do que se pretende medir. A diferença de tempo entre os relógios das duas capturas equivalem ao tempo gasto para se executar o trecho do código alvo. Essa técnica, apesar de oferecer implementação rápida e simplificada, pode não ser adequada quando se pretende mensurar execuções com durações muito curtas.

Desde o lançamento da família Intel Pentium, os processadores padrão x86 apresentam um registrador interno de 32 bits (hoje em dia com 64 bits) que é incremen-

tado a cada ciclo de *clock*. Este registrador também está presente em processadores de outros fabricantes que seguem o padrão x86. Ao se consultar a contagem no registrador imediatamente antes e depois de um trecho de código e, conhecendo-se a exata frequência interna do processador, pode-se estimar com boa precisão o tempo de execução pela razão da equação 5.1. Esta técnica é conhecida como RDTSC (*Read Time-Stamp Counter*).

$$Time = \frac{Counter_2 - Counter_1}{CPUFrequency} \tag{5.1}$$

Medidas de tempo que fazem uso deste recurso possuem características que a tornam mais adequadas às situações envolvidas neste projeto. É possível citar algumas destas características:

- Menor custo computacional devido a consulta a um registrador interno (RDTSC) ao processador do que uma consulta ao relógio do sistema;
- Devido aos escalonamentos do sistema e a necessidade de tradução ao se operar com o relógio, a flutuação de tempo de consultas a esse relógio possui maior escala que as consultas ao registrador TSC;
- Consulta ao registrador interno é feita por instruções ASSEMBLY e não por chamadas de sistema;
- Os tempos de respostas das consultas ao registrador TSC permitem melhores resoluções e medições de fenômenos com durações mais curtas.

Um cuidado deve ser tomado no momento da inserção do código de medição de tempo, pois os compiladores podem fazer modificações no local de posicionamento

na versão binária do código. Para maiores detalhes sobre esta técnica de medida ver em [Pao10].

No caso de ambientes de processamento paralelo, o cuidado deve ser maior ainda. Para os casos de *hardware* com múltiplos núcleos existe o problema do uso do registrador "TSC" em medições de tempo. Apesar de os fabricantes garantirem que o contador TSC é incrementado em uma unidade a cada ciclo de *clock* interno, não se pode garantir que os registradores de cada núcleo foram iniciados juntos e podem apresentar uma discrepância entre os valores armazenados nos registradores de diferentes núcleos. Em sistemas operacionais multitarefas, o escalonador de tarefas pode migrar uma execução entre diferentes núcleos, podendo resultar em erros nas medições de tempo por operar com registradores TSC diferentes.

Outro problema no uso do registrador TSC para medições de tempo é o recurso de escalonamento de frequência apresentado por muitos processadores. Este recurso altera a frequência interna dos núcleos, geralmente de forma independente, o que impede a dedução do tempo de execução com base na quantidade de ciclos gastos pela aplicação.

A necessidade de mensurar eventos extremamente rápidos neste trabalho exigiu a insistência no uso da técnica de medição de tempo por meio do TSC. Os problemas mencionados foram contornados por meio de duas ações, cada uma atuando diretamente sob um dos dois fatores descritos anteriormente:

- Para se evitar o escalonamento de frequência dos núcleos, os módulos controladores deste recurso para a CPU usada foram carregados e sua frequência de operação foi "bloqueada" no máximo, evitando oscilações durante as execuções dos processos. Essa técnica não foi necessária quando usado o *hardware* padrão, uma vez que o processador Xeon E5550 não oferecia o recurso de escalonamento de frequência.

- A garantia das medições de tempo por meio da leitura do registrador TSC do mesmo núcleo onde se iniciou a execução da tarefa foi feita por meio da configuração da "afinidade" do núcleo com o *ID* do processo.

Esse recurso é oferecido pelo sistema operacional usado e acionado por meio da chamada *taskset*.

### 5.3.2 Validação da técnica RDTSC

O uso da técnica de medição de tempo RDTSC [TTF08] se justificou principalmente pela necessidade em mensurar módulos do programa extremamente rápidos desde sua versão serial. O conhecimento da frequência interna da CPU usada é um ponto importante para se calcular o tempo de duração de um trecho de código executado.

Para esta implementação houve um cuidado especial objetivando validar a técnica RDTSC no *hardware* utilizado. Esta validação se deu por meio de medições da execução de uma função cujo tempo é "conhecido" e extremamente superior ao que se quer conhecer, podendo estimar-se o *clock* interno da CPU e compará-lo com a informação de sua frequência interna em sua documentação técnica.

O processo de escalonamento do *kernel* linux associado ao mecanismo interno da função *sleep* garante que o tempo de sua execução seja pelo menos o valor solicitado em segundos, acrescido em no máximo um período de escalonamento. A imprecisão promovida pelo escalonador pode ser minimizada a medida em que o tempo de *sleep* aumenta.

A validação se deu por meio das seguintes etapas:

- Uma vez "conhecida" a frequência interna teórica de operação da CPU usada (*datasheet*), foi desenvolvido um código em linguagem C que usa a técnica de medição RDTSC para mediar exclusivamente o tempo de uma chamada *sleep* e calcular a frequência de operação interna da CPU;

- Mensurar o tempo de uma chamada da função *sleep* de 1 segundo e calcular o *clock*;

• Repetir o processo anterior sucessivamente, aumentando o tempo de *sleep* e calculando os respectivos *clocks*;

• Organizar os dados em forma de gráfico, traçando um "ajuste exponencial" e obtendo seus coeficientes;

• Comparar *clock* real mensurado com a documentação técnica da CPU usada.

Segundo a informação do *kernel* linux sobre o *clock* real da CPU usada, calculado no momento do *boot*, a frequência interna de operação era de 2.666,384 MHz. O resultado das medições é exposto na figura 5.8, onde se nota que valor oferecido pela ferramenta de *fitting* para o coeficiente "A" tende ao valor informado pelo *kernel* à medida que se eleva o tempo da medição. A equação 5.2, que ofereceu o melhor ajuste para os dados encontrados, apresenta os coeficientes usados no ajuste. Caso o tempo de medição tenda a infinito o coeficiente "A" representaria o *clock* interno medido.

$$y = A.e^{\frac{b}{x+c}} \tag{5.2}$$

## 5.4 Hard RT vs Soft RT vs Não RT

Duas são as principais razões para a introdução do conceito de tempo real (RT - *Real-Time*) no projeto. Como as análises de cada imagem ocorrem em um período de tempo da ordem de microssegundos, a priorização pelo sistema operacional da execução do *software* criado pode promover uma troca de contexto com o objetivo de reduzir a interferência do escalonador com outros processos em execução na máquina, que irão certamente produzir melhores resultados nas medições de tempo.

Outra razão para considerar o conceito de tempo real é a possível redução de tempo ao se "alarmar" a presença de um MARFE para uma futura atuação de um sistema

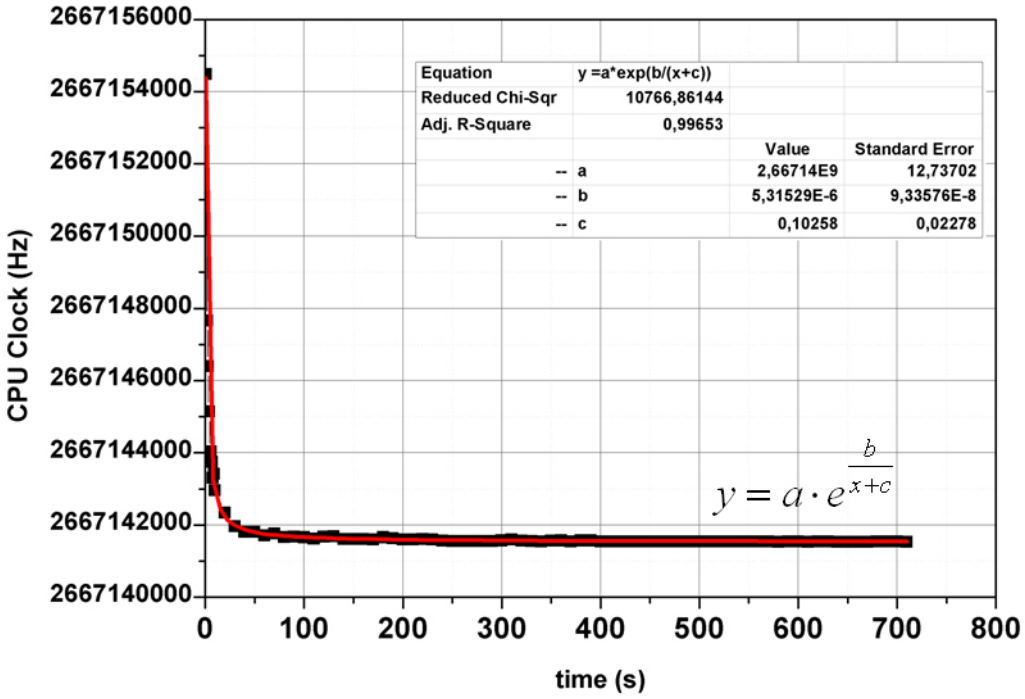



**Figura 5.8:** Ajuste exponencial para estimativa de *clock* interno real da CPU por meio de medições sucessivas da função *sleep* para vários tempos de espera determinados.

de controle (e.g. do sistema de confinamento do campo magnético). Quanto mais rápida for anunciada a detecção, mais tempo hábil haverá para a atuação dentro de um limite estabelecido pelo sistema de tempo real. A verificação do ganho de tempo efetivo para reportar o MARFE em um sistema RT, após sua detecção, e a depreciação do desempenho do *software* nesse sistemas é uma das etapas futuras do projeto, as quais não serão abordadas neste documento.

O conceito de tarefa em tempo real está intimamente ligado ao limite de tempo de resposta de um sistema para um determinado evento, esse tempo também é conhecido como *deadline*. Em geral, a caracterização de *hard* ou *soft* RT está ligada à possibilidade de se perder um determinado evento rápido, conforme mostrado abaixo:

- *Hard* RT: Executa a tarefa com "boa" margem de tempo para o *deadline*, a perda de somente um único evento é suficiente para determinar a falha no sistema;

• *Firm* RT: Executa a tarefa com alguma margem de tempo para o *deadline*, a perda de poucos eventos pode ser tolerada sem falha no sistema mas com degradação na qualidade do serviço;

• *Soft* RT: Executa a tarefa com "pouca" margem de tempo para a *deadline*, a perda de alguns eventos não é desejada mas pode ser tolerada sem falha no sistema e ocasionalmente sem degradação na qualidade do serviço;

• Pseudo RT ou não RT: Sistemas "não RT" podem executar uma tarefa com pequena ou grande margem de tempo para o *deadline* porém não garantem uma constante nesta execução, podendo ter um certo evento perdido. Se uma determinada tarefa é executada, de forma satisfatória, em um sistema não preparado como RT, pode-se dizer que para esta determinada tarefa esse sistema pode ser considerado RT.

Em sistemas operacionais, o grande aumento da taxa de escalonamentos nos sistemas RT, somado aos demais ajustes necessários para a promoção das garantias RT prejudicam invariavelmente o desempenho geral dos sistemas (operacional e aquele que necessita da resposta em tempo real). Aliado a este fato, as dificuldades nas alterações do algoritmo de detecção de MARFE para uma operação junto ao *kernel*, em *hard* RT, descartou o uso desse recurso nessa etapa do projeto. Junto a estas razões, a complexidade dos códigos gerados forçaram a abordagem da promoção do "tempo real" por meio da elevação de privilégios (prioridade) dos processos, além da garantia que o *hardware* estava completamente dedicado ao processamento dos códigos desenvolvidos.

Os recursos usados para a elevação de privilégios dos aplicativos de detecção de MARFE foram promovidos pelo próprio sistema operacional usado, por meio das chamadas *nice* e *chrt*. Dependendo do tempo necessário à resposta a um evento e a performance atingida pelo algoritmo, uma abordagem verdadeiramente em tempo

real poderá ser necessária no futuro. Porém, estes limites de atuação em tempo real terão que ser obtidos junto a equipe do JET .

## 5.5 Paralelização do algoritmo de detecção de MARFEs

Tendo como base a análise proveniente dos estudos de *speedups* resultantes dos testes com produtos matriciais, a tecnologia de paralelização adequada será aplicada no algoritmo de detecção de MARFE ou em partes dele.

