UNIVERSIDADE FEDERAL DO AMAZONAS - UFAM
INSTITUTO DE COMPUTAÇÃO - ICOMP
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA - PPGI

Integração de Características Preemptivas à Técnica de Escalonamento Dinâmico de Tensões e Frequências Intra-Tarefa

por

Rawlinson da Silva Gonçalves

Manaus - Amazonas

10 de julho de 2015

PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO AMAZONAS - UFAM
INSTITUTO DE COMPUTAÇÃO - ICOMP
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA - PPGI

Integração de Características Preemptivas à Técnica de Escalonamento Dinâmico de Tensões e Frequências Intra-Tarefa

por

Rawlinson da Silva Gonçalves

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Instituto de Computação da Universidade Federal do Amazonas como requisito para a obtenção do grau de Mestre em Engenharia de Software e Sistemas Embarcados.

Orientador: Dr. Raimundo da Silva Barreto

Manaus - Amazonas
10 de julho de 2015

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Goncalves, Rawlinson da Silva
G635i Integração de Características Preemptivas à Técnica de Escalonamento Dinâmico de Tensões e Frequências Intra-Tarefa / Rawlinson da Silva Goncalves. 2015
160 f.: il. color; 29,7 cm.

Orientador: Raimundo da Silva Barreto
Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. Sistemas Embarcados. 2. Sistemas de Tempo Real. 3. DVFS Intra-Tarefa. 4. Redução do Consumo de Energia. 5. Tratamento de Preempções. I. Barreto, Raimundo da Silva II. Universidade Federal do Amazonas III. Título

PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
INSTITUTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

UFAM

FOLHA DE APROVAÇÃO

"Integração de Características Preemptivas à Técnica Escala Dinâmica de Voltagem e Frequência Intra-Tarefa"

RAWLINSON DA SILVA GONÇALVES

Diessertação de Mestrado defendida e aprovada pela banca examinadora contituída pelos Professores:

Prof. Raimundo da Silva Barreto - PRESIDENTE

Prof. Lucas Carvalho Cordeiro - MEMBRO INTERNO

Prof. Rivalino Matias Júnior - MEMBRO EXTERNO

Manaus, 10 de Julho de 2015

Dedico esta dissertação aos meus pais, que tanto investiram durante anos em minha formação humana e acadêmica, sempre me apoiando para que eu pudesse alcançar os meus objetivos.

Agradecimentos

Primeiramente a Deus por todos esses anos ter me proporcionado saúde, força e perseverança, para que eu pudesse percorrer todo esse caminho e ter chegado até aqui, mesmo diante de algumas "topadas" que levei durante essa jornada.

Aos meus pais, familiares e amigos que sempre acreditaram no meu potencial, me dando todo o apoio, carinho, atenção e incentivo para a realização dos meus sonhos. Gostaria de agradecer especialmente as minhas tias Maria Antônia, Inês e Luíza, ao tio José, aos meus primos Diego e Débora, a minha irmã Eliza e, principalmente, a minha mãe Joana por terem me ajudado a cuidar do meu pai ao longo desses anos, seja cuidando dele no dia-a-dia, levando-o para ter momento de lazer ou nas consultas médicas. Dessa forma, consegui ter mais tempo para me dedicar as atividades do mestrado, sabendo que ele estava em boas mãos.

Ao Prof. Dr. Raimundo da Silva Barreto, que ao longo desses anos contribuiu significativamente para minha formação acadêmica, profissional e pessoal, desde a graduação (meados de 2007). Acima de tudo, pelas orientações, conselhos, ensinamentos e críticas que foram primordiais para a conclusão desta dissertação.

Aos meus amigos Diego, Gabriel, Herbert, Spósito e Valentin, as minhas amigas Daniella e Odette (todos do laboratório GISE) que compartilharam seus conhecimentos como pesquisadores, para que eu amadurecesse mais rapidamente.

Aos meus amigos do Projeto da Samsung Anderson, Alberto, Erick, Larissa Ayres e Larissa Bentes pelos momentos de aprendizado e trabalho em equipe durante o desenvolvimento do projeto.

Aos meus amigos dos PPGI, em especial aos grupos ExperTS e Labotim, Nilmara, Bruno, Vitor, Rallyson, Juan, Jhonathan, Anderson e Rayner pelas horas de estudo e aprendizado.

Enfim, a todos pelos momentos únicos de descontração durante as refeições, lanches e *happy hours*, sempre mostrando companheirismo e amizade.

À FAPEAM pelo apoio financeiro. A UFAM por prover a infraestrutura necessária para o desenvolvimento desta dissertação. Aos meus amigos da secretaria do PPGI Elienai, Frank e Helen pela ajuda com os procedimentos administrativos e documentações referentes ao mestrado.

Principalmente, a todos que me deram o apoio necessário para ajudar a amenizar a perda do meu Pai, Carlos Gonçalves Viana (em memória), que fique registrado: Ele foi um grande homem. Não tinha muito estudo, mas era muito sábio e tinha uma visão única de ver o mundo e as pessoas.

A todos os demais professores, onde tive o privilégio de ser aluno e que também tiveram sua parcela de contribuição para que eu fosse capaz de concluir esta dissertação.

Enfim, a todos que direta ou indiretamente puderam contribuir para o desenvolvimento desta dissertação. Meu muito obrigado!

"Só sabemos com exatidão, quando sabemos pouco; à medida que vamos adquirindo conhecimentos, instala-se a dúvida."
(Johan Wolfgang Von Goethe)

RESUMO

Os sistemas embarcados têm evoluído significativamente nos últimos anos, principalmente devido aos avanços da tecnologia, a redução dos custos dos equipamentos eletrônicos e a popularização dos dispositivos móveis. Muitos desses sistemas dependem da energia provenientes de baterias para manter o funcionamento dos seus diversos componentes. No entanto, para que esses dispositivos tenham uma boa autonomia, várias técnicas e metodologias têm sido propostas para melhor gerenciar o consumo de energia do sistema como um todo. Essa necessidade tem contribuído para o surgimento de diversas linhas de pesquisa, principalmente na área de sistemas de tempo real, onde o fator complicante não está somente em reduzir o consumo de energia, mas também em respeitar as restrições temporais de todas as tarefas em execução no sistema.

Sendo assim, este trabalho tem como objetivo diminuir o consumo de energia do processador utilizando a técnica de escalonamento dinâmico de tensões e frequências do processador intra-tarefa, também conhecido como DVFS intra-tarefa (em inglês, *Dynamic Voltage and Frequency Scaling*). A metodologia *online* proposta visa realizar o gerenciamento das trocas de tensões e frequências do processador, através de uma abordagem colaborativa entre as aplicações de tempo real e o sistema operacional. Dessa forma, ambos podem trabalhar em conjunto, dentro do núcleo do sistema, para diminuir os tempos de resposta dos chaveamentos de tensões e frequências do processador, principalmente diante de sucessivas preempções entre as aplicações de tempo real em execução no sistema. Os resultados experimentais dessa metodologia, utilizando o C-Benchmarck, mostraram que é possível diminuir cerca de 6% o consumo de energia do processador, mesmo executando todas as tarefas no pior caso.

Palavras-Chave: Sistemas Embarcados, Sistemas de Tempo Real, DVFS Intra-Tarefa, Redução do Consumo de Energia, Tratamento de Preempções.

ABSTRACT

Embedded systems have evolved significantly in recent years, mainly due to advances in technology, cost reduction of electronic equipment, and mainly the popularization of mobile devices. Many of these systems require energy resources from battery to maintain the operation of their various components. However, for these devices to have a good autonomy, several techniques and methodologies have been implemented to better manage energy consumption of the system as a whole. This need has contributed to the rise of various lines of research, mainly in the area of real-time systems, where the complicating factor is not only reducing energy consumption but also respect the time constraints of all tasks running on the system.

Thus, this work aims to maximize energy gains from the use of intra-task dynamic voltage and frequency scaling technique, also known as intra-task DVFS. The proposed online methodology aims to achieve better management of exchanging voltages and frequency of the processor, through a collaborative approach between real-time applications and the operating system. Therefore, both can work together, within the kernel of the system, to reduce the response times of the processor context switches, mainly after preemptions. The experimental results, using the C-Benchmarck, showed that it is possible to decrease about 6% processor power consumption even performing all tasks in the worst case.

Keywords: Embedded Systems, Real Time Systems, DVFS Intra-Task, Low Power Consumption, Preemptions Treatment.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Códigos Fonte

Lista de Abreviações e Acrônimos

ACEP	*Average-Case Execution Path*
ACM	*The Association for Computing Machinery*
API	*Application Programming Interface*
AVS	*Automatic Voltage Scaler*
BSORT	*Bubble Sort*
DVFS	*Dynamic Voltage and Frequency Scaling*
DVS	*Dynamic Voltage Scaling*
CDVS	*Combined DVS*
CDVS-NS	*CDVS No Sleep State*
CDVS-S	*CDVS Sleep State*
CFG	*Control-Flow Graph*
CHP	*Common Hot Path*
CK	*Checkpoint*
CNT	*Count*
CMOS	*Complementary Metal-Oxide-Semiconductor*
CPU	*Central Processing Unit*
CPUID	*CPU IDentification*
CPUSTAT	*CPU Statistic*
EDF	*Earliest Deadline First*
FPPT	*Fixed-Priority scheduling with Preemption Threshold*
HAL	*Hardware Abstraction Layer*
ID	*IDentification*
IEEE	*The Institute of Electrical and Electronics Engineers*
IntraDVS	*Intra Dynamic Voltage Scaling*
IO	*Input and Output*
IPC	*Inter-Process Communication*
ItcaEDF	*Intra-Task Characteristics Aware EDF*
LaIntraDVS	*Look Ahead IntraDVS*
LXRT	*LinuX Real Time*
MATMULT	*Matrix Multiplication*
MMC	Mínimo Múltiplo Comum
MOSFET	*Metal Oxide Semiconductor Field Effect Transistor*

MRTC	*Mälardalen Real Time Research Centre*
VSP	*Voltage Scaling Point*
PCP	*Priority Ceiling Protocol*
PID	*Process Identifier*
PMH	*Power Management Hint*
PMP	*Power Management Point*
pWCEC	*Probabilistic Worst-Case Execution Cycle*
RAEP	*Remaining Average-Case Execution Path*
RM	*Rate-Monotonic*
RMS	*Rate-Monotonic Scheduler*
ROEP	*Remaining Optimal-Case Execution Path*
RT	*Real Time*
RTAI	*Real-Time Application Interface*
RTLPower	*Real-Time Low Power*
RWCEC	*Remaining Worst Case Execution Cycle*
RWEP	*Remaining Worst-Case Execution Path*
SEC	*Saved Execution Cycles*
TALk	*Temperature Aware Leakage*
UID	*User IDentification*
VSP	*Voltage Scaling Point*
WCEC	*Worst Case Execution Cycle*
WCEP	*Worst Case Execution Path*
WCET	*Worst Case Execution Time*
WCR	*Worst-Case Remaining*

Capítulo 1

Introdução

Os avanços da tecnologia e a redução dos custos dos equipamentos eletrônicos proporcionaram uma maior popularização dos sistemas embarcados. Isso fez com que esses dispositivos passassem a estar cada vez mais presentes no cotidiano das pessoas, principalmente pela mobilidade, facilidade ao acesso à informação, comunicação, entre outros benefícios. Dessa forma, a necessidade por maior capacidade de processamento, maior autonomia de bateria e redução do consumo de energia vêm crescendo significativamente nos últimos anos (Pillai e Shin, 2001).

Essas necessidades têm gerado diversas linhas de pesquisa, principalmente, na área de baixo consumo de energia, que tem se tornado uma métrica importante de qualidade no projeto de sistemas embarcados e sistemas de tempo real (Alipour *et al.*, 2011; Cohen *et al.*, 2012). Uma das técnicas mais utilizadas para reduzir o consumo de energia é o escalonamento dinâmico de tensões e frequências (em inglês, *Dynamic Voltage and Frequency Scaling* - DVFS) que possibilita alterar em tempo de execução, via *software*, a tensão e frequência do processador de acordo com alguma política pré-definida, sendo possível ainda gerenciar o consumo de energia (Kim *et al.*, 2008; Baums e Zaznova, 2008).

Dentro do *Kernel* do Linux, a técnica DVFS é implementada através dos *Governors*. Os *Governors* são políticas predefinidas responsáveis por realizar o gerenciamento das tensões e frequências que serão utilizadas pelo processador. A grande maioria dessas políticas funciona por meio da análise da carga de trabalho do sistema, dentro de uma determinada janela de tempo pré-definida. Esse aspecto tornam os *Governors* reativos, pois necessitam analisar informações que ocorreram no passado para poder estimar a frequência que deverá ser aplicada sobre o processador, a fim de atender as demandas computacionais exigidas pelos programas em execução no sistema (Love, 2010). Dentre todos os *Governors* presentes no *Kernel* do Linux, o *Userspace Governor* é a única política que permite as aplicações definirem as tensões e frequências que deverão ser utilizadas pelo processador. Sendo assim, não há um canal de comunicação efetivo entre

as aplicações e o *Governor*, de modo que o *Governor* possa ter o controle das tensões e frequências utilizadas por cada uma das tarefas em execução no sistema.

Dentro do contexto de sistemas de tempo real, o uso da técnica de baixo consumo de energia dentro das aplicações é conhecido como DVFS intra-tarefa, cujo principal objetivo é fornecer às aplicações um conjunto de ferramentas para controlar as tensões e frequências do processador, mesmo estando no espaço do usuário (Shin *et al.*, 2001a). Essa técnica deu origem a várias outras técnicas, cujo o cerne é analisar o código fonte da aplicação e encontrar os melhor lugares para inserir esses pontos de chaveamento de tensões e frequências, de forma que haja redução no consumo de energia do processador (Takase *et al.*, 2011; Tatematsu *et al.*, 2011; Mohan *et al.*, 2010; Yang *et al.*, 2009; Zitterell e Scholl, 2008).

A técnica mais difundida na literatura é feita através da análise do grafo de fluxo de controle (em inglês, *Control-Flow Graph* - CFG) da aplicação e, em seguida, são extraídos os pontos do código onde deverão ser inseridos os pontos de controle (em inglês, *Checkpoints* ou também chamado de *Voltage Scaling Point* - VSP) (Shin e Kim, 2001; Shin *et al.*, 2001a; Lee *et al.*, 2002; Shin *et al.*, 2001b; Buss *et al.*, 2003; Seo *et al.*, 2004). Os pontos de controle são trechos de código responsáveis por realizar as trocas de tensões e frequências do processador de acordo com a necessidade da aplicação. O principal ganho dessa metodologia está na detecção de trechos de código que deixaram de ser executados, dependendo do fluxo de execução da aplicação (exemplo: estruturas de repetição que deixaram de executar algumas interações, pois suas condições de término foram atendidas antes de alcançar o número máximo de interações). Dessa forma, os ganhos identificados são repassados para o processador, diminuindo a carga de processamento e, consequentemente, o consumo de energia (Shin *et al.*, 2001a).

Essa metodologia tem diversas aplicabilidades quando consideramos um contexto de Sistemas de Tempo Real com múltiplas tarefas, pois além de diminuir o consumo de energia é possível diminuir o tempo ocioso do processador, sem violar as premissas temporais das tarefas em execução no sistema (Shin *et al.*, 2001a). Dentro desse contexto, o menor consumo de energia será obtido quando o tempo de computação de cada tarefa for o mais próximo possível do seu *deadline*, pois dessa forma é possível obter o menor tempo ocioso do processador utilizando tensões e frequências ideais de acordo com a tarefa em execução.

A problemática desse cenário está no tratamento de preempções, pois o sistema operacional não fornecem chamadas de sistema para que as aplicações possam realizar o tratamento eficiente desse tipo de interrupção, como por exemplo: ajustar as tensões e frequências do processador já levando em consideração o tempo em que a tarefa ficou preemptada. Portanto, o objetivo central desta dissertação é diminuir o consumo de energia do processador, através da criação de um novo canal de comunicação entre as aplicações de tempo real e o controlador de tensões e frequências do processador do

sistema operacional, com o intuito de maximizar os ganhos da técnica DVFS intra-tarefa, através do melhor gerenciamento e tratamento de preempções entre as tarefas de tempo real em execução no sistema.

1.1 Descrição do Problema

O baixo consumo de energia tem se tornado uma importante métrica dentro de sistemas de tempo real, mas precisa ser implementada com cautela, pois necessita garantir que todas as premissas temporais das aplicações em execução no sistema sejam respeitadas (Cohen *et al.*, 2012). Nesse cenário, o menor consumo de energia será obtido com a utilização de tensões e frequências ideais para cada tarefa que venha a ser executada no sistema. Essas tensões e frequências ideias garantem que cada tarefa termine o seu processamento antes do seu *deadline*, deixando o processador o menor tempo possível ocioso (Awan e Petters, 2012; Cohen *et al.*, 2012; Tatematsu *et al.*, 2011; Mohan *et al.*, 2010).

Esse cenário se torna ainda mais complexo quando consideramos múltiplas tarefas preemptivas e um escalonamento baseado em prioridade fixa, pois é necessário realizar testes de escalonabilidade para saber se um dado grupo de tarefas é ou não escalonável. Neste caso, diferentes variáveis devem ser levadas em consideração como as interferências ocasionadas por tarefas de maior prioridade; o tempo de processamento das tarefas no pior caso; o cálculo das tensões e frequências que deverão ser utilizadas por cada tarefa; entre outros (Cohen *et al.*, 2012; Mohan *et al.*, 2010).

Um dos grandes problemas é manter as tarefas executando o maior tempo possível usando suas tensões e frequências ideais diante da ocorrência de preempções, pois as tarefas não conseguem realizar o tratamento desse tipo de evento de forma eficiente, principalmente quando é analisado o tempo de resposta para o restabelecimento das tensões e frequências ideais das tarefas que está retornando de preempção. Esse fator se torna mais agravante devido à inexistência de uma canal de comunicação entre as aplicações de tempo real e o controlador de tensões e frequência do processador do sistema operacional.

O problema a ser tratado nesta dissertação pode ser expresso pela seguinte pergunta: é possível reduzir o consumo de energia do processador através da criação de uma abordagem colaborativa entre as aplicações de tempo real e o sistema operacional de forma que ambos possam trabalhar em conjunto no restabelecimento eficiente das tensões e frequências do processador principalmente diante de preempções?

1.2 Contexto

O contexto desta dissertação está em criar um canal de comunicação mais efetivo entre as aplicações de tempo real e o controlador de tensões e frequências do processador do sistema operacional de forma que ambos possam trabalhar em conjunto, a fim de trazer parte do controle das tensões e frequências das aplicações para dentro do núcleo do sistema e diminuir o tempo de resposta do chaveamento de contexto do processador, principalmente no momento que as tarefas preemptadas retornam para o estado executando. Isso irá garantir que as tarefas executem por mais tempo utilizando suas tensões e frequências ideais, proporcionando a técnica DVFS intra-tarefa o suporte a preempções.

A Figura 1.1 mostra uma visão macro, por camadas, dos canais de comunicação existentes no *Kernel* do Linux, onde as características de tempo real são garantidas pela ferramenta RTAI (em inglês, *Real-Time Application Interface*), que por sua vez é responsável em gerenciar as tarefas de tempo real em execução no sistema.

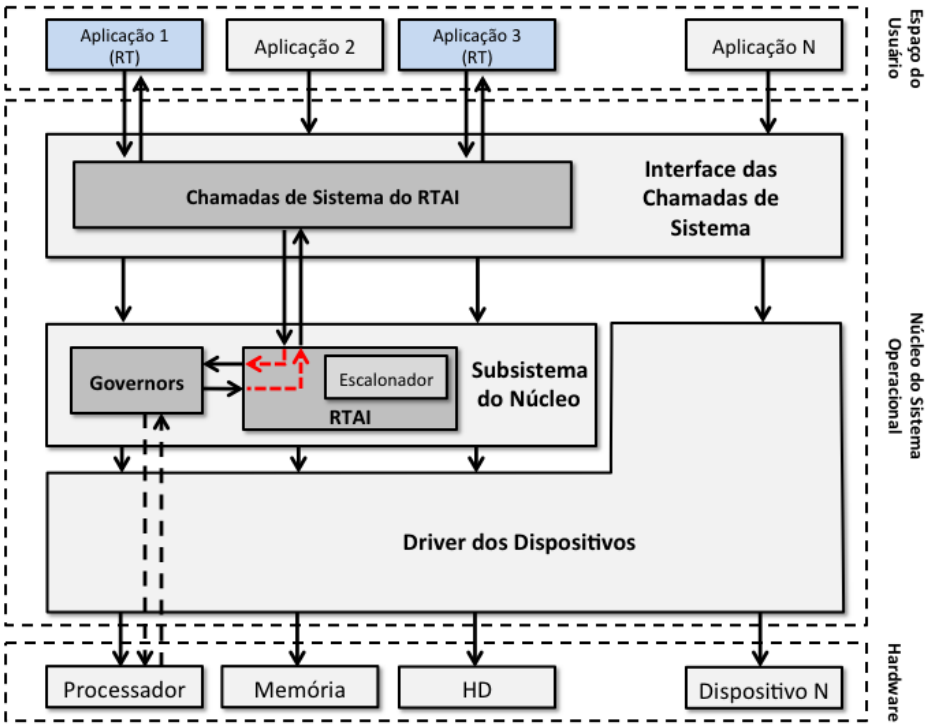

Figura 1.1: Visão geral da contextualização do trabalho, baseada na arquitetura do Linux proposta no livro de Love (2010).

Essa arquitetura deixa mais evidente os problemas relatados na Seção 1.1, onde:

1. Os *Governors* existentes no Kernel do Linux não têm acesso aos dados das aplicações que estão em execução no sistema operacional e nem fornecem uma API para que elas possam trocar informações em tempo de execução (como por exemplo: as tensões e frequências ideais de cada aplicação);

2. A arquitetura mostra também que os *Governors* não têm um canal de comunicação com o escalonador do sistema para realizar o tratamento de preempções.

1.3 Motivação e Justificativa

O projeto e desenvolvimento de sistema embarcados são complexos e precisam atender um conjunto de restrições, como por exemplo: tamanho, custo, consumo de energia, entre outros requisitos. Dentre os vários componentes presentes nesse tipo de sistema, o processador é o responsável por consumir grande parte dos recursos energéticos, principalmente devida a grande maioria deles utilizarem a tecnologia CMOS (em inglês, *Complementary Metal-Oxide-Semiconductor*) onde o consumo de energia ocorre nos pulsos de relógio de entrada na CPU e parte da energia é dissipada por seus milhares de transistores (Lee e Krishna, 1999). A Equação 1.1 define uma aproximação do consumo de energia dinâmica do processador segundo Shin *et al.* (2001a).

$$E \propto C_l \times N_{cycle} \times V_{dd}^2 \tag{1.1}$$

Onde C_l é a capacitância de carga, N_{cycle} é o número de ciclos executados e V_{dd}^2 é a tensão fornecida. Analisando a Equação 1.1, a tensão aplicada sobre o processador, por ser um termo quadrático, irá demandar maior quantidade de energia. Portanto desenvolver um controle mais refinado sobre essa variável implicará diretamente na diminuição quadrática do consumo de energia do dispositivo Shin *et al.* (2001b). Essas características tornam a técnica DVFS uma poderosa ferramenta na redução do consumo de energia do processador (Cohen *et al.*, 2012; Tatematsu *et al.*, 2011; Takase *et al.*, 2011; AbouGhazaleh *et al.*, 2003b; Shin *et al.*, 2001b).

Esses motivos levaram ao desenvolvimento desta dissertação, onde o foco principal de pesquisa convergiu para redução do consumo de energia do processador por meio da integração de características preemptivas à técnica DVFS intra-tarefa.

1.4 Objetivos

O objetivo geral desta proposta é diminuir o consumo de energia do processador através da criação de uma abordagem colaborativa entre as aplicações de tempo real e o núcleo do sistema operacional, procurando reduzir o tempo de resposta do chaveamento das tensões e frequências do processador, principalmente no instante em que as tarefas preemptadas retornam para o estado de executando.

Sendo assim, o objetivo geral decompõe-se nos seguintes objetivos específicos:

1. Prover um canal de comunicação entre as aplicações de tempo real e o controlador de tensões e frequências do processador do sistema operacional, a fim de que ambos possam trabalhar em conjunto na redução do consumo de energia do processador;

2. Reduzir o tempo de resposta de chaveamento de contexto do processador migrando o controle da técnica DVFS de dentro das aplicações para o núcleo do sistema operacional. Dessa forma, o sistema operacional passa a ter o controle das tensões e frequências que serão utilizadas no processador, com base nos dados informados pelas aplicações, e, além disso, introduzindo o tratamento de preempções à técnica DVFS intra-tarefa;

3. Demonstrar experimentalmente que a metodologia proposta é capaz de reduzir o consumo de energia do processador e maximizar os ganhos obtidos com a técnica DVFS intra-tarefa diante de preempções.

1.5 Método Proposto

O método proposto nesta dissertação visa resolver os problemas apresentados nas Seções 1.1 e 1.2 cujo foco principal é fornecer à técnica DVFS intra-tarefa o suporte a preempções e um chaveamento de contexto do processador com menos *overheads*[1], de acordo com a aplicação em execução no sistema, trazendo o gerenciamento e o controle das tensões e frequências do processador de dentro das aplicações em tempo real para dentro do núcleo do sistema operacional de forma que as tomadas de decisão sejam feitas com um menor tempo de resposta.

Em resumo, o cerne da solução proposta está em uma reestruturação do núcleo do sistema operacional de forma que o *Governor*, o escalonador e as aplicações de tempo real possam trabalhar em conjunto para prover uma solução eficiente aos problemas apresentados e reduzir o consumo de energia do processador.

A Figura 1.2 mostra uma visão geral do método proposto, que consiste na adição de três novos componentes à arquitetura do sistema operacional, são eles:

- **Chamadas de Sistema do Raw Governor:** fornecem as interfaces (API) para que as aplicações de tempo real possam trocar informações com o controlador de tensões e frequências do processador (como por exemplo: as tensões e frequências ideais de cada aplicação). Dessa forma, as aplicações passam ao sistema operacional todas as informações necessárias sobre as tarefas de tempo real que estão em execução no sistema, permitindo assim que as trocas de contexto do processador possam ser feitas diretamente no núcleo do sistema.

[1] *Overhead* é todo e qualquer tempo de computação ou armazenamento que venha a ser gasto para realizar uma determinada tarefa.

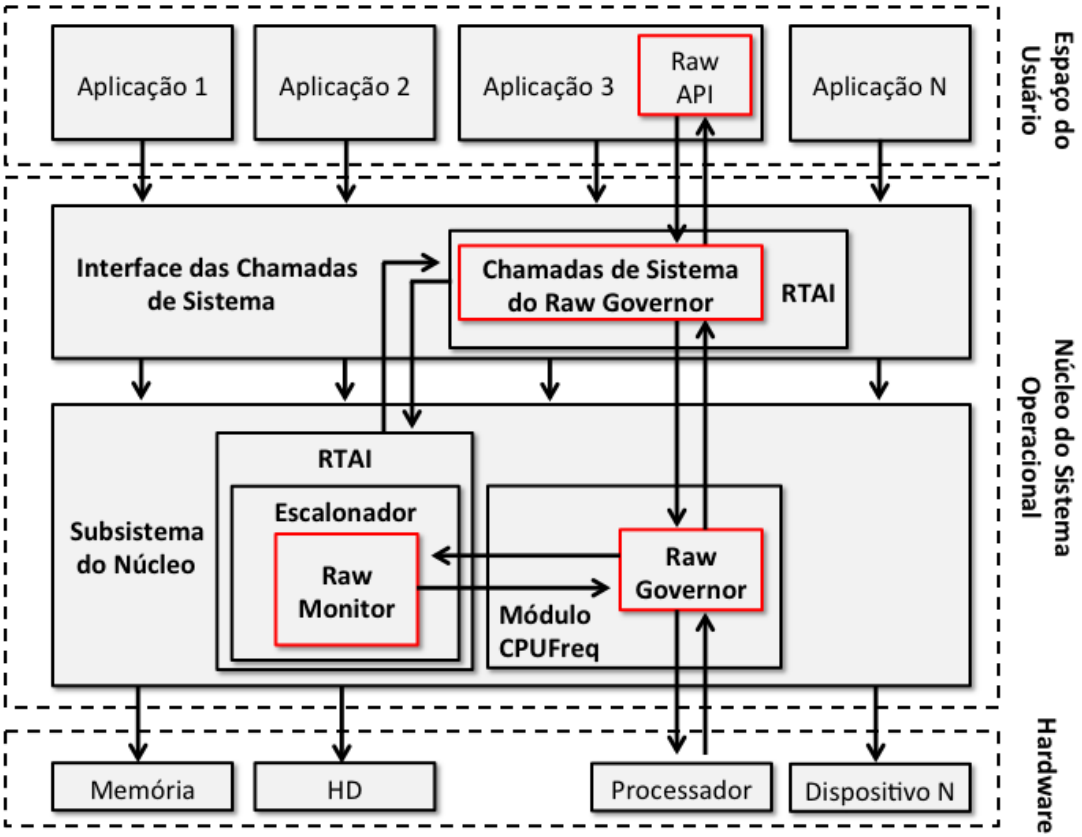

Figura 1.2: Visão geral da arquitetura proposta.

Assim, é possível diminuir o tempo de chaveamento de contexto do processador, pois o gerenciamento das tensões e frequências deixam de ser feitas no nível de usuário pelas aplicações e passam a ser feitas no núcleo do sistema operacional com menos *overheads*, ou seja, com menor tempo de resposta;

- **Raw Governor:** esta é a nova política de gerenciamento do processador com suporte a preempção. Sua função é aplicar as tensões e frequências ideais (informadas pelas aplicações) de forma personalizada de acordo com a tarefa em execução;

- **Raw Monitor:** esta é uma extensão do *Raw Governor* dentro do escalonador da RTAI. O seu papel é informar ao *Governor* as tarefa que estão retornando de preempção para que seja feito o chaveamento de contexto do processador.

1.6 Organização do Trabalho

O material desta dissertação está dividido em capítulos, para o melhor entendimento do trabalho, e a seguir será feito uma breve descrição do que será abordado em cada um deles. No Capítulo 2 serão discutidos os principais conceitos e definições que serão

utilizados no desenvolvimento desta dissertação. No Capítulo 3 serão analisados os principias trabalhos correlatos. No Capítulo 4 será apresentado o método proposto. No Capítulo 5 serão apresentados os resultados experimentais. Por fim, no Capítulo 6 são expostas as conclusões finais, bem como as principais contribuições, limitações e trabalhos futuros.