O processo de paralelização do algoritmo de detecção de MARFEs foi beneficiado devido à implementação modular da versão serial. Na tentativa de se aproveitar uma maior parte de código já escrita e ainda implementar o paralelismo, foi adotada uma estratégia de paralelização com a divisão primária das tarefas em dois grupos, cada grupo subdividindo-se em processos ou *threads* diferentes, objetivando as etapas apresentadas a na figura 5.9.

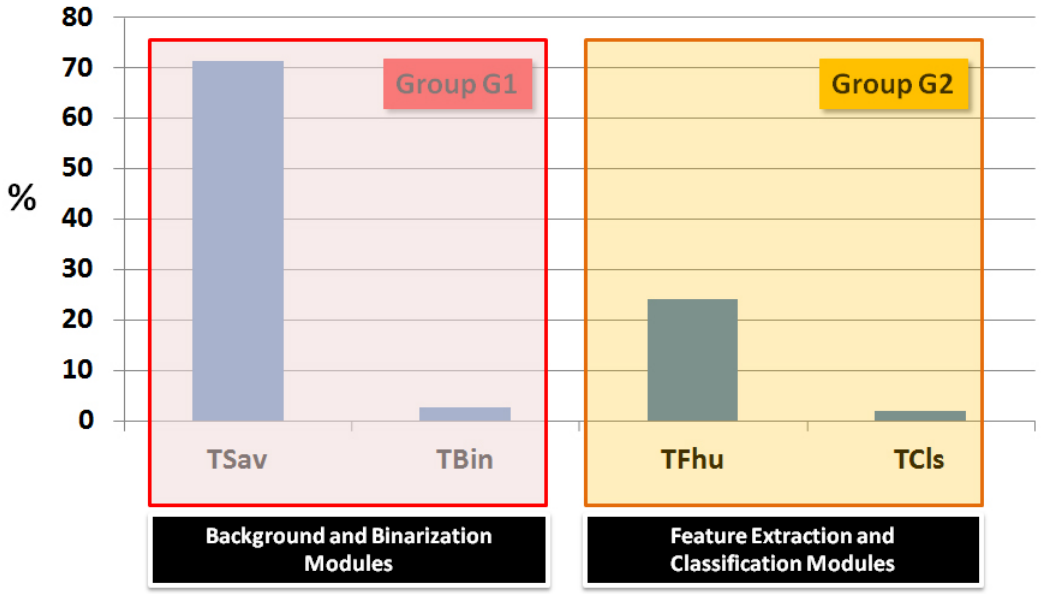



**Figura 5.9:** Divisão paralela primária em dois grupos de ações, responsáveis por diferentes etapas na detecção de MARFEs.

A implementação da técnica de paralelização por meio de *threads* se deu nas tarefas do grupo "1" (G1), por ter apresentado um cenário propício às modificações necessárias para tal. As tarefas do grupo "2" (G2) foram paralelizadas por meio do uso de *fork* + *shm* devido a forma com a qual estes módulos foram originalmente criados, dificultando a manipulação das variáveis originais.

A paralelização da etapa denominada de TSav (*background*) se deu por meio da simples divisão das novas imagens a serem tratadas em três partes, onde cada chamada *thread* se responsabilizaria pelo cálculo da média das imagens anteriores e sua respectiva subtração do seu terço responsável em relação a imagem original. O código foi otimizado para que o cálculo levasse em conta somente a imagem final e a atual no processamento, evitando o retrabalho de processar tod o grupo imagens repetitivamente. Em estudos anteriores foi avaliado que um grupo com 22 imagens apresentou melhores resultados quanto a precisão das classificações. O uso de subprocessos leves *threads* nessa etapa permitiu o aproveitamento de boa parte do código escrito anteriormente.

Para garantir o sincronismo dos processos de *background* foi criado um processo "*master*" que se responsabiliza pelo controle dos subprocessos *threads*, também denominados de *slaves*.

A figura 5.10 ilustra a paralelização dos módulos TSav e TBin, compondo o grupo G1.

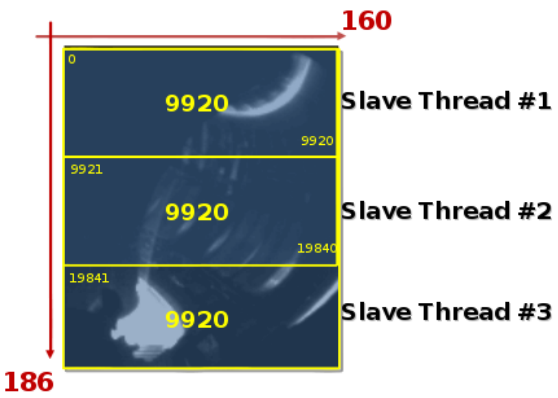



**Figura 5.10:** Ilustração da divisão das subtarefas das *threads* do grupo G1.

O uso de chamadas *forks* nas tarefas do grupo G2 também possibilitou o aproveitamento de parte do código desenvolvido anteriormente. Nesta etapa o aproveitamento de códigos anteriores foi de extrema importância devido às técnicas de processamento de imagens avaliadas na sua versão serial. Por apresentar uma maior quantidade e complexidade de ações, na identificação e classificação dos MARFEs (ou outros objetos) nas imagens, a tecnologia *fork* se mostrou mais propícia. As facilidades adicionais para a promoção da paralelização foram implementadas com pouca intrusão nos códigos anteriores, adicionando recursos quase que separadamente e sem alterações das variáveis originalmente criadas, o que seria mais complexo de ser feito no caso de utilização de *threads*. Uma visão geral das divisões e subdivisões de tarefas da versão paralela do algoritmo de detecção de MARFEs é exposta na figura 5.11.

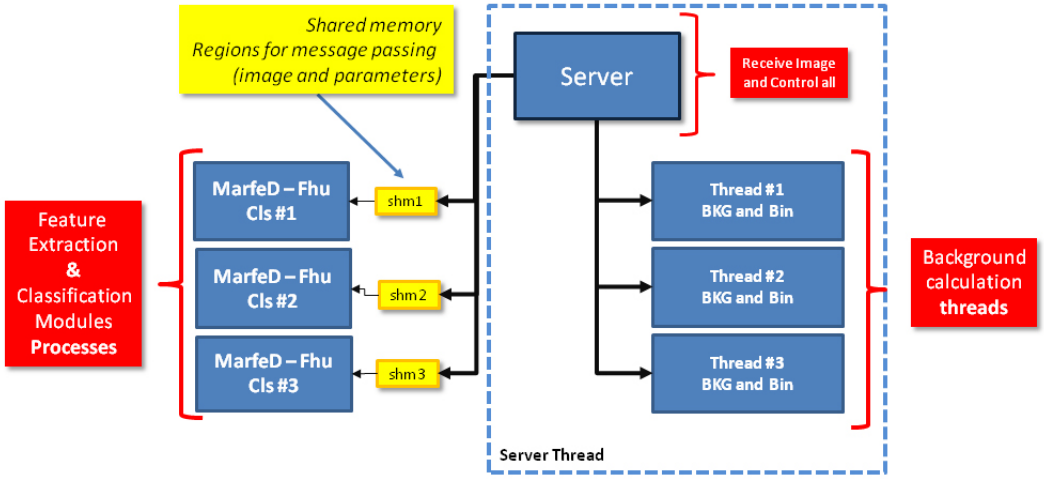



**Figura 5.11:** Ilustração da divisão das tarefas e subtarefas do algoritmo paralelo de detecção de MARFEs, para os grupos G1 e G2.

### 5.5.1 Determinação do número de *threads* e *forks*

Para uma otimização do algoritmo paralelo e o melhor aproveitamento do *hardware* foi elaborado um estudo dos intervalos de tempo individuais de cada subtarefa e suas componentes. Um diagrama com o estudo dos tempo de execução foi elaborado a partir das medidas realizadas, buscando reduzir ao máximo a ociosidade dos núcleos utilizados. A figura 5.12 apresenta os tempos e as divisões praticadas neste

projeto. Como as tarefas do grupo G2 são independentes entre si e dependentes do término das tarefas do G1, os blocos de tempos foram distribuídos de modo a ter um núcleo do grupo G2 disponível ao final de cada conjunto de execução do grupo G1, ocupando os núcleos (G1 + G2) quase que em tempo integral.

A figura mostra uma medição para três imagens processadas sucessivamente (552, 553, 554), em um *hardware* com oito núcleos disponíveis. Os núcleos de número 1, 2 e 3 ficaram responsáveis pelas thredas do grupo G1 (TSav e TBin). Os núcleos de número 4, 5 e 6 se responsabilizaram pala execução dos *forks* do grupo G2 (TFHu e TCls). O núcleo de número 0 ficou responsável por executar o processo *master* e encarregado das medições de tempo, restando ao núcleo de numero 7 as tarefas do sistema operacional. É possível observar um intervalo de tempo de 84 $\mu$s entre o processamento de duas imagens sucessivas. Este intervalo é variável e dependerá certamente do conteúdo das imagens. Uma análise da distribuição estatística deste intervalo de tempo será apresentada no capítulo 6. Um outro intervalo de tempo de 67 $\mu$s (*idle*) pode ser avaliado entre tarefas de um mesmo núcleo do grupo G2. O balanco de um *master*, três *threads* e três *forks* foi encontrado pela avaliação deste diagrama. Se for adicionado um novo núcleo ao grupo G2 o espaço de tempo disponível irá aparecer nas tarefas do grupo G1.

Um estudo mais aprofundado da distribuição de tarefas pelos três núcleos do grupo G2 pode ser vista na figura 5.12. Os intervalos de tempo de 67 $\mu$s podem ser considerados como um limite das variações de tempo de execução das tarefas do próprio grupo G2, uma vez que certamente elas não são todas executadas no tempo de 156 $\mu$s.

Este arranjo teórico permitiu manter três núcleos atarefados em praticamente 100% do tempo e outros três com baixa taxa de ociosidade. Sob estas condições pode-se obter uma avaliação inicial da velocidade de classificação da existência ou não de MARFEs em um intervalo de tempo estimado de 84$\mu$s (i.e. uma taxa de processamento teórica de 11.904 fps).