Capítulo 2

Conceitos e Definições

Esta seção apresenta os principais conceitos e definições abordados nesta dissertação, são eles: Sistemas Embarcados, Sistemas de Tempo Real, Escalonamento Dinâmico de Tensão e Frequência (DVFS), Grafo de Fluxo de Controle, *Kernel* do Linux e RTAI. Estes conceitos são fundamentais para que haja um melhor entendimento deste trabalho.

2.1 Sistemas Embarcados

Os sistemas embarcados, na sua grande maioria, possuem *hardwares* mais limitados e, consequentemente, um poder de processamento mais restrito por serem sistemas voltados para aplicações específicas. Durante a fase de projeto desse tipo de sistema é levado em consideração o seu custo, tamanho, desempenho e consumo de energia (Lee *et al.*, 2008).

Esses sistemas estão presentes em quase todo tipo de equipamento eletrônico, como por exemplo: geladeiras, micro-ondas, máquinas de lavar, televisores, celulares, etc. Estes sistemas também são aplicados em ambientes que possuem restrições temporais, como por exemplo: no controle de aviões, no freio de carros, no controle de elevadores, entre outros (Lee *et al.*, 2008).

2.2 Sistemas de Tempo Real

Os sistemas de tempo real possuem restrições temporais a serem cumpridas por suas tarefas. Esses sistemas não dependem apenas dos resultados computados, mas também do instante de tempo em que esses resultados foram produzidos (Cheng, 2002), pois necessitam respeitar as restrições temporais impostas para cada tarefa em execução. Os sistemas de tempo real podem ser classificados de duas formas:

- **Sistemas de tempo real crítico (*Hard real-time*):** requer que todas as tarefas ou suas instâncias concluam suas execuções dentro do prazo especificado, caso contrário os prejuízos serão catastróficos, como por exemplo: um avião poderá cair, o freio de um carro poderá não funcionar, entre outros graves problemas.
- **Sistemas de tempo real não-críticos (*Soft real-time*):** permite que algumas de suas tarefas ou suas instâncias percam seus prazos, sem grandes prejuízos para o funcionamento do sistema, como por exemplo: a perda de algum pacote de *streaming* na transmissão de áudio e vídeo.

2.2.1 Modelo de Tarefas e suas Propriedades Temporais

No contexto de sistemas de tempo real, as tarefas podem ser classificadas de três formas, são elas:

- **Periódicas:** são tarefas que possuem várias instâncias ou iterações, onde seus períodos são fixos entre duas execuções consecutivas da mesma tarefa;
- **Esporádicas:** são tarefas que possuem zero ou mais instâncias e existe um intervalo de tempo mínimo entre duas execuções consecutivas da mesma tarefa;
- **Aperiódicas:** são tarefas que não possuem um intervalo mínimo ou máximo entre duas execuções da mesma tarefa, ou seja não possui um prazo específico para ativação da tarefa, como por exemplo: o aparecimento de um objeto em uma tela de radar.

Dentro do contexto temporal, as tarefas possuem as seguintes propriedades (ver Figura 2.1):

- ***Deadline* ou Meta Temporal:** é o intervalo de tempo máximo que a tarefa terá para concluir o seu processamento, além dos tempos de bloqueio e precedências;
- **Tempo de Execução ou Tempo de Computação:** é o intervalo de tempo que a tarefa levou para concluir seu processamento;
- **Tempo Ocioso ou Tempo de Folga:** é o intervalo de tempo não utilizado pela tarefa, ou seja, é a diferença entre a meta temporal (*deadline*) e o seu tempo de execução. Caso este valor seja negativo, implica dizer que a meta temporal da tarefa foi violada.

Dessa forma, as tarefas ainda podem ser representadas como sendo a tripla (c_i, p_i, d_i), onde c_i é o tempo de computação da tarefa, p_i é o período e d_i é a sua meta temporal.

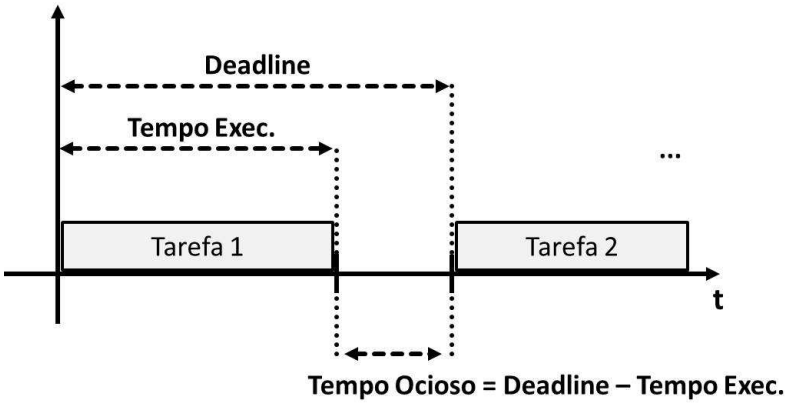

Figura 2.1: Visão geral das premissas temporais associadas a uma tarefa de tempo real.

Com base nessas características é possível extrair o fator de utilização do sistema, que é descrito na Equação 2.1 onde n é o número de tarefas.

$$U = \sum_{i=1}^{n} \frac{c_i}{p_i} \tag{2.1}$$

2.2.2 Relação de Precedência e de Exclusão Mútua

A grande maioria das aplicações de tempo real possuem implicações semânticas entre as tarefas. Tais implicações podem restringir a ordem de execução, como é o caso das relações de precedência. Uma tarefa τ_i é dita preceder a tarefa τ_j ($\tau_i \rightarrow \tau_j$), se τ_j somente puder iniciar sua execução após o término da tarefa τ_i.

Outra relação importante a ser considerada no escalonamento de tempo real, é a relação de exclusão mútua. Tal relação ocorre quando existe algum recurso compartilhado em exclusão mútua. Dessa forma, temos que uma tarefa τ_i exclui τ_j quando a execução de uma seção crítica de τ_j, que manipula o recurso compartilhado, não pode executar porque τ_i ainda está utilizando o recurso.

Portanto, as relações de precedência e de exclusão mútua entre as tarefas integrantes ao problema são especificadas a partir da definição do modelo de tarefas.

2.2.3 Algoritmo de Escalonamento

Antes de abordar sobre o algoritmo de escalonamento de tempo real é necessário ter o conhecimento claro dos seguintes conceitos: escalonamento, escala e escalonador. A política de escalonamento é a forma ou a maneira de ordenar as tarefas que encontram-se disponíveis na fila de prontos. A escala é uma lista ordenada que indica a

ordem de ocupação do processador pelas tarefas "prontas". O escalonador é o componente do sistema operacional responsável por realizar a gestão do processador, além de implementar uma política de escalonamento de tarefas para realizar a ocupação do mesmo.

As políticas de escalonamento definem as regras que irão produzir as escalas, a fim de atender às restrições temporais impostas pelo modelo de tarefas. Essas políticas de escalonamento para sistemas de tempo real podem ser classificadas como:

- **Análise Dinâmica (*Online*):** podem sofrer mudanças em tempo de execução, ou seja, os parâmetros podem ter seus valores alterados com a evolução do sistema.
- **Análise Estática (*Offline*):** são determinadas em fase de projeto, ou seja, os parâmetros que definem o cálculo de escala são fixas e definidas em tempo de projeto.

Nesse contexto, as tarefas de tempo real ainda podem ser classificadas como:

- **Tarefas Preemptivas:** são tarefas que podem ser interrompidas por tarefas de maior prioridade.
- **Tarefas Não-Preemptivas:** são tarefas que não podem ser interrompidas por nenhuma outra tarefa.

Nesta dissertação, a política de escalonamento utilizada será a *Rate-Monotonic*, também conhecida como RM. Segundo Cheng (2002), o *Rate-Monotonic* é a política de escalonamento de tempo real mais popular, cujo o objetivo é escalonar tarefas periódicas de prioridade fixa. A RMS executa com maior prioridade a instância da tarefa pronta de período mais curto. Em caso de empate entre duas ou mais tarefas, a RMS selecionará aleatoriamente uma para a próxima execução.

Exemplo. Considerando três tarefas periódicas com os seguintes tempos de chegada (S), tempos de computação (c), períodos (p) e *deadlines* (d):

$$\begin{aligned} \tau_1 &: S_1 = 0; c_1 = 2; p_1 = d_1 = 5; \\ \tau_2 &: S_2 = 1; c_2 = 1; p_2 = d_2 = 4; \\ \tau_3 &: S_3 = 2; c_3 = 2; p_3 = d_3 = 20; \end{aligned} \tag{2.2}$$

Para esse grupo de tarefas, temos que no instante de tempo 0, τ_1 é a única tarefa pronta que está programada para ser executada. No instante de tempo 1, τ_2 chega. Desde que $p_2 < p_1$, τ_2 tem maior prioridade, então τ_1 é preemptada e τ_2 inicia sua execução. No instante de tempo 2, τ_2 termina sua execução e chega τ_3. Como $p_3 > p_1$, τ_1 agora tem a

maior prioridade, por isso, continua a execução. No instante de tempo 3, τ_1 termina a execução. Neste momento, τ_3 é a única tarefa pronta, então ela inicia sua execução. No instante de tempo 4, τ_3 ainda é a única tarefa, então ela continua a executar até terminar sua execução no instante de tempo 5. Nesse momento, as segundas instâncias de τ_1 e τ_2 estão prontas para executar novamente. Como $p_2 < p_1$, τ_2 tem a maior prioridade, então τ_2 inicia a sua execução. No instante de tempo 6, a segunda instância de τ_2 termina a sua execução. Neste momento, a segunda instância τ_1 é a única tarefa pronta para começar sua execução, terminando no instante de tempo 8. Esse processo segue até o instante de tempo igual ao MMC (em português, Mínimo Múltiplo Comum) dentre os períodos de todas as tarefas, que neste caso é o $MMC(4, 5, 20) = 20$. Em outras palavras, a partir do instante de tempo 20 o escalonamento irá se repetir, não sendo mais necessário continuar realizando a geração da escala de execução das tarefas. Esse instante de tempo também é conhecido na literatura como hiper período[1] (em inglês, *Hyperperiod*). A Figura 2.2 ilustra o diagrama de temporização das tarefas J_i, segundo a política de escalonamento RM, bem como o instante de tempo que ocorre a repetição do escalonamento (no instante de tempo igual a 20) (Cheng, 2002).

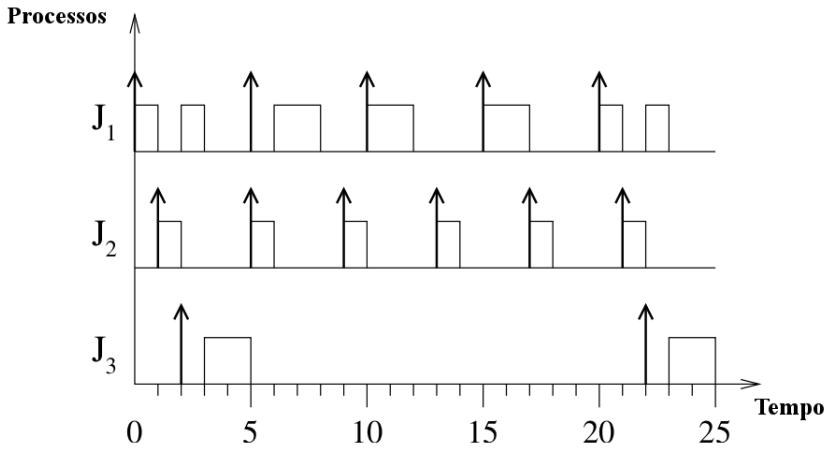

Figura 2.2: Exemplo de uma escala de tarefas J_i utilizando a política de escalonamento *Rate-Monotonic* (Cheng, 2002).

Esse processo de geração do diagrama de temporização das tarefas pode ser muito trabalhoso, dependendo da quantidade de tarefas e suas propriedades temporais. Na próxima seção serão abordados alguns testes de escalonabilidade para saber se um dado grupo de tarefas de tempo real é ou não escalonável.

2.2.4 Teste de Escalonabilidade

O teste de escalonabilidade é uma das etapas mais importantes do processo de escalonamento, pois ele irá determinar se um determinado grupo de tarefas é ou não

[1] O hiper período é o instante de tempo no qual a escala de execução das tarefas, dentro do escalonador, se repete.

escalonável, antes de se definir uma escala factível (ver Figura 2.2). Esses testes têm como objetivo analisar situações de picos do sistema no pior caso de execução das tarefas e essas análises vão depender diretamente do modelo de tarefas considerado e da política de escalonamento definida para o problema em questão.

Basicamente, existem três tipos de classificações para testes de escalonabilidade, são eles:

- **Suficiente:** se o teste for positivo, garante que o conjunto de tarefas é sempre escalonável, caso negativo ainda podem existir um ou mais conjuntos de tarefas escalonáveis;
- **Necessário:** se o teste for negativo, garante que o conjunto de tarefas é sempre não escalonável;
- **Exato:** os testes de escalonabilidade exatos são mais complexos e determinam se um dado conjunto de tarefas é ou não escalonável.

Segundo Cheng (2002), existem quatro tipos de teste de escalonabilidade para política RM, são eles:

Teste de Escalonabilidade 1: Dado um conjunto de N tarefas independentes, preemptivas e periódicas sobre um único processador de tal forma que seus *deadlines* sejam iguais ou maiores do que seus respectivos períodos, desde que eles sejam números inteiros múltiplos entre si, e considerando U a utilização total do modelo de tarefas definido. Uma condição necessária e suficiente para um escalonamento viável desse conjunto de tarefas é definido por:

$$U = \sum_{i=1}^{n} \frac{c_i}{p_i} \leq 1 \tag{2.3}$$

Exemplo. Considerando o modelo de tarefas definido na Equação 2.4.

$$\begin{aligned} \tau_1 &: S_1 = 0; c_1 = 2; p_1 = d_1 = 5; \\ \tau_2 &: S_2 = 1; c_2 = 1; p_2 = d_2 = 4; \\ \tau_3 &: S_3 = 2; c_3 = 2; p_3 = d_3 = 20; \end{aligned} \tag{2.4}$$

Temos que os períodos das tarefas são múltiplos exatos entre si ($p_2 < p_1 < p_3$, $p_1 = 2p_2$, $p_3 = 4p_2 = 2p_1$) e que este conjunto de tarefas pertence a classe especial de tarefas definidos pelo teste de escalonabilidade 1. Sendo assim, $U = \frac{1}{4} + \frac{1}{2} + \frac{2}{8} = 1 \leq 1$, logo esse modelo de tarefas é escalonável pela política RM.

Teste de Escalonabilidade 2: Dado um conjunto de N tarefas independentes, preemptivas e periódicas sobre um único processador, sendo U a utilização total do

modelo de tarefas definido. Uma condição suficiente para o escalonamento viável deste conjunto de tarefas é $U \leq n(2^{\frac{1}{n}} - 1)$

No entanto, o uso desse simples teste de escalonabilidade pode subutilizar o sistema, permitindo que um dado conjunto de tarefas exceda o limite de utilização e ainda possa ser escalonável pela política RM. Portanto, passamos a derivar uma condição suficiente e necessária para o escalonador usando o algoritmo RM. Suponha que temos três tarefas, todos iniciando no instante de tempo 0. A tarefa τ_1 tem o menor período, seguido por τ_2 e τ_3. É intuitivo ver que τ_1 terá um escalonamento viável, se o tempo de computação for inferior ou igual ao seu período, portanto, a condição necessária e suficiente deve ser:

$$c_i \leq p_i \tag{2.5}$$

Para τ_2 terá um escalonamento viável, é preciso encontrar um tempo disponível suficiente no intervalo de $[0, p_2]$ que não seja usado por τ_1. Supondo que τ_2 conclua a sua execução no tempo t. Em seguida, o número total de iterações de τ_1 no intervalo de $[0, t]$ é:

$$\left\lceil \frac{t}{p_1} \right\rceil \tag{2.6}$$

Para garantir que τ_2 possa concluir sua execução no tempo t, cada iteração de τ_1 no intervalo de $[0, t]$ deve ser preenchida e deve haver tempo disponível suficiente para τ_2 termine a sua execução. Este tempo disponível é c_2. Portanto, temos:

$$t = \left\lceil \frac{t}{p_1} \right\rceil c_1 + c_2 \tag{2.7}$$

Da mesma forma, deve ser feito para τ_3, onde deve haver tempo suficiente restante para executar τ_3 antes do escalonamento de τ_1 e τ_2:

$$t = \left\lceil \frac{t}{p_1} \right\rceil c_1 + \left\lceil \frac{t}{p_2} \right\rceil c_2 + c_3 \tag{2.8}$$

A próxima questão é como determinar o tempo t de forma que exista um escalonamento viável para que um conjunto de tarefas possa ser construído. Note que há um número infinito de pontos em cada intervalo, se for assumido um tempo não discreto. No entanto, o valor do limite máximo é definido pela Equação 2.6.

Esse valor somente muda em múltiplos de p_1, com o aumentar de c_1. Então é necessário apenas mostrar que existe um valor k, tal que:

$$kp_1 \geq kc_1 + c_2 \quad e \quad kp_1 \leq p_2 \tag{2.9}$$

Portanto, é preciso verificar que:

$$t \geq \left\lceil \frac{t}{p_1} \right\rceil c_1 + c_2 \tag{2.10}$$

para algum t que seja múltiplo de $p1$, tal que $t \leq p_2$. Se este valor for encontrado, então nós temos a condição necessária e suficiente para um escalonamento viável de τ_2 usando o algoritmo RM. Esta verificação é finita, uma vez que existe um número finito múltiplo de p_1 que são menores ou iguais a p_2. Seguindo o mesmo processo para τ_3, é necessário verificar se a seguinte condição é válida:

$$t \geq \left\lceil \frac{t}{p_1} \right\rceil c_1 + \left\lceil \frac{t}{p_2} \right\rceil c_2 + c_3 \tag{2.11}$$

Com base nessas condições, é possível definir a condição necessária e suficiente para o escalonamento viável de uma tarefa periódica (ver teste de escalonabilidade 3).

Teste de Escalonabilidade 3: Com base nas condições anteriores, podemos definir uma equação de teste de escalonabilidade iterativa (ver Equação 2.12).

$$w_i(t) = \sum_{k=1}^{i} c_k \left\lceil \frac{t}{p_k} \right\rceil, \quad 0 < t \leq p_i \tag{2.12}$$

Onde,

$$w_i(t) \leq t \tag{2.13}$$

e qualquer instante de tempo t é escolhido da seguinte forma:

$$\begin{gathered} t = kp_j; \\ j = 1, ..., i; \\ k = 1, ..., \left\lfloor \frac{p_i}{p_j} \right\rfloor; \end{gathered} \tag{2.14}$$

Portanto, se e somente se a tarefa τ_i é escalonável pela política RM, se $d_i \neq p_i$ então podemos substituir p_i pelo $min(d_i, p_i)$ na Equação 2.14.

O exemplo a seguir aplica esta condição suficiente e necessária para verificar a escalonabilidade de quatro tarefas usando o algoritmo RM.

Exemplo. Considerando que todas as tarefas periódicas chegam no instante de tempo 0 e que o período de cada tarefa é igual ao seu *deadline*.

$$\begin{gathered}\tau_1 : c_1 = 10; p_1 = d_1 = 50; \\ \tau_2 : c_2 = 15; p_2 = d_2 = 80; \\ \tau_3 : c_3 = 40; p_3 = d_3 = 110; \\ \tau_4 : c_4 = 50; p_4 = d_4 = 190;\end{gathered} \tag{2.15}$$

Usando o teste de escalonabilidade 2, passamos a verificar se cada tarefa é escalonável usando o algoritmo RM, começando com as tarefas que têm o menor período.

Para τ_1, $i = 1, j = 1, ..., i = 1$, então:

$$k = 1, ..., \left\lfloor \frac{p_i}{p_j} \right\rfloor = 1, ..., \lfloor \frac{50}{50} \rfloor = 1; \tag{2.16}$$

Assim, $t = kp_1 = 1(50) = 50$. A tarefa τ_1 é RM escalonável se e somente se:

$$c_1 \leq 50; \tag{2.17}$$

Portanto, $c_1 = 10 \leq 50$, τ_1 é RM escalonável.

Para τ_2, $i = 2, j = 1, ..., i = 1, 2$, então:

$$k = 1, ..., \left\lfloor \frac{p_i}{p_j} \right\rfloor = 1, ..., \lfloor \frac{80}{50} \rfloor = 1; \tag{2.18}$$

Assim, $t = 1p_1 = 1(50) = 50$, ou $t = 1p_2 = 1(80) = 80$. A tarefa τ_2 é RM escalonável se e somente se:

$$\begin{gathered}c_1 + c_2 \leq 50; ou \\ 2c_1 + c_2 \leq 80;\end{gathered} \tag{2.19}$$

Portanto, $c_1 = 10$ e $c_2 = 15, 10+15 \leq 50$ (ou $2(10)+15 \leq 80$), então τ_2 é RM escalonável junto com τ_1.

Para τ_3, $i = 3, j = 1, ..., i = 1, 2, 3$, então:

$$k = 1, ..., \left\lfloor \frac{p_i}{p_j} \right\rfloor = 1, ..., \lfloor \frac{110}{50} \rfloor = 1, 2; \tag{2.20}$$

Assim, $t = 1p_1 = 1(50) = 50$, ou $t = 1p_2 = 1(80) = 80$, ou $t = 1p_3 = 1(110) = 110$, ou $t = 2p_1 = 2(50) = 100$. A tarefa τ_3 é RM escalonável se e somente se:

$$\begin{gathered} c_1 + c_2 + c_3 \leq 50; ou \\ 2c_1 + c_2 + c_3 \leq 80; ou \\ 2c_1 + 2c_2 + c_3 \leq 100; ou \\ 3c_1 + 2c_2 + c_3 \leq 110; \end{gathered} \tag{2.21}$$

Portanto, $c_1 = 10$, $c_2 = 15$ e $c_3 = 40$, teremos $2(10)+15+40 \leq 80$(ou $2(10)+2(15)+40 \leq 100$ ou $3(10) + 2(15) + 40 \leq 110$). Logo τ_3 é RM escalonável junto com $\tau_1 e \tau_2$.

Para τ_4, $i = 4, j = 1, ..., i = 1, 2, 3, 4$, então:

$$k = 1, ..., \left\lfloor \frac{p_i}{p_j} \right\rfloor = 1, ..., \lfloor \frac{190}{50} \rfloor = 1, 2, 3; \tag{2.22}$$

Assim, $t = 1p_1 = 1(50) = 50$, ou $t = 1p_2 = 1(80) = 80$, ou $t = 1p_3 = 1(110) = 110$, ou $t = 1p_4 = 1(190) = 190$, ou $t = 2p_1 = 2(50) = 100$, ou $t = 2p_2 = 2(80) = 160$, ou $t = 3p_1 = 3(50) = 150$. A tarefa τ_4 é RM escalonável se e somente se:

$$\begin{gathered} c_1 + c_2 + c_3 + c_4 \leq 50; ou \\ 2c_1 + c_2 + c_3 + c_4 \leq 80; ou \\ 2c_1 + 2c_2 + c_3 + c_4 \leq 100; ou \\ 3c_1 + 2c_2 + c_3 + c_4 \leq 110; ou \\ 3c_1 + 2c_2 + 2c_3 + c_4 \leq 150; ou \\ 4c_1 + 2c_2 + 2c_3 + c_4 \leq 160; ou \\ 4c_1 + 3c_2 + 2c_3 + c_4 \leq 190; \end{gathered} \tag{2.23}$$

Substituindo os valores de c_1, c_2, c_3 e c_4 na Equação 2.23, temos que nenhuma das equações podem ser satisfeita. Logo τ_4 não é RM escalonável. Isto é um fato, pois:

$$U = \frac{10}{50} + \frac{15}{80} + \frac{40}{110} + \frac{50}{190} = 1,014 > 1 \tag{2.24}$$

Portanto, nenhum escalonador poderá escalonar estas tarefas. Agora, se a tarefa τ_4 for ignorada, a utilização passará a ser $U = 0,75$, que já satisfaz o teste de Escalonabilidade 2. Sendo assim, a Figura 2.3 mostra a escala de execução das três primeiras tarefas.

Teste de Escalonabilidade 4: Segundo Bertolotti e Manduchi (2012), essa é uma outra forma de representar a equação iterativa do teste de escalonabilidade da política RM. Assumindo que:

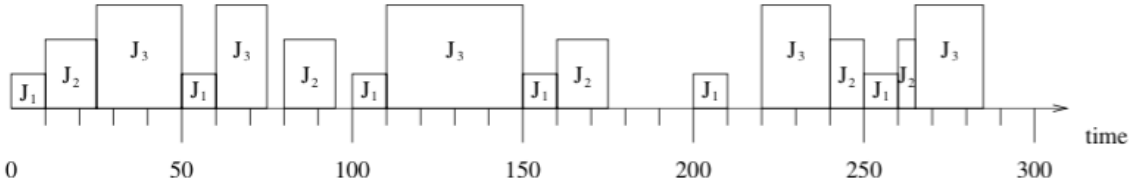

Figura 2.3: Representação da escala de execução das tarefas τ_1, τ_2 e τ_3.

$$\begin{aligned} w_i^0 &= c_i \quad e \\ w_i^{x+1} &= c_i + \textstyle\sum_{k \in hp(i)} c_k \left\lceil \frac{w_i^x}{p_k} \right\rceil \end{aligned} \tag{2.25}$$

onde $hp(i)$ é o conjunto de tarefas com prioridades superiores a prioridade da tarefa τ_i. Esse sistema só possui solução se o resultado do tempo de resposta da iteração $x+1$ convergir e for igual a iteração x, ou seja $(w_i^{x+1} = w_i^x)$, e se o tempo de resposta $w_i^{x+1} \leq d_i$. Caso contrário, as tarefas não são RM escalonáveis.

Exemplo. Considerando que todas as tarefas periódicas chegam no instante de tempo 0 e que o período de cada tarefa é igual ao seu *deadline*.

$$\begin{aligned} \tau_1 &: c_1 = 3; p_1 = d_1 = 7; \\ \tau_2 &: c_2 = 3; p_2 = d_2 = 12; \\ \tau_3 &: c_3 = 5; p_3 = d_3 = 20; \end{aligned} \tag{2.26}$$

Usando o teste de escalonabilidade da Equação 2.25 e começando com as tarefas que têm o menor período.

Para τ_1, temos:

$$w_1^0 = c_1 \ \rightarrow \ w_1^0 = 3 \tag{2.27}$$

Como $w_1^0 \leq d_1 \ \rightarrow \ 3 \leq 7$, logo τ_1 é RM escalonável.

Para τ_2, temos:

$$\begin{aligned} w_2^0 &= c_2 = 3; \\ w_2^1 &= c_2 + c_1 \left\lceil \frac{w_2^0}{p_1} \right\rceil = 3 + 3 \left\lceil \frac{3}{7} \right\rceil = 6; \\ w_2^2 &= c_2 + c_1 \left\lceil \frac{w_2^1}{p_1} \right\rceil = 3 + 3 \left\lceil \frac{6}{7} \right\rceil = 6; \end{aligned} \tag{2.28}$$

Como $w_2^2 = w_2^1$ e $w_2^2 \leq d_2 \ \rightarrow \ 6 \leq 12$, logo τ_2 é RM escalonável.

Para τ_3, temos:

$$
\begin{aligned}
w_3^0 &= c_3 = 5; \\
w_3^1 &= c_3 + c_1 \left\lceil \frac{w_3^0}{p_1} \right\rceil + c_2 \left\lceil \frac{w_3^0}{p_2} \right\rceil = 5 + 3 \left\lceil \frac{5}{7} \right\rceil + 3 \left\lceil \frac{5}{12} \right\rceil = 11; \\
w_3^2 &= c_3 + c_1 \left\lceil \frac{w_3^1}{p_1} \right\rceil + c_2 \left\lceil \frac{w_3^1}{p_2} \right\rceil = 5 + 3 \left\lceil \frac{11}{7} \right\rceil + 3 \left\lceil \frac{11}{12} \right\rceil = 14; \\
w_3^3 &= c_3 + c_1 \left\lceil \frac{w_3^2}{p_1} \right\rceil + c_2 \left\lceil \frac{w_3^2}{p_2} \right\rceil = 5 + 3 \left\lceil \frac{14}{7} \right\rceil + 3 \left\lceil \frac{14}{12} \right\rceil = 17; \\
w_3^4 &= c_3 + c_1 \left\lceil \frac{w_3^3}{p_1} \right\rceil + c_2 \left\lceil \frac{w_3^3}{p_2} \right\rceil = 5 + 3 \left\lceil \frac{17}{7} \right\rceil + 3 \left\lceil \frac{17}{12} \right\rceil = 20; \\
w_3^5 &= c_3 + c_1 \left\lceil \frac{w_3^4}{p_1} \right\rceil + c_2 \left\lceil \frac{w_3^4}{p_2} \right\rceil = 5 + 3 \left\lceil \frac{20}{7} \right\rceil + 3 \left\lceil \frac{20}{12} \right\rceil = 20;
\end{aligned}
\tag{2.29}
$$

Como $w_3^5 = w_3^4$ e $w_3^5 \leq d_3 \ \rightarrow \ 20 \leq 20$, logo τ_3 é RM escalonável. Portanto, o grupo de tarefas é RM escalonável.

2.3 Escalonamento Dinâmico de Tensão e Frequência (DVFS)

Atualmente, a técnica DVFS só é possível devido a grande maioria dos processadores utilizarem as tecnologias CMOS (em inglês, *Complementary Metal Oxide Semiconductor*) e MOSFET (em inglês, *Metal Oxide Semiconductor Field Effect Transistor*), pois elas permitem controlar via *software* as tensões e frequências que são utilizadas pelo processador, permitindo que sejam feitas trocas constantes de frequência à medida que o sistema necessite de mais ou menos processamento (Kang e Leblebici, 2002).

Há vários processadores disponíveis no mercado que visam o baixo consumo de energia, onde é possível associar para um determinado nível de tensão um determinando nível de frequência. A Tabela 2.1 mostram as tensões e frequências do processador *AMD Athlon II X2 250*, por exemplo para se alcançar uma frequência de $3.000\,MHz$ é necessário aplicar uma tensão de $1,425\,V$ no processador que irá consumir uma potência de $60,2\,W$, essa interpretação se aplica as demais linhas da tabela (AMD, 2009).