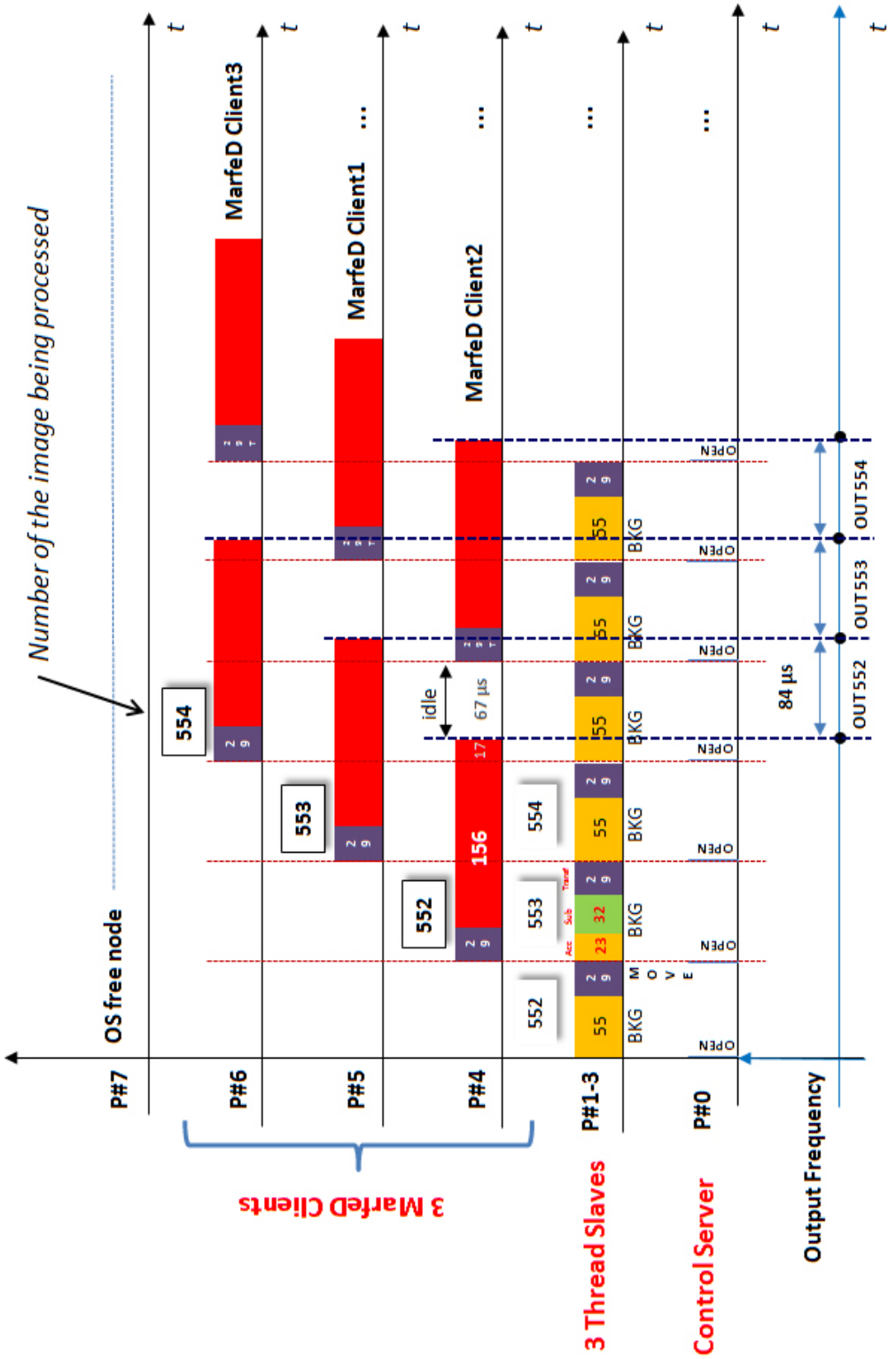



**Figura 5.12:** Distribuição temporal das tarefas e suas componentes principais e respectivas alocações em núcleos.

# Capítulo 6

# Resultados obtidos com o paralelismo computacional

Este capítulo apresenta os resultados alcançados com os estudos e testes elaborados. Tabelas comparativas e observações adicionais sobre os mesmos também serão discutidas aqui.

## 6.1 Precisão das classificações

Possivelmente o objetivo que mais influenciou nas decisões sobre alterações de códigos e norteou o desenvolvimento do projeto foi o compromisso em se manter, ou melhorar, a precisão alcançada anteriormente com a versão serial do algoritmo. De nada adiantaria elevar a taxa de processamento das imagens se a confiabilidade fosse depreciada.

Como exposto nas tabelas 4.4 e 4.5, a versão serial do algoritmo apresentou um total de 1527 objetos detectados e uma classificação correta de 1466 objetos. Isso representa uma taxa que supera 96% de acertos na detecção geral de ocorrência de MARFEs, Não-MARFEs e outros. Caso esse universo seja limitado ao conjunto

de imagens com ocorrência de MARFE, foram detectados corretamente 284 dos 290 objetos presentes na base de dados, refletindo em um taxa de acerto próxima de 98%.

As tabelas 6.1 e 6.2 apresentam os resultados das taxas de acerto para a versão paralela do algoritmo. A tabela 6.2 apresenta uma comparação destes resultados com sua versão anterior serial. A diferença de resultados dos dois algoritmos pode ser explicada como consequência do processo de corte da imagem por uma região de interesse (ROI), uma vez que na imagem completa diversas regiões foram corretamente classificadas como Não-MARFEs e descartadas após o corte. Com relação ao conjunto de objetos pré-classificados como MARFE, a tabela 6.3 apresenta os resultados da taxa de acerto da versão paralela do algoritmo e os compara com a versão serial. No universo de 290 objetos catalogados como MARFE, 287 foram classificados e dentre estes houve um acerto de 283 objetos, representando uma taxa de detecção correta de 97,6%. A possível justificativa para esta pequena diferença (correspondente a somente 1 MARFE perdido em relação a versão serial) se da devido a mudança do algoritmo de extração de característica das regiões binarias. O novo algoritmo (cvBlobs) permitiu um ganho de desempenho em tempo de execução da ordem de 7 vezes no tempo de execução do módulo de extração de características FHu (0,675 ms para 0,100 ms, em valores médios conforme apresentados nas figuras 4.2 e 4.4).

**Tabela 6.1:** Matriz confusão para a avaliação da precisão do algoritmo de execução paralela na classificação dos MARFEs (NMf=Não-MARFE; Mf=MARFE; Oth=Outros/Indeterminado)

| BD / PI | **NM** | **Mf** | **Oth** |
|---|---|---|---|
| **NM** | 697 | 3 | 39 |
| **Mf** | 9 | 283 | 5 |
| **Oth** | 2 | 1 | 8 |

**Tabela 6.2:** Precisão do algoritmo paralelo na classificação da ocorrência de MARFE ou ausência do mesmo, dado em porcentagem, baseada em um banco de dados pré-classificado do JET e comparado à versão serial.

| | Serial | Paralelo |
|---|---|---|
| Total | 1527 objetos | 1047 objetos |
| **TP** | **96,01%** | **94,36%** |
| FP | 0,98% | 1,34% |
| FN | 3,01% | 4,30% |

**Tabela 6.3:** Precisão do algoritmo paralelo na classificação da ocorrência de MARFE, dado em porcentagem, baseada em um banco de dados pré-classificado do JET, para um universo restrito às imagens com a presença de MARFE e comparado à versão serial.

| | Serial | Paralelo |
|---|---|---|
| Total | 290 objetos | 290 objetos |
| **Correct Detection** | **97,9%** | **97,6%** |
| Misclassified | 0,69% | 1,38% |
| Not found | 1,38% | 1,03% |

## 6.2 Performance

Para realizar esta análise de performance foram processadas 9950 imagens com um tempo de execução média de 871,92 ms. A taxa média de análise obtida foi de 11.411,6 fps (87,6 $\mu$s). Esse valor por si já excede com larga margem de folga a taxa necessária para garantir 10 análises de imagens (130,7 $\mu$s) dentro do intervalo de tempo de ocorrência de um MARFE, ilustrada na figura 6.1.

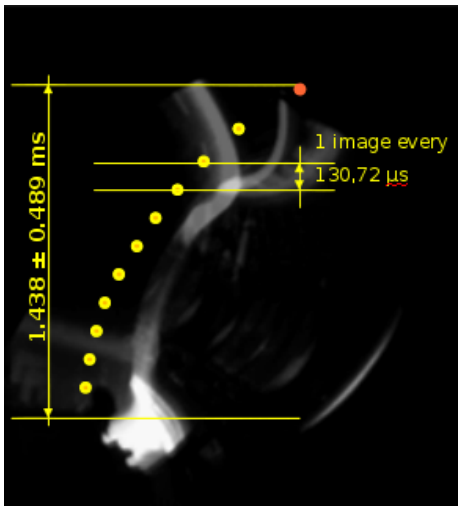



**Figura 6.1:** Disposição para a garantia de 10 frames detectáveis ao longo de uma ocorrência de MARFE.

Para uma análise da performance mais detalhada, foram considerados alguns fatores particulares da forma de operação do algoritmo e da distribuição obtida nos resultados. Como visto na figura 4.8 e, devido a esta forma de análise do algoritmo, a detecção do fenômeno ocorreu exclusivamente abaixo de 133 $\mu$s. Nesta região também ocorreram 94,2% de todas as análises efetuadas.

A figura 6.2 expõe a distribuição do intervalo de tempo de processamento de cada uma das 9950 imagens, obtido com a versão paralela do algoritmo. Ela também destaca a porcentagem de imagens tratadas abaixo e acima do limiar de 133 $\mu$s. Foram processadas 9369 (94,2%) imagens abaixo do limiar de 133 $\mu$s. O ajuste dos pontos neste intervalo por uma distribuição gaussiana levou a um valor de 74,41 $\mu$s para a velocidade de processamento, o que corresponde a uma taxa média de 13439,05 fps. No entanto, 581 (5,8%) imagens foram processadas acima do limiar de 133 $\mu$s. No pior caso o intervalo de tempo do processamento foi de 342,3 $\mu$s, o que corresponderia a uma taxa média de 2920 fps.

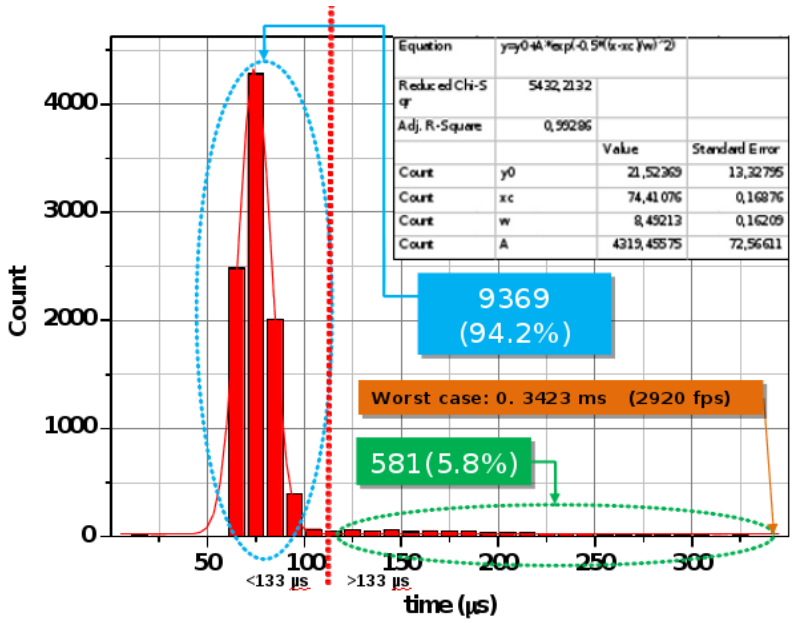



**Figura 6.2:** Distribuição temporal das análises e classificações na detecção de MARFEs. Na região onde ocorreram todas as detecções de MARFE, abaixo de 133$\mu$s, encontram-se em 94,2% das análises.

A figura 6.3 apresenta a distribuição do tempo de duração das ocorrência de todos os MARFEs presentes no banco de dados. Como pode ser observado, há um conjunto de MARFEs que vai do intervalo de aproximadamente 1 ms até o intervalo

de 2,2 ms. Também pode ser observado nesta figura que um único caso ocorreu com um intervalo de 300 $\mu$s. Este caso estava presente na base de imagens e foi constatado que o mesmo se iniciou e não progrediu até a parte inferior da imagem onde se encontra o diversor.

É possível fazer uma análise conjunta do resultado obtido na figura 6.2 com a figura 6.3. Para o caso do limiar inferior aos 133 $\mu$s, em que o processamento se dá com intervalo médio de 74,41 $\mu$s em 94,2% do tempo de processamento, no pior caso (1 ms) seria possível processar 13 imagens. Cabe ressaltar que não está sendo considerada a situação de MARFE que ocorreu com o intervalo de 300 $\mu$s, onde ainda seria possível processar 4 imagens neste intervalo. Por outro lado, com um intervalo de 342,3 $\mu$s (pior caso da figura 6.2, i.e. dois casos em 9950), seria possível processar aproximadamente 3 imagens de MARFEs e seria perdido completamente o fenômeno quando este acontecesse como citado para aquele com o intervalo de tempo de 300 $\mu$s.

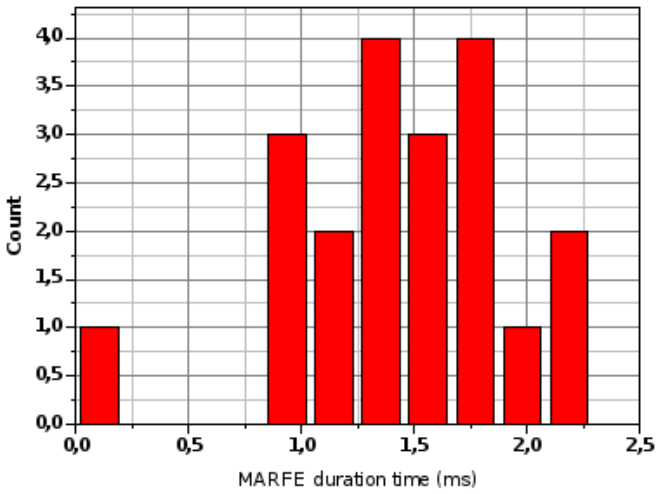



**Figura 6.3:** Distribuição do tempo de duração das ocorrência de todos os MARFEs presentes no banco de dados.

### 6.2.1 *Real Time*

O processo de elevação de privilégios na execução do algoritmo, configurando-o como RT na fila de prioridades do sistema operacional, aliado à configuração de afinidade de CPU e ajuste de comportamento *nice* não somente contribuiu nas medições internas de tempos dos módulos mas também promoveu um pequeno ganho de desempenho. Sem estes ajustes o algoritmo de detecção paralela apresentaria uma depreciação na ordem de 5% aos resultados apresentados no item 6.2.

Em virtude das performances obtidas, este sistema de processamento pode ser considerado como RT. Para 94,2% do casos processados foi possível realizar mais de 10 análises, garantindo assim uma operação com grande margem de tempo para o limite de *deadline* (duração mínima de um MARFE, aproximadamente 1 ms).

# Capítulo 7

# Conclusão

Este trabalho propôs uma abordagem para solucionar o problema da detecção do fenômeno MARFE em tempo hábil e com alta probabilidade de acerto no laboratório de fusão nuclear JET. A versão inicial, desenvolvida por [AAC+12] em processamento serial, obteve um desempenho médio de 650 fps e uma taxa de acerto de 97,9% para a detecção do MARFE. A versão proposta neste trabalho, de execução paralela, alcançou uma taxa média de 11.411 fps mantendo a mesma taxa de acerto na classificação do MARFE. Para a implementação desta versão paralela as etapas de processamento das imagens foram divididas em dois grupos: (i) subtração da imagem média de fundo para detecção de movimento e binarização (segmentação por um *threshold* fixo) e (ii) extração de características dos objetos na imagem binária e sua classificação pelo algoritmo de SVM (Maquinas de Vetores de Suporte). Estes dois grupos tiveram seus códigos implementados por duas técnicas distintas de paralelismo computacional. O primeiro foi desenvolvido em *thread* e o segundo em *fork*. Este sistema foi testado e caracterizado em um ambiente computacional SMP de 8 núcleos [1] sendo um dedicado ao controle, três para as tarefas do grupo 1, três para as as tarefas do grupo 2 e um núcleo restante exclusivo para a sustentação do sistema operacional. A caracterização permitiu uma avaliação do desempenho estatístico dos dois grupos para um conjunto de 9.950 imagens onde se verificou o

[1] *SMP: symmetric multiprocessing (SMP) computer-architecture.*

percentual da taxa de processamento de imagens para atingir o objetivo de operar com pelo menos 10 análises dentro do intervalo de ocorrência de um MARFE.

Foi empregado um cuidado adicional na implementação das técnicas de medidas de tempo para a caracterização dos módulos de processamento de imagens, uma vez que estes são extremamente rápidos apresentando consequentemente maiores complexidades em serem medidos, inviabilizando o uso das ferramentas mais comuns. Esta característica particular obrigou a realização de um estudo e a validação da acurácia, não somente da técnica de medição mas também das informações de frequência de *clock* oferecidas pelo sistema operacional sobre o *hardware* utilizado. Esta preocupação com as medidas de tempo está também associada ao fato de se utilizar uma arquitetura computacional paralela onde os sistemas de medidas de tempo não são, a princípio, sincronizados. Ademais, era nossa preocupação caracterizar os limites de funcionamento do sistema de detecção de MARFE para operação em tempo real.

Os problemas inerentes ao acesso múltiplo à informações compartilhadas entre processos concorrentes (*forks* e *threads*) foram tratados por aplicação de zonas críticas aos acessos competitivos. Uma especial atenção também foi dada ao desenvolvimento de cada parte do programa de execução paralela, balanceando sua demanda de processamento e respectiva distribuição de carga para cada CPU, minimizando o tempo ocioso para o sistema SMP utilizado.

A discrepância entre a precisão da classificação obtida na versão serial e a encontrada na versão paralela do algoritmo pode ser consequência da otimização imposta pelo corte da imagem a ser processada no início, que originalmente foi implementada apenas na versão paralela e interfere nos cálculos de posicionamento e tolerância dos alvos detectados. Ainda que não exatamente iguais, o banco de imagens pré-catalogadas garante a precisão da taxa de acerto, onde a versão paralela se manteve em níveis muito próxima da anterior. Isso pode ser observado claramente em uma análise geral ou somente considerando as imagens com presença de MARFE, pela tabelas 6.2 e 6.3. Com os números apresentados até então, pode-se dizer que o objetivo da manutenção da precisão de classificação do algoritmo serial foi alcan-

çado. Ainda que houvesse uma maior discrepância entre as versões, uma taxa de acerto na versão paralela de 97,6% dos objetos catalogados como MARFE é um índice aceitável para os parâmetros conhecidos do projeto.

Outro objetivo proposto inicialmente foi de se analisar ao menos 10 imagens dentro do intervalo de tempo da ocorrência de um único MARFE, como exposto no item 1.1 e ilustrado na figura 6.1. Esta capacidade de processamento e classificação não somente aumentaria a probabilidade de detecção do fenômeno como também possibilitaria sua detecção em etapas mais iniciais, promovendo um maior tempo para atuação de outros sistemas de controle externos para garantir a estabilidade do processo de fusão.

A taxa média de processamento e classificação de 11.411 fps já seria suficiente para atender ao objetivo traçado de 7.649,5 fps. Uma atenção especial deve ser dada à figura 6.2 que apresenta a distribuição dos intervalos de tempo de processamento e classificação obtidos nos testes realizados. Pode-se observar que 94,2% das análises estiveram abaixo do limite de 133 $\mu$s (limite para processar até 10 imagens dentro do intervalo de ocorrência de um MARFE). O valor médio encontrado para estes casos é de 74,41 $\mu$s e um desvio inferior a $\pm$ 0,17 $\mu$s, levando a uma taxa de processamento na melhor das hipóteses de 13.439 fps. Também pode-se observar na própria figura que 5,8% dos casos ocorreram acima do limite proposto de 133 $\mu$s, sendo que no pior caso tem-se uma taxa de processamento de 2.920 fps (342,3 $\mu$s), mesmo assim ainda seria possível processar até duas imagens dentro do intervalo de ocorrência do MARFE mais rápido dentre àqueles catalogados na base de dados do JET (entre os MARFEs que se manifestaram por completo, da parte superior até a inferior do reator). Em todos os casos (melhor e pior) a taxa de acerto nas classificações dos padrões será de 97,2%.

**Tabela 7.1:** Tabela comparativa das taxas de análises (desempenho) apresentadas neste projeto. A velocidade de processamento apresentada como "média parcial" corresponde ao valor médio para 94,2% dos MARFEs, que ocorreram abaixo do limite de 10 imagens processadas dentro do intervalo de tempo proposto como objetivo.

| Resultados Obtidos (fps) | | | |
|---|---|---|---|
| Anterior | Objetivo | Média Total | Média Parcial |
| 650 | 7.650 | 11.411 | 13.439 |

## 7.1 Propostas Futuras

Uma análise conjunta das figuras 6.2 e 4.8 nos permite uma avaliação complementar. Pode-se afirmar que a detecção dos MARFEs acontece em intervalos de tempos inferiores a 150 $\mu$s e 95% das análise ocorreram abaixo dos 133 $\mu$s. Uma proposta de implementação futura seria o descarte do processamento das imagens, por exemplo por meio de uma interrupção do processo quando o tempo ultrapassasse os 133 $\mu$s. Essa otimização poderia ser estimada mas não foi efetivamente implementada.

O encerramento forçado do tratamento de uma imagem caso seja detectado a presença de um MARFE também seria uma proposta adicional de otimização. Uma vez detectado o fenômeno não há, a princípio, motivos para se continuar a analisar a mesma sequência de imagens.

Outra implementação quase imediata seria a conversão da tecnologia híbrida de paralelismo para uma tecnologia única e otimizada. O uso exclusivo de uma tecnologia de paralelização com um menor *parallel-overhead* implicará certamente em uma redução do tempo de processamento.

Já existem *hardwares* com relativa facilidade de aquisição computadores com um número maior de núcleos (*cores*) que àquele usado nesse projeto. A implementação das divisões de tarefas em mais unidades de processamento locais adicionará mais desempenho nas análises. Com uma frequência interna de 2,66 GHz, a simples execução do algoritmo desenvolvido em um *hardware* semelhante em número de núcleos porém com uma maior frequência poderá oferecer um maior desempenho nas análises, ou o uso de um *hardware* com suporte a memórias com latências menores que o modelo usado por esse trabalho, DDR2-800.

Outro campo que possivelmente carece de melhoria é a reavaliação das atuais divisões de carga. Foi detectado que alguns núcleos podem permanecer ociosos boa parte do tempo. A redistribuição, ainda que executada para o mesmo *hardware*, poderá trazer um significativo ganho de desempenho.

Um dos módulos mais custosos computacionalmente é o de *background* e *open image*. Com a implementação do tratamento e classificação das imagens diretamente no *hardware* de aquisição, na câmera, as chances de um desempenho melhor ficaria condicionada à capacidade de processamento do equipamento. Um estudo adicional poderia avaliar a implementação desses algoritmos em dispositivos eletrônicos como FPGA (*Field-Programmable Gate Array)*, DSP (processadores de sinais) ou GPUs (Unidades processadoras gráficas). Exisitirá no entanto um estudo adicional em como integrar esses dispositivos com os sistemas de aquisição de imagens do JET.

# Referências Bibliográficas


[AAC+12] M. Portes Albuquerque, M. Portes Albuquerque, Germano T. Chacon, A. Murari, and JET-EFDA Contributors. High-speed image processing algorithms for real-time detection of MARFEs on JET. *IEEE TRANSACTIONS ON PLASMA SCIENCE*, 40(12):3485–3492, december 2012.

[AAS+13a] M. Portes Albuquerque, M. Portes Albuquerque, M. G. M. Souza, A. Murari, Nilton Alves, and JET-EFDA Contributors. A 10.000-image-per-second parallel algorithm for real-time detection of MARFEs on JET. *IEEE TRANSACTIONS ON PLASMA SCIENCE*, 41(2):341–349, february 2013.

[AAS+13b] M. Portes Albuquerque, M. Portes Albuquerque, M. G. M. Souza, A. Murari, Nilton Alves, and JET-EFDA Contributors. A 10.000-image-per-second parallel algorithm for real-time detection of MARFEs on JET. *Internal Paper*, page 3, 2013.

[ABD+09] Krste Asanovic, Rastislav Bodik, James Demmel, Tony Keaveny, Kurt Keutzer, John Kubiatowicz, Nelson Morgan, David Patterson, Koushik Sen, John Wawrzynek, David Wessel, and Katherine Yelick. A view of the parallel computing landscape. *Magazine - Communications of the ACM*, 52(10):56–67, October 2009.

[AH29] R. D. E. Atkinson and F. G. Houtermans. Zur frage der aufbaumöglichkeit der elemente in sternen. 54(9-10):656–665, 1929.

[Ama11] Cássio Henrique dos Santos Amador. Determinação da posição da separatriz magnética em TOKAMAKs através de reflectometria de microondas. Doutorado, Centro Brasileiro de Pesquisas Físicas, CBPF, Brasil, Br, 2011.

[Amd67] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. *Proceeding AFIPS '67 (Spring) Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485, August 1967.

[Bar12] Blaise Barney. Posix threads programming. Technical document, Lawrence Livermore National Laboratory - LLNL, feb 2012.

[Cha12] Germano Chacon. Aplicação de técnicas de processamento digital de imagens para a detecção de MARFEs no JET. Master's thesis, Instrumentação Científica, Centro Brasileiro de Pesquisas Físicas, CBPF, 2012.

[Con91] Russell G. Congalton. A review of assessing the accuracy of classifications of remotely sensed data. *REMOTE SENS. ENVIRON.*, 37(2-3):35–46, 1991.

[Dij68] Edsger W. Dijkstra. The structure of the "THE"-multiprogramming system. *Communications of the ACM*, 11(5):314–346, May 1968.

[Ein05] Albert Einstein. Ist die trägheit eines körpers von seinem energieinhalt abhängig. *Annalen der Physik*, (18):639–641, set 1905.

[FM11] Samuel H. Fuller and Lynette I. Millett. *The Future of Computinf Performance - Game Over or Next Level?* National Academy of Sciences, Washington-DC, USA, 3 edition, 2011.