O surgimento dessa técnica permitiu que os sistemas operacionais passassem a estabelecer políticas pré-definidas para realizar o gerenciamento dinâmico das tensões e

Tabela 2.1: Lista de tensões e frequências do processador *AMD Athlon II X2 250*.

Frequências (MHz)	**Tensões (V)**	**Potência (W)**
3000	1,425	60,2
2300	1,325	51,2
1800	1,225	37,2
800	1,125	23,1

frequências utilizadas pelo processador (Kim *et al.*, 2008; Baums e Zaznova, 2008). Dentro do contexto de sistemas de tempo real, as duas principais técnicas DVFS voltadas para o baixo consumo de energia do processador são (Quinõnes *et al.*, 2011; Lee *et al.*, 2008):

- **DVFS intra-tarefa:** as tensões e frequências são definidas pelas tarefas de tempo real, levando em consideração apenas os dados locais da aplicação. Uma das formas de se obter essas tensões e frequências é através das análises estáticas do grafo de fluxo de controle (em inglês, *Control-Flow Graph* - CFG), ver mais detalhes na Seção 2.4.
- **DVFS inter-tarefa:** as tensões e frequências são definidas tarefa por tarefa a cada instante de atuação do escalonador do sistema, levando em consideração as informações de todas as tarefas em execução no sistema. A principal diferença com a técnica intra-tarefa é que os tempos de folga (ver definição na Seção 2.2.1) obtidos com a técnica inter-tarefa podem ser usados na tarefa atual ou nas tarefas seguintes, enquanto que os tempos de folga obtidos com a técnica intra-tarefa só poderão ser utilizados pelas tarefas que as gerou.

2.4 Grafo de Fluxo de Controle

Os grafos de fluxo de controle são abstrações estáticas do fluxo de execução de uma determinada aplicação, extraídos a partir do código fonte. O CFG é um grafo direcionado e a resposta de saída da execução da aplicação vai depender do caminho percorrido nesse grafo (Lee *et al.*, 2008). Os componentes básicos de uma CFG são:

- **Blocos Básicos:** são trechos de código no qual a aplicação não possui nenhum desvio condicional.
- **Blocos de Desvio:** são trechos de código que correspondem a desvios condicionais e desvios incondicionais.
- **Arestas:** representam as relações de dependência entre cada bloco ou nó.

A Figura 2.4 mostra um exemplo de como é feita a extração do grafo de fluxo de controle de uma aplicação. Vale destacar que os b_i, do item b da figura, contém os trechos de código da aplicação agrupados de acordo com os desvios condicionais, ou seja, b_2 possui os trechos de código da primeira condição, b_6 da terceira condição e assim sucessivamente. Além disso, cada b_i possui um peso, que indica a quantidade de ciclos de execução no pior caso. Essa informação será utilizada no item c para calcular o pior caso de execução da aplicação no geral, ou seja, assumindo o pior caminho de execução como $b_1, b_{wh}, b_3, b_4, b_5, b_{wh}, b_3, b_4, b_5, b_{wh}, b_3, b_4, b_5, b_{wh}, b_{if}, b_6$ e b_7 resultará em um total de 160 ciclos, somando os custos de cada nó. A partir dessa estimativa, o grafo do item c também tem o objetivo de mostrar ao lado de cada nó a quantidade de ciclos que faltam para a aplicação ser concluída, exemplo: estando em b_1 e levando em consideração o pior caminho de execução a quantidade de ciclos restantes é 160; passando o fluxo de execução para b_{wh}, que possui um custo de 10 ciclos, a quantidade restante passa a ser 150 ciclos; e assim sucessivamente até que todo o código seja executado. Dessa forma, é possível ter um controle da quantidade de ciclos restantes à medida que a aplicação vai sendo executada.

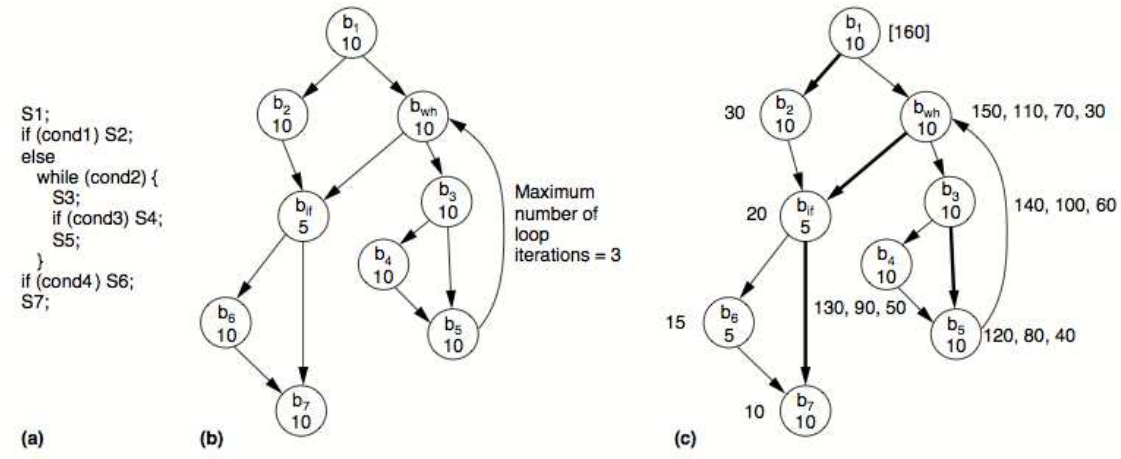

Figura 2.4: Processo de extração do grafo de fluxo de controle de uma aplicação, onde: (a) Mostra o seu código fonte; (b) O CFG extraído a partir do código fonte e seus componentes básicos; e (c) Mostra o processo de análise da quantidade de ciclos de execução da tarefa no pior caso (Shin e Kim, 2001).

2.5 *Kernel* do Linux

O *Kernel* é o núcleo de qualquer sistema operacional, em outras palavras, é o componente central que visa estabelecer uma ligação entre as aplicações e o processamento dos dados no nível do hardware. As principais responsabilidades do *Kernel* são realizar o gerenciamento do escalonamento dos processos, gerenciar as memórias e os dispositivo de entrada e saída, além de prover um encapsulamento de acesso para todos os componentes de *hardware* (Cheng, 2002). Dentre os principais

componentes e recursos utilizados nesta dissertação (ver Figura 2.5), será feito um breve resumo sobre os seguintes tópicos: Escalonador, Chamadas de Sistema e Módulo CPUFreq.

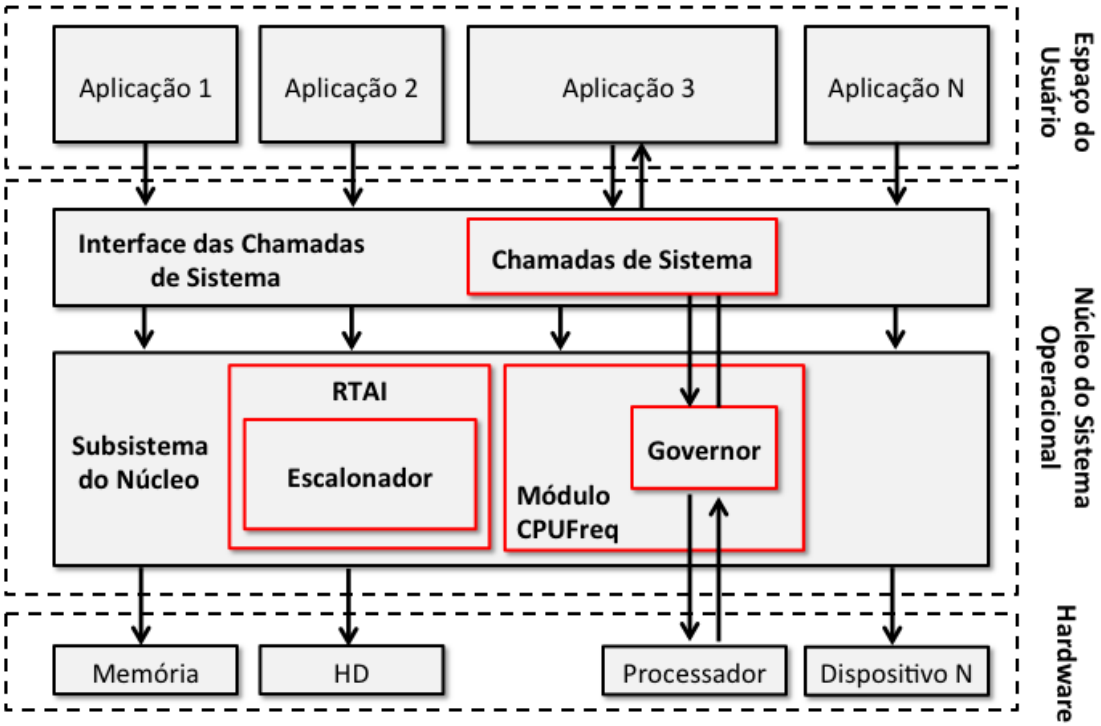

Figura 2.5: Visão geral dos principais componentes e recursos do *Kernel* do Linux utilizados nesta dissertação.

2.5.1 Escalonador

Dentre todos os componentes presentes no *Kernel*, um dos mais importantes é o escalonador. Ele é a base de funcionamento de qualquer sistema operacional e seu papel é decidir qual processo será executado, quando e por quanto tempo ele ficará executando no processador (Love, 2010). Todo escalonador possui um algoritmo de escalonamento (ver Subseção 2.2.3), que podem ser: (1) não preemptivo, quando os processos executam até o fim sem ser interrompidos; e (2) preemptivo, quando os processos executam em fatias de tempo (chamadas de *quantum*) definidas pelo sistema operacional. Esse componente é útil para o método proposto, pois nele será introduzido um monitor de tarefas que terá o objetivo de auxiliar o *Governor* nas trocas de tensões e frequências do processador quando ocorrerem preempções entre as tarefas em tempo real.

2.5.2 Chamadas de Sistema

As chamadas de sistema (em inglês, *System Calls*) têm a função de estabelecer uma comunicação entre o espaço do usuário (em inglês, *Userspace*) e o núcleo do sistema operacional (em inglês, *Kernelspace*). Elas fornecem um conjunto de interfaces que possibilitam os processos em execução interagirem com o sistema. Através das chamadas de sistema é possível criar novos processos, se comunicar com outros processos, obter informações do sistema operacional, dentre outros recursos.

Essas interfaces agem como mensageiros entre as aplicações e o núcleo do sistema operacional, além de servir como peça chave para garantir a estabilidade do sistema, impedindo que as aplicações fiquem livres para fazerem o que quiserem dentro do núcleo do sistema (Cheng, 2002). A Figura 2.6 mostra o processo de execução de uma chamada de sistema, desde a função no nível de usuário até ela ser executada no núcleo do sistema operacional.

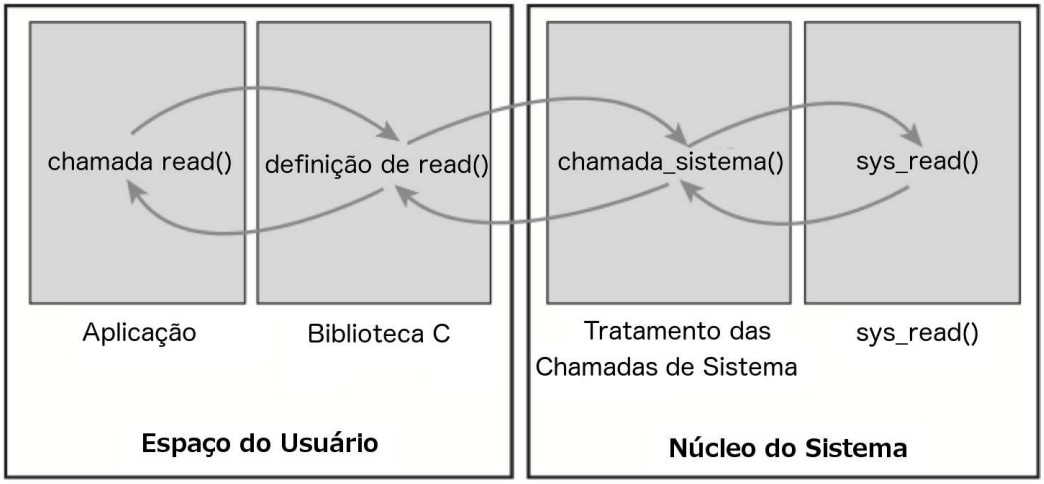

Figura 2.6: Visão geral do processo de execução de uma chamada de sistema no Linux (Cheng, 2002).

Dentro da método proposto nesta dissertação, as chamadas de sistema irão estabelecer a comunicação entre as aplicações de tempo real e o controlador de tensões e frequências do sistema operacional (*Governor*), para que ele possa realizar o gerenciamento do processador de acordo com a aplicação em execução no sistema.

2.5.3 Módulo CPUFreq

O módulo CPUFreq é o responsável por realizar o escalonamento de tensões e frequências do processador, dentro do *Kernel* do Linux. Esse módulo permite que sejam feitos chaveamentos de contexto do processador de acordo com políticas chamadas de *Governors* (Brodowski e Golde, 2013).

Os *Governors* são políticas pré-definidas de escalonamento de frequências do processador com propósitos bem específicos. Os principais *Governors* pré-definidos pelo módulo CPUFreq são (Brodowski e Golde, 2013):

- ***Performance*:** este *Governor* define estaticamente o processador para a frequência mais alta dentro dos limites permitidos.
- ***Powersave*:** este *Governor* define estaticamente o processador para a frequência mais baixa dentro dos limites permitidos.
- ***Userspace*:** este *Governor* permite que o usuário, ou qualquer programa do espaço do usuário, com UID "*root*" (ou seja, com privilégios de administrador), possam definir uma frequência específica no processador. Dessa forma, o controle das tensões e frequências do processador é feita pela aplicação de tempo real.
- ***Ondemand*:** este *Governor* é definido dependendo da utilização atual do processador. Para fazer isso, o processador deve ter a capacidade de trocar frequências rapidamente.
- ***Conservative*:** este *Governor* é muito semelhante ao *Ondemand*, exceto pela maneira de como é feito o incremento e decremento da frequência do processador, pois os saltos de incremente e decremento são menores.

É importante ressaltar que nenhum dos *Governors* citados possuem um canal de comunicação direto com as aplicações do nível de usuário. O único que chega a ter alguma comunicação com o espaço do usuário é o *Userspace Governor*, mas os comandos são feitos via terminal e necessitam de privilégios de administrador do sistema para serem executados.

Portanto, o módulo CPUFreq será a base para a implementação de um novo *Governor*, que realizará o escalonamento das tensões e frequências do processador baseado nos dados informados pelas aplicações de tempo real, de forma *online*.

2.6 RTAI

RTAI significa Interface para Aplicações de Tempo Real (em inglês, *Real Time Application Interface*). Ela surgiu para dar ao Linux o suporte a premissas temporais, mas para isso foi necessário fazer algumas alterações no código fonte do *Kernel*, principalmente nas políticas de tratamento de interrupção e no escalonador (Bucher *et al.*, 2014), para torná-lo totalmente preemptivo. Dessa forma, é possível obter uma plataforma de tempo real, com uma baixa latência e altas exigências de previsibilidade.

O RTAI oferece os mesmos serviços do núcleo do *Kernel* do Linux, acrescentando as características de um sistema de tempo real industrial (Bucher *et al.*, 2014). Ele consiste basicamente de um despachante de interrupção.

As modificações feitas no *Kernel* não chegam a ser invasivas, pois o RTAI utiliza o conceito de HAL (em inglês, *Hardware Abstraction Layer*) para obter informações do Linux e controlar algumas funções fundamentais. Dessa forma, o RTAI considera o Linux como uma tarefa em segundo plano em execução quando não ocorre nenhuma atividade em tempo real (Bucher *et al.*, 2014).

Para se comunicar com as aplicações do nível de usuário, o RTAI utiliza o serviço LXRT (em inglês, *LinuX Real Time*), que por sua vez é um módulo que permite as aplicações utilizarem todos os serviços disponíveis pelo RTAI e seus escalonadores no espaço do usuário, podendo ser utilizado em aplicações de tempo real crítico e não-crítico. A Figura 2.7 mostra uma visão geral da arquitetura do RTAI e seus principais componentes. Na parte inferior, acima do hardware, existe uma camada de abstração de *hardware* (HAL) que é instalado pelo RTAI, cuja função é preparar o código fonte do Linux para assumir o controle completo de eventos externos gerados pelo *hardware*. Os itens na cor verde estão especificando todos os módulos pertencentes ao RTAI. Com a utilização do HAL os módulos do RTAI conseguem cuidar do tratamento de interrupção (as rotinas de interrupção de serviço), escalonamento de tarefas, criação de tarefas, comunicação entre processos (IPC) e controle de acesso de recursos. Toda essa infra-estrutura pode ser acessada pelas tarefas em tempo real por meio da LXRT API (Bergsma, 2009).

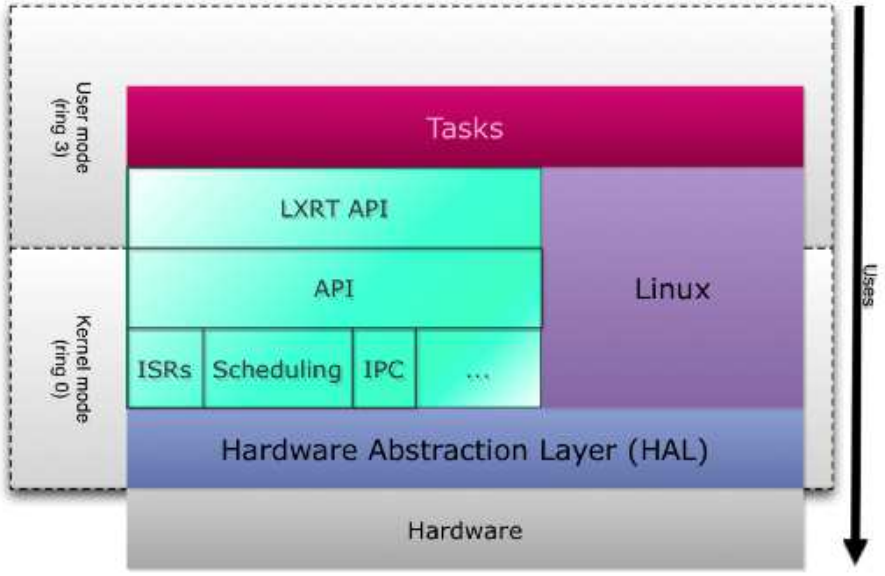

Figura 2.7: Arquitetura do RTAI e seus principais componentes(Bergsma, 2009).

Dentro do contexto da arquitetura proposta nesta dissertação, o RTAI terá o papel de fornecer o ambiente em tempo real dentro do Linux para a realização dos experimentos.

2.7 Resumo

Este capítulo descreveu os principais conceitos necessários para o entendimento deste trabalho. Na Seção 2.1 foram mostrados os principais conceitos que definem sistemas embarcados. Na Seção 2.2 foram introduzidos os conceitos de sistemas de tempo real, como: os tipos de classificação desses sistemas, os tipos de tarefas, as propriedades temporais das tarefas, conceitos básicos de relação de precedência e de exclusão mútua e também comenta sobre o algoritmo de escalonamento *Rate-Monotonic* e quais os testes de escalonabilidade existentes para este tipo de escalonamento. Na Seção 2.3 foram apresentados os principais conceitos relacionados a técnica de escalonamento dinâmico de tensões e frequências do processador. Na Seção 2.4 foram introduzidos os conceitos de grafo de fluxo de controle. Esse grafo ajuda a identificar todos os possíveis caminhos de execução de uma determinada aplicação. Na Seção 2.5 foram apresentados os conceitos básicos sobre o *Kernel* do Linux, que serão fundamentais para o entendimento da arquitetura do método proposto por esta dissertação. Por fim, na Seção 2.6 foram detalhados os principais conceitos sobre a ferramenta RTAI, utilizada na etapa de experimentações desta dissertação para fornecer ao *Kernel* do Linux uma interface para executar aplicações de tempo real.

Capítulo 3

Trabalhos Correlatos

Nesta seção serão apresentados e discutidos os principais trabalhos existentes na literatura que utilizam a técnica DVFS intra-tarefa como base para a construção de suas técnicas. Para alcançar resultados com valor científico, foi decidido realizar uma revisão sistemática (em inglês, *Systematic Review*) baseado no trabalho de Kitchenham *et al.* (2004), que introduziu o conceito em Engenharia de Software Baseada em Evidência (ESBE). Uma revisão sistemática "é um meio de identificar, avaliar e interpretar toda pesquisa disponível e relevante sobre uma questão de pesquisa, um tópico ou um fenômeno de interesse, e tem por objetivo apresentar um avaliação justa de um tópico de pesquisa, usando uma metodologia confiável, rigorosa e auditável" (Kitchenham *et al.*, 2004).

A revisão sistemática requer um esforço considerável quando comparada com uma revisão de literatura conduzida de forma *ad-hoc*, sem planejamento e com critérios de seleção estabelecidos sem nenhuma metodologia pré-definida, enquanto que a revisão sistemática segue um protocolo formal para conduzir uma pesquisa sobre um determinado tema, com uma sequência bem definida de passos metodológicos (Mafra e Travassos, 2006). Este protocolo é construído considerando um tema específico que representa o elemento central da investigação. Os passos da pesquisa, as estratégias definidas para coletar as evidências e o foco das questões de pesquisa são definidos explicitamente, de tal forma que outros pesquisadores sejam capazes de reproduzir o mesmo protocolo de pesquisa e, também, de julgar a adequação dos padrões adotados no estudo (Biolchini *et al.*, 2005).

Em razão disso, foi conduzida uma revisão sistemática com o objetivo de identificar e conhecer as metodologias que utilizam a técnica DVFS intra-tarefa, dentro do contexto de sistemas de tempo real, para diminuir o consumo de energia do processador. O processo para a condução de revisões sistemáticas envolve três etapas (Mafra e Travassos, 2006):

1. **Planejamento da Revisão**: os objetivos da pesquisa são listados e o protocolo da revisão é definido.

2. **Condução da Revisão**: nesta atividade, as fontes para a revisão sistemática são selecionadas, os estudos primários são identificados, selecionados e avaliados de acordo com os critérios de inclusão, exclusão e de qualidade estabelecidos durante a definição do protocolo da revisão.

3. **Análise dos Resultados**: os dados dos estudos são extraídos e sintetizados para análise e apresentação dos resultados.

As próximas seções darão ênfase a essas três etapas. Vale ressaltar que este capítulo abordará apenas os tópicos de maior relevância da revisão sistemática feita, a fim de proporcionar mais coesão e concisão a esta dissertação. A descrição completa e detalhada da revisão sistemática realizada, está disponível em Gonçalves e Barreto (2014).

3.1 Planejamento da Revisão Sistemática

O protocolo e as padronizações utilizadas para a realização dos estudos foram derivados dos trabalhos produzidos por Santos (2008) e Kitchenham e Charters (2007). Dessa forma, nesta seção serão apresentados os principais tópicos do planejamento da revisão sistemática que são constituídos de dois itens: a questão de pesquisa investigada e o ambiente no qual foi realizado o levantamento bibliográfico.

Como o cerne deste trabalho está na técnica DVFS intra-tarefa, a principal questão de pesquisa da revisão sistemática visa responder a seguinte pergunta: "Quais são as técnicas que utilizam como base a técnica DVFS intra-tarefa para reduzir o consumo de energia do processador dentro do contexto de sistemas de tempo real?". Com as respostas à essa pergunta foi possível catalogar o estado da arte em relação a técnica DVFS intra-tarefa e dar embasamento teórico para concepção desta dissertação.

Todas as publicações catalogadas foram extraídas a partir de bibliotecas digitais. Neste caso a biblioteca definida foi a Scopus, acessível em *http://www.scopus.com*. Segundo a editora Elsevier (2013) (Elsevier, 2013b), a Scopus é uma das maiores bases de dados de resumos e citações da literatura de pesquisa com revisão por pares (em inglês, *peer-reviewed*) com mais de 20.500 títulos de mais de 5.000 editoras internacionais. Dentre estas editoras podemos citar: Springer (Springer, 2013); IEEE Xplore Digital Library (IEEE, 2013); ACM Digital Library (ACM, 2013); ScienceDirect/Elsevier (Elsevier, 2013a); Wiley Online Library (Sons, 2013); British Computer Society (Society, 2013); dentre outras. A biblioteca Scopus também inclui aproximadamente 5.3 milhões de conferências de artigos de *proceedings* e *journals*, 400 publicações comerciais, 360 séries de livros e publicações aceitas são disponibilizadas *online* antes da publicação oficial em mais de 3.850 periódicos. Ainda segundo a editora Elsevier (Elsevier, 2013b), a Scopus tem aproximadamente 2 milhões de novos arquivos adicionados a cada ano, com atualizações diárias.

A próxima etapa, após o levantamento bibliográfico feito na Scopus, foi realizar a definição de uma série de critérios para a inclusão e exclusão de publicações. O objetivo desses critérios são selecionar apenas as publicações mais relevantes para a pesquisa. Esses critérios foram divididos em dois filtros. O 1º filtro tem por objetivo selecionar publicações que abordem técnicas para minimizar o consumo de energia do processador, dentro do contexto de sistemas de tempo real, através da análise preliminar apenas do título, resumo / *abstract* e as palavras-chave da publicação. O 2º filtro tem por objetivo selecionar as publicações que utilizem a técnica DVFS intra-tarefa com base para construção de suas técnicas, através da leitura na integra da publicação. Os critérios de inclusão e exclusão utilizados em cada um dos filtros estão descritos em Gonçalves e Barreto (2014).

A partir das publicações selecionadas no 2º filtro, os seguintes dados foram extraídos:

- Dados da publicação:
  - Título;
  - Autor(es);
  - Palavras-chave;
  - Fonte de publicação;
  - Ano de publicação.
- Resumo da publicação:
  - Uma breve descrição do estudo.
- Dados derivados a partir das questões de pesquisa:
  - Método(s) utilizados: técnicas e/ou métodos utilizados;
  - Ferramenta(s): caso tenha sido desenvolvido alguma ferramenta para comprovar os resultados experimentais da metodologia proposta;
  - Impacto (positivo x negativo): indicação dos pontos positivos e negativos da metodologia proposta;
  - Validação do método: descreve como se deu o processo de validação da metodologia proposta;
  - Limitações do método: descreve as limitações da metodologia proposta.
- Dados selecionados para se obter um melhor entendimento dos resultados:
  - Integração de métodos: se o método foi gerado a partir da integração com outro(s) método(s);
  - Modo de aplicação do método: se a técnica é executada de forma *online, offline* ou híbrida;

  - Perspectivas futuras: questões de pesquisa sugeridas como trabalhos futuros, se houver alguma.

- Comentários adicionais do pesquisador.

3.2 Condução da Revisão Sistemática

Nesta etapa, foram feitas as análises quantitativas quanto ao processo de condução da revisão sistemática, onde foi possível gerar gráficos com os resultados das publicações identificadas na máquina de busca da Scopus (ver Código 3.1), bem como o número de publicações aceitas em cada um dos filtros executados (ver Figura 3.1).

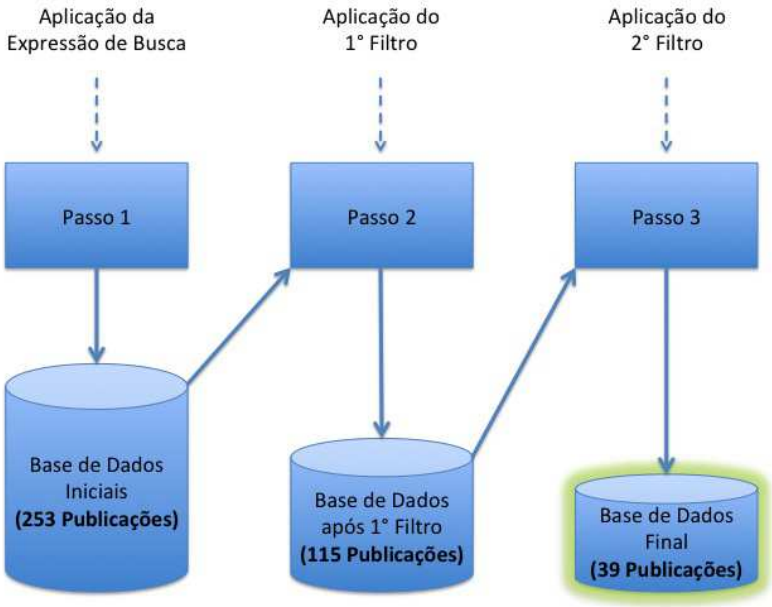

Figura 3.1: O diagrama mostra uma visão global do processo de seleção das publicações e uma analise quantitativa de cada etapa.

Código 3.1: Expressão de busca utilizada na Scopus para realizar a coleta das 253 publicações retornadas na fase inicial da revisão sistemática.

```
1 (ALL("hard real-time" OR "soft real-time" OR "real-time system" OR "real
2 time system" OR "real-time application" OR "real time application" OR "real
3 -time embedded system" OR "real time embedded system") AND ALL("DVFS" OR
4 "dynamic voltage and frequency scaling" OR "dynamic frequency scaling technique"
5 OR "dynamic voltage frequency scaling" OR "DVS" OR "dynamic voltage scaling"
6 OR "DFS" OR "dynamic frequency scaling" OR "voltage scheduling" OR "frequency
7 scheduling" OR "frequency scaling" OR "voltage scaling"OR "dynamic power
8 management" OR "dynamic voltage" OR "dynamic frequency" OR "frequency control"
9 OR "frequency scaling" OR "voltage control" OR "processor frequency" OR
10 "processor voltage" OR "voltage-clock scaling" OR "voltage clock scaling")
11 AND ALL("intra-task" OR "intra task")) AND ( LIMIT-TO(SUBJAREA, "COMP")
12 OR LIMIT-TO(SUBJAREA, "ENGI") OR LIMIT-TO(SUBJAREA, "ENER"))
```

Em resumo, os dados ilustrados na Figura 3.1 mostram que dentre as 253 publicações retornadas pela máquina de busca apenas 39 delas foram selecionados para compor a base de dados final da revisão sistemática. Em outras palavras, apesar das 253 publicações tratarem de metodologias de baixo consumo de energia, apenas 39 publicações aplicam a técnica DVFS intra-tarefa para diminuir o consumo de energia do processador dentro do contexto de sistemas de tempo real. Portanto, essa base de dados final contém as publicações que possuem a maior relevância com a linha de pesquisa desta dissertação.

A partir da base de dados inicial, foi possível fazer a análise quantitativa em relação ao interesse da comunidade científica na linha de pesquisa definida na revisão sistemática. A Figura 3.2 apresenta um gráfico com a visão geral do número total de publicações por ano. Com base nesta informação podemos observar que por volta de 2005 houve um declínio no número de publicações, demonstrando assim que área está chegando ao seu ponto de saturação, onde propor novas contribuições está sendo cada vez mais desafiador.

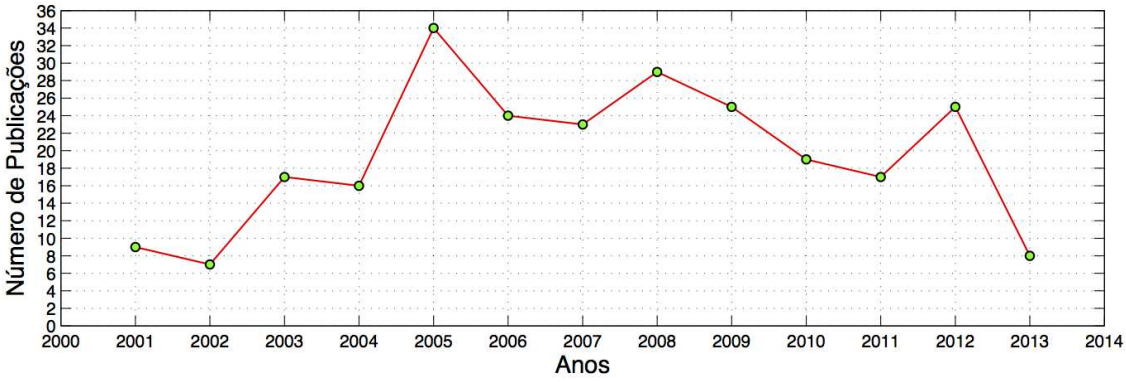

Figura 3.2: Número de publicações por ano.

Vale enfatizar que a base de dados inicial, composta de 253 publicações retornadas pela Scopus, foram extraídas de 25 diferentes editoras, tais como: IEEE, Springer e ACM. A Figura 3.3, detalha o número de publicações por editoras e na Tabela 3.1 é possível consultar o nome completo das editoras catalogadas.

Após a conclusão das análises e aplicações de todos os critérios de exclusão definidos no 1º e 2º filtros, iniciou-se a fase de extração dos dados. As Tabelas 3.2, 3.3 e 3.4 mostram todas as 39 publicações que compõem a base de dados final da revisão sistemática. Logo, com a definição e catalogação das publicações selecionadas após a execução do 2º filtro, encerrou-se a etapa de condução da revisão sistemática.

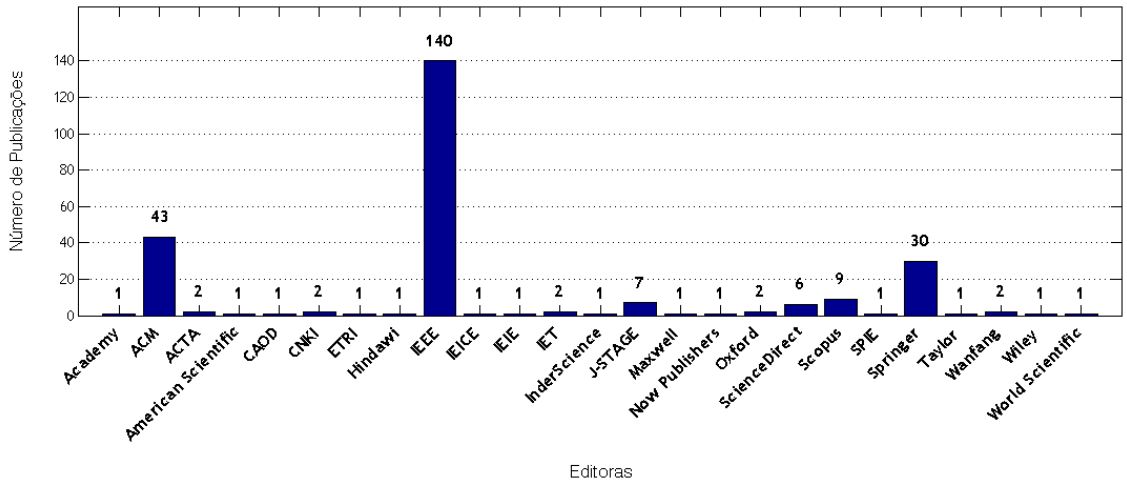

Figura 3.3: Número de publicações por editora.

Tabela 3.1: Nome completo das editoras sem abreviações.

Abreviação / Sigla	**Nome Completo da Editora**
Academy	Academy Publisher
ACM	The Association for Computing Machinery
ACTA	ACTA Press
American Scientific	American Scientific Publishers
CAOD	China/Asia On Demand
CNKI	National Knowledge Infrastructure
ETRI	Electronics and Telecommunications Research Institute
Hindawi	Hindawi Publishing Corporation
IEEE	The Institute of Electrical and Electronics Engineers
IEICE	The Institute of Electronics, Information and Communication Engineers
IEIE	The Institute of Electronics and Information Engineers
IET Digital Library	The Institution of Engineering and Technology
InderScience	InderScience Publishers
J-STAGE	Japan Science and Technology Information Aggregator, Electronic
Maxwell	Maxwell Scientific Organization
Now Publishers	Now Publishers
Oxford	Oxford Journals
ScienceDirect	ScienceDirect (Elsevier)
Scopus	Scopus (Elsevier)
SPIE	Digital Library
Springer	Springer
Taylor	Taylor & Francis Group
Wanfang	Wanfang Data
Wiley	Wiley Online Library
World Scientific	World Scientific Publishing

Tabela 3.2: Publicações que compõem a base de dados final da revisão sistemática (Parte 1).

ID	Título	Autores	Ano	Editora
[P01]	Online Intra-Task Device Scheduling for Hard Real-Time Systems	Muhammad Ali Awan, Stefan M. Petters	2012	IEEE
[P02]	Algorithms for combined inter- and intra-task dynamic voltage scaling	Seo H., Seo J., Kim T.	2012	Oxford University
[P03]	A Car Racing Based Strategy for the Dynamic Voltage and Frequency Scaling Technique	David Cohen, Eduardo Valentin, Raimundo Barreto, Horácio Oliveira, and Lucas Cordeiro	2012	IEEE
[P04]	TALk: A temperature-aware leakage minimization technique for real-time systems	Yuan L., Leventhal S.R., Gu J., Qu G.	2011	IEEE
[P05]	An integrated optimization framework for reducing the energy consumption of embedded real-time applications	Takase H., Zeng G., Gauthier L., Kawashima H., Atsumi N., Tatematsu T., Kobayashi Y., Kohara S., Koshiro T., Ishihara T., Tomiyama H., Takada H.	2011	IEEE
[P06]	Checkpoint extraction using execution traces for intra-task DVFS in embedded systems	Tatematsu T., Takase H., Zeng G., Tomiyama H., Takada H.	2011	IEEE
[P07]	Parametric timing analysis and its application to dynamic voltage scaling	Mohan S., Mueller F., Root M., Hawkins W., Healy C., Whalley D., Vivancos E.	2010	ACM
[P08]	Real-time power management for a multi-performance processor	Ishihara T.	2009	IEEE
[P09]	Energy efficient intra-task dynamic voltage scaling for realistic CPUs of mobile devices	Yang C.-C., Wang K., Lin M.-H., Lin P.	2009	Scopus (Elsevier)
[P10]	Stochastic voltage scheduling of fixed-priority tasks with preemption thresholds	He X., Jia Y., Wa H.	2009	IEEE
[P11]	Efficient algorithms for jitterless real-time tasks to DVS schedules	Chen D.-R., Hsieh S.-M., Lai M.-F.	2008	IEEE
[P12]	Expected energy consumption minimization in DVS systems with discrete frequencies	Chen J.-J.	2008	ACM
[P13]	Improving energy-efficient real-time scheduling by exploiting code instrumentation	Zitterell T., Scholl C.	2008	IEEE
[P14]	Task partitioning algorithm for intra-task dynamic voltage scaling	Oh S., Kim J., Kim S., Kyung C.-M.	2008	IEEE
[P15]	Efficient algorithms for periodic real-time tasks to optimal discrete voltage schedules	Chen D.-R., Hsieh S.-M., Lai M.-F.	2008	IEEE
[P16]	System level voltage scheduling technique using UML-RT model	Neishaburi M.H., Daneshtalab M., Nabi M., Mohammadi S.	2007	IEEE

Tabela 3.3: Publicações que compõem a base de dados final da revisão sistemática (Parte 2).

ID	Título	Autores	Ano	Editora
[P17]	Optimizing intratask voltage scheduling using profile and data-flow information	Shin D., Kim J.	2007	IEEE
[P18]	Static WCET analysis based compiler-directed DVS energy optimization in real-time applications	Yi H., Chen J., Yang X.	2006	Springer
[P19]	Energy-efficient task scheduling algorithm for mobile terminal	Zhang L., Qi D.	2006	IET Digital Library
[P20]	Runtime distribution-aware dynamic voltage scaling	Hong S., Yoo S., Jin H., Choi K.-M., Kong J.-T., Eo S.-K.	2006	IEEE
[P21]	Dynamic voltage scaling for multitasking real-time systems with uncertain execution time	Xian C., Lu Y.-H.	2006	ACM
[P22]	Toward the optimal configuration of dynamic voltage scaling points in real-time applications	Yi H.-Z., Yang X.-J.	2006	Springer
[P23]	Intra-task scenario-aware voltage scheduling	Gheorghita S.V., Basten T., Corporaal H.	2005	ACM
[P24]	An intra-task DVS algorithm exploiting program path locality for real-time embedded systems	Kumar G.S.A., Manimaran G.	2005	Springer
[P25]	Optimal dynamic voltage scaling for wireless sensor nodes with real-time constraints	Cassandras C.G., Zhuang S.	2005	SPIE - Digital Library
[P26]	Optimal integration of inter-task and intra-task dynamic voltage scaling techniques for hard real-time applications	Seo J., Kim T., Dutt N.D.	2005	IEEE
[P27]	Optimizing intra-task voltage scheduling using data flow analysis	Shin D., Kim J.	2005	IEEE
[P28]	Optimizing the configuration of dynamic voltage scaling points in real-time applications	Yi H., Yang X.	2005	Springer
[P29]	ParaScale: Exploiting parametric timing analysis for real-time schedulers and dynamic voltage scaling	Mohan S., Mueller F., Hawkins W., Root M., Healy C., Whalley D.	2005	IEEE
[P30]	The optimal profile-guided greedy dynamic voltage scaling in real-time applications	Yi H., Yang X., Chen J.	2005	Springer
[P31]	Intra-task voltage scheduling on DVS-enabled hard real-time systems	Shin D., Kim J.	2005	IEEE
[P32]	Profile-based optimal intra-task voltage scheduling for hard real-time applications	Seo J., Kim T., Chung K.-S.	2004	ACM and IEEE
[P33]	Collaborative operating system and compiler power management for real-time applications	Aboughazaleh N., Mosse D., Childers B., Melhem R., Craven M.	2003	IEEE

Tabela 3.4: Publicações que compõem a base de dados final da revisão sistemática (Parte 3).

ID	Título	Autores	Ano	Editora
[P34]	Exploring efficient operating points for voltage scaled embedded processor cores	Buss M., Givargis T., Dutt N.	2003	ACM and IEEE
[P35]	Energy management for real-time embedded applications with compiler support	AbouGhazaleh N., Childers B., Mosse D., Melhem R., Craven M.	2003	ACM
[P36]	An intra-task dynamic voltage scaling method for SoC design with hierarchical FSM and synchronous dataflow model	Lee S., Yoo S., Choi K.	2002	ACM
[P37]	Low-energy intra-task voltage scheduling using static timing analysis	Shin D., Kim J., Lee S.	2001	ACM
[P38]	Intra-task voltage scheduling for low-energy hard real-time applications	Shin D., Kim J., Lee S.	2001	IEEE
[P39]	A profile-based energy-efficient intra-task voltage scheduling algorithm for hard real-time applications	Shin D., Kim J.	2001	ACM and IEEE

3.3 Análise dos Resultados

Essa seção tem como objetivo mostrar um breve resumo e fazer uma análise crítica sobre os principais trabalhos catalogados na revisão sistemática. As publicações foram citadas em ordem cronológica crescente, para que se tenha uma visão melhor de como se deu a evolução do estado da arte, ao longo dos anos e foram agrupadas quanto ao modo de execução das metodologias, que podem ser:

- ***Online*:** são métodos que podem sofrer modificações ao longo da execução da aplicação;
- ***Offline*:** são métodos que funcionam em tempo de compilação e são aplicados estaticamente no código fonte da aplicação;
- ***Híbrida*:** são métodos implementados parte *offline* e parte *online*.

3.3.1 Metodologias *Online*

No trabalho de Zitterell e Scholl (2008), os autores propõem um escalonamento mais eficiente de energia para processadores com frequências discretas, chamado de ItcaEDF (em inglês, *Intra-Task Characteristics Aware EDF*). Ela se baseia na integração das técnicas inter e intra tarefas para diminuição dos tempos ociosos do processador e os tempos de folgas das tarefas. No algoritmo intra-tarefa, os autores focam na quantidade de ciclos economizados e em um contador de ciclos, que possibilita contabilizar os diferentes caminhos dentro de um *loop*, permitindo diminuir os níveis de frequência do processador de acordo com as invariantes do *loop*. Quanto ao algoritmo inter-tarefa, os autores implementam um conjunto de bibliotecas que permitem as tarefas compartilhar informações umas com as outras, contribuindo para um melhor escalonamento global e diminuição do tempo ocioso do processador.

No trabalho de Cohen *et al.* (2012), os autores apresentam uma nova política de escalonamento de tarefas de tempo real, que leva em consideração preempções. Essa nova técnica consegue economizar energia mesmo executando as tarefas no seu pior caso. Em resumo, os autores fazem uma analogia do escalonador do sistema operacional com uma corrida de carros, onde o objetivo da corrida é que todos os carros (uma analogia as tarefas) cheguem juntos na linha de chegada, utilizando as menores velocidades (uma analogia a tensões e frequências do processador) sem violar nenhuma premissa temporal. Essa nova política de escalonamento foi experimentada apenas em ambiente simulado e utilizando casos de teste gerados pelos próprios autores.

No trabalho de Awan e Petters (2012), os autores propõem um algoritmo *online* de escalonamento intra-tarefa, cuja principal funcionalidade é ligar e desligar dispositivos

do *hardware*, permitindo que eles sejam utilizados somente quando necessário. Essa metodologia se aplica a sistemas de tempo real crítico e funciona basicamente explorando os tempos de folga entre as execuções das tarefas, a fim de realizar o melhor gerenciamento dos acionamentos e desligamentos dos dispositivos, melhorando assim a performance de economia de energia do sistema. Os experimentos mostram um ganho de economia de energia acima dos 90% em comparação com outras técnicas presentes na literatura.

3.3.2 Metodologias *Offline*

No trabalho de Shin e Kim (2001) foi um dos precursores no desenvolvimento de ferramentas para análise do WCET intra-tarefa em aplicações de tempo real. A principal finalidade dos algoritmos da ferramenta é controlar a velocidade de execução da aplicação baseado nos caminhos de execução do caso médio (em inglês, *Average-Case Execution Path - ACEP*), que são os caminhos mais frequentemente executados. Com essa abordagem os autores conseguiram provar que o algoritmo proposto é mais eficaz na redução do consumo de energia que o algoritmo original intraVS, chamado pelos autores de *(RWEP)-based IntraVS* (em inglês, *Remaining Worst-Case Execution Path-based Intra VS*), onde mesmo utilizando as ACEPs é possível satisfazer as restrições temporais das aplicações de tempo real. Esse método se baseia no perfil de comportamento da aplicação, através da análise dos caminhos de execução mais utilizados (ou *hot paths*), chamado de *(RAEP)-based IntraVS* (em inglês, *Remaining Average-Case Execution Path-based IntraVS*). Sua principal contribuição está na exploração das probabilidades de cada caminho de execução da aplicação e garantir que as restrições temporais sejam respeitadas mesmo executando o pior caso. Os experimentos mostram que o *RAEP-based* é 34% mais eficiente energeticamente que o *RWEP-based*.

No trabalho de Shin *et al.* (2001a), os autores propõem um novo algoritmo de escalonamento de tensão intra-tarefa que controla a tensão de alimentação do processador durante a execução da tarefa, através da exploração dos tempos de folga. Esse método se baseia na análise do tempo de execução estático [1] e na inserção de códigos, dentro do código fonte da aplicação, para a realização dos chaveamentos de tensões e frequências do processador, de forma que o consumo geral de energia seja reduzido. Esses códigos de chaveamento de tensão são definidos para cada um dos blocos de código selecionados a partir do grafo de fluxo de controle da aplicação (CFG). Dessa forma, é possível definir as tensões e frequências para cada bloco de código, aproximando assim o tempo de execução ao deadline da tarefa, sempre respeitando as restrições temporais de todas as tarefas em execução. Neste trabalho os autores introduziram uma nova perspectiva para analisar as CFGs, que consiste em mapear os blocos de código por estruturas condicionais (chamado de *B-types*) e por estruturas de

[1] As análises de tempo de execução estáticas são feitas sem a execução propriamente dita da aplicação.

repetição (chamado de *L-types*), dessa forma é mais fácil analisar e predizer os cálculos do WCEC (em inglês, *Worst Case Execution Cycle*) e RWCEC (em inglês, *Remaining Worst Case Execution Cycle*). Todas essas análises foram introduzidas na ferramenta AVS (em inglês, *Automatic Voltage Scaler*), desenvolvida pelos próprios autores. O único ponto negativo no estudo realizado é a falta de métricas para avaliar os reais impactos causados pela inserção de códigos adicionais dentro das aplicações.

No trabalho de Shin *et al.* (2001b), os autores propõem uma nova metodologia para analise do WCET (em inglês, *Worst Case Execution Time*) em aplicações de tempo real e tomaram com base o trabalho de Shin *et al.* (2001a). Essa nova análise é feita em tempo de compilação, de modo *offline*, utilizando o grafo de fluxo de controle da aplicação, onde o cálculo do WCET é feito para cada nó da CFG, enquanto que no trabalho anterior a estimativa do WCET era feita tendo como base o programa inteiro.

Esses três trabalhos (Shin e Kim, 2001; Shin *et al.*, 2001a,b) foram uns dos primeiros a abordar metodologias que utilização a técnica DVFS intra-tarefa e juntos possuem cerca de 186 citações na literatura, que é facilmente justificável, pois foram os precursores em propor metodologias baseadas nessa técnica.

No trabalho de Buss *et al.* (2003), os autores propõem a exploração e seleção de potenciais pontos de escalonamento de tensão que possam atuar na diminuição eficiente do consumo de energia em aplicações de tempo real não críticos. A problemática desse método está em selecionar esses pontos de controle para atuar em conjunto com a técnica DVS intra-tarefa, proporcionando redução do consumo de energia do processador. Esse método se baseia basicamente em três passos: (1) fazer a análise estática da aplicação e atribuir um fator de desaceleração ideal para cada bloco; (2) computar as frequências de operação com base na análise da aplicação inteira; (3) reatribuir os fatores de aceleração para cada bloco, com base nas frequências de operação válidas e computadas no passo 2. Essa abordagem é muito semelhante à técnica de Shin *et al.* (2001a), onde a principal diferença está na metodologia de definição dos fatores de desaceleração. Esta metodologia não deixa claro se dá suporte a alguns tipos de estruturas como *loops* ocultos e recursões.

No trabalho de Seo *et al.* (2004), os autores propõem uma técnica baseada no perfil de execução da tarefa, onde os níveis de tensão são definidos para cada bloco de código. Esse método tem como objetivo gerenciar melhor os *overheads* de transição, que são totalmente ou parcialmente ignorados nos outros trabalhos presentes na literatura, e obter melhores níveis de redução do consumo de energia do processador. Essa técnica é chamada de "ROEP-*based technique*" (em inglês, ROEP - *Remaining Optimal-Case Execution Path*), que é uma melhoria da metodologia *RAEP-based* proposta por (Shin e Kim, 2001), cuja principal diferença está na redução dos desperdícios de energia com as trocas excessivas de tensões e frequências do processador e com a diminuição dos *overheads* inseridos dentro das aplicações. Seguindo a escala cronológica dos trabalhos

catalogados nesta revisão sistemática, esta foi uma das primeiras abordagens a otimizar estes parâmetros.

No trabalho de Shin e Kim (2005a), os autores melhoraram a eficiência do método *RAEP-based* proposto por eles em Shin e Kim (2001). Nesta nova abordagem, a principal diferença está nas otimizações de *overheads* para a realização das transições de tensão, que antes eram feitas de forma *offline* e agora o método de atribuição de tensões passou a ser *online* e mais eficiente. Os autores utilizaram os mesmos casos de teste para realização dos experimentos e fizeram alterações na ferramenta AVS para adaptá-la a nova abordagem. Um fato interessante a ser relatado é que os autores começaram a introduzir o conceito de ciclos salvos ou ciclos economizados (em inglês, *Saved Cycles ou* C_{saved}), ou seja, são trechos de código que deixaram de ser executados dentro da aplicação. Esse conceito será melhor explicado em Shin e Kim (2005b).

No trabalho de Shin e Kim (2005b), os autores propõem uma otimização na técnica intraDVS usando informações de fluxo de dados da aplicação de tempo real. A metodologia visa melhorar a eficiência energética antecipando os pontos de escalonamento de tensão (em inglês, *Voltage Scaling Points* - VSPs), baseadas nos resultados das análises do fluxo de dados da aplicação. Essa técnica foi chamada de *LaIntraDVS* (em inglês, *Look Ahead IntraDVS*). Em outras palavras, o método proposto antecipa os pontos de controle para maximizar os ganhos de energia da técnica intraDVS, como por exemplo: analisar uma estrutura de repetição e predizer quantas interações serão necessárias e, em seguida, aplicar as tensões e frequências ideais para o bloco de código antes que ele seja realmente executado. No entanto, essa metodologia fornece ganhos mais efetivos apenas quando as distâncias entre os pontos de escalonamento de tensão são grandes.

No trabalho de Kumar e Manimaran (2005), os autores propõem um novo algoritmo DVS intra-tarefa de consumo de energia consciente cujo o objetivo central é explorar os caminhos mais comuns e frequentemente executados dentro de uma aplicação de tempo real. Esse algoritmo foi chamado de CHP (em inglês, *Common Hot Path*). Essa técnica considera todos os caminhos mais executados (ou *hot-paths*) e para cada um deles são atribuídas probabilidades que irão indicar os caminhos mais utilizados. Dessa forma, a técnica consegue combinar todos os *hot paths* em um único caminho base que é comum em comprimento com a maioria dos *hot paths*, assim é possível descobrir qual o caminho que leva a melhores taxas de minimização do consumo de energia, pois nem sempre o caminho mais curto é o mais eficaz para minimização do consumo de energia. No entanto, essa técnica tem dificuldades para lidar com *loops* sem profundidade definida.

No trabalho de Hong *et al.* (2006), os autores propõem uma nova técnica de escalonamento de tensão (DVS) intra-tarefa que não visa apenas explorar as distribuições de tempo de execução da aplicação, mas também o fluxo de dados e a arquitetura. Em outras palavras, essa abordagem utiliza os dados da aplicação e da

arquitetura para predizer o RWCEC e, em seguida, aplicar a técnica de predição de VSPs. Portanto, com este trabalho os autores introduziram o conceito de perfil estatístico de ciclos de execução dentro da técnica DVS intra-tarefa ao invés de ciclos de execução no pior caso (WCEC).

No trabalho de Huizhan *et al.* (2006), os autores propõem uma ferramenta chamada HEPTANE, cuja função é realizar a análise estática do WCET (em inglês, *Worst Case Execution Time*), inserir os códigos da técnica DVFS e definir o perfil de consumo de energia da aplicação. Essa ferramenta trabalha em conjunto com o simulador de energia e performance chamado *Sim-Panalyzer*, que roda em um ambiente *RTLPower* (em inglês, *Real-Time Low Power*), cuja função é simular o ambiente de experimentação para rodar os casos de teste criados pelos autores. Analisando de forma mais incisiva o trabalho, não ficou claro como a ferramenta HEPTANE trata as invariantes de *loops*, na definição do perfil de consumo de energia da aplicação.

No trabalho de Shin e Kim (2007), os autores propõem duas melhorias sobre a técnica IntraDVS. A primeira delas é uma melhoria da técnica chamada RAEP-IntraDVS (em inglês, *Remaining Average-case Execution Path*), que visa otimizar o escalonamento de tensões através da análise das informações da aplicação, levando em consideração o caminho de execução de caso médio remanescente. A outra melhoria é sobre a técnica LaIntraDVS, citada no trabalho Shin e Kim (2005b), que passa a levar em consideração as informações do fluxo de dados para gerar otimizações sobre os pontos de chaveamento de tensão (em inglês, *Voltage Scaling Points* - VSPs), principalmente através da predição das VSPs antes de estruturas condicionais e *loops*.

No trabalho de Neishaburi *et al.* (2007), os autores apresentam uma otimização sobre o escalonamento de tensões intra-tarefa, através da análise do fluxo de dados e do fluxo de controle da aplicação. A partir dessa análise, a metodologia é capaz de antecipar os pontos de escalonamento de tensão, enquanto que a técnica DVFS intra-tarefa tradicional apenas localiza os pontos de controle. Essa metodologia permite adicionar menos *overheads* no código fonte da aplicação.

No trabalho de Chen *et al.* (2008b), os autores propõem uma técnica que visa minimizar o consumo de energia através da análise do fluxo de dados da aplicação, tanto do ponto de vista inter quanto intra tarefa. Essa abordagem consiste basicamente de três fases: (1) primeiramente é feita a transformação harmônica dos períodos de todas as tarefas, em seguida é feita a validação e compartilhamento dos tempos de folga entre as demais tarefas, utilizando um escalonamento definidos pelos autores de *Jitterless* [2] *Schedule*; (2) o próximo passo é calcular a utilização total dado os novos parâmetros das tarefas definidos no passo 1; por último (3) é feita a computação das

[2] *Jitterless* são interferências causadas pela chegada de sucessivas instâncias de uma mesma tarefa (Chen *et al.*, 2008b).

características de cada tarefa, tais como o início e fim relativos, com o objetivo de ajustar as tensões e frequências, evitando que restrições temporais venham ser violadas.

No trabalho de Oh *et al.* (2008), os autores propõem um novo algoritmo de particionamento de tarefas baseado na técnica DVS intra-tarefa, onde o seu principal objetivo é dividir de maneira mais eficiente os blocos de código da aplicação de forma que seja possível diminuir o número de chaveamento de tensões e frequências do processador. Essa abordagem, primeiramente, divide o código fonte da aplicação em um número máximo de seções de código. Em seguida, são calculados os ciclos de execução de cada nó, por meio de simulações estáticas, e são atribuídas penalidades para cada nó que venha a falhar em suas predições. Essas penalidades são utilizadas para decidir se determinados nós deverão ser agrupados ou não. Com essa metodologia os autores conseguiram reduzir o número de chaveamentos de tensões e frequências do processador.

No trabalho de Chen *et al.* (2008a), os autores propõem um algoritmo intra-tarefa e um inter-tarefa para diminuir o consumo de energia durante o escalonamento das tarefas. Essa técnica tem como finalidade diminuir os *overheads* e os tempos de folga entre as tarefas, dando mais previsibilidade e otimizações *offline* para o escalonamento. Para facilitar a geração do escalonamento, as tarefas com períodos arbitrários são transformados em períodos harmônicos para que os tempos de preempção, início e término de cada tarefa possam ser facilmente derivados, principalmente para tratar o que o autor chama de *Jitterless Schedule*. Essa abordagem foi desenvolvida a partir do trabalho de Chen *et al.* (2008b).

No trabalho de Ishihara (2009), o autor propõe uma metodologia baseada em uma nova arquitetura contendo vários núcleos de processamento, chamada *Architecture of Multi-Performance Processor*, onde cada núcleo trabalha em uma frequência e tensão específica. Dessa forma, o processador não perde tempo chaveando tensões e frequências, que são em média na casa das centenas de microsegundos. Essa arquitetura quando integrada a técnica DVFS intra-tarefa permite maximizar os ganhos de economiza de energia, através da diminuição dos *overheads* da técnica DVFS, permitindo fazer chaveamentos de tensão e frequência na casa dos 1.5 microsegundos e dissipando apenas 10 nano-joules. Essa técnica, reduziu cerca de 25% de energia em comparação com a técnica DVS convencional do processador. No entanto, essa arquitetura é mais cara, pois necessita de mais componentes como memórias secundárias, *caches* para cada núcleo de processamento e um *software* mais específico e robusto para realizar todo esse gerenciamento.

No trabalho de Tatematsu *et al.* (2011), os autores propõem uma metodologia que analisa o código fonte da aplicação e lista todos os possíveis locais para a inserção de pontos de controle (também chamado pelos autores de *checkpoints*). Em seguida, todos esses pontos são analisado e os que não trazem ganhos energéticos são removidos. Por fim,

a metodologia compara essa listagem de pontos de controle com uma tabela RWCEC (em inglês, *Remaining Worst Case Execution Cycles*), também extraída da aplicação, para então calcular as tensões e frequências que deverão ser utilizadas no processador. Durante a fase de experimentações, os resultados mostraram que a inserção de alguns *checkpoints* na aplicação podem ocasionar a perda de *deadlines*.

3.3.3 Metodologias Híbridas

No trabalho de AbouGhazaleh *et al.* (2003a), os autores propõem uma técnica que explora as variações dos tempos de execução em diferentes caminhos de execução da aplicação. Esta é uma abordagem híbrida que depende do compilador e do sistema operacional para melhor gerenciar o desempenho e a redução do consumo de energia do processador. O compilador insere os chamados PMHs (em inglês, *Power Management Hints*), que são trechos de código responsáveis por fornecer e coletar informações em tempo de execução da aplicação para o sistema operacional, além de estimar o desempenho da aplicação no pior caso. Dessa forma, o sistema operacional invoca os PMPs (em inglês, *Power Management Points*) para realizar o chaveamento de tensão e frequência do processador com base nas informações passadas pelos PMHs. Durante a etapa de experimentações os autores constataram que o desempenho da aplicação pode ser prejudicado dependendo da quantidade de instruções inseridas pelo compilador.