[Gai84] Jason Gait. Semaphores outside the kernel. *ACM SIGPLAN Notices*, 19(10):12–21, 1984.

[GB05] D. A. Gurnett and A. Bhattacharjee. *Introduction to Plasma Physics: With Space and Laboratory Applications*. Cambridge, Cambridge, UK, 2005.

[GRC+08] E. Gomez, D.A. Rani, C.R. Cheeseman, D. Deegan, M. Wise, and A.R. Boccaccini. Thermal plasma technology for the treatment of wastes: A critical review. *Journal of Hazardous Materials*, 161(2-3):614–626, 2008.

[HH97] C. Hughs and T. Hughes. *Object Oriented Multithreading using C++:architectures components*. John Wiley e Sons, Canada, 2ª edition, aug 1997.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Magazine Communications of the ACM*, 21(8):666–677, August 1978.

[IBM06] IBM. Unix-type – interprocess communication (ipc) apis. Ibm systems - iseries, IBM Corporation, 2006.

[IEE04] IEEE. POSIX IEEE Std 1003.1, 2004 edition. Standard padronization, Institute of Electrical and Electronics Engineers - IEEE, feb 2004.

[Ili07] Christian Iliadis. *Nuclear Physics of Stars*. WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim, DE, 2007.

[Jon06] M. Tim Jones. Boost socket performance on linux. Developerworks, IBM Corporation, feb 2006.

[Jon10] M. Tim Jones. Anatomy of linux kernel shared memory. Developerworks, IBM Corporation, abr 2010.

[LCH+09] R. Layne, N Cook, D. Harting, D.C. McDonald, C. Tidy, and EFDA contributions. The jet intershot analysis: Current infrastructure and future plans. *JET White Paper*, June 2009.

[LH89] Kai Li and Paul Hudak. Memory coherence in shared virtual memory systems. *Journal ACM Transactions on Computer Systems (TOCS)*, 7(4):321–358, November 1989.

[LLM+84] B. Lipshultz, B. LaBombard, E. S. Marmar, M. M. Pickrell, J. L. Terry, R. Watterson, and S. M. Wolfe. MARFE: an edge plasma phenomenon. *Nuclear Fusion*, 24(8):977–988, august 1984.

[LMMS+98] A. Loarte, R. D. Monk, J. R. Martín-Solís, D. J. Campbell, S. Clement, S. J. Davies, J. Ehrenberg, S. K. Erents, H. Y. Guo, P. J. Harbour, L. D. Horton, L. C. Ingesson, H. Jäckel, J. Lingertat, C. G. Lowry, C. F. Maggi, G. F. Matthews, K. McCormick, D. P. O'Brien, R. Reichle, G. Saibene, R. J. Smith, D. Stamp, M. F. Stork, and G. C. Vlases. Plasma detachment in JET mark i divertor experiments. *Nuclear Fusion*, 38(3):331–371, set 1998.

[LW02] R. Layne and M. Wheatley. New data storage and retrieval systems for JET data. *Fusion Engineering and Design*, 60:333–339, November 2002.

[Moo65] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[Mur10] Andrea Murari. Algorithms for the automatic identification of MARFEs and UFOs in JET database of visible camera videos. *IEEE TRANSACTIONS ON PLASMA SCIENCE*, 41(2):341–349, february 2010.

[Pao10] Gabriele Paoloni. How to benchmark code execution times on intel ia - 32 and ia - 64 instruction set architectures. *White Paper*, set 2010.

[Pit08] Marcos Pitanga. *Construindo Supercomputadores com Linux*. Brasport, Rio de Janeiro-RJ, Brasil, 3ª edition, 2008.

[RL77] C. V. Ramamoorthy and H. F. Li. Pipeline architecture. *Journal - ACM Computing Surveys (CSUR)*, 9(1):61–102, March 1977.

[SCS+08] Larry Seiler, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, Adam Lake, Jeremy Sugerman, Robert Cavin, Roger Espasa, Ed Grochowski, Toni Juan, and Pat Hanrahan. Larrabee: A many-core x86 architecture for visual computing. *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2008*, 27(3), August 2008.

[SFH05] K. Scherer, H. Fichtner, and B. Heber. *Space Weather: The Physics Behind a Slogan*, volume 656. Springer, Berlin, DE, 2005.

[Smi09] V.P. Smirnov. Tokamak foundation in USSR/Russia 1950–1990. *IOP Publishing and International Atomic Energy Agency*, (50), 2009.

[Tan03] Andrew Stuart Tanenbaum. *Sistemas Operacionais Modernos*. Pearson Prentice Hall, São Paulo, BR, 2003.

[TTF08] Guo-Song Tian, Yu-Chu Tian, and Colin Fridge. High-precision relative clock synchronization using time stamp counters. pages 69–78, 2008.

[Wes06] John Wesson. The science of JET the achievements of the scientists and engineers who worked on the joint european torus 1973-1999. *JET Joint Undertaking*, March 2006.

# Apêndice A

# Publicações

A pesquisa desenvolvida também gerou uma publicação [AAS+13a] na revista *IEEE Transations on Plasma Science*, volume 41, número 2, páginas 341 à 349, de fevereiro de 2013. O conteúdo desse artigo pode ser visto, em sua íntegra, neste apêndice, em forma de *preprint* publicado no JET.

EFDA–JET–PR(11)57

M. Portes de Albuquerque, A. Murari, M. Giovani, Nilton Alves Jr.
and JET EFDA contributors

# A 10,000 Images per Second Parallel Algorithm for Real Time Detection of MARFEs on JET

# A 10,000 Images per Second Parallel Algorithm for Real Time Detection of MARFEs on JET

M. Portes de Albuquerque[1], A. Murari[2], M. Giovani[1], Nilton Alves Jr.[1]
and JET EFDA contributors*

***JET-EFDA, Culham Science Centre, OX14 3DB, Abingdon, UK***

*[1]Centro Brasileiro de Pesquisas Físicas, CBPF/MCT, Rua Dr. Xavier Sigaud 150 Urca, Rio de Janeiro 22290180, Brazil.*
*[2]Consorzio RFX-Associazione EURATOM ENEA per la Fusione, I-35127 Padova, Italy.*
** See annex of F. Romanelli et al, "Overview of JET Results", (23rd IAEA Fusion Energy Conference, Daejon, Republic of Korea (2010)).*

Preprint of Paper to be submitted for publication in
IEEE Transactions on Plasma Science

**ABSTRACT**

This work presents a very high speed image processing algorithm applied to MARFEs detection on JET. The algorithm was built in serial and parallel versions and written in C/C++ using OpenCV, cvBlob and LibSVM libraries. The code implemented was characterized by its accuracy and run-time performance. The final result of the parallel version achieves a correct detection rate of 97.6% for MARFE identification and an image processing rate of more than 10,000 frames/s. The parallel version divides the image processing chain into two groups and seven tasks. One group is responsible for Background Image Estimation and Image Binarization modules and the other for region Feature Extraction and pattern Classification. At the same time and to maximize the workload distribution, the parallel code uses data parallelism and pipeline strategies for these two groups respectively. A master thread is responsible for opening, signaling and transferring images between both groups. The algorithm has been tested in a dedicated Intel Symmetric MultiProcessing (SMP) computer-architecture with a Linux™ operating system.

## 1. INTRODUCTION

Digital image processing has become an important tool in many areas of scientific instrumentation. However, the implementation of image processing techniques can be relatively complex and is often computationally expensive. Several of these techniques can be implemented in electronic devices such as FPGAs (Field-Programmable Gate Array) or DSPs (Digital Signal Processor), or use the potentiality of high-performance computing systems based on multicore processors. High performance computing is a strong candidate mainly because there exists an important set of computing libraries to build an entire and complex processing chain. Although parallelism depends on the problem, the low cost and easy availability of multicore systems and parallel software make it much more attractive than in years past. On the other hand, FPGA is still an interesting option but its adoption in high performance tasks is currently limited by the complexity of FPGA design compared to conventional software.

Magnetic Confinement Nuclear Fusion (MCNF) is one of the recent fields in which digital image processing has become a fundamental tool in scientific instrumentation. Indeed image processing is nowadays very important not only for the interpretation of the experiments but also for pattern retrieval in large data bases [1][2]. In the Joint European Torus, about 30 new cameras have been installed for the current experiments with the new metallic wall. One of the most challenging characteristics of cameras as scientific instruments is the large amount of data that they produce. For example, JET global image data base contains more than 90 terabytes of data of which at least 50% are images [3][4].

In [5] we presented an image processing algorithm to process JET videos of a fast visible camera at high speed. In this first work we described and discussed an image processing algorithm based on several techniques for acquisition, segmentation and classification of patterns to detect a MARFE. The algorithm was characterized in terms of execution time and accuracy. In this context, accuracy

is the system error rate in identifying objects, whereas execution time is the measure of the system speed (which depends on the speed of its constituent parts). That serial algorithm was developed in C/C++ language using the OpenCV library for image processing, the LibSVM library for pattern classification and was tested on a dedicated Intel 64 bits Linux™ computing platform. In its final version, the algorithm presented a correct detection rate of 93.3% and an average of 650 frames processed per second.

Also in that work, the developed algorithm was used to process all frames offline, which in real time would mean processing all visible camera (KL8) images sequentially every 33$\mu$s. With the image processing rate of the serial algorithm, on the other hand, we would reach an analysis of one frame every 46 images in a real time environment. However, the average duration of a MARFE is $1.438 \pm 0.489$ ms. This would imply to analyze only one image within a period of a MARFE, greatly reducing the chance of detection and classification. Considering the performance of the classifier, a good compromise would be a real-time analysis of at least ten frames within a MARFE period. This would lead us to an acquisition rate of 7,575 frames per second (i.e. a period of 132$\mu$s).

Although we obtained a good rate of correct classification, the required performance in terms of computational speed would be unreachable with the serial version, indicating that this kind of problem is a suitable candidate for parallel computing. In this paper we will describe the new serial version with a new Feature Extraction library and its implementation in a parallel algorithm in a multicore systems using SMP architecture.

### *1.1 CAMERA AND JET IMAGE DATA-BASE*

The JET experiment is equipped with a fast visible camera which can grab an image with a rate of 30,000 frames per second. At an extreme limit, the processing algorithm should be able to read a gray level image (224 of width and 256 of height), process and recognize a MARFE within a period of 33$\mu$s. This approach would lead us to process all images acquired by the fast video camera, i.e. a significant technological challenge for real time image analysis. It is part of the study presented in this paper to evaluate techniques to obtain acceptable processing rates at the same time maintaining a high efficiency of MARFE recognition. In this context, speed is critical because MARFEs tend to move very fast in the field of view and, if not enough frames are captured, it becomes very difficult for the recognition algorithm to identify the MARFE. The time response of the actuators is not a major issue in this case because MARFEs have been detected even half a second before a disruption. So speed is necessary for the reliability of the automatic identification not for the response of the actuators.

A data base of videos has been built in JET in order to evaluate different image processing algorithms. It comprises 22 videos with identical optical settings for a total of 4,236 binary regions, containing information about MARFEs only and consisting of: 718 (16.9%) MARFEs, 3,373 (79.7%) Non-MARFEs and 145 (3.4%) other patterns that have morphological characteristics at the border between the first two and could be classified as either. All major plasma events which can affect the

images captured by the visible camera, such ELMs, UFOs etc, are included in this set of discharges [5]. The algorithm proposed in this paper uses this data base as a reference for image analysis [6]. In this work we use a total of 2,523 regions to define the classification model (training) and 1,713 to characterize the algorithm performances.

## 2. MARFE INSTABILITY

MARFE is a fast instability that appears in Tokamaks as a toroidal ring of increased radiation. Since MARFEs cause a significant growth in impurity radiation, they leave a quite clear signature in the frames of visible cameras and can be taken by visible spectrum high-speed cameras used in JET. The videos of these cameras provide essential information for the experiment control and its physical interpretation. An example of MARFE frames from JET fast visible camera, showing the top down movement of a MARFE, is reported in figure 1.

With the aim of characterizing the time evolution of MARFEs as seen by JET fast visible camera, figure 2 reports the MARFE duration in ms obtained from the abovementioned data base. The average time duration of a MARFE is 43.15 ± 14.67 frames (1.438 ± 0.489 ms) with a maximum time of 65 frames (2.17ms) and a minimum of 3 frames (0.1 ms).