No trabalho de AbouGhazaleh *et al.* (2003b), os autores tomaram como base o trabalho de AbouGhazaleh *et al.* (2003a), onde o foco principal da técnica passou a ser a colaboração entre o compilador e o sistema operacional. A principal contribuição em relação ao trabalho anterior está no sistema operacional, que passa a monitorar periodicamente os chaveamentos de tensões e frequências do processador baseado nas informações providas pelos *PMHs*.

No trabalho de Mohan *et al.* (2005), os autores propõem uma nova técnica chamada *ParaScale*, que permite fazer análises de tempo paramétrico em conjunto com o escalonamento. Essas análises permitem detectar dinamicamente os limites dos *loops* e o limite inferior do WCET (em inglês, *Wrost Case Execution Time*), durante o tempo de execução da tarefa. Portanto, o ganho desta metodologia está, principalmente, sobre os tempos de folga obtidos sobre as estruturas de repetição. Dentre os trabalhos catalogados nessa revisão sistemática este foi o primeiro a trabalhar com limites paramétricos de *loops*, permitindo ter um melhor controle dos tempos de folga dentro de estruturas de repetição.

No trabalho de Seo *et al.* (2005), os autores propõe uma nova técnica DVS que combinam as técnicas DVS intra-tarefa e inter-tarefa, chamada de *DVS-intgr*. Essa técnica examina os limites inferiores de consumo de energia baseada na técnica DVS intra-tarefa e com essas propriedades são definidos os tempos de execução ideais de cada tarefa. Em seguida,

as tarefas são divididas em vários grupos de trabalho de tal forma que cada tarefa possa ser executada dentro do limite de tempo preestabelecido para cada grupo. Por fim, todas as tarefas são gerenciadas através da utilização da técnica DVS inter-tarefa melhorada para produzir o melhor escalonamento de forma que haja a redução no consumo de energia e nenhuma premissa temporal seja violada.

No trabalho de Gheorghita *et al.* (2005), os autores propõem uma abordagem proativa que visa melhorar a performance do algoritmo de escalonamento intra-tarefa, explorando os tempos de folga que aparecem em tempo de execução, o que possibilita diminuir as tensões e frequências do processador através de trechos de código inseridos na aplicação original, chamados de pontos de escalonamento de tensão ou VSPs (em inglês, *Voltage Scaling Points*). Essa abordagem consiste, basicamente, em quatro etapas: (1) identificar os parâmetros que poderiam ter um impacto sobre o tempo de execução da aplicação; (2) calcular o máximo de impacto destes parâmetros sobre o WCEC da aplicação; (3) particionar o aplicativo em possíveis cenários e, em seguida, selecionar apenas cenários que, isoladamente, reduz o consumo de energia, já levando em consideração os parâmetros calculados no passo 2; por fim, (4) computar o escalonamento DVS para cada cenário selecionado no estágio 3 e combiná-los com o escalonamento global da aplicação de tempo real.

No trabalho de Xian e Lu (2006), os autores propõem uma abordagem que visa integrar as técnicas de escalonamento de tensão intra-tarefa e inter-tarefa. O conceito principal do método proposto é que cada tarefa possa contribuir com informações individuais para que seja possível melhorar o escalonamento individual das demais tarefas em execução, sempre tomando como base as informações globais passadas pelas demais tarefas. Dessa forma, a abordagem é dividida, basicamente, em duas etapas: (1) é calculado estatisticamente o escalonamento de frequência ótimo para múltiplas tarefas periódicas utilizando o escalonamento EDF (em inglês, *Earliest Deadline First*), para processadores que conseguem mudar suas frequências de forma contínua; e (2) para processadores que possuem uma faixa limitada de frequências discretas, é apresentado um algoritmo heurístico específico para construção do escalonamento de frequências baseado em informações de distribuição de probabilidade e restrições de escalonabilidade globais.

No trabalho de Zhang e QI (2006), os autores propõem um algoritmo de escalonamento de tarefas baseado em otimizações genéticas para diminuir o consumo de energia quando são especificados os *deadlines* e os ciclos de execução das tarefas. Esse algoritmo genético híbrido integra as técnicas inter e intra tarefas visando mensurar o pWCEC (em inglês, *Probabilistic Worst-Case Execution Cycle*), a fim de encontrar o melhor coeficiente de escalonamento das tarefas de forma que todas as restrições temporais sejam obedecidas e, ao mesmo tempo, se obtenha uma minimização do consumo de energia do processador. No entanto, os resultados experimentais mostraram que nem sempre é possível alcançar bons resultados de economia de energia, dependendo dos parâmetros inicias das tarefas.

No trabalho de Chen (2008), o autor apresenta uma nova abordagem para minimizar o consumo de energia utilizando funções de densidade de probabilidade com base nas cargas de trabalho das tarefas de tempo real. Para o escalonamento intra-tarefa foi feito um algoritmo eficiente para obter a frequência ideal para uma única tarefa, de modo que o consumo de energia seja minimizado. Enquanto que o algoritmo de escalonamento inter-tarefa, chamado *M-Greedy*, foi desenvolvido com base em uma abordagem de programação linear cuja finalidade é obter as melhores soluções para as tarefas de tempo real baseada em quadros, visando diminuir os tempos de folga.

No trabalho de He *et al.* (2008), os autores exploram os tempos de execução variáveis de tarefas, dentro da política de escalonamento FPPT (em inglês, *Fixed-Priority scheduling with Preemption Threshold*). Essa política de escalonamento permite reduzir os custos com preempções desnecessárias das tarefas. Então, os autores desenvolveram um algoritmo para analisar todas as possibilidades de carga de trabalho para cada tarefa. Em seguida, utilizou esses dados estocásticos para definir as tensões e frequências do processador de acordo com o tamanho da tarefa e sua distribuição de probabilidade, com o intuito de minimizar o consumo de energia no caso médio.

No trabalho de Mohan *et al.* (2010), os autores propõem um técnica que remove as restrições sobre as invariantes de *loops* através de analises paramétricas, com o objetivo de maximizar a identificação dos tempos de folga das tarefas e minimizar o consumo de energia do processador. Dessa forma, os ganhos dessa abordagem está diretamente relacionada com a redução do número de interações dos *loops* mapeados dentro das aplicações. No entanto, os resultados experimentais mostram que a metodologia tem dificuldades para lhe dar com um número de pequeno de interações dentro dos *loops*.

No trabalho de Takase *et al.* (2011), os autores desenvolvem um *framework* com o objetivo de melhor realizar o chaveamento entre performance e consumo de energia do processador. As configurações ótimas do processador são definidas de acordo com cada etapa de execução da tarefa. Além disso, esse *framework* aplica técnicas de otimização sobre a alocação de memória da aplicação, visando diminuir *overheads* de IO (em inglês, *Input and Output*). Dessa forma, todas as características e comportamentos da aplicação são analisados tando do ponto de vista inter quanto intra tarefa. Os resultados dessa análise, reduzem o consumo de energia em tempo de execução de acordo com o comportamento da aplicação. Os resultados experimentais utilizando um sistema de vídeo conferência conseguiram reduzir, em média, o consumo de energia em 44,9% em comparação com outros estudos de caso criados pelos próprios autores.

No trabalho de Yuan *et al.* (2011), os autores propõem um algoritmo de escalonamento intra-tarefa que visa diminuir a temperatura do processador e minimizar o consumo de energia em sistema de tempo real. Essa técnica foi chamada de *TALk* (em inglês, *Temperature Aware Leakage*). A ideia básica do algoritmo é aumentar a frequência quando a temperatura do chip estiver baixa ou quando a carga de trabalho for alta e

colocar o processador em baixo consumo de energia quando a temperatura do chip estiver alta ou com carga de trabalho leve. Para fazer isso, o algoritmo *TALk* foi dividido em duas partes: (1) O *offline*, que usa métodos de programação dinâmica para alcançar os melhores níveis de economia de energia e de temperatura; (2) O *online* tem como objetivo determinar o modo de operação do processador com base na sua temperatura corrente e na quantidade de ciclos remanescentes das tarefas em execução. Com esse algoritmo os autores conseguiram melhorar a economia de energia em cerca de 18% em comparação com a técnica DVFS tradicional. Durante a fase de experimentações, os resultados mostraram que para alguns *benchmarks* o algoritmo *offline* pode violar as restrições temporais de algumas tarefas.

No trabalho de Seo *et al.* (2012), os autores apresentam uma técnica de baixo consumo de energia que se baseia na combinação simultânea entre inter e intra tarefas, também chamada de DVS combinado (em inglês, *Combined* DVS - CDVS). Essa nova abordagem leva em consideração o estado do sistema dormindo (em inglês, *Sleep State* - CDVS-S) e não dormindo (em inglês, *No Sleep State* - CDVS-NS). Ela consiste basicamente de 4 etapas: (1) Aplicar a técnica CDVS-NS para determinar os intervalos de execução das tarefas de modo que o consumo total de energia seja minimizado, sem estados de dormindo; (2) Realizar a análise estática dos blocos de código da tarefa, a fim de identificar os tempos ociosos e as tensões e frequências que deverão ser utilizadas; (3) Combinar os intervalos de tempo salvos na segunda etapa com o maior tempo possível no qual o sistema possa estar no estado ocioso de forma eficiente; e por último (4) Monitorar dinamicamente todas as instâncias das tarefas em execução que concluíram sua execução antes do prazo final e, em seguida, irá decidir se coloca a tarefa em estado ociosa ou em estado de dormindo, dependendo do que for mais econômico energeticamente. Em geral, essa metodologia conseguiu reduzir o consumo de energia em média de 7% com a técnica CDVS-S e de 12% com a técnica CDVS-NS em comparação com outros trabalhos presentes na literatura.

3.3.4 Análises Quantitativas

A partir dos dados catalogados na base de dados final da revisão sistemática foi possível fazer várias análises relevantes para melhor caracterização do estado da arte. A primeira análise quantitativa feita foi quanto aos modos de execução das abordagens. A Figura 3.4 apresenta o gráfico com o resultado da classificação dos modos de execução das metodologias, onde apenas 8% (3 publicações) delas são totalmente *online* (Awan e Petters, 2012; Cohen *et al.*, 2012; Zitterell e Scholl, 2008). É importante observar que as metodologias híbridas, tem grandes chances de alcançar ou até ultrapassar a quantidade de abordagem que utilizam o modo de execução *offline*, em um futuro não muito distante. Principalmente, devido ao uso mais recorrente da integração das técnicas inter e intra tarefas para maximizar os ganhos energéticos sobre o processador.

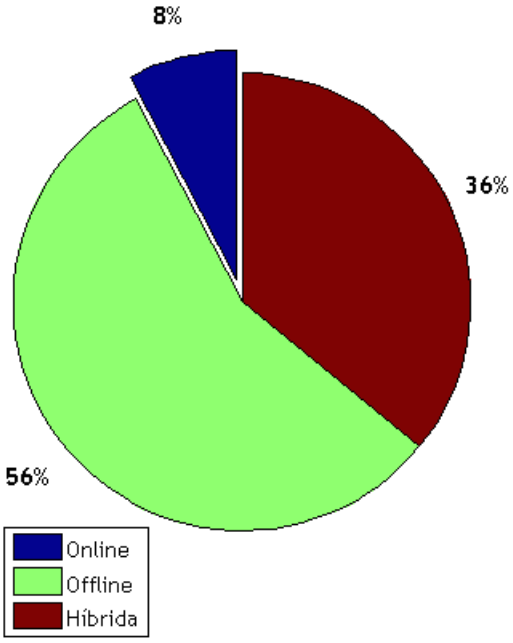

Figura 3.4: Análise quantitativa dos modos de execução das técnicas catalogadas.

A segunda análise feita, avalia a disponibilidade do apoio ferramental, onde apenas 45% (21 publicações) delas estão disponíveis. A Figura 3.5 apresenta o gráfico quantitativo mais detalhado do percentual de publicações que oferecem apoio ferramental.

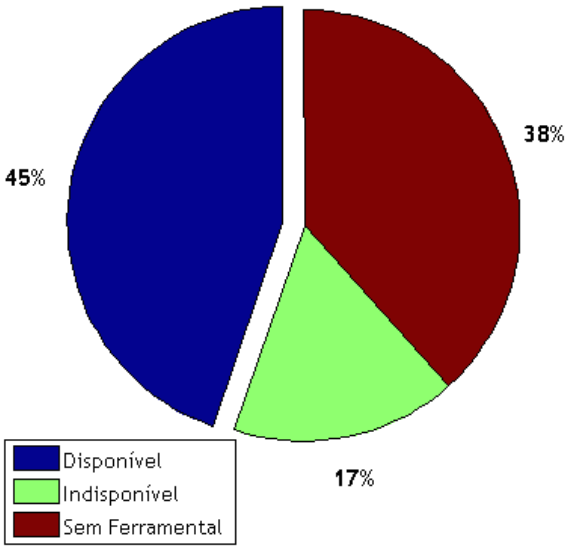

Figura 3.5: Analise quantitativa da disponibilidade ferramental das publicações catalogadas.

A terceira análise feita foi quanto as técnicas que dão suporte a preempção, onde apenas 10% (4 publicações) fornecem esse tipo de suporte (AbouGhazaleh *et al.*, 2003b; Chen *et al.*, 2008a,b; Cohen *et al.*, 2012), enquanto que 13% (5 publicações) dão suporte parcial, ou seja, consideram um ambiente com múltiplas tarefas preemptivas em execução, mas

não deixa claro na técnica como foi implementado (Takase *et al.*, 2011; Chen, 2008; He *et al.*, 2008; Awan e Petters, 2012; Zitterell e Scholl, 2008). A Figura 3.6 ilustra melhor essa análise e, além disso, deixa mais evidente que essa é uma linha de pesquisa pouco explorada pela comunidade científica.

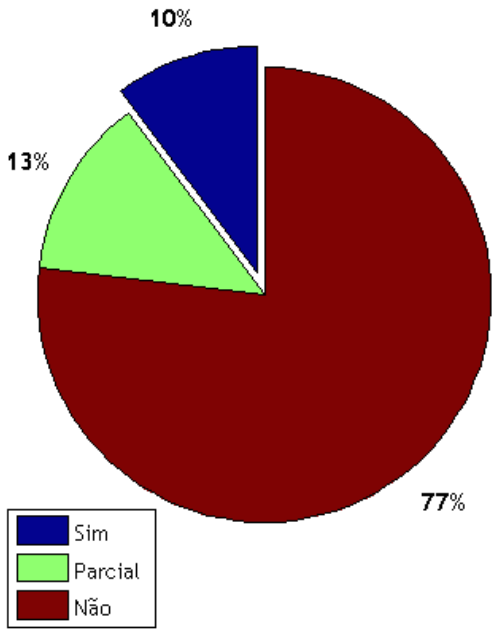

Figura 3.6: Análise quantitativa das técnicas que dão suporte a preempções.

A grande maioria das técnicas não dão suporte a preempções, pois os métodos consideram um ambiente mono tarefa. Por outro lado, as publicações que deixam claro em suas técnicas que dão suporte a preempções tiveram que integrar outras técnicas, com por exemplo a técnica DVFS inter-tarefa. Esses argumentos ressaltam que ainda há muitas linhas de pesquisa a serem exploradas dentro desse contexto.

3.3.5 Análises Qualitativas

Visando obter uma visão geral, no que diz respeito a completude de cada uma das abordagens, efetuamos uma comparação qualitativa entre as 39 publicações que compõem a base de dados final da revisão sistemática. Vale ressaltar que o principal objetivo desta comparação é medir a cobertura das abordagens diante das métricas propostas e não sua eficácia ou desempenho, ou seja, identificar as abordagens que satisfaçam o maior número de métricas.

Diante disto, a Tabela 3.5 apresenta as métricas definidas, sendo que cada coluna da tabela significa: (I) o identificador da métrica (ID); (II) o nome da métrica (Nome); (III) as opções definidas para cada métrica (Opções); (IV) pontuação atribuída a métrica (Pontuação de Cobertura), onde a pontuação é definida da seguinte forma: cada "+" representa 5 pontos; e (V) pontuação máxima permitida pela métrica.

Tabela 3.5: Lista de métricas estabelecidas para realizar a comparação de completude entre as publicações que compõem a base de dados final da revisão sistemática.

Métricas				
ID	**Nome**	**Opções**	**Pontuação de Cobertura**	**Pontuação Máxima**
1	Ferramental	Não possui / Não identificado	0	15
		Possui, mas não foi identificada sua disponibilidade	++	
		Possui e está disponível	+++	
2	Tipo do *Benchmark* Utilizado (abrev. T Bench)	Nenhum ou *Benchmarks* do próprio artigo	0	15
		Benchmarks da literatura	+	
		Benchmarks da indústria	++	
		Combinação entre os tipos de *benchmarks*	+++	
3	Mais de uma fonte de *Benchmark* (abrev. >Bench)	Sim	++	10
		Não	0	
4	Comparação com outras abordagens (abrev. Compara a Abordagem)	Sim	++	10
		Não	0	
5	Suporte à Compartilhamento de Recursos	Não / Não identificado	0	10
		Parcial	+	
		Sim	++	
6	Suporte a Preempções	Não / Não identificado	0	10
		Parcial	+	
		Sim	++	
			Total Pontuação Máxima:	**70**

As métricas apresentadas na Tabela 3.5 tem por objetivo identificar: ID=1 se a abordagem proposta possui um apoio ferramental e se este está disponível; ID=2 o tipo do *benchmark* utilizado na avaliação prática da abordagem, ou seja, se foi um *benchmark* desenvolvido pelo próprio autor, provido da literatura, provido da indústria ou uma combinação entre estes tipos de *benchmarks*; ID=3 se mais de uma fonte de *benchmark* foi utilizada; ID=4 se a abordagem proposta foi comparada com outras na prática; ID=5 se a abordagem proposta dá suporte a compartilhamento de recursos, como por exemplo dispositivos de I/O; e por último ID=6 se a abordagem proposta dá suporte a preempções.

Com base nas métricas apresentadas na Tabela 3.5, todas as 39 publicações foram analisadas e classificadas seguindo os critérios definidos na Tabela 3.6. O resultado dessa análise está ilustrada na Tabela 3.7, onde está divida em quatro partes: (1) são os códigos de identificação das 39 publicações retornadas no segundo filtro, que por sua vez são compostos de 3 partes, por exemplo, para o ID = "Shin_Kim_Lee_2001_a" temos que: (I) são os principais autores da publicação; (II) seguido pelo ano da publicação; e por último (III) um código único para identificar a publicação, visto que alguns autores possuem várias publicações em um mesmo ano (esses códigos vão de "a" até "z"); (2) é a avaliação ferramental da abordagem; (3) é quanto a avaliação experimental da abordagem; e por último (4) é uma avaliação quanto as limitações / suporte da abordagem.

Tabela 3.6: Critérios de classificação das publicações selecionadas no 2º filtro.

Critérios de Classificação das Publicações
1. Maior pontuação geral.
2. Maior pontuação no item "Suporte a Preempções".
3. Maior pontuação no item "Ferramental".
4. Maior pontuação no item "Tipo do *Benchmark* Utilizado".
5. Maior pontuação no item "Comparação com outras abordagens".
6. Publicação mais recente.

A Tabela 3.7 mostra que os trabalhos de AbouGhazaleh *et al.* (2003b) e AbouGhazaleh *et al.* (2003a) possuem maiores completudes em relação as métricas definidas, alem disso estes dois trabalhos são as principais referencias bibliográficas para o desenvolvimento desta dissertação.

Tabela 3.7: Comparação de completude entre as abordagens pertencentes a base de dados final da revisão sistemática.

ID	**Ferramental**	**Avaliação Experimental**			**Limitações / Suporte**		**Pontuação**
		T Bench	**>Bench**	**Compara a Abordagem**	**Suporta Compartilhamento de Recursos**	**Suporte a Preempções**	
Aboughazaleh_2003_b	+++	+++	++	++	0	++	60
Aboughazaleh_2003_a	+++	+++	++	++	0	0	50
Yi_Chen_Yang_2006_a	++	+++	++	++	0	0	45
Takase_2011_a	++	+++	++	0	0	+	40
Chen_2008_a	0	+++	++	++	0	+	40
Buss_Givargis_Dutt_2003_a	+++	+++	++	0	0	0	40
Ishihara_2009_a	+++	+++	++	0	0	0	40
Yuan_2011_a	0	+++	++	++	0	0	35
Xian_Lu_2006_a	0	+++	++	++	0	0	35
He_2008_a	+++	0	0	++	0	+	30
Seo_Seo_Kim_2012_a	+++	+	0	++	0	0	30
Tatematsu_2011_a	+++	+	0	++	0	0	30
Mohan_2010_a	+++	+	0	++	0	0	30
Mohan_Mueller_Root_2005_a	+++	+	0	++	0	0	30
Awan_Petters_2012_a	+++	0	0	0	+	+	25
Seo_Kim_Dutt_2005_a	+++	0	0	++	0	0	25
Shin_Kim_2005_b	++	+	0	++	0	0	25
Shin_Kim_2001_a	++	+	0	++	0	0	25
Hong_Yoo_Choi_Kong_2006_a	++	+	++	0	0	0	25
Gheorghita_2005_a	0	+++	++	0	0	0	25
Chen_Hsieh_Lai_2008_a	0	0	0	++	0	++	20
Chen_Hsieh_Lai_2008_b	0	0	0	++	0	++	20
Neishaburi_2007_a	+++	+	0	0	0	0	20
Lee_Yoo_Choi_2002_a	+++	+	0	0	0	0	20
Cohen_2012_a	0	0	0	0	+	++	15
Zitterell_2008_a	0	0	0	++	0	+	15
Shin_Kim_Lee_2001_a	++	+	0	0	0	0	15
Shin_Kim_Lee_2001_b	++	+	0	0	0	0	15
Oh_Kim_Kim_Kyung_2008_a	0	+	0	++	0	0	15
Shin_Kim_2007_a	0	+	0	++	0	0	15
Kumar_Manimaran_2005_a	0	+	0	++	0	0	15
Shin_Kim_2005_a	0	+	0	++	0	0	15
Seo_Kim_Chung_2004_a	0	+	0	++	0	0	15
Zhang_2006_a	++	0	0	0	0	0	10
Yang_2009_a	0	0	0	++	0	0	10
Cassandras_Zhuang_2005_a	0	0	0	++	0	0	10
Yi_Yang_Chen_2005_a	0	0	0	++	0	0	10
Yi_Yang_2006_a	0	0	0	0	0	0	0
Yi_Yang_2005_a	0	0	0	0	0	0	0

Para finalizar esta seção e a etapa de análises sobre a base de dados final da revisão sistemática, foi feito um diagrama para caracterizar as evoluções do estado da arte ao longo dos últimos anos, ver Figura 3.7. As principais contribuições exibidas nesse

diagrama foram extraídas a partir dos dados catalogados de cada artigo, procurando seguir a ordem cronológica nas quais elas foram apresentadas a comunidade científica.

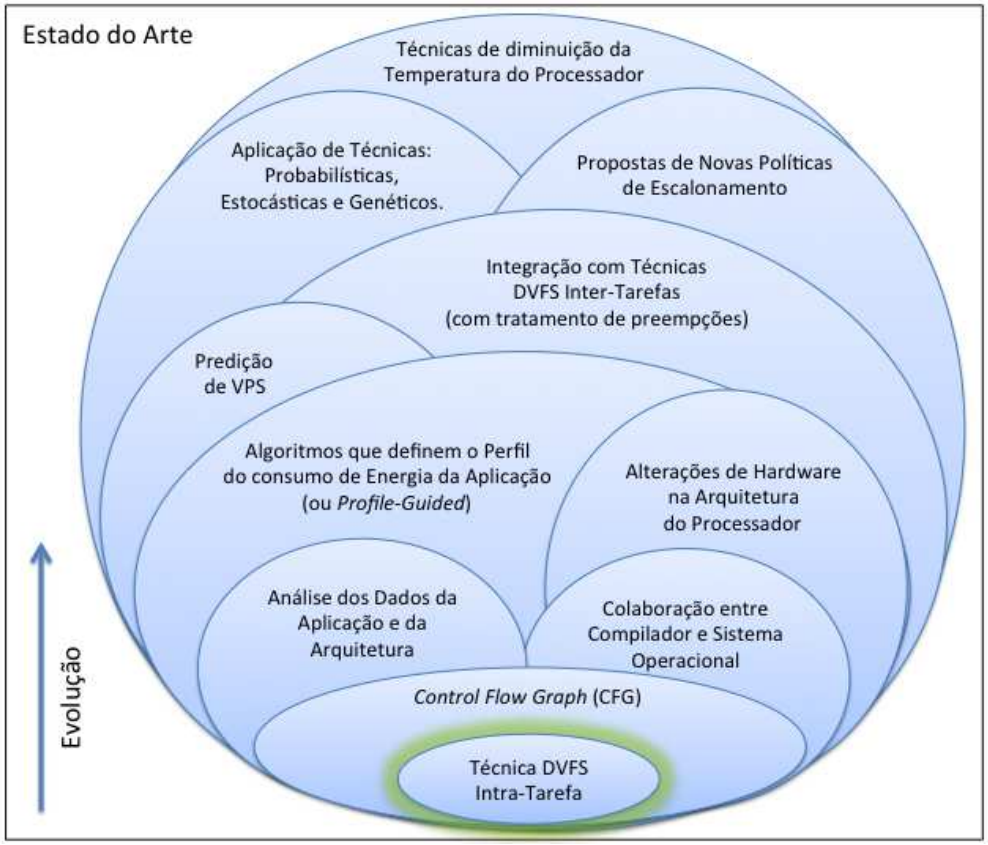

Figura 3.7: O diagrama mostra uma visão mais abrangente da evolução do estado da arte na área de baixo consumo de energia, do ponto de vista da técnica DVFS intra-tarefa.

3.4 Resumo

Este capítulo descreveu e analisou os principais trabalhos que utilizam a técnica DVFS intra-tarefa como base para a concepção de seus métodos com o objetivo de reduzir o consumo de energia do processador. Para alcançar resultados científicos mais concretos na coleta e catalogação dos trabalhos foi realizada uma revisão sistemática. Dessa forma, outros pesquisadores podem auditar todo o processo de levantamento do estado da arte, além de permitir que a comunidade científica possa atualizar com maior facilidade o estado da arte em relação ao uso da técnica DVFS intra-tarefa. Sendo assim, na Seção 3.1 foram apresentados os objetivos da pesquisa e um resumo do protocolo utilizado para realizar o levantamento bibliográfico do estado da arte. Na Seção 3.2 mostrou todo o processo de condução da revisão sistemática, que evolveu o levantamento dos trabalhos relacionados, aplicação dos critérios de seleção dos trabalhos candidatos para compor o estado da arte e, por fim, procurou mostrar uma análise quantitativa parcial sobre os

dados coletados, após a execução de todo o protocolo da revisão sistemática. Por fim, na Seção 3.3 foi apresentado um breve resumo sobre os trabalhos identificados na revisão sistemática e algumas análises quantitativas e qualitativas sobre os dados catalogados de cada publicação. O principal objetivo deste capítulo foi mostrar como está o estado da arte atual na área de baixo consumo de energia do processador do ponto de vista da utilização da técnica DVFS intra-tarefa.

Capítulo 4

Raw Governor

A principal problemática ao utilizar a técnica DVFS intra-tarefa, em um ambiente de tempo real com múltiplas tarefas em execução, é realizar o tratamento de preempções, devido a inexistência de um canal de comunicação entre as tarefas de tempo real, o escalonador do sistema e os *Governors*, de forma que as aplicações possam ter o controle sobre esse tipo de interrupção visando diminuir o tempo de resposta das trocas de tensões e frequências do processador diante de preempções. Em geral, por se tratar de um ambiente de tempo real, além do suporte a preempções é necessário garantir que as restrições temporais de todas as tarefas sejam respeitadas e que haja redução do consumo de energia do processador, principalmente através da redução dos tempos de folga das tarefas. A integração de todas essas características em uma única metodologia é desafiador, principalmente quando são introduzidos parâmetros específicos do sistema operacional e do hardware.

Neste capítulo, é apresentado uma metodologia que visa integrar características preemptivas à técnica DVFS intra-tarefa, por meio de uma abordagem colaborativa entre as aplicações de tempo real e o sistema operacional. Em resumo, esse método visa estabelecer um canal de comunicação entre as tarefas de tempo real em execução com o *Governor* (ver Seção 2.5.3) do sistema operacional com intuito de diminuir os códigos estáticos inseridos dentro das aplicações de tempo real e diminuir o tempo de resposta do chaveamento de tensões e frequências do processador. O método tem como cerne retirar das aplicações o controle das trocas de tensões e frequências do processador e centralizar esse mecanismo dentro do *Governor*, no núcleo do sistema operacional. A fim de reduzir os tempos de chaveamento de contexto [1] do processador que levam mais tempo para serem executados a partir do espaço do usuário em comparação com as trocas feitas no núcleo do sistema.

[1] Troca de contexto do processador é uma analogia a chaveamento de tensões e frequências do processador.

As próximas seções descrevem detalhadamente o método proposto nesta dissertação, onde a Seção 4.1 irá abordar sobre as definições e modelagens que serão utilizadas na formulação do método proposto. A Seção 4.2 irá apresentar a modelagem do método proposto. Por fim, a Seção 4.3 irá descrever o processo metodológico seguido para compor a solução aos itens de pesquisa levantados (ver Seção 1.4).

4.1 Definições e Modelos Adotados

Nesta seção serão definidos todos os parâmetros e modelos matemáticos que serão utilizados como base para formulação do método proposto desta dissertação.

4.1.1 Modelo do Sistema e Modelo de Tarefas

O modelo de sistema leva em consideração tarefas de tempo real críticas e não-críticas, que sejam periódicas e independentes entre si, pois o método proposto não está levando em consideração o compartilhamento de recursos. O modelo também necessita de um CPU C que implemente a tecnologia DVFS e esteja 100% dedicada para realizar o gerenciamento das tarefas de tempo real em execução no sistema. Este CPU C é definido pela tripla (V, F, R), onde V contém a lista de tensões discretas disponíveis (em *Volts*), F contém a lista de frequências discretas disponíveis (em *Hz*) e R é uma lista que contém tuplas de correlação entre V e F. Eles foram modelados respectivamente como: $V = \langle v_0, v_1, v_2, ..., v_m \rangle$, $F = \langle f_0, f_1, f_2, ..., f_m \rangle$ e $R = \langle r_0, r_1, r_2, ..., r_n \rangle$, onde $r_i = \{(v_x, f_y)$ desde que $v_x \in V$ e $f_y \in F\}$.

Já o modelo de tarefas em execução nesse sistema foi definido como $T = \langle \tau_1, \tau_2, \tau_3, ..., \tau_N \rangle$, onde N é o número de tarefas periódicas. Cada tarefa τ_i é definida pela sêxtupla $\langle C_i, P_i, D_i, W_i, p_i, F_i \rangle$, que significa respectivamente:

- C_i é o tempo de execução da tarefa no pior caso (ou WCET);
- P_i é o período da tarefa;
- D_i é o *deadline* da tarefa;
- W_i é a quantidade de ciclos de execução da tarefa no pior caso (ou WCEC);
- p_i é a prioridade da tarefa;
- F_i é a frequência ideal da tarefa (em *Hz*) para que haja minimização do consumo de energia do processador. Este valor somente poderá ser reduzido pela tarefa diante da identificação de SECs, pois caso contrário as premissas temporais das demais tarefas em execução poderão ser violados.

Quanto a geração da escala de execução das tarefas, foi baseado no escalonamento *Rate-Monotonic* (ver Seção 2.2.3), onde as tarefas sempre executarão no seu pior caso. No entanto, a política de escalonamento não é suficiente para definir se um grupo de tarefas é ou não escalonável, para realizar essa validação foi utilizado o teste de escalonabilidade definido na Equação 2.25, que é um teste exato baseado no tempo máximo de resposta das tarefas (ver Seção 2.2.4), garantindo que para toda tarefa τ_i teremos que $w_i^{x+1} \leq D_i$.

O teste de escalonabilidade, além de determinar se um grupo de tarefas é ou não escalonável, serve como parâmetro para se obter uma visão geral do tempo de utilização do processador e do tempo em que ficará ocioso durante o escalonamento das tarefas. Vale ressaltar que esse teste de escalonabilidade prevê as tarefas executando no pior caso e não leva em consideração otimizações de hardware ou software, como por exemplo otimizações em *pipeline* ou em caches de memória, para aumentar o nível de utilização do processador. Em outras palavras, se o modelo levasse em consideração essas otimizações seria possível alcançar uma utilização maior do processador, com menor consumo de energia. No entanto, o teste de escalonabilidade se baseia apenas nas informações intrínsecas de cada tarefa τ_i.

4.1.2 Modelos de *Overhead*

Todo e qualquer processamento ou trocas de tensões e frequências do processador envolvem custos com *overheads*. O método proposto neste trabalho foca basicamente em dois modelos: (1) no tempo de computação dos pontos de chaveamento de tensão (VSPs) inseridos nos códigos fonte das aplicações; e (2) no tempo de chaveamento de contexto do processador durante os retornos de preempções das tarefas de tempo real em execução no sistema.

O modelo 1 visa reduzir, principalmente, os pontos de validações de tensões e frequências do processador, como pode ser visto na Figura 4.1 proposto por Yi *et al.* (2005), através do uso das funções *CKs*. Em comparação com o método proposto nesta dissertação, todos os pontos de validações de tensões e frequências [2] do processador poderão ser removidos, pois o sistema operacional passará a ser responsável por auxiliar as aplicações neste processo, diminuindo assim os códigos estáticos inseridos dentro das aplicações de tempo real.

Já o modelo 2, visa reduzir o tempo de resposta do chaveamento de tensões e frequências do processador durante as trocas de contexto entre as tarefas em execução, principalmente diante de preempções ocasionadas por tarefas de maior prioridade. Esse gerenciamento é feito através do uso das informações passadas pelas tarefas ao sistema

[2] Os pontos de validações de tensões e frequências são trechos de código inseridos dentro das aplicações de tempo real para validade se a tensão e frequência atual do processador está compatível com as necessidades atuais da aplicação.

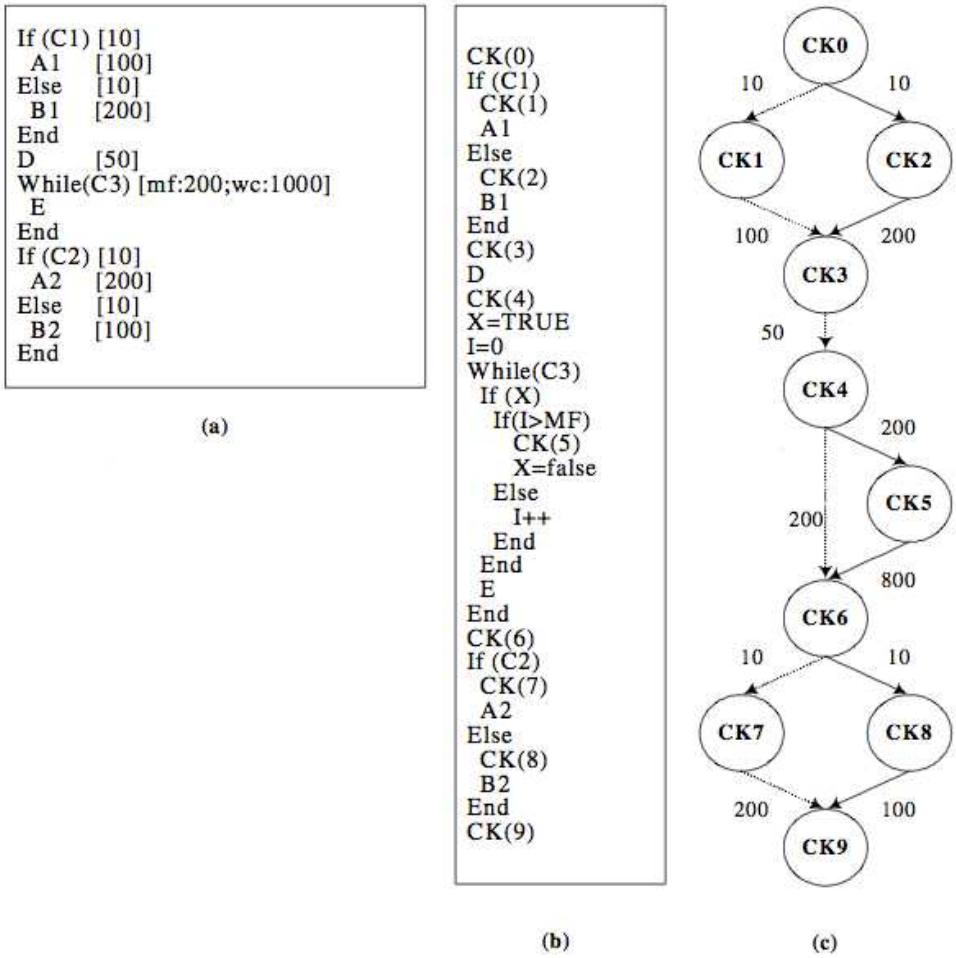

Figura 4.1: Exemplo de uma tarefa de tempo real, onde: (a) mostra o código fonte; (b) mostra o processo de inserção de pontos de controle (*CKs*), proposto por Yi *et al.* (2005); e (c) o CFG extraído a partir do código fonte do item b, onde as arestas contém os custos computacionais de cada caminho no pior caso.

operacional e pelo restabelecimento mais eficientes das tensões e frequências ideais da tarefa que está retornando de preempção, visto que as tarefas que estão em execução no espaço do usuário não sabem o tempo exato no qual uma determinada tarefa estará retornando de preempção. Dessa forma, é possível simular VSPs dinâmicas e *online* personalizadas para cada tarefa de tempo real em execução no sistema, sempre tendo como base os dados informados pelas tarefas. Sendo assim, boa parte do controle do chaveamento de contexto do processador passará a ser feito pelo sistema operacional, o que abre a possibilidade de se reduzir o número de VSPs estáticas inseridas no código fonte da aplicação, ressaltando teoricamente as afirmações feitas no modelo 1.

4.1.3 Modelo de Energia

O modelo de energia desta dissertação se baseia em processadores que utilizam a tecnologia CMOS (ver Seção 2.3), onde a potência consumida é dada por:

$$P_{CMOS} = P_{estatica} + P_{dinamica} \tag{4.1}$$

A potência estática é dissipada devido a corrente de fuga, que está diretamente ligada ao limiar da tensão aplicada e a arquitetura no qual o processador foi desenvolvido (Shin e Kim, 2007). Segundo Shin e Kim (2007), a potência estática pode ser ignorada, pois com o avanço da tecnologia esses valores tendem a serem constantes de acordo com o modelo de processador utilizado. Logo, a potência estática do processador também será desconsiderada do modelo de energia desta dissertação.

Dessa forma, o modelo de energia do método proposto irá considerar apenas a potência dinâmica, que por sua vez é definida pela equação (Shin e Kim, 2007):

$$P_{dinamica} = \alpha \cdot C_L \cdot V_{dd}^2 \cdot f_{clk} = C_{eff} \cdot V_{dd}^2 \cdot f_{clk} \tag{4.2}$$

Onde α é o fator de chaveamento de contexto (ou seja, é o número médio de transições de alta para baixas tensões dentro de um período do relógio), C_L é a capacitância de carga do processador, V_{dd} é a tensão de alimentação do processador, f_{clk} é a frequência do relógio do processador e C_{eff} é a capacitância de carga eficaz do processador.

Analisando a Equação 4.2 temos que a potência dinâmica nos processadores que utilizam a tecnologia CMOS é diretamente proporcional a capacitância de carga, frequência de *clock* e a tensão de alimentação ao quadrado. A partir dessa equação derivou-se o modelo final do consumo de energia do processador em função do tempo T, que é dado por:

$$E_{dinamica} = \int_0^T P_{dinamica}(t)dt = C_{eff} \cdot V_{dd}^2 \cdot f_{clk} \cdot T \tag{4.3}$$

sabendo que o número de ciclos executados de uma tarefa de tempo real é dado por $N_c = f_{clk} \cdot T$, logo:

$$E_{dinamica} = C_{eff} \cdot V_{dd}^2 \cdot N_c \tag{4.4}$$

Portanto, este será o modelo final do consumo de energia do processador baseado no número de ciclos das tarefas em execução no sistema.

4.2 Modelagem do Método Proposto

O método proposto desta dissertação se baseia em um ambiente colaborativo entre as aplicações de tempo real e o sistema operacional, para realizar o chaveamento de contexto

eficiente do processador, principalmente diante de preempções entre as aplicações. Para idealizar esse ambiente, o sistema precisará que todas as tarefas de tempo real τ_i informem os dados definidos no modelo de tarefas (ver Seção 4.1.1), pois essas informações serão fundamentais para que sejam aplicadas as tensões e frequências ideais no processador, para cada tarefa de tempo real que venha a entrar em execução no sistema.

Dessa forma, o gerenciador de tensões e frequências do sistema operacional, chamado de *Governor* no *Kernel* do Linux (ver Seção 2.5.3), poderá atuar como uma VSP dinâmica dentro do núcleo do sistema, permitindo que VSPs estáticas inseridas no código fonte das aplicações possam ser removidas, gerando menos *overheads*. Assim, é possível estender a técnica DVFS intra-tarefa para núcleo do sistema operacional com o suporte a preempção.

Portanto, a partir do contexto apresentado e dos modelos definidos na Seção 4.1 foi possível chegar a modelagem do método proposto ilustrado na Figura 4.2, que tomou como base a arquitetura do RTAI integrada ao *Kernel* do Linux (ver Seção 2.6). Essa nova arquitetura tem como objetivo estabelecer um novo canal de comunicação entre as aplicações de tempo real e o *Governor* do sistema operacional a fim de proporcionar o tratamento de preempções *online* e personalizado para cada tarefa que venha a entrar em execução no sistema.

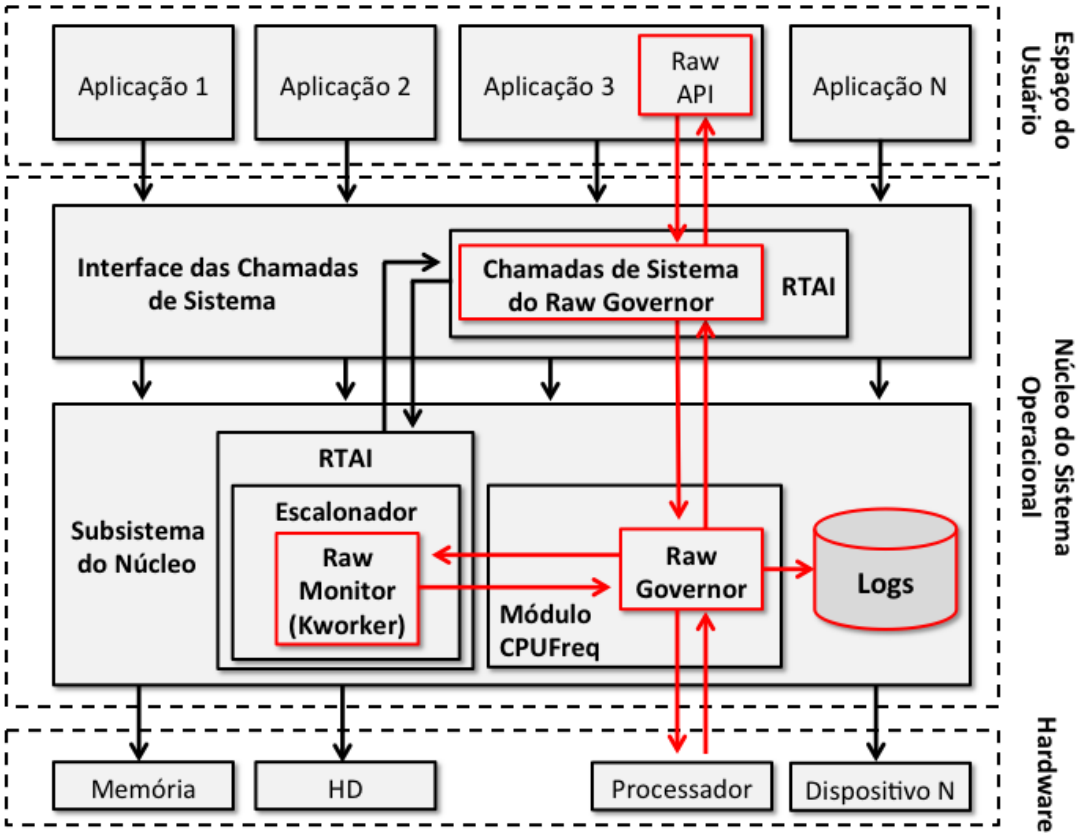

Figura 4.2: Modelagem do método proposto.

Os itens em vermelho, na Figura 4.2, mostram os novos componentes que foram criados para permitir que o *Governor* possa monitorar e atuar como VSPs dinâmicas com suporte a preempções. Esses novos componentes receberam o prefixo **Raw**[3], apenas por questões de padronização das nomenclaturas. O objetivo central dessa arquitetura é proporcionar ao sistema operacional um melhor controle sobre as tarefas de tempo real em execução no sistema, de forma que ela possa atuar dinamicamente no tratamento de preempções de acordo com as características intrínsecas de cada tarefa.

As seções a seguir irão descrever detalhadamente cada um dos novos componentes presentes na modelagem proposta na Figura 4.2.

4.2.1 Chamadas de Sistema do *Raw Governor*

As chamadas de sistema têm como objetivo intermediar a comunicação entre as aplicações de tempo real e o núcleo do sistema operacional. Ela fornece um conjunto de funções que permite: (1) definir o WCEC, o RWCEC, as tensões e frequências ideais utilizadas por uma tarefa específica; (2) obter as tensões e frequências atuais do processador; (3) obter as escalas de tempo do sistema (ou *tick timer*), dessa forma as tarefas podem calcular o tempo restante de processamento até o final do período corrente; e por fim, (4) obter a fatia de tempo (ou *time slice*) do período atual de uma determinada tarefa. Dessa forma, as aplicações de tempo real passam para o núcleo do sistema operacional as informações necessárias para que o *Governor* possa realizar o gerenciamento das trocas de contexto do processador de acordo com a tarefa que venha a entrar em execução no sistema.

Além disso, as chamadas de sistema tem o papel de modularizar e criar uma camada de abstração com o intuito de evitar que as aplicações tenham acesso direto ao núcleo do sistema e permitir que sejam feitas validações sobre os dados informados. O Apêndice B mostra algumas das chamadas de sistema do *Raw Governor*.

4.2.2 *Raw Governor*

O *Raw Governor* é uma política de escalonamento de tensões e frequências do processador implementada dentro do módulo CPUFreq (ver Seção 2.5.3). Em comparação com os outros *Governors* presentes no Linux, este proposta trata de um *Governor* dinâmico e reativo a preempções. Dentre suas atribuições, ele é responsável por:

- Aplicar tensões e frequências específicas, em um determinado núcleo do processador, de acordo com as necessidades da tarefa que está em execução.

[3]O prefixo **Raw** tem o mesmo conceito utilizado no *Socket Raw* aplicado no contexto de rede de computadores, ou seja, permitir o envio e o recebimento direto de informações sem quaisquer interferências do *Kernel* (Hartkopp *et al.*, 2015). Logo, este prefixo representa o novo canal de comunicação criado entre as aplicações em tempo real e o controlador de tensões e frequências do sistema operacional.

- Fazer um tratamento sobre os dados informados para que valores incorretos de tensões e frequências não venham a ser aplicados no processador. Diante de dados inconsistentes, o *Raw Governor* está pré-programado para aplicar o valor válido superior mais próximo ao informado, pois assumir valores menores podem acarretar na perda de *deadline* de uma ou mais tarefas.

- E por fim, prover um mecanismo mais eficiente para realizar a troca de contexto do processador, de forma personalizado para cada tarefa que venham a entrar em execução no sistema, principalmente as que estão retornando de preempção, evitando que as tarefas executem com valores incoerentes de tensões e frequências, geralmente definidos pela última tarefa que estava em execução no sistema. O Algoritmo 1 mostra o pseudocódigo do *Raw Governor* que realiza o gerenciamento das trocas de contexto do processador das tarefas que estão retornando de preempção.

Algoritmo 1: Pseudocódigo utilizado pelo *Raw Governor* para realizar o tratamento de preempções.

```
travar_semaforo_governor();
tarefa = obter_dados_tarefa_real_time();
se tarefa->pid > 0 então
    se tarefa->wcec > 0 então
        freq_ideal = obter_freq_ideal(tarefa);
        __cpufreq_trocar_contexto_processador(freq_ideal);
        tarefa->frequencia_corrente = freq_ideal;
    fim
    /* Atualiza a estrutura de dados da tarefa, indicando que o tratamento
       de preempcao já foi feito pelo Governor.                          */
    atualizar_flags_monitoramento_tarefa(tarefa);

    /* Inicializa as flags de monitoramento para detectar novos retornos
       de preempção.                                                     */
    limpar_flags_monitoramento_governor();
fim

destravar_semaforo_governor();
```

Esse sistema de tratamento de preempções conta com um serviço de monitoramento integrado ao escalonador do sistema operacional, chamado *Raw Monitor* (será melhor detalhado na Seção 4.2.3). No entanto, para que o *Raw Governor* e o *Raw Monitor* tenham um bom desempenho e venham proporcionar ganhos a técnica DVFS intra-tarefa é necessário que todas as tarefas τ_i em execução mantenham seus valores de tensões e frequências ideais sempre atualizados.

O outro papel do *Raw Governor* é gerar *logs* de todo o processo de gerenciamento do processador, como por exemplo: a carga de trabalho atribuída a cada frequência válida do

processador, o tempo no qual a CPU ficou ociosa, o número total de trocas de contexto do processador e o consumo total de energia dinâmica. Essas informações são fundamentais para a realização das análises e validações dos resultados experimentais. O Apêndice F mostra alguns exemplos dos dados estatísticos gerados pelo *Raw Governor*.

4.2.3 Raw Monitor

O *Raw Monitor* é implementando dentro do *Raw Governor*, no entanto ele possui um mecanismo de colaboração com o escalonador do sistema operacional, que por sua vez é responsável por alertá-lo sobre as tarefas que entrarão em execução ou que estão retornando de preempção, devido interrupções geradas por tarefas de maior prioridade. O Algoritmo 2 mostra o pseudocódigo inserido dentro do escalonador do sistema operacional para se comunicar com o *Raw Governor*.

Algoritmo 2: Pseudocódigo utilizado para estabelecer comunicação entre o escalonador do sistema operacional e o *Raw Governor*.

```
/* O primeiro núcleo do processador sempre será de uso exclusivo do RTAI.   */
CPUID_RTAI = 0;
...
/* O monitoramento de preempções é feito apenas para as tarefas em execução no
   processador do RTAI.                                                      */
se (cpuid == CPUID_RTAI) então
    tarefa_corrente = obter_tarefa_corrente(CPUID_RTAI);
    dados_tarefa = obter_dados_tarefa_tempo_real(tarefa_corrente->pid);
    se (dados_tarefa) e (tarefa_corrente->flagReturnoPreempcao == 1) então
        /* Será enviada uma mensagem para o Raw Monitor informando os dados da
           tarefa de tempo real, para que ele possa ajustar a tensão e
           frequência do processador da tarefa que está retornando de
           preempção.                                                        */
        obj_raw_governor = obter_governor(CPUID_RTAI);
        /* Verificando se as funções necessárias para a comunicação com o Raw
           Governor estão disponíveis ao escalonador.                        */
        se (obj_raw_governor) e (obj_raw_governor->acordar_raw_monitor) então
            obj_raw_governor->acordar_raw_monitor(obj_raw_governor,
            dados_tarefa);
        fim
    fim
fim
```

Dessa forma, o monitor consegue controlar mais eficientemente as trocas de contexto do processador de acordo com a tarefa que esteja em execução no sistema. Sendo assim, seus objetivos específicos podem ser decompostos em três, são eles:

1. Monitorar as trocas de contexto das tarefas de tempo real feitas pelo escalonador do RTAI;

2. Auxiliar o *Raw Governor* nas trocas das tensões e frequências ideais do processador, para cada tarefa τ_i que esteja retornando de preempção;

3. Diminuir o tempo de resposta das trocas de tensões e frequências do processador, pois o *Raw Monitor* sabe o momento exato em que deve ser feito o chaveamento de contexto.

Dessa forma, as tarefas de tempo real em execução no sistema não precisam se preocupar com preempções, pois sempre serão aplicados os valores de tensões e frequências ideais informados por elas ao sistema operacional. Além disso, a modelagem do método proposto permite que as tarefas possam ajustar esses valores em tempo de execução, dando mais robustez e flexibilidade para a solução proposta.

Vale ressaltar que esses valores de tensões e frequências definidos para cada uma das tarefas de tempo real do sistema, garantem que mesmo executando no pior caso nenhuma das tarefas irá violar suas premissas temporais, independe do tempo que venham a ficar preemptadas pelo sistema operacional, pois todas as possibilidades foram levadas em consideração durante os testes de escalonabilidade.

4.3 Metodologia

Esta seção tem como objetivo resumir os passos de aplicação da solução proposta com base nos modelos definidos na Seção 4.1 e na arquitetura do método proposto definida na Seção 4.2. Os passos são:

1. Definir a lista de tarefas de tempo real que serão executadas no sistema e os parâmetros de cada uma delas, conforme descrito na Seção 4.1.1, exceto as tensões e frequências ideais que serão definidas no próximo passo.

2. Definir as tensões e frequências ideais de cada tarefa, sempre assumindo o pior caso de execução. Para realizar a obtenção desses parâmetros foi utilizada a ferramenta *Smartenum*, proposta no trabalho de Valentin e Barreto (2010), cuja função é analisar todas as possíveis escalas de execução dado um conjunto de tarefas, a fim de identificar as escalas que utilizam os menores valores de tensões e frequências e que possuem o menor consumo de energia. O Apêndice C mostra como utilizar a ferramenta *Smartenum* para obter as tensões e frequências ideais de um grupo de tarefas.

3. Aplicar um teste de escalonabilidade exato (ver Seção 2.2.4) para validar os parâmetros definidos no passo anterior, a fim de garantir que todas as tarefas serão escalonáveis e que não haverá violações de premissas temporais, pois os dados obtidos com a ferramenta *Smartenum* não levou em consideração as

otimizações de *hardware* e *software* (como por exemplo: tecnologias de *pipeline*, *cache* de instruções e de memória) na definição das tensões e frequências ideais das tarefas. É importante ressaltar que esse teste de escalonabilidade deve ser feito utilizando os tempos de computação reais das tarefas dentro do sistema operacional, assumindo como base as tensões e frequências definidas pela ferramenta *Smartenum*, pois dessa forma as características da arquitetura estarão intrínsecas nos tempos de computação de cada tarefa. Exemplo, uma tarefa com 1000 ciclos executando a uma frequência de $1000Hz$ teoricamente levaria 1 segundo para ser concluída, no entanto ao executar na arquitetura A o tempo de computação foi de $1,1$ segundos e na arquitetura B o tempo de computação foi de $0,89$ segundos, isso mostra que vários fatores tanto de *software* quanto de *hardware* estão diretamente ligados ao tempo de computação real da tarefa (Nogueira *et al.*, 2014). Por esse motivo é que as tensões e frequências obtidas pela ferramenta *Smartenum* devem ser validadas. O Apêndice D mostra o *script* feito no MatLab[4] para realizar o teste de escalonabilidade baseado no tempo de resposta das tarefas.

4. Inserir trechos de código, manualmente, nas tarefas de tempo real para passar as informações calculadas ao sistema operacional, preferencialmente no início da função principal de cada tarefa. Dessa forma, o sistema terá acesso a uma base de dados detalhada de cada tarefa que venha a entrar em execução no sistema, permitindo que tanto no início da execução da tarefa quanto no retorno de preempções, os valores das tensões e frequências ideais do processador sejam restabelecidos de forma mais eficiente. A Figura 4.3 mostra o código fonte de uma aplicação de tempo real, onde o trecho de código selecionado na cor azul exemplifica a passagem de dados locais para o sistema operacional.

5. Por fim, colocar as tarefas de tempo real em execução no sistema operacional, já modificado com os novos componentes propostos na modelagem definida na Seção 4.2.

Dessa forma, é possível integrar características preemptivas à técnica DVFS intra-tarefa. As principais referências deste trabalho são AbouGhazaleh *et al.* (2003a) e AbouGhazaleh *et al.* (2003b), que também propõem técnicas de baixo consumo de energia através de uma abordagem colaborativa entre as aplicações de tempo real e o sistema operacional. A Figura 4.4 ilustra o método proposto por AbouGhazaleh *et al.* (2003b), onde cada um dos blocos vermelhos mostram as interrupções periódicas geradas pelo sistema para realizar as aplicações e validações das tensões e frequências do processador.

O principal ganho da técnica proposta nesta dissertação em relação aos trabalhos de AbouGhazaleh *et al.* (2003b) está no sistema de monitoramento e detecção de preempções

[4]http://www.mathworks.com

```
void *init_task_matmult(void *arg)
{
    RT_TASK *Task_Matmult;
    struct thread_param *config = (struct thread_param*) arg;
    int idTask = config->idTask;
    unsigned long pidTask = 0;
    ...
    if(!(Task_Matmult=rt_thread_init(idTask, prioridade, 0, SCHED_FIFO, CPU_ALLOWED)))
    {
        printf("[ERRO] Não foi possível criar a tarefa MatMult (C-Benchmark).\n");
        exit(1);
    }
    ...
    rt_allow_nonroot_hrt();
    rt_task_make_periodic(Task_Matmult, Tinicio, Tperiodo_Matmult);
    rt_change_prio(Task_Matmult, prioridade);
    pidTask = rt_cfg_get_pid(Task_Matmult);

    init_info(Task_Matmult, WCEC_Matmult, config->cpuFrequenciaIdeal,
              config->cpuVoltagemIdeal);

    /** INICIO: PROCESSAMENTO DA TAREFA... **/
    InitSeedMatMult(idTask);
    TestMatMult(Task_Matmult, idTask, ArrayA, ArrayB, ResultArray);
    /**   FIM: PROCESSAMENTO DA TAREFA...  **/

    rt_task_wait_period(); // **** WAIT

    rt_task_delete(Task_Matmult);

    return 0;
}
```

Figura 4.3: O código mostra um exemplo de como as aplicações passam informações locais ao sistema operacional.

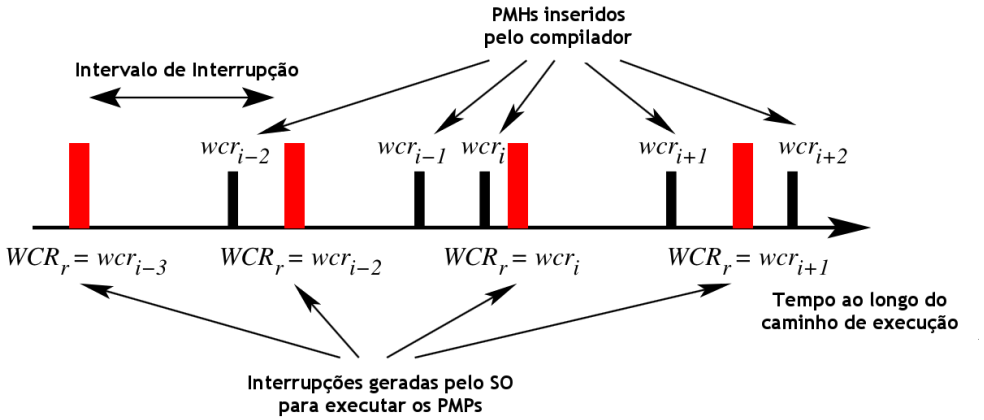

Figura 4.4: Método proposto por AbouGhazaleh *et al.* (2003b).

que gera menos *overheads*, pois o *Raw Monitor* trabalha dentro do escalonador do sistema operacional, não necessitando criar interrupções periódicas para validar as tensões e frequências do processador, pois o *Raw Governor* atua somente diante de dois tipos de eventos: (1) trocas de contexto entre tarefas pelo escalonador do RTAI; e (2) chamadas de sistemas requisitadas a partir das aplicações de tempo real em execução (ver Seção 4.2.1).

Outra vantagem dessa metodologia é a redução do número de trocas de mensagens entre as aplicações e o núcleo do sistema operacional. A Figura 4.5 monstra uma visão geral de como ocorrem as trocas de mensagem, entre as camadas do sistema operacional, do ponto de vista das técnicas que tratam o chaveamento de contexto do processador diante de preempções no espaço do usuário e no núcleo do sistema. Analisando de forma mais pontual essas trocas de mensagem, é possível verificar que o gerenciamento feito no núcleo do sistema exige menos trocas de mensagem quando comparado com o gerenciamento feito no espaço do usuário.