## 3. SERIAL IMAGE PROCESSING ALGORITHM

The serial image processing system is divided into a series of modules, each aimed at a specific treatment and detailed in [6], figure 3. An additional Time Analysis and Performance Analyzer modules have been built and are dedicated to the measurement of the code execution time and accuracy respectively. These modules have been implemented in C/C++ language for Linux™ O.S. Three main libraries have been used to assemble the entire image processing system: OpenCV, cvBlob and LibSVM. OpenCV stands for Open Source Computer Vision and is a library of programming functions for real time computer vision and image processing [7][8]. LIBSVM is an integrated software package for pattern classification [9]. The cvBlob library [10] implements the algorithm defined in [11], and obtain an external and internal contours of each blob in the same pass and labels them with 8-connectivity components. The centroid is then computed for each blob that satisfies the filter interval area criterion (only blobs with an area greater than 70 and smaller than 10,000 pixels are taken into account [5]).

### *3.1. ACCURACY AND TIME RESPONSE*

The accuracy of the entire system has been determined using the Expanded Confusion Matrix method. The image processing algorithm has found and processed 1,527 regions, resulting in a performance of 96.01% correctly classified, 0.98% of false positives and 3.01% of false negatives. If we consider only the MARFE samples presented in the test set of the data base (290), we have a total of 97.9% regions correctly detected, 1.38% misclassified and 0.69% not found by the image processing algorithm.

We have placed a series of start and stop probes between each module, in order to analyze their execution time performance. In figure 4, the overall analysis for each module is presented.

Modules Feature Extraction (FHu), Opening Image (Op) and Background Image Subtraction (SAv) together represent 98% of the total run time. The average run time is 1.069 ± 0.059ms with an image processing average rate of 937.98 ± 49.82 images per second. Table 1 shows the complete processing time for all 14 videos used in the time performance analysis. If we do not take into account the Open Image module, the final image processing rate achieves an average value of 2,427.05 ± 206.81 frames per second, Table 1.

However, this sequential code processes only 3.0% of the total images acquired by the fast visible camera. The parallel programming techniques can be used to improve response times. This will be the subject of the next section, where we will detail the development of the parallel version of the MARFE detection algorithm.

## 4. PARALLEL IMAGE PROCESSING ALGORITHM

Even workloads that do have efficient and scalable parallel implementations, these are often larger, more complex, and much more difficult to validate than are their single-threaded counterparts. The two main approaches to parallelism are pipelining and data parallelism. Pipelining partitions the workload in time, so that a given unit of work flows through the processors. In data parallelism approaches, the task is partitioned in space, so that different sets of data are handled by different CPUs. We have used both techniques to build the parallel version of the code.

Another important consideration is about real-time and real-fast systems. The objective of fast computing is to minimize the average response time of a given set of tasks. However, the objective of real-time computing is to meet the individual timing requirement of each task. Rather than being fast (which is a relative term anyway), the most important property of a real-time system is predictability; that is, its functionality and timing behavior should be as deterministic as necessary to satisfy the system main specifications. Fast computing is helpful in meeting stringent timing specifications, but fast computing alone does not guarantee predictability [12], [13] and [14].

In the presented parallel development approach, we are more concerned to real-fast systems. However, we have to keep in mind the time restriction required for the MARFE detection and its subsequent decision making process. At this time we aim to detect the phenomenon as quick as possible and try, for example to correlate it with plasma disruptions. In the best case, the restriction for a real-time system would be a MARFE detection within a limit of 132μs (indicating an effective position of MARFE within 10 acquired images). Or we can also consider a detection in up to 1 ms (indicating a MARFE occurrence with at least one image in this interval). It will be then important to characterize the system related to these limits.

### *3.1. CONSIDERATIONS ABOUT TIME MEASUREMENTS IN PARALLEL SYSTEMS*

Measuring time in parallel applications in a multicore systems requires great care, especially when

they are applied to scientific instruments. There are several ways to measure time on computers. The safest way is to consider the whole system as a complete instrument (hardware and software) and use the wall-time measurement method. In the implementation of the Time Analysis module, we can use the time stamp counter (rdtsc) technique or the clock_gettime() for clock and timer measurement on Linux™ . The tsc is a 64-bit register present on all x86 processors and is an excellent, low-overhead high resolution way of getting CPU timing information. Nevertheless, we have to be very carefully when using rdtsc technique (e.g. you have to serialize the instructions before calling the rdtsc function; use the 64 bits version of the tsc register, guarantee a fix clock rate during the code execution, among others). Please refer to Intel document [15] for more detailed information about benchmarking an Intel system with the rdtsc technique. For exactly all of these reasons, we decided to use the POSIX clock_gettime function, with CLOCK_MONOTONIC argument as a system clock parameter. The use of CLOCK_MONOTONIC will protect the Time Analysis module against an incorrect measurement if any change of the system clock occurs (for example by system administrator or ntp - network time protocol), [16]. However, in a multicore environment, any type of high resolution time measurements deserves even more attention. All measurements must be done in a single core, since there is no guarante that the timestamp counter registers of multiple CPUs on a single motherboard will be synchronized.

### *3.2. PARALLEL PRIORITIES ADJUSTMENTS*

Before executing the code, it is first necessary to set the real-time scheduling priority. We have used the SCHED_FIFO priority to run all the processes in different CPUs with the maximum process scheduling priority (normally -20 on Linux). On a multicore system, it is also important to direct the process to a specific CPU (also known as hardware thread). We have implemented these requirements with taskset command in order to set for each process a CPU affinity. In some applications we should use mlockall() system call in order to avoid page faults. With this procedure it is possible to lock, as memory-resident, all of the process´s current memory and all future mapping.

### *3.3. PARALLEL ALGORITHM STRATEGY*

The five modules presented in section III have been adapted to the parallel version of the code. In parallel execution, load balancing is important for maximizing the workload distribution [17]. We will detail this code in the next sections.

#### *Open Image module*

This parallel version stores, in the program address space, a set of 9950 images. At this point of the processing chain, the image can be considered as a set of pixels with independent brightness values. One of the main goals of the opening image module is the transferring of the images from a memory region to the part of the code where it will be processed by the subsequent modules. This technique would be equivalent to a dedicated hardware for image acquisition, controlled by the algorithm. It is also a main goal of this study to determine the maximum time for transferring an

image from an acquisition system into the program, forming a complete and integrated acquisition and processing system for MARFE detection.

Since a MARFE can occur only in a reduce region of the image, this parallel version has been implemented by performing a crop in the original image defining a Region of Interest (RoI) where the processing is concentrated. This RoI was defined based on the JET image data base information, and from now on only these cropped images will be considered as shown in figure 5. It is worth mentioning that no image mask has been used in this work.

***Parallel Modules***

We have divided the image processing modules into two groups. Both groups have been re-divided into 3 tasks to be executed in parallel. Background Image Subtraction (SAv) and Image Binarization (Bin) have been brought together in Group1 and The Feature Extraction (FHu) and Classification (Cls) modules in Group 2. All processes were scheduled with the maximum Linux priority and in real-time, arranged in a FIFO queue and each one restricted to a specific CPU. Finally, one exclusive CPU controls the whole execution. The overall scheme is presented in figure 6 and detailed below.

- **Group 1:** Background Image Subtraction and Image Binarization have been parallelized using data parallelism strategy. The image is divided into three equal regions in order to be processed at the same time by three different CPUs. For the whole process, we implement a thread synchronization and protection of the shared variables using critical section programming techniques. A critical section corresponds to the part of the code that accesses a shared resource that must not be concurrently accessed by more than one thread. Critical sections have also been used to control the allocation of images for Group 2 tasks.

  The working image was divided into three slices and each slave task evaluates its part of the background image. They have to compute a new binary image and recalculate the next background image as the mean of the previous N frames (N = 23), figure 7. A binary image is the resulting image of the difference between the current image and the background one. The binary image is then generated, whether the resulting value, of a per pixel operation, is greater (or not) than the previous established threshold value of 13 [5]. At the end of each slave calculation a shared and protected variable is set in order to synchronize the threads with the master control task.

- **Group 2:** the Feature Detection and Classification Modules have been parallelized using a pipelining policy. When the binary image calculation is ended by the Group 1 tasks, it is ready to be transferred to a Group 2 one. The algorithm performs a polling procedure to determine which of the three CPUs is available, and transfers the entire binary image to be processed by the Feature Extraction and Classification (SVM) modules. It is worth mentioning that in a given instant of time, Group 2 is working on three different images. This is the main gain of the pipelining processing method.

### *3.4. PERFORMANCE ANALYSIS*

We have used G++ GNU Compiler version 4.4.5, OpenCV library version 2.2, cvBlob 0.10.3 and LibSVM library version 3.1 (04/2011). The computer test platform is a Linux™ cluster node with the following characteristics: Supermicro with 2 motherboard, 8GB of RAM memory each, with 2x Intel CPU Xeon E5430 HarperTown Quad-Core 2666.431MHz (8 cores/ motherboard), 6MB of cache memory and 1TB of HD SATAII. O.S.: Unix-Like x86_64 / Ubuntu 10.10, kernel: 2.6.35-22-SMP, x84_64. All the results presented in this paper have been obtained with this 64 bits computing platform in a 8 core SMP architecture. Seven cores were used to implement and run this parallel image processing code. It should be noted that in this work we have left one CPU to be used by the operating system tasks.

**Accuracy**

The classification error rate of the parallel version is similar from that obtained with the serial version. However, in the parallel version we have worked with cropped images and the number of the total blobs regions analyzed by the algorithm is obviously lower. The final result for the system performance is shown in Table 2. The parallel image processing algorithm has found and processed 1,047 blobs regions, resulting in a performance of 94.4% correctly classified, 1.34% of false positives and 4.3% of false negatives. If we consider only the MARFE samples presented in the test set of the data base (290), we have a total of 97.6% MARFEs correctly detected, 1.38% misclassified and 1.03% not found by the algorithm, figure 8.

***Time Analysis***

In the time analysis we are concerned with the evaluation of the algorithm execution time and its capability to achieve the best performance in terms of number of frames per second. The algorithm is divided into seven tasks, placed in eight available CPUs. The operational timing diagram is shown in figure 9. It should be mentioned that this is an estimated diagram, since we cannot guarantee the synchronization of all seven CPU´s clocks in the system. However, the final measurement of the processing speed can be obtained observing the opening image rate, which is held by the Master Control Task. A new image will only be opened and processed after the completion of the previous one, by Group 1 and its posterior allocation for Group 2. In this figure, Group 1 tasks (SAv and Bin) are executed by processors 1 to 3 while the second group of tasks (FHu and Cls) are performed by processors 4, 5 and 6 for a sequence of three frames (images 552, 553 and 554 are used to illustrate this working example). The CPU number 0 is responsible for opening, signaling and transferring images between Group 1 and Group 2. The figure shows, schematically, the output sequence of Group 2 tasks (Output time line). The interval between them, for this example, was 84$\mu$s.

With this approach, it is possible also to analyze the workload distribution of the three CPUs dedicated to Group 2 processes. For this example (a run test of 9950 images), we observe that CPU P#4 is requested 45.9%, P#5 43.4% and P#6 10.8% of the time. This suggests that other strategies can be tested in order to reach better performances.

Another analysis is the evaluation of the execution times: total and for each individual frame. A total of 9950 images have been processed with a rate of 11,411.55 frames per seconds (87.63 ± 31.83 μs). For the individual case, it is possible to evaluate the distribution of the opening image interval. This is the most appropriate time to be measured, since it can be observed in a single CPU during the whole process. The distribution of these processing values can be seen in figure 10. The central value of the Gaussian fitting function was 79.82 ± 9.20 μs (i.e., a frame rate of 12,528.19 ± 1,463,43). In this figure, a dashed vertical line is placed at time 132 μs, and indicates that for 9449 images (94.96%) the execution time is below this value and for 501 ones (5.04%) above. The worst case is 363.8 μs (equivalent to a rate of 2748 fps). The inset graph presents the decay of the distribution for values above 132 μs and up to 363 μs.