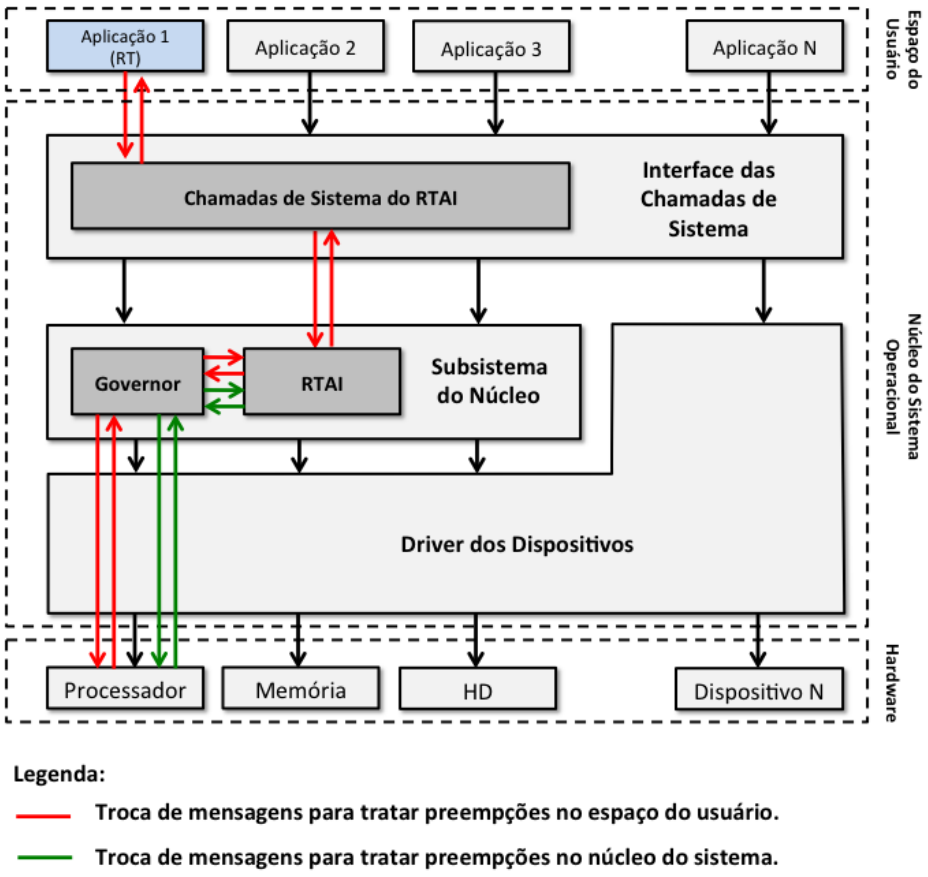

Figura 4.5: Visão geral das trocas de mensagens entre as técnicas que realizam o tratamento de preempção no espaço do usuário e no núcleo do sistema.

Portanto, todos os ganhos obtidos com a diminuição dos tempos de resposta nos chaveamentos de contexto diante de preempções e a diminuição da quantidade de trocas de mensagens entre as aplicações e o sistema operacional proporcionam ao método proposto desta dissertação uma metodologia mais otimizada para integrar características preemptivas a técnica DVFS intra-tarefa abrindo grandes possibilidades de redução do consumo de energia do processador, que serão comprovadas através de experimentações no Capítulo 5.

4.4 Resumo

Este capítulo apresentou uma nova metodologia que visa integrar características preemptivas a técnica DVFS intra-tarefa, através de uma abordagem colaborativa entre as aplicações e o núcleo do sistema operacional, de forma que ambos possam trabalhar em conjunto na redução do consumo de energia do processador. Em outras palavras, o cerne dessa metodologia está em prologar o tempo de utilização das tensões e frequências ideais de cada tarefa que venha a entrar em execução no sistema e reduzir o tempo de chaveamento de contexto do processador diante de preempções, visando assim reduzir o tempo ocioso das tarefas e, consequentemente, o tempo ocioso do processador. Na Seção 4.1 foram apresentadas todas as definições e modelagens que caracterizam a problemática da pesquisa e, além disso, delimitam o contexto de aplicação da metodologia proposta. Na Seção 4.2 foi descrito a modelagem do método proposto e a descrição de todos os novos componentes criados para estabelecer um ambiente de colaboração entre as aplicações em tempo real e o núcleo do sistema operacional. Por fim, na Seção 4.3 foi apresentado o passo-a-passo para aplicação da metodologia proposta por esta dissertação.

Capítulo 5

Resultados Experimentais

O objetivo deste capítulo é verificar, através de experimentações, se o método proposto no Capítulo 4 pode reduzir o consumo de energia do processador. Esta fase da dissertação foi subdividida nas seguintes seções: como o ambiente de experimentação foi concebido, quais foram os *hardwares* utilizados, como se deu a definição dos *Benchmarks* utilizados nas experimentações, como se deu a definição dos estudos de caso para realizar a validação do método proposto e, por fim, a última seção mostrará os resultados obtidos após a realização dos experimentos.

5.1 Ambiente de Experimentação

O ponto de partida para o desenvolvimento do ambiente de experimentação desta dissertação foi todo baseado na arquitetura do *Kernel* do Linux, como já foi citado no Capítulo 4. Esse ambiente foi construído sobre o ***Kernel* 2.6.38.8** *Vanilla* [1]. No entanto, o *Kernel Vanilla* do Linux não possui características de tempo real definidas por padrão. Para torná-lo um ambiente de tempo real foi necessário instalar a ferramenta RTAI (do inglês, *Real Time Application Interface*, ver mais detalhes da ferramenta na Seção 2.6).

Após a fase de configuração do ambiente de tempo real, iniciou-se a implementação da arquitetura do método proposto (ver Figura 4.2). As principais modificações feitas no Linux [2] foram:

- Reservar o primeiro núcleo disponível do processador para uso exclusivo das aplicações de tempo real. Enquanto que os demais núcleos são utilizados para executar as tarefas do sistema operacional e as tarefas do espaço do usuário que

[1] O *Kernel* 2.6.38.8 *Vanilla* está disponível no site "*The Linux Kernel Archives*", através do endereço eletrônico: *https://www.kernel.org/*;

[2] O código-fonte do Linux está disponível em Gonçalves e Barreto (2015a);

não sejam de tempo real. Em outras palavras, essa arquitetura exige no mínimo dois núcleos de processamento. Essa abordagem garante que as tarefas de tempo real executem sem quaisquer interferências externas tanto do sistema operacional (exceto processos que controlam interrupções) quanto de outras aplicações em execução no espaço do usuário.

- Dentro das estruturas de dados das tarefas do sistema foram adicionados as informações definidas no modelo de tarefa (ver Seção 4.1.1) e alguns atributos para melhor gerenciar as trocas de contexto do processador diante de preempções. Dessa forma, tanto o *Governor* quanto o escalonador têm acesso as informações das tarefas de tempo real em execução no sistema.
- Por fim, dentro do módulo CPUFreq (ver Seção 2.5.3) foi criada uma nova política de escalonamento de tensões e frequências chamado "Raw" (conforme descrito na Seção 4.2.2).

Dentro da RTAI [3], as principais modificações foram:

- Adicionar novos atributos na estrutura de dados das tarefas de tempo real do RTAI para se adequar ao modelo de tarefas definidos no método proposto.
- Criar as chamadas de sistema definidas na Seção 4.2.1.
- Por fim, dentro do escalonador do RTAI foi criado um monitor de tarefas, com complexidade $O(1)$ (ou seja, o tempo de computação é uma constante) conforme descrito no Algoritmo 2, para alertar o *Raw Governor* sobre o PID [4] da tarefa que está retornando de preempção, a fim de que ele possa realizar a troca de contexto do processador de acordo a tarefa que entrará em execução. Já as tarefas que não estão retornando de preempção têm suas tensões e frequências definidas durante sua inicialização, ao utilizar a API do *Raw Governor* para passar as informações iniciais, como a tensão e frequência ideal, ao sistema operacional.

No Apêndice A está descrito todo o processo de instalação e configuração desse ambiente, assim como os endereços eletrônicos para o *download* do código-fonte de ambas as ferramentas.

5.2 Configuração do *Hardware*

A configuração do *hardware* foi definida a partir dos equipamentos disponíveis no laboratório do GISE (Grupo de Interesse em Sistemas Embarcados) e pela sua

[3] O código-fonte do RTAI está disponível em Gonçalves e Barreto (2015b);

[4] PID vem do inglês e significa *Process Identifier*. Ela é uma chave única que identifica as tarefas no sistema;

compatibilidade com os requisitos do ambiente de experimentação definidos na Seção 5.1. Portanto, os *hardwares* utilizados, para a realização das experimentações, foram:

- **Processador:** AMD Athlon II X2 250 [5];
- **Placa Mãe:** PCWARE APMCP68 [6];
- **Memória RAM:** 4 Gb
- **HD:** 500 Gb

5.3 Definição dos *Benchmarks* e dos Estudos de Caso

O *benchmark* utilizado durante a realização das experimentações foi o *C-Benchmark*[7] do grupo MRTC (em inglês, *Mälardalen Real Time Research Centre*). Esse *benchmark* foi selecionando a partir dos dados catalogadas na revisão sistemática, onde os critérios de seleção foram: (1) códigos em C ; e (2) alto tempo de computação.

Dentre as várias aplicações disponíveis no *C-Benchmark* foram selecionadas três para compor o estudo de caso utilizado nas experimentações, são elas:

- **CNT:** é uma aplicação que conta os números não negativos dentro de uma matriz;
- **BSORT:** é uma aplicação que implementa a técnica de ordenação *Bubblesort* em matrizes unidimensionais;
- **MATMULT:** é uma aplicação que implementa a multiplicação entre duas matrizes de mesma dimensão.

Essas aplicações foram selecionadas devido à capacidade de terem seus tempos de computação elevados com o aumentar das dimensões de suas matrizes. Isso permitiu criar estudos de caso com utilização do processador acima dos 90%, principalmente devido à escala de operação do processador utilizado no experimento trabalhar na faixa dos *GHz*.

Dessa forma, a definição final dos estudos de caso para análise do método proposto se deu com a utilização de 15 tarefas, baseada nas três aplicações citadas, porém utilizando parâmetros diferentes. A Tabela 5.1 mostra a lista de tarefas e os parâmetros de configuração utilizados como estudo de caso.

Vale fazer algumas ressalvas sobre o estudo de caso proposto:

[5] O *Data Sheet* do processador AMD Athlon II X2 250 está disponível em AMD (2009);
[6] As especificações da PCWARE APMCP68 estão disponíveis em PCWARE (2015);
[7] O *C-Benchmark* está disponível em: http://www.mrtc.mdh.se/projects/wcet/benchmarks.html;

Tabela 5.1: Lista de tarefas que compõem o estudo de caso desta dissertação.

ID	Tipo da Tarefa	WCEC (ciclos)	Período = *Deadline* (segundos)	Prioridade	Frequência Ideal (GHz)	Quantidade Total de Execuções
00	CNT	1421126000	16	04	1.8	45
01	CNT	1421126000	20	06	1.8	36
02	CNT	1421126000	30	12	2.3	24
03	CNT	1421126000	16	02	1.8	45
04	CNT	1421126000	20	08	2.3	36
05	MATMULT	6910262639	30	09	0.8	24
06	MATMULT	6910262639	30	11	1.8	24
07	MATMULT	6910262639	30	13	1.8	24
08	MATMULT	6910262639	30	15	0.8	24
09	BSORT	3000210009	16	03	0.8	45
10	BSORT	3000210009	16	05	2.3	45
11	BSORT	3000210009	30	10	1.8	24
12	BSORT	3000210009	20	07	2.3	36
13	BSORT	3000210009	30	14	2.3	24
14	CPUSTAT	1000	10	01	0.8	–

- As tarefas foram replicadas com a finalidade de aumentar o tempo de utilização do processador, diminuir o tempo em que ele fica ocioso e aumentar o nível de complexidade dos casos de teste dos experimentos.

- A quantidade total de execuções foi calculada com base em 3 hiper períodos (ver Seção 2.2.3), para aumentar o tempo total das experimentações do estudo de caso.

- A tarefa 14, chamada de "CPUSTAT", foi criada para controlar o número de execuções de cada uma das tarefas, coletar dados estatísticos do processador, finalizar a realização do experimento e gerar o relatório final da experimentação. O Apêndice F mostra exemplos dos relatórios gerados pelo CPUSTAT;

- As prioridades das tarefas foram definidas conforme a política de escalonamento do *Rate-Monotonic* (ver Seção 2.2.3), ou seja, as tarefas de menor período têm a maior prioridade, que neste caso quanto menor o número atribuído maior é a prioridade da tarefa;

- Os valores das tensões foram desconsiderados, pois o processador utilizado nas experimentações (ver Tabela 2.1) possui relação única entre tensões e frequências. Em outras palavras, o processador ajusta automaticamente os valores de tensão para se alcançar uma determinada frequência, procurando sempre aplicar os menores níveis de forma que haja minimização do consumo de energia.

- As frequências ideais foram definidas assumindo como base os dados obtidos pela ferramenta *Smartenum*. No entanto, executando as tarefas de tempo real com as frequências calculadas na ferramenta, a utilização do processador foi de $79,36\%$ (ver Código D.1). Portanto, para gerar um estudo de caso mais crítico, foram feitas alterações manuais nas frequências de algumas tarefas, com o auxílio do *script* de teste de escalonabilidade desenvolvida no MatLab (ver Apêndice D), com o intuito de alcançar maiores utilizações do processador e gerar mais preempções entre as

tarefas. Dessa forma, foi possível alcançar uma utilização do processador de cerca de $96,82\%$ respeitando as premissas temporais de todas as tarefas (ver Código D.3). Em outras palavras, o *overhead* do sistema operacional deve ser mínimo (tanto do *Raw Governor* quanto do *Raw Monitor*), na casa dos microssegundos, caso contrário as premissas temporais das tarefas serão violadas.

O passo seguinte foi implementar as tarefas dentro do ambiente de experimentação (proposto na Seção 5.1), utilizando a linguagem C, conforme os parâmetros definidos na Tabela 5.1. A partir do estudo de caso definido na Tabela 5.1 foram criados 41 casos de teste para realizar a validação do método proposto. A Tabela 5.2 mostra cada um dos casos de teste criados com o objetivo de avaliar o desempenho do *Raw Governor* através da ativação e desativação do serviço de monitoramento. Além disso, os casos de teste visam mostrar um comparativo entre o *Raw Governor* e o gerenciamento de contexto do processador feito no espaço do usuário, através do uso de VSPs estáticas dentro dos códigos fonte das tarefas. Vale observar que o caso de teste sem *Raw Monitor* e sem VSP não foi criado, pois uma aplicação não conseguiria realizar o chaveamento e validação do contexto do processador sem a utilização de nenhuma dessas técnicas.

Os casos de teste foram criados até o momento que houvesse uma convergência e estabilização do consumo de energia das trocas de contexto e tratamentos de preempção feitas no espaço do usuário. Para realizar a validação e o chaveamento de contexto do processador no espaço do usuário, foram feitas algumas modificações na política de inserção de VSPs baseadas no trabalho de Yi *et al.* (2005) (ver exemplo na Figura 4.1 e considerar VSPs equivalente a CKs). As modificações foram:

- A quantidade de VSPs foi limitada de acordo com o caso de teste definido na Tabela 5.2. Essa inserção parcial de VSPs proporcionam dados que permitem fazer uma melhor análise qualitativa do método proposto, pois é possível ter uma visão melhor do desempenho entre as trocas de contexto feitas no espaço do usuário e núcleo do sistema, a medida que são adicionado novas VSPs;
- A política de distribuição de VSPs ocorreu da seguinte forma: (1) o código fonte da tarefa de tempo real foi dividido em blocos de código de igual tamanho (aproximadamente a mesma quantidade de ciclos), de acordo com a quantidade de VSPs definidas no caso de teste; por fim, (2) as VSPs são inseridas entre esses blocos, seguindo a política de Yi *et al.* (2005). A Figura 5.1 mostra um exemplo do processo de inserção de 3 VSPs dentro do código fonte de uma aplicação;
- Dentre todas as tarefas que compõem o estudo de caso mostrado na Tabela 5.1, apenas a tarefa "CPUSTAT" não receberá VSPs, pois ela não sofre interferências das demais tarefas em execução no sistema, por ser a tarefa de maior prioridade.

Tabela 5.2: Lista dos 41 casos de teste que compõem o processo de validação do método proposto (parte 1).

ID do Experimento	***Raw Monitor* Habilitado?**	***VSPs* Habilitadas?**	**Quantidade de VSP(s)**
01	Sim	Não	00
02	Sim	Sim	01
03	Não	Sim	01
04	Sim	Sim	02
05	Não	Sim	02
06	Sim	Sim	03
07	Não	Sim	03
08	Sim	Sim	04
09	Não	Sim	04
10	Sim	Sim	05
11	Não	Sim	05
12	Sim	Sim	06
13	Não	Sim	06
14	Sim	Sim	07
15	Não	Sim	07
16	Sim	Sim	08
17	Não	Sim	08
18	Sim	Sim	09
19	Não	Sim	09
20	Sim	Sim	10
21	Não	Sim	10
22	Sim	Sim	11
23	Não	Sim	11
24	Sim	Sim	12
25	Não	Sim	12
26	Sim	Sim	13
27	Não	Sim	13
28	Sim	Sim	14
29	Não	Sim	14
30	Sim	Sim	15
31	Não	Sim	15
32	Sim	Sim	16
33	Não	Sim	16
34	Sim	Sim	17
35	Não	Sim	17
36	Sim	Sim	18
37	Não	Sim	18
38	Sim	Sim	19
39	Não	Sim	19
40	Sim	Sim	20
41	Não	Sim	20

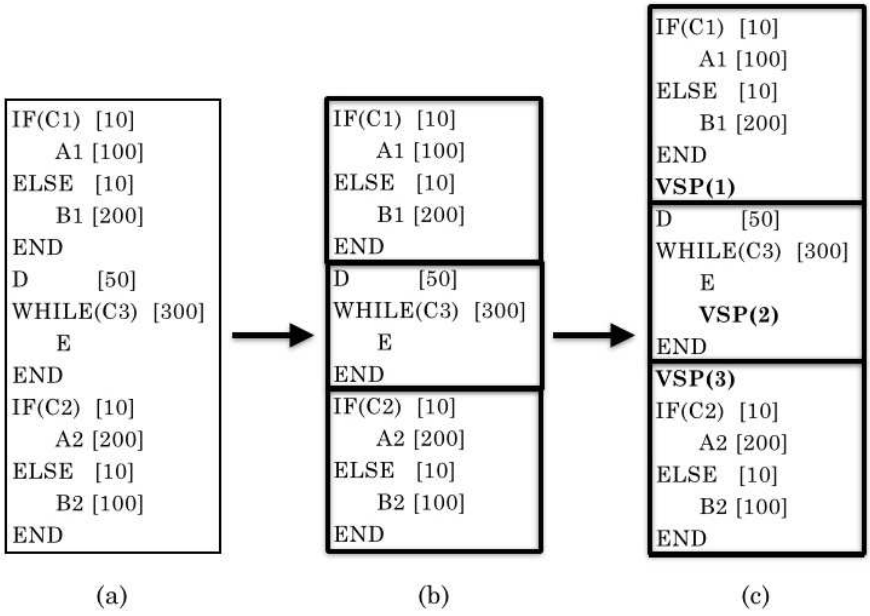

Figura 5.1: Exemplo do processo de inserção de 3 VSPs no código fonte da aplicação, onde: (a) mostra o código fonte original; (b) mostra o processo de divisão dos blocos de código da aplicação; e (c) mostra a inserção das VSPs em cada bloco.

5.4 Resultados Experimentais

Os resultados experimentais dos 41 casos de teste, mostrados na Tabela 5.2, levaram mais de 8 horas ininterruptas para serem obtidos. Os resultados do consumo de energia e do tempo ocioso do processador de cada caso de teste executado no ambiente de experimentação são mostrados na Tabela 5.3. Vale ressaltar que o consumo de energia foi baseado no modelo definido na Subseção 4.1.3, o tempo ocioso do processador foi mensurado utilizando as estatísticas fornecidas pelo módulo CPUFreq do Linux e nenhum dos casos de teste executados tiveram suas restrições temporais violadas. O Apêndice E mostra o *script* utilizado para realizar a computação dos dados de cada caso de teste.

A Figura 5.2 mostra o gráfico do consumo de energia do processador em função do número de VSPs com e sem a atuação do *Raw Monitor*, baseado nos dados da Tabela 5.3. Analisando o gráfico em vermelho é possível constatar que os chaveamentos de contexto do processador feitas pela *Raw Governor* em conjunto com o *Raw Monitor* e mais eficiente energeticamente em comparação com o gráfico azul, independente da quantidade de VSPs inseridas dentro das aplicações em tempo real.

O comportamento do gráfico azul foi estabilizando o consumo de energia a medida que foram sendo inseridos os VSPs, pois a aplicação passou a ter mais pontos de validação e de trocas de tensões e frequências do processador, resultando em um melhor gerenciamento.

Tabela 5.3: Mostra o consumo de energia e o tempo ocioso do processador para cada um dos 41 casos de teste (dados ordenados pelas colunas 2 e 4 em destaque).

ID do Experimento	*Raw Monitor* Habilitado?	*VSPs* Habilitadas?	Quantidade de VSP(s)	Consumo de Energia (kJ)	Tempo Ocioso Processador (segundos)
03	Não	Sim	01	**8,096445**	**207,4243**
05	Não	Sim	02	**8,064078**	**209,4523**
07	Não	Sim	03	**8,026955**	**206,9121**
09	Não	Sim	04	**8,033695**	**207,6989**
11	Não	Sim	05	**8,061638**	**209,4618**
13	Não	Sim	06	**8,061533**	**209,5324**
15	Não	Sim	07	**7,675999**	**208,9555**
17	Não	Sim	08	**7,694207**	**209,5575**
19	Não	Sim	09	**7,675938**	**208,9078**
21	Não	Sim	10	**7,688361**	**209,3664**
23	Não	Sim	11	**7,687015**	**209,4337**
25	Não	Sim	12	**7,669599**	**208,6862**
27	Não	Sim	13	**7,671919**	**208,4204**
29	Não	Sim	14	**7,659584**	**205,9666**
31	Não	Sim	15	**7,644148**	**207,4589**
33	Não	Sim	16	**7,644476**	**207,3444**
35	Não	Sim	17	**7,648635**	**206,9822**
37	Não	Sim	18	**7,648654**	**206,7894**
39	Não	Sim	19	**7,638374**	**207,2033**
41	Não	Sim	20	**7,643660**	**206,5481**
01	Sim	Não	00	**7,185524**	**57,4464**
02	Sim	Sim	01	**7,633486**	**206,6439**
04	Sim	Sim	02	**7,630280**	**206,8330**
06	Sim	Sim	03	**7,631821**	**206,3541**
08	Sim	Sim	04	**7,631038**	**206,3243**
10	Sim	Sim	05	**7,630637**	**206,6576**
12	Sim	Sim	06	**7,630734**	**206,5188**
14	Sim	Sim	07	**7,632553**	**206,1646**
16	Sim	Sim	08	**7,629486**	**206,5492**
18	Sim	Sim	09	**7,631695**	**206,2064**
20	Sim	Sim	10	**7,638358**	**204,0086**
22	Sim	Sim	11	**7,629542**	**206,6066**
24	Sim	Sim	12	**7,630398**	**206,4521**
26	Sim	Sim	13	**7,631478**	**206,1358**
28	Sim	Sim	14	**7,631996**	**205,9906**
30	Sim	Sim	15	**7,630448**	**206,4027**
32	Sim	Sim	16	**7,631415**	**205,9713**
34	Sim	Sim	17	**7,632294**	**205,8428**
36	Sim	Sim	18	**7,632241**	**205,9199**
38	Sim	Sim	19	**7,633033**	**205,7040**
40	Sim	Sim	20	**7,632292**	**205,9256**

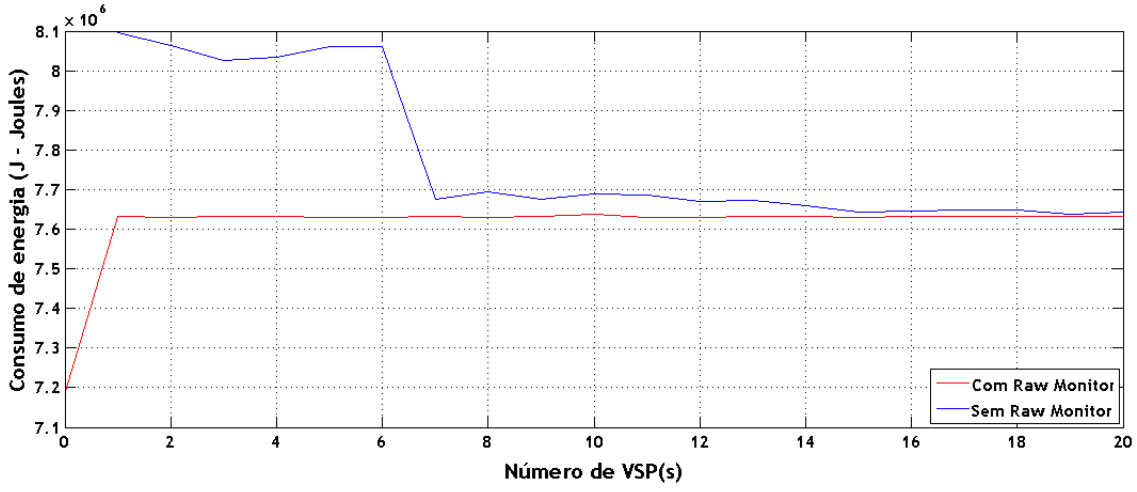

Figura 5.2: O gráfico mostra o consumo de energia dinâmica do processador em função do número de VSPs com e sem a atuação do *Raw Monitor*.

Já o comportamento do gráfico vermelho foi crescendo devida a atuação da técnica VSP em conjunto com o *Raw Governor*, ainda sim é possível constatar que a atuação do *Raw Governor* em conjunto com o *Raw Monitor* foi mais eficiente energeticamente que utilizando somente as VSPs, mostrado no gráfico em azul.

No entanto, a comparação de desempenho energético deve ser feita entre o estudo de caso $ID = 0$ (usando somente o *Raw Monitor*), onde o consumo de energia foi de $7,185524\,kJ$, com o ponto de estabilização do consumo de energia da função sem o *Raw Monitor*, que foi cerca de $7,632292\,kJ$. É necessário aguardar a estabilização do consumo de energia sem o *Raw Monitor*, pois as VSPs foram inseridas gradativamente no código fonte da aplicação, justamente para analisar o seu comportamento a medida que as VSPs são inseridas. A Figura 5.3 mostra o comparativo entre o consumo de energia final do processador com e sem a atuação do *Raw Monitor*, onde houve uma redução de cerca de 6% sobre o consumo de energia dinâmica do processador.

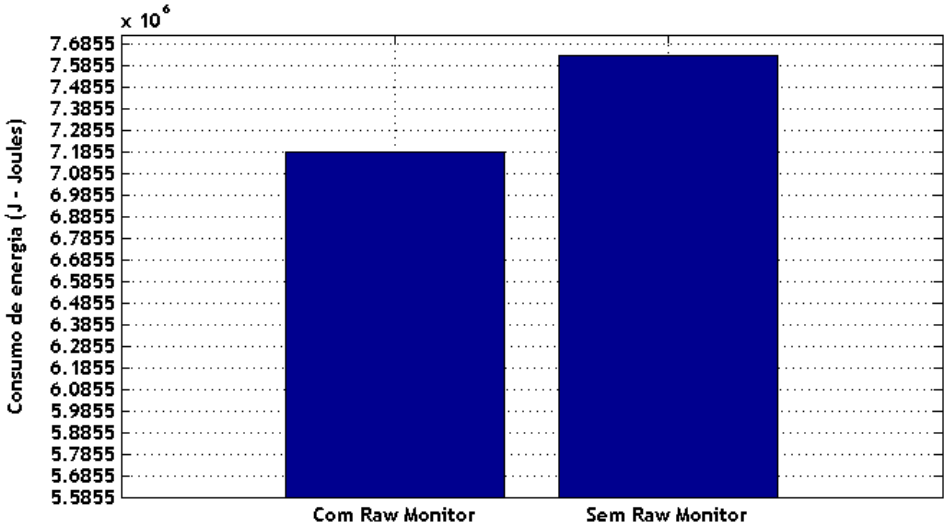

Figura 5.3: O gráfico em barras mostra o consumo de energia final do processador com e sem a atuação do *Raw Monitor*.

Essa redução no consumo de energia é bastante factível quando comparamos o tempo de resposta do chaveamento de tensões e frequências do processador entre o espaço do usuário e o núcleo do sistema. A Figura 5.4 mostra o resultado dessa comparação, onde os dados foram coletados da seguinte forma: (1) primeiramente, foi alterada uma aplicação em tempo real para calcular o tempo total que leva desde a chamada de sistema até o processador assumir a nova tensão e frequência informada pela aplicação; (2) o segundo passo consistiu em gerar os logs desses tempos; (3) em seguida, foram feitos os mesmos passos 1 e 2 dentro do *Raw Governor*; e por fim, (4) foram tiradas as médias de dez amostras, tanto da aplicação quanto do *Raw Governor*, e em seguida os dados foram comparados.

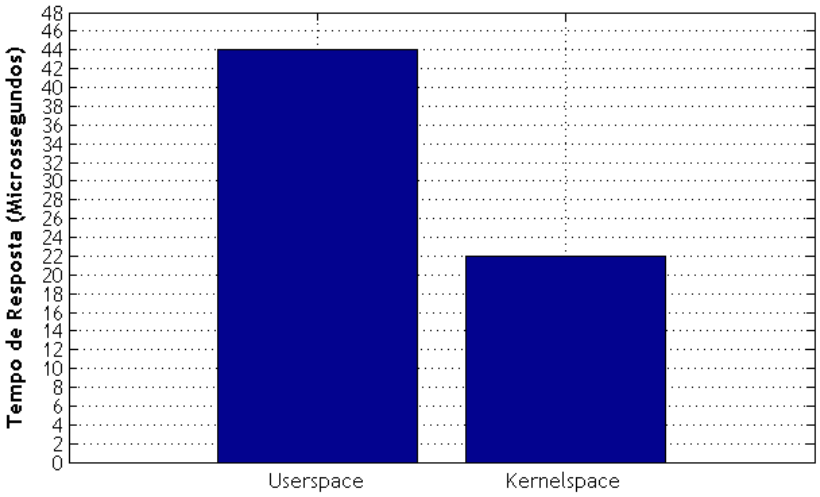

Figura 5.4: O gráfico mostra a comparação entre o tempo de resposta do chaveamento de contexto do processador entre o espaço do usuário e o núcleo do sistema.