It is also important to analyze this performance in relation to the MARFE duration time presented in figure 2. The average time of a MARFE duration is 1.438 ms; and the minimum value found for the distribution, shown in figure 2, is 0.933 ms. We do not take into account the lowest case presented in the histogram of figure 2. In this only one pulse sequence, based on JET data base, a MARFE starts and does not keep a continuous behavior up to the bottom of the image. This unique case presents a MARFE duration of 100 $\mu$s (single bar in the left of the histogram of figure 2). With the current performance, the algorithm will sample, at least 10 images for 94.8% of MARFEs cases. For the average values, the system will sample 16 images within a MARFE duration period. In 5.2% of the cases the algorithm will sample at a rate less than 10 samples within a MARFE period. Based on the worst rate of processing (363.8 μs, i.e., 0.01% - 1 images in 9950 processed) and for a MARFE duration of 28 frames (0.933 ms) it is still possible to acquire at least two images in this rare event. In all of these cases we will have a hit rate of 97.6% as described in the previous section.

## CONCLUSIONS AND FURTHER DEVELOPMENTS

In this work we have described a very high speed image processing algorithm for MARFE detection. A serial and parallel version of the algorithm have been presented with their respectively execution time and performance rates. The parallel version achieves an average execution rate of more than 10,000 frames per second with a correct detection of 97.6% for MARFE identification.

The processing in a real environment still depends on the system settings of the high performance camera. Since the procedure for transferring images from camera to computer is not yet defined, we can estimate the transfer limits between the camera acquisition system and the computer. In order to keep the strategy to process ten images inside a MARFE period, with the current processing rate of 11.411,55 fps, we can estimate a transfer delay of 50 $\mu$s (based on the worst case of MARFE duration, i.e., 0.933 ms). An FPGA integrated in a PCI-e card can be an appropriate approach to solve this problem. For example, the PCI-e 1.0a has a transfer rate of 250 MB/s per lane (a full-duplex byte stream). With three lanes we can reach this speed. Furthermore we can also transfer part or the entire Group 1 tasks (background image estimation and image binarization) processing to be executed in this electronic device.

Another approach, to increase even more the processing rate, is to test new parallels strategies. One of the most interesting, which can be derived from figure 9, is to add a new CPU to Group 1 tasks. With this approach we can certainly better equalize the use of the three CPUs executing Group 2 tasks and reduce the idle time between two successive tasks. A new test in a new parallel Intel SMP computer with more cores and higher clocks can also be considered in order to increase the overall performance. For example, if we use an Intel Xeon processor (5690 Series), with a clock rate of 3.46 GHz and 12 cores (2 processors), we can use 2 more cores for Group 1 tasks and in that situation we can probably achieve a frame rate of about 21,000 fps.

Nevertheless, based on the current version implemented, it is still important to add a control module in order to safeguard Group 2 tasks execution time. This new module main goal would be the controlling of the maximum execution time for Group 2 tasks. This will avoid the rare event of outputting out-of-order results.

**ACKNOWLEDGMENT**

The authors thank F. M. Barcellos de Sousa for his useful advice in electronic devices, the support from the National Council for Scientific and Technological Development (CNPq) and the Brazilian Research and Projects Financing (FINEP) of the Brazilian Ministry of Science and Technology. This work has been carried out under the European Fusion Development Agreement under the Brazil - EURATOM collaboration agreement on fusion research, coordinated by the Brazilian Fusion Network and by the Brazilian Nuclear Energy Commission. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

**REFERENCES**

[1]. J. Vega, A. Murari, G.A. Rattá, P. Castro, A. Pereira, and A. Portas; "Structural Pattern Recognition Techniques for Data Retrieval in Massive Fusion Databases"; AIP Conf. Proc. Vol **988**, pp. 481-484, Burning Plasma Diagnostics: An International Conf., March 12; 2008

[2]. J. Vega; "Intelligent methods for data retrieval in fusion databases"; Fusion Engineering and Design. vol. **83**; pp. 382-386; 2008

[3]. R. Layne et al; "New data storage and retrieval systems for JET data"; Fusion Engineering and Design, vol. **60**, Issue 3, June, pp.333-339; 2002

[4]. R. Layne et al, "The JET Intershot Analysis: Current infrastructure and future plans "; Fusion Engineering and Design, vol. **85**, Issues 3-4, July, pp.403-409, 2010

[5]. Murari, A.; Camplani, M.; Cannas, B.; Mazon, D.; Delaunay, F.; Usai, P.; Delmond, J.F.; , "Algorithms for the Automatic Identification of MARFEs and UFOs in JET Database of Visible Camera Videos," Plasma Science, IEEE Transactions on , vol.**38**, no.12, pp.3409-3418, Dec. 2010

[6]. M. Portes de Albuquerque, A. Murari, E.L. Farias, G. Chacon, Marcelo P. de Albuquerque; "High Speed Image Processing Algorithms for Real Time Detection of MARFEs on JET", submitted to IEEE Transactions on Plasma Science - available in http://www.iop.org/Jet/fulltext/EFDP11031.pdf ; 2011

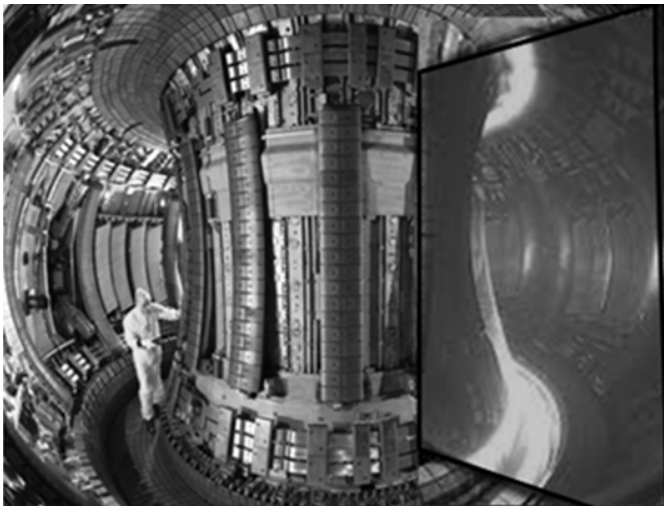
[7]. G. Bradski and A. Kaehler; "Learning OpenCV: Computer Vision with the OpenCV Library", 978-0596516130, O'Reilly Media; 2008
[8]. OpenCV (Open Source Computer Vision); Web site: http://opencv.willowgarage.com/wiki/ ; 2011
[9]. Chih-Chung Chang and Chih-Jen Lin; "LIBSVM library for support vector machines"; Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm; 2001
[10]. C. Carnero Linan; cvBlob: a library for computer vision to detect connected regions in binary digital images. It performs connected component analysis and features extraction; http://code.google.com/p/cvblob/ ; 2011
[11]. Fu Chang, Chun-Jen Chen and Chi-Jen Lu; "A linear-time component-labeling algorithm using contour tracing technique"; Computer Vision and Image Understanding; vol. **93**, Issue 2, pp.206-220; February; 2004
[12]. P. E. McKenney; "'Real Time' versus 'Real Fast': How to Choose?"; Proceedings of the 11th Linux Symposium - Dresden, Germany, Sept. 2009
[13]. K. Koolwal; "Myths and Realities of Real-Time Linux Software Systems"; Proceedings of the 11th Linux Symposium - Dresden, Germany, Sept. 2009
[14]. Gambier, A.; "Real-time control systems: a tutorial"; 5th Asian Control Conference, vol.**2**, pp. 1024- 1031, July; 2004
[15]. G. Paoloni; " How to Benchmark Code Execution Times on Intel® IA-32 and IA-64 Instruction Set Architectures"; http://download.intel.com/embedded/software/IA/324264.pdf; Sept. 2010
[16]. McKenney; "When Do Real Time Systems Need Multiple CPUs?" Proceedings of the 11th Linux Symposium - Dresden, Germany, Sept. 2009
[17]. Agner Fog; "Optimization manuals: Optimizing software in C++: An optimization guide for Windows, Linux and Mac platforms"; Copenhagen University College of Engineering; Jun/2011

| | | | with Open Image Module | | without Open Image Module | |
|---|---|---|---|---|---|---|
| | JET Video | Total Image Processed | Frames/s | Mean Execution Time (ms) | Frames/s | Mean Execution Time (ms) |
| 1 | KL8_70029V2 | 400 | 908.32 | 1.101 | 2458.845 | 0.407 |
| 2 | KL8_70032V6 | 400 | 1001.345 | 0.999 | 2567.394 | 0.389 |
| 3 | KL8_70033V3 | 400 | 1008.314 | 0.992 | 2602.692 | 0.384 |
| 4 | KL8_70053V3 | 400 | 918.734 | 1.088 | 2377.358 | 0.421 |
| 5 | KL8_70053V4 | 400 | 936.736 | 1.068 | 2513.052 | 0.398 |
| 6 | KL8_70053V6 | 400 | 973.861 | 1.027 | 2537.475 | 0.394 |
| 7 | KL8_70054V3 | 400 | 940.514 | 1.063 | 2408.289 | 0.415 |
| 8 | KL8_70054V5 | 400 | 971.685 | 1.029 | 2524.28 | 0.396 |
| 9 | KL8_70055V2 | 400 | 869.918 | 1.150 | 2250.744 | 0.444 |
| 10 | KL8_70055V3 | 400 | 919.656 | 1.087 | 2514.332 | 0.398 |
| 11 | KL8_70055V4 | 400 | 967.827 | 1.033 | 2479.897 | 0.403 |
| 12 | KL8_70056V4 | 400 | 970.645 | 1.030 | 2526.832 | 0.396 |
| 13 | KL8_70056V5 | 400 | 916.8 | 1.091 | 2439.233 | 0.41 |
| 14 | KL8_70097V2 | 2000 | 827.301 | 1.209 | 1778.216 | 0.562 |

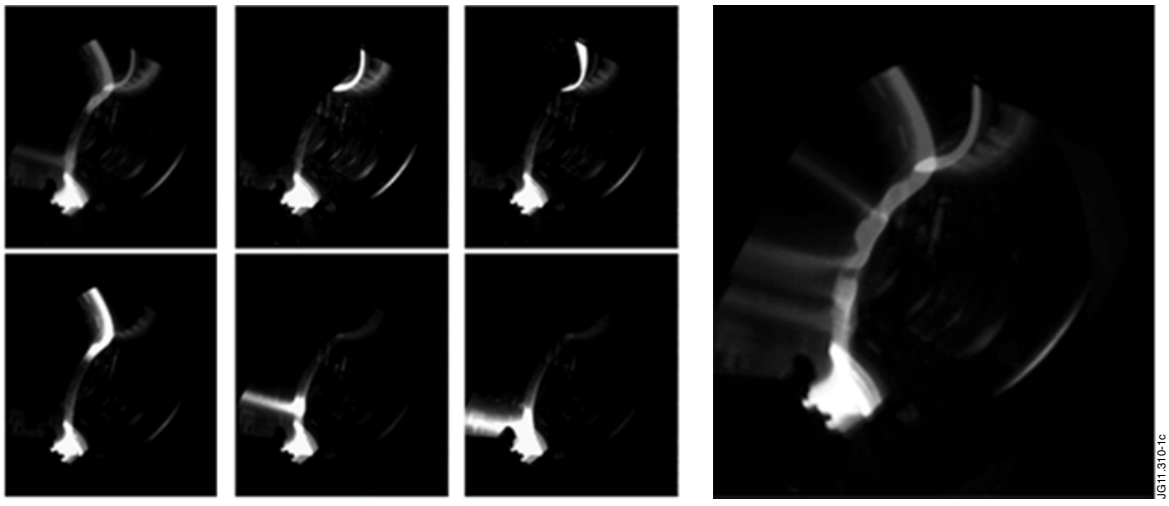
*Table 1: Mean execution time (and frame/s) for 14 JET videos (7,200 images).*

| PARALLEL VERSION | | | | |
|---|---|---|---|---|
| | | BD | | |
| | | Non-MARFE | MARFE | Other |
| PI | Non-MARFE | 697 | 3 | 39 |
| | MARFE | 9 | 283 | 5 |
| | Other | 2 | 1 | 8 |

*Table 2: Confusion Matrix for the performance analysis of the parallel version of the code. It contains information about current image processing algorithm (IP) and predicted (BD) classifications. We rely on an image data base previously classified with the expected result for Non-MARFE, MARFE and Other.*



(a)




(b) (c)

*Figure 1: (a) Overall view of the inside of the JET Tokamak. The highlighted region corresponds to the image obtained by the fast visible camera (http://www.jet.efda.org). (b) From left to right, top to bottom, a typical sequence of a MARFE as captured by JET fast visible camera; (c) Superimposed sequence of MARFEs images. The top down signature of a MARFE can be observed in the left region of the image.*

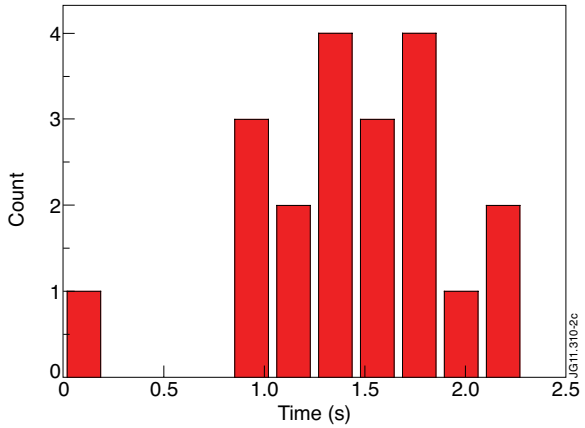



*Figure 2: MARFE duration distribution for 22 videos from JET data base.*

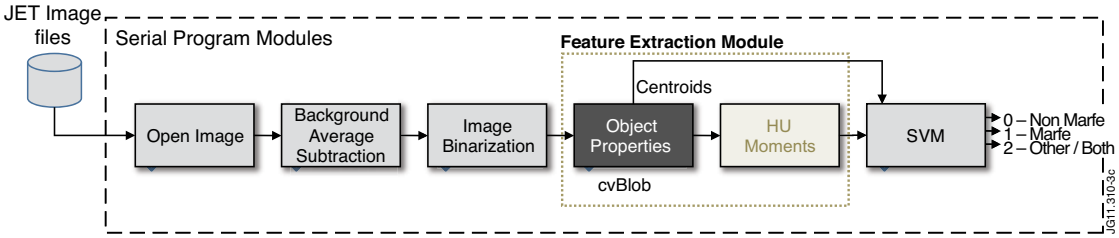



*Figure 3: Modules of the serial version of the Image Processing algorithm for MARFE detection.*

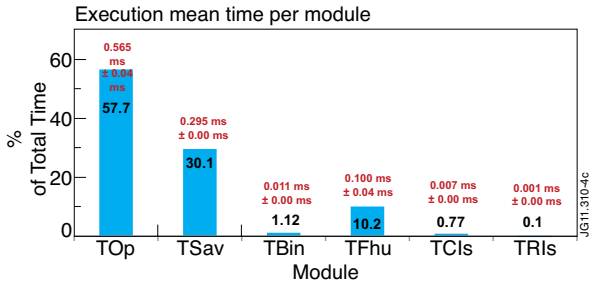



*Figure 4:Percentages of total time spent in each image processing module for 7,200 frames in 14 JET videos.*

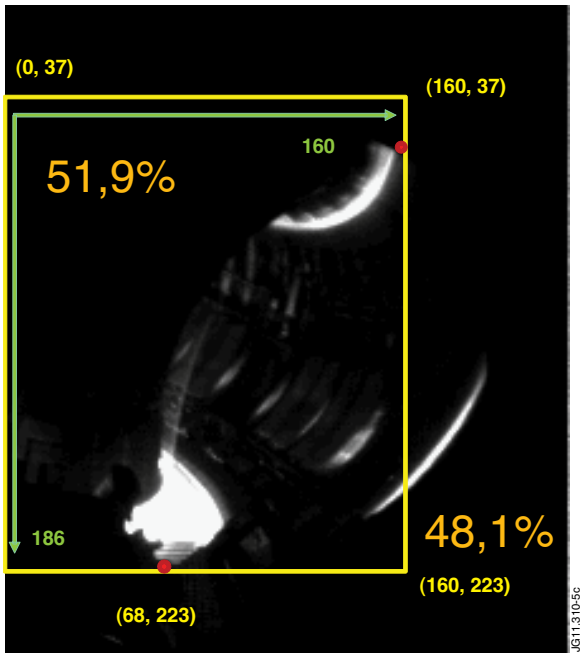



*Figure 5: The new Region of Interest for the parallel algorithm implementation. The final image size is 160 ×186 pixels and is 51.9% smaller than the original one.*

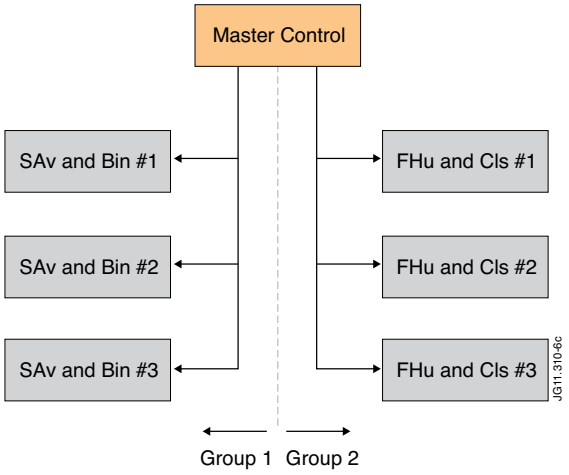



*Figure 6: Parallel strategy: the code was divided into two groups (1: Background Subtraction Average and Image Binarization modules and 2: Feature Extraction and Classification modules). Each group was re-divided and executed in three exclusive CPUs. A Master thread, running also in a exclusive CPU, controls the whole processing dividing the tasks among all other processors.*

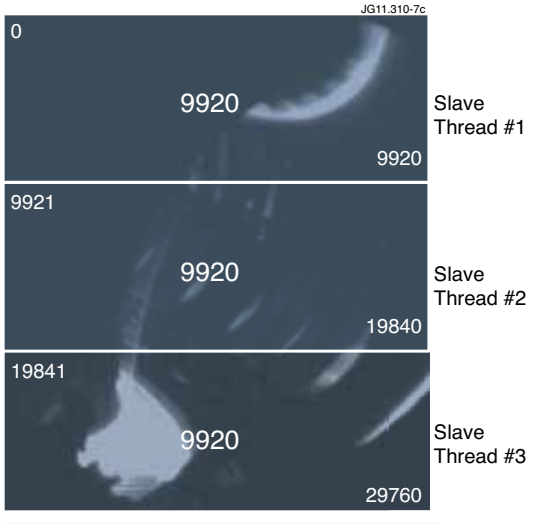



*Figure 7: Parallel strategy for background calculation and image binarization. The cropped image represents the Region of Interest to be processed. Three regions of 9920 pixels are obtained from an image size of 160 x 186 pixels.*

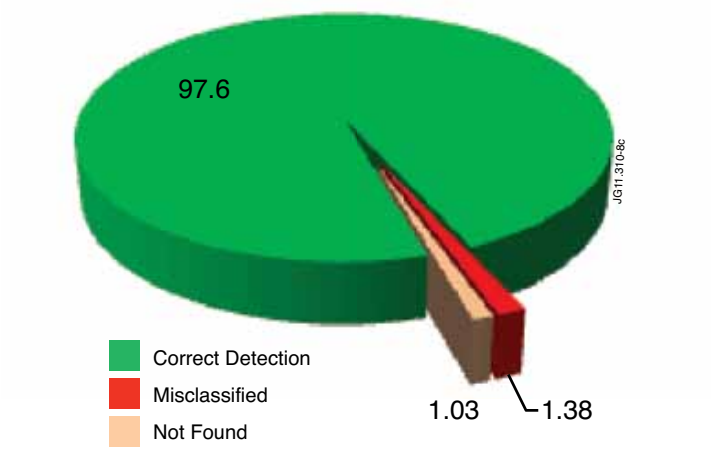



*Figure 8: Percentages of the MARFE detection parallel algorithm.*

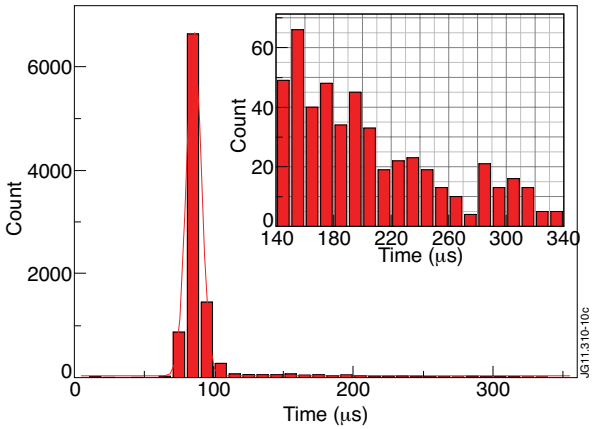



*Figure 9 - Timing Diagram of the parallel version of the algorithm for an sequence of three images (552, 553 and 554). Group 1 tasks (SAv and Bin) use processors 1 to 3 in data parallelism; Group 2 tasks (FHu and Cls) are processed by nodes 4, 5 and 6 in a pipeline fashion. Processor number 0 is responsible for controlling the whole activity. The three numbers inside the time line of Group 1 (23, 32 and 29µs) show the time spent for background accumulation, subtraction and image transfer to Group 2 respectively.*

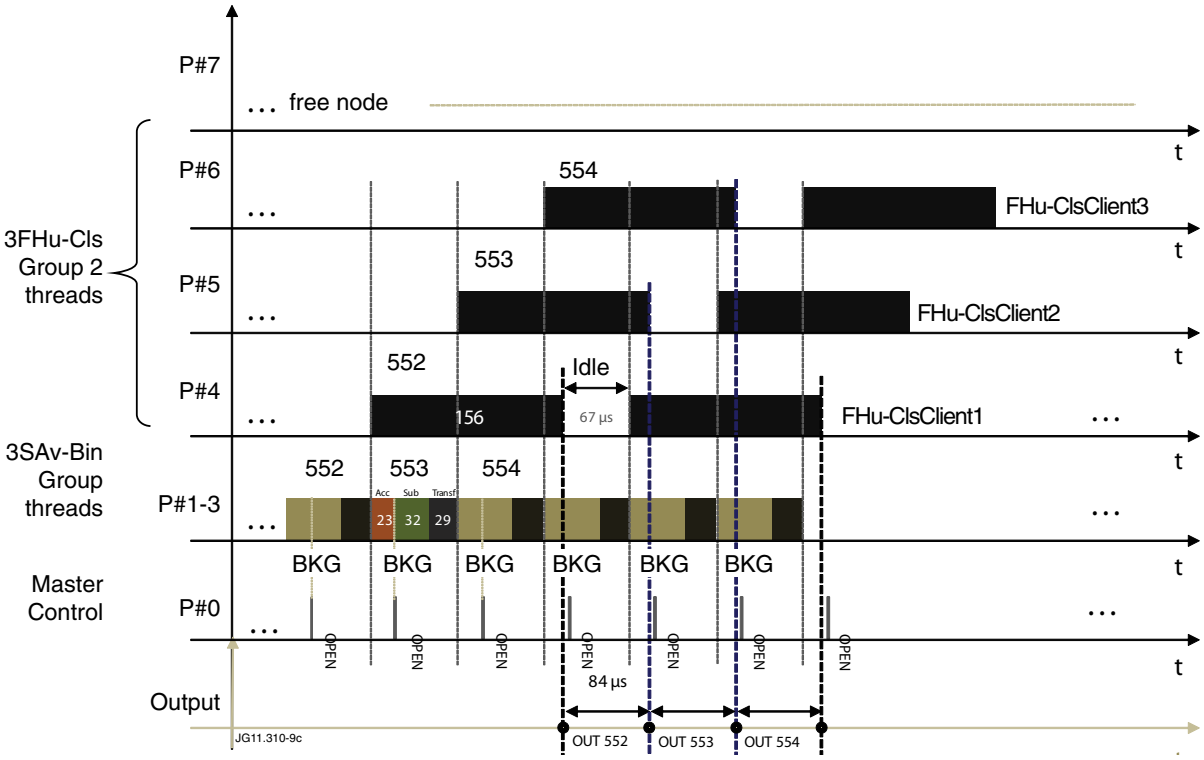



*Figure 10 - Timing diagram of the parallel version of the algorithm. The vertical bar indicates the limit of 132µs. A percentage of 94.96% images were processed below this limit. The inset shows the distribution decay for values above this limit. The average value was 87.63µs (11,411.55 frames per second).*