Já a Figura 5.5 mostra o gráfico do tempo ocioso do processador em função do número de VSPs com e sem a atuação do *Raw Monitor*, também baseado nos dados da Tabela 5.3.

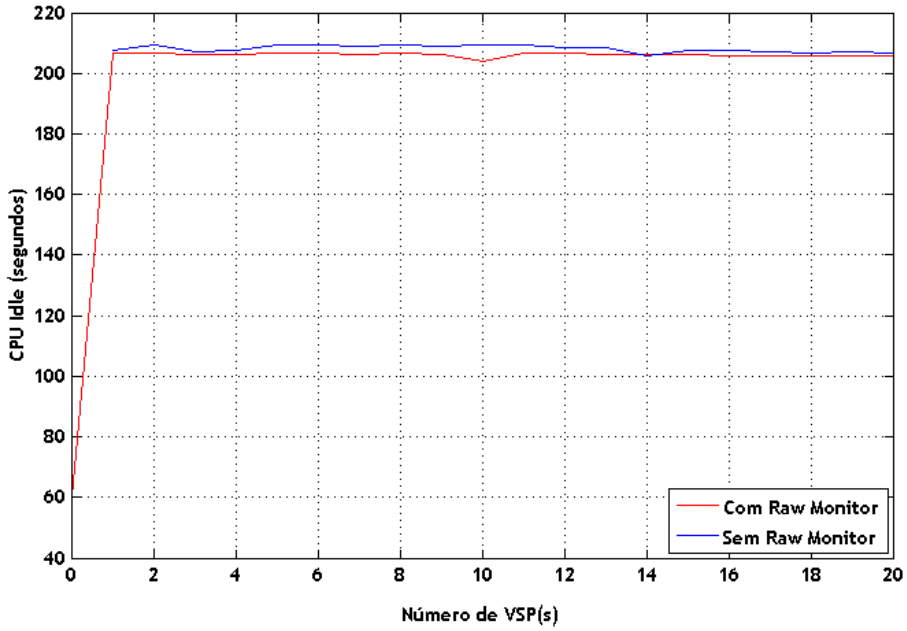

Figura 5.5: O gráfico mostra o tempo ocioso do processador em função do número de VSPs com e sem a atuação do *Raw Monitor*.

Analisando a Figura 5.5 é possível verificar que houve uma redução do tempo ocioso do processador com o uso do *Raw Monitor*. As trocas de contexto mais eficiente do processador, principalmente mediante preempções, permitiu que as tarefas utilizassem suas respectivas tensões e frequências ideais por maior período de tempo, reduzindo assim

o tempo ocioso das tarefas e, consequentemente, o tempo ocioso do processador, como foi exemplificado na Figura 2.1.

Nesta segunda análise, a comparação do tempo ocioso do processador também deve ser feita entre o caso de teste $ID = 0$ (usando somente o *Raw Monitor*), onde o tempo ocioso foi de $57,4464$ segundos, com o ponto de estabilização do tempo ocioso da função sem o *Raw Monitor*, que foi cerca de $205,9666$ segundos. A Figura 5.6 mostra o comparativo entre o tempo ocioso final do processador com e sem a atuação do *Raw Monitor*, onde houve uma redução de cerca de 72%.

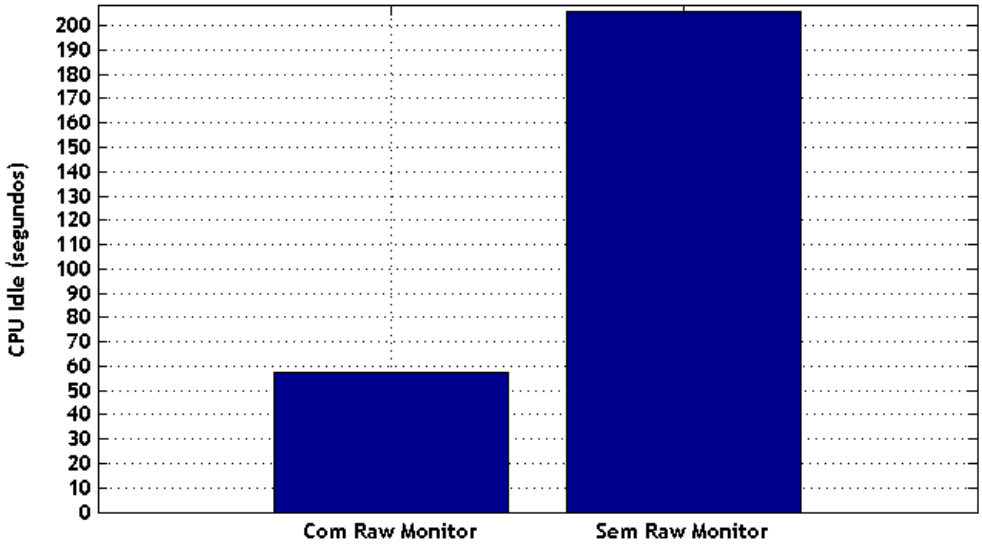

Figura 5.6: O gráfico em barras mostra o tempo ocioso final do processador com e sem a atuação do *Raw Monitor*.

Em resumo, foram reduzidos, ao logo de 8 horas de experimentações, cerca de 148 segundos do tempo ocioso do processador, o que resultou em uma economia de 6% do consumo de energia. Portanto, os resultados obtidos experimentalmente fornecem dados que comprovam a eficácia do método proposto no Capítulo 4, mesmo executando todas as tarefas em tempo real no pior caso, e, além disso, garantem que todos os objetivos específicos (ver Seção 1.4) foram alcançados com êxito.

5.5 Resumo

O objetivo deste capítulo foi aplicar o método proposto na prática, através de experimentações reais dentro de um sistema operacional de tempo real. As experimentações utilizaram o *C-Benchmark*, do grupo *Mälardalen Real Time Research Centre*, tais como: CNT, BSORT e MATMULT. Eles foram selecionados com o intuito de gerarem experimentações com altas cargas de processamento capazes de utilizar

quaisquer frequências disponíveis pelo processador (ver Tabela 2.1). Na Seção 5.1 foram descritas as principais modificações feitas no sistema operacional para transformar o *Kernel* do Linux em um ambiente capaz de executar aplicações de tempo real. Na Seção 5.2 foram apresentadas as configurações do *hardware* utilizado nas experimentações. Na Seção 5.3 foram definidos os *benchmarks* e os estudos de casos utilizados nas experimentações para comprovar a eficácia do método proposto por esta dissertação. Vale observar que todos os estudos de casos foram criados para executar no pior caso, sempre levando em consideração múltiplas tarefas preemptivas, justamente para criar um ambiente de experimentação crítico e complexo para validação do método proposto. Por fim, na Seção 5.4 foram apresentados os resultados obtidos durante as experimentações, onde foi possível verificar que mesmo executando todas as tarefas no pior caso houve uma redução no tempo ocioso do processador e uma redução do consumo de energia em cerca de 6%. Isso mostra que uma abordagem colaborativa entre as aplicações e o núcleo do sistema operacional permite gerenciar melhor as trocas de contexto do processador, principalmente diante de preempções, de acordo com a tarefa que esteja em execução no sistema.

Capítulo 6

Conclusões

Um dos principais desafios no desenvolvimento de aplicações de tempo real é a otimização do consumo de energia do sistema e, ao mesmo tempo, garantindo que as restrições temporais de todas as tarefas sejam respeitadas. Uma das técnicas mais utilizadas dentro deste contexto é o escalonamento dinâmico de tensões e frequências intra-tarefa, no entanto ela tem dificuldades para realizar esse escalonamento diante de preempções. Isso ocorre devido o fato dessa técnica ser implementada dentro das aplicações de tempo real e mesmo com o auxílio do sistema operacional elas não têm como prever de forma eficiente se uma determinada tarefa foi ou não preemptada e nem o tempo de duração da preempção.

Portanto, este trabalho apresentou uma abordagem colaborativa entre as aplicações de tempo real e o sistema operacional com o intuito de realizar o melhor gerenciamento das trocas de contexto do processador com suporte ao tratamento de preempções. Dessa forma, ambos podem trabalhar em conjunto para minimizar o consumo de energia do processador. A principal vantagem dessa metodologia é a realização do tratamento de preempção das tarefas em tempo real no núcleo do sistema de forma dinâmica e reconfigurável de acordo com a tarefa em execução, graças ao monitor inserido dentro do escalonador do sistema. Além disso, essa metodologia permite diminuir o número de trocas de mensagens entre o espaço do usuário e o núcleo do sistema (ver Figura 4.5), otimizando o tempo de resposta para realizar o tratamento de preempções.

Sendo assim, o método proposto por esta dissertação teve o intuito de otimizar o gerenciamento das trocas de contexto do processador, principalmente diante de preempções, a fim de reduzir a quantidade de códigos estáticos inseridos dentro do código fonte das tarefas de tempo real para realizar as validações das tensões e frequências a serem utilizadas pelo processador. Essas otimizações proporcionam uma diminuição dos tempos de folga das tarefas e, consequentemente, uma minimização do consumo de energia dinâmica do processador.

As seções a seguir irão abordar de forma mais pontual as reais contribuições desta dissertação (ver Seção 6.1), assim como suas limitações (ver Seção 6.2) e, por fim, algumas propostas de trabalhos futuros (ver Seção 6.3).

6.1 Contribuições

As principais contribuições desta dissertação, em relação ao estado da arte mapeado, são:

- Uma nova metodologia para dar suporte ao tratamento de preempção à técnica DVFS intra-tarefa.
- Um novo canal de comunicação entre as aplicações de tempo real e o controlador de tensões e frequências do processador do sistema operacional, de forma que ambos possam trocar informações em tempo de execução.
- A diminuição do tempo de resposta do chaveamento de contexto do processador, principalmente diante de preempções.
- A diminuição do *overheads* estáticos inseridos no código fonte da aplicação em tempo real para realizar a validação de tensões e frequências do processador, diante de preempções, visto que este processo passa a ser feito no núcleo do sistema operacional de forma *online* e não mais de forma estática através do uso das VSPs.
- A diminuição do consumo de energia do processador, após a realização dos 41 casos de teste executando sempre no pior caso. Essa diminuição do consumo ocorre devido a redução dos *overheads* estáticos e da redução dos tempos de folga das tarefas, pois elas passam a executar por mais tempo com as tensões e frequências ideais, como são mostrados nas Figuras 5.4 e 5.5.
- E por fim, o ambiente de experimentação feito para validação do método proposto, pois é uma nova plataforma que pode ser utilizadas pela comunidade científica para o desenvolvimento de novas técnicas que venham a contribuir com o estado da arte na área de baixo consumo de energia. Todo o ambiente é de código livre e está disponível para *download*, ver Apêndice A.

Vale ressaltar que o suporte a preempções, a diminuição do tempo de resposta do chaveamento de contexto do processador, o processo de validação das tensões e frequências de acordo com a tarefa em execução no sistema e a redução do tempo de folga das tarefas são contribuições alcançadas graças aos novos componentes adicionados ao núcleo do sistema operacional.

6.2 Pontos Limitantes

Apesar dos bons resultados obtidos experimentalmente, abaixo segue uma lista de limitações do presente trabalho:

- O ideal seria implementar o gerenciador de troca de contexto do processador dentro do escalonador do sistema operacional, pois o tempo de resposta seria menor ainda, principalmente por ele ser o responsável por realizar o tratamento de preempções, evitando o uso de intermediários com o *Raw Governor*. No entanto, essa proposta teria que alterar toda a arquitetura do Linux e seria necessário retirar completamente o módulo CPUFreq do sistema, responsável por realizar as trocas de contexto do processador. Isso demandaria muito mais esforço, tempo e conhecimentos avançados de Linux para que todas essas modificações fossem concluídas.

- Algumas experimentações realizadas com cerca de 50 tarefas de tempo real, com WCEC na casa do bilhões de ciclos executando no pior caso, fizeram o sistema operacional travar e as causas do travamento não ficaram claras no processo de *debug* do sistema utilizando as mensagens impressas no arquivo *dmesg*, principalmente por ser um procedimento complexo e que envolvem muitas variáveis do sistema operacional.

- O método proposto não levou em consideração o compartilhamento de recursos entre tarefas, pois necessitaria a integração de técnicas inter-tarefas, que aumentariam a complexidade dos experimentos e das ferramentas responsáveis por realizar o teste de escalonabilidade das tarefas, pois teriam que levar em consideração as novas interferências e relações de precedência entre tarefas e recursos.

- A metodologia também não leva em consideração a inversão de prioridades entre as tarefas de tempo real em execução no sistema, pois demandariam mais *overheads* estáticos dentro das aplicações de tempo real. Uma possível solução para este item seria utilizar a técnica *Priority Ceiling Protocol* (PCP).

- O cálculo do consumo de energia do processador feito através do modelo definido na Subseção 4.1.3, poderia ter sido comparado com o seu consumo de energia real através do uso de equipamentos específicos ligados diretamente ao processador, que tornaria os dados experimentais muito mais precisos. No entanto, muitos desses equipamentos são específicos do fabricante do processador e possuem um alto custo de aquisição.

6.3 Trabalhos Futuros

O método proposto nesta dissertação abre novas linhas de pesquisa, como:

- Trocar a política de escalonamento utilizada no método proposto de *Rate-Monotonic* para EDF (do inglês, *Earliest Deadline First*) e verificar como ele irá se comportar com essa nova política.
- Criar uma metodologia para avaliar os impactos gerados por sucessivas trocas de contexto do processador e qual seria o impacto na sua temperatura interna. Visto que o aumento dos chaveamentos de contexto do processador resulta, diretamente, no maior consumo de energia estática (ver Seção 4.1.3) e, consequentemente, uma maior liberação de calor aumentando a temperatura interna do *chip*.
- Aplicar e integrar técnicas inter-tarefas ao método proposto a fim de maximizar os ganhos energéticos do processador.
- Criar um compilador que integre de forma automatizada as aplicações em tempo real ao *Raw Governor*. Assim, os programadores não precisam se preocupar com os cálculos das tensões e frequências ideais e nem o tratamento de preempções.

Referências Bibliográficas

AbouGhazaleh N., Childers B., Mosse D., Melhem R. e Craven M. Energy management for real-time embedded applications with compiler support. In *ACM SIGPLAN Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, pages 284–293. ACM, 2003a.

AbouGhazaleh N., Mossé D., Childers B., Melhem R. e Craven M. Collaborative operating system and compiler power management for real-time applications. In *9th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2003*, number 1203045, pages 133–141. IEEE, 2003b.

ACM. ACM Digital Library. Disponível em http://dl.acm.org/, 2013. Acessado em 29 de Dezembro.

Alipour M., Moshari K. e Bagheri M. Performance per power optimum cache architecture for embedded applications, a design space exploration. In *Networked Embedded Systems for Enterprise Applications (NESEA), 2011 IEEE 2nd International Conference on*, pages 1–6, 2011.

AMD. *AMD Family 10h Desktop Processor Power and Thermal Data Sheet*. Advanced Micro Devices (AMD), Inc., 2008-2009., June 2009.

Awan M. A. e Petters S. M. Online intra-task device scheduling for hard real-time systems. In *7th IEEE International Symposium on Industrial Embedded Systems, SIES 2012*, number 6356569, pages 48–56. IEEE, 2012.

Baums A. e Zaznova N. Power optimization of embedded real-time systems and their adaptability. In *Automatic Control and Computer Sciences*, pages 59–73. Springer, 2008.

Bergsma M. Extending rtai linux with fpds. Disponível em http://alexandria.tue.nl/extra2/afstversl/wsk-i/bergsma2009.pdf, July 2009.

Bertolotti I. C. e Manduchi G. *Real-Time Embedded Systems: Open-Source Operating Systems Perspective*. Taylor and Francis Group, 2012.

Biolchini J., Mian P. G. e Natali A. C. C. Systematic Review in Software Engineering. Technical Report RT-ES 679/05, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, May 2005.

Brodowski D. e Golde N. Cpu frequency and voltage scaling code in the linux(tm) kernel. http://www.kernel.org/doc/documentation/cpu-freq/governors.txt, February 2013.

Bucher R., Dozio L., Gasperini D., Mayer H., Mantegazza P., Masarati P., Neuhauser M., Racciu G., Schleef D. e Soetens P. Rtai: a beginner's guide. https://www.rtai.org/?Documentation, February 2014.

Buss M., Givargis T. e Dutt N. Exploring efficient operating points for voltage scaled embedded processor cores. In *24th IEEE International Real-Time Systems Symposium RTSS 2003*, pages 275–281. IEEE, 2003.

Chen D.-R., Hsie S.-M. e Lai M.-F. Efficient algorithms for jitterless real-time tasks to dvs schedules. In *Parallel and Distributed Computing, Applications and Technologies, PDCAT Proceedings*, number 4710997, pages 319–322. IEEE, 2008a.

Chen D.-R., Hsieh S.-M. e Lai M.-F. Efficient algorithms for periodic real-time tasks to optimal discrete voltage schedules. In *IPDPS Miami 2008 - Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium, Program and CD-ROM*, number 4536543. IEEE, 2008b.

Chen J.-J. Expected energy consumption minimization in dvs systems with discrete frequencies. In *ACM Symposium on Applied Computing*, pages 1720–1725, 2008.

Cheng A. M. K. *Real-Time Systems Scheduling, Analysis, and Verification.* ISBN 0-471-18406-3. John Wiley and Sons, Inc., Hoboken, New Jersey., 2002.

Cohen D., Valentin E., Barreto R., Oliveira H. e Cordeiro L. A car racing based strategy for the dynamic voltage and frequency scaling technique. In *IEEE International Symposium on Industrial Electronics*, number 6237186, pages 774–779. Oxford Journals, 2012.

Elsevier. Digital Library ScienceDirect. Disponível em http://www.sciencedirect.com/, 2013a. Acessado em 29 de Dezembro.

Elsevier. What does Scopus cover? Disponível em http://www.info.sciverse.com/scopus/scopus-in-detail/facts, 2013b. Acessado em 29 de Dezembro.

Gheorghita S. V., Basten T. e Corporaal H. Intra-task scenario-aware voltage scheduling. In *CASES 2005: International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 177–184. ACM, 2005.

Gonçalves R. e Barreto R. Linux with raw governor integrated. https://github.com/nosnilwar/linux-raw-gov-2.6.38.8.git, 2015a.

Gonçalves R. e Barreto R. Rtai with raw governor integrated. https://github.com/nosnilwar/rtai-raw-gov-3.9.1.git, 2015b.

Gonçalves R. e Barreto R. Caracterização do estado da arte sobre as metodologias que utilizam como base a técnica dvfs intra-tarefa. Cornell University Library (ARXIV), 2014.

Hartkopp O., Thuermann U., Kizka J., Grandegger W., Schwebel R., Kleine-Budde M. e Spranger B. How to use socketcan. http://www.kernel.org/doc/documentation/networking/can.txt, July 2015.

He X., Jia Y. e Wa H. Stochastic voltage scheduling of fixed-priority tasks with preemption thresholds. In *2008 International Conference on Wireless Communications, Networking and Mobile Computing, WiCOM 2008*, number 4679276. IEEE, 2008.

Hong S., Yoo S., Jin H., Choi K.-M., Kong J.-T. e Eo S.-K. Runtime dvfs control with instrumented code in power-scalable cluster systemme distribution-aware dynamic voltage scaling. In *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, number 4110236, pages 587–587. IEEE, 2006.

Huizhan Y., Juan C. e Xuejun Y. Static wcet analysis based compiler-directed dvs energy optimization in real-time applications. In *11th Asia-Pacific Conference on Advances in Computer Systems Architecture, ACSAC 2006;*, volume 4186 LNCS, pages 123–136. Springer, 2006.

IEEE. IEEE Xplore Digital Library. Disponível em http://ieeexplore.ieee.org, 2013. Acessado em 29 de Dezembro.

Ishihara T. Real-time power management for a multi-performance processor. In *International SoC Design Conference, ISOCC 2009*, number 5423892, pages 147–152. IEEE, 2009.

Kang S.-M. e Leblebici Y. *CMOS Digital lntegrated Circuits Analysis and Design.* McGraw-Hill, 3rd edition, 2002.

Kim W., Gupta M. S., yeon Wei G. e Brooks D. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *in International Symposium on High-Performance Computer Architecture*, pages 123–134. IEEE, 2008.

Kitchenham B. e Charters S. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 2007.

Kitchenham B. A., Dyba T. e Jorgensen M. Evidence-Based Software Engineering. In *Proceedings of the 26th International Conference on Software Engineering*, pages 273–281. IEEE, 2004.

Kumar G. S. A. e Manimaran G. An intra-task dvs algorithm exploiting program path locality for real-time embedded systems. In *12th International Conference on High Performance Computing, HiPC 2005*, volume 3769 LNCS, pages 225–234. Springer, 2005.

Lee I., Leung J. Y.-T. e Son S. H. *Handbook of Real-Time and Embedded Systems*. ISBN-10: 1-58488-678-1. Chapman and Hall; CRC Taylor and Francis Group, 2008.

Lee S., Yoo S. e Choi K. An intra-task dynamic voltage scaling method for soc design with hierarchical fsm and synchronous dataflow model. In *Proceedings of the International Symposium on Low Power Electronics and Design, Digest of Technical Papers - ISLPED 02*, pages 84–87. ACM and IEEE, 2002.

Lee Y.-H. e Krishna C. Voltage-clock scaling for low energy consumption in real-time embedded systems. In *Real-Time Computing Systems and Applications, 1999. RTCSA '99. Sixth International Conference on*, pages 272 –279. IEEE, 1999.

Love R. *Linux Kernel Development*. Pearson Education, Inc., 3rd edition, 2010.

Mafra S. N. e Travassos G. H. Estudos Primários e Secundários Apoiando a Busca por Evidência em Engenharia de Software. Disponível em www.cos.ufrj.br/uploadfiles/1149103120.pdf. Technical Report RT-ES 679/05, PESC - COPPP/UFRJ, Rio de Janeiro, RJ, Brasil, 2006.

Mohan S., Mueller F., Hawkins W., Root M., Healy C. e Whalley D. Parascale: Exploiting parametric timing analysis for real-time schedulers and dynamic voltage scaling. In *Real-Time Systems Symposium*, number 1563111. IEEE, 2005.

Mohan S., Mueller F., Root M., Hawkins W., Healy C., Whalley D. e Vivancos E. Parametric timing analysis and its application to dynamic voltage scaling. In *Transactions on Embedded Computing Systems*, volume 10. ACM, 2010.

Neishaburi M., Daneshtalab M., Nabi M. e Mohammadi S. System level voltage scheduling technique using uml-rt model. In *2007 IEEE/ACS International Conference on Computer Systems and Applications, AICCSA 2007*, number 4231003, pages 500–505. IEEE, 2007.

Nogueira P. E., Matias, Jr. R. e Vicente E. An experimental study on execution time variation in computer experiments. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 1529–1534, 2014.

Oh S., Kim J., Kim S. e Kyung C.-M. Task partitioning algorithm for intra-task dynamic voltage scaling. In *IEEE International Symposium on Circuits and Systems*, number 4541646, pages 1228–1231. IEEE, 2008.

PCWARE. Especificação técnica do modelo apmcp68, 2015.

Pillai P. e Shin K. G. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 89–102. ACM, 2001.

Quinõnes E., Abella J., Cazorla F. J. e Valero M. Exploiting intra-task slack time of load operations for dvfs in hard real-time multi-core systems. In *ACM SIGBED Review - Work-in-Progress (WiP) Session of the 23rd Euromicro Conference on Real-Time Systems (ECRTS 2011)*, volume 8, pages 32–35. ACM, 2011.

Santos G. Ambientes de engenharia de software orientados à corporação. In *Tese (Doutorado em Ciências em Engenharia de Sistemas e Computação) – Universidade Federal do Rio de Janeiro*, 2008.

Seo H., Seo J. e Kim T. Algorithms for combined inter- and intra-task dynamic voltage scaling. In *Computer Journal*, volume 55, pages 1367–1382. IEEE, 2012.

Seo J., Kim T. e Chung K.-S. Profile-based optimal intra-task voltage scheduling for hard real-time applications. In *DAC '04 Proceedings of the 41st annual Design Automation Conference*, pages 87–92. ACM and IEEE, 2004.

Seo J., Kim T. e Dutt N. D. Optimal integration of inter-task and intra-task dynamic voltage scaling techniques for hard real-time applications. In *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, volume 2005, pages 449–454. IEEE, 2005.

Shin D. e Kim J. A profile-based energy-efficient intra-task voltage scheduling algorithm for hard real-time applications. In *International Symposium on Low Electronics and Design (ISLPED'01)*, pages 271–274. ACM and IEEE, 2001.

Shin D. e Kim J. Intra-task voltage scheduling on dvs-enabled hard real-time systems. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 24, pages 1530–1549. IEEE, 2005a.

Shin D. e Kim J. Optimizing intra-task voltage scheduling using data flow analysis. In *Asia and South Pacific Design Automation Conference, ASP-DAC*, volume 2, pages 703–708. IEEE, 2005b.

Shin D. e Kim J. Optimizing intratask voltage scheduling using profile and data-flow information. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 26, pages 369–385. IEEE, 2007.

Shin D., Kim J. e Lee S. Intra-task voltage scheduling for low-energy hard real-time applications. In *IEEE Design and Test of Computers*, volume 18, pages 20–29. IEEE, 2001a.

Shin D., Kim J. e Lee S. Low-energy intra-task voltage scheduling using static timing analysis. In *38th Design Automation Conference; Las Vegas, NV; United States;*, pages 438–443. IEEE, 2001b.

Society B. C. The Chartered Institute for IT, Enabling the information society. Disponível em http://onlinelibrary.wiley.com/, 2013. Acessado em 29 de Dezembro.

Sons J. W. . Wiley Online Library. Disponível em http://onlinelibrary.wiley.com/, 2013. Acessado em 29 de Dezembro.

Springer. Springer Link, Part of Springer Science+Business Media. Disponível em http://link.springer.com/, 2013. Acessado em 29 de Dezembro.

Takase H., Zeng G., Gauthier L., Kawashima H., Atsumi N., Tatematsu T., Kobayashi Y., Kohara S., Koshiro T., Ishihara T., Tomiyama H. e Takada H. An integrated optimization framework for reducing the energy consumption of embedded real-time applications. In *Proceedings of the International Symposium on Low Power Electronics and Design*, number 5993648, pages 270–276. IEEE, 2011.

Tatematsu T., Takase H., Zeng G., Tomiyama H. e Takada H. Checkpoint extraction using execution traces for intra-task dvfs in embedded systems. In *6th IEEE International Symposium on Electronic Design, Test and Application, DELTA 2011*, number 5729533, pages 19–24. IEEE, 2011.

Valentin E. e Barreto R. A branch-and-bound algorithm for optimum frequency set establishment in real-time dvfs. In *Workshop de Tempo-Real e Sistemas Embarcados (WTR)*, 2010.

Xian C. e Lu Y.-H. Dynamic voltage scaling for multitasking real-time systems with uncertain execution time. In *ACM Great Lakes Symposium on VLSI, GLSVLSI*, volume 2006, pages 392–397. ACM, 2006.

Yang C.-C., Wang K., Lin M.-H. e Lin P. Energy efficient intra-task dynamic voltage scaling for realistic cpus of mobile devices. In *Journal of Information Science and Engineering*, volume 25, pages 251–272. JISE, 2009.

Yi H., Yang X. e Chen J. The optimal profile-guided greedy dynamic voltage scaling in real-time applications. In *ICESS'05 Proceedings of the Second international conference on Embedded Software and Systems*, volume 3820 LNCS, pages 708–719. Springer, 2005.

Yuan L., Leventhal S. R., Gu J. e Qu G. Talk: A temperature-aware leakage minimization technique for real-time systems. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 30, pages 1564–1568. IEEE, 2011.

Zhang L. e QI D. Energy-efficient task scheduling algorithm for mobile terminal. In *IET Conference Publications*, volume 152. IET Digital Library, 2006.

Zitterell T. e Scholl C. Improving energy-efficient real-time scheduling by exploiting code instrumentation. In *Proceedings of the International Multiconference on Computer Science and Information Technology, IMCSIT 2008*, volume 3, pages 763–771. IEEE, 2008.

Apêndice A

Processo de instalação do ambiente de experimentação

Este apêndice irá descrever o processo de instalação do ambiente de experimentação utilizado nesta dissertação. A base do sistema operacional adotada foi o Debian 6.0 (*Squeeze*). Segue abaixo, primeiramente, o processo de instalação do *Kernel* modificado de acordo com a modelagem do método proposto feita na Seção 4.2:

1. Baixar o *Kernel* 2.6.38.8 direto do GitHub do projeto, disponível em Gonçalves e Barreto (2015a);
2. O processo de instalação do novo *Kernel* é o mesmo que se aplica a qualquer *Kernel Vanilla*. Segue o endereço eletrônico de um tutorial que explica o passo-a-passo desse processo: http://www.cafw.ufsm.br/ roberto/wp-content/uploads/2010/11/capitulo_20.pdf;
3. Reiniciar o sistema operacional selecionando o novo *Kernel* instalado;
4. O próximo passo é deixar o primeiro núcleo do processador para uso exclusivo das tarefas de tempo real. Logo, é necessário criar uma nova política no *cgroups* para realizar esse gerenciamento, executando o comando "nano /etc/cgconfig.conf" e adicionando o trecho de Código A.1. Esse código cria uma política no *cgroups* chamada de "os_default", com permissões de administrador (*root*), para colocar todos os processos para executar no processador número 1, exceto os processos que controlam as interrupções e as filas de execuções das tarefas. Deixando assim, o processador número 0 exclusivo para aplicações em tempo real criadas no nível de usuário.

Código A.1: Política padrão do *cgroups* para o ambiente de experimentação.

```
group os_default {
        perm {
                task {
                        uid = root;
                        gid = root;
                }
                admin {
                        uid = root;
                        gid = root;
                }
        }
        cpuset {
                cpuset.cpus = 1;
                cpuset.cpu_exclusive = 1;
        }
}

mount {
  cpu = /mnt/cgroups/cpu;
  cpuacct = /mnt/cgroups/cpuacct;
  devices = /mnt/cgroups/devices;
  cpuset = /mnt/cgroups/sistema_operacional;
}
```

5. Após a criação da política que define os processadores padrões para as tarefas que não são de tempo real é necessário definir a regra de aplicação da política, executando o comando "nano /etc/cgrules.conf" e adicionando o trecho de Código A.2 no final do arquivo.

Código A.2: Regra padrão do *cgrules* para o ambiente de experimentação.

```
# /etc/cgrules.conf
#
#Each line describes a rule for a user in the forms:
#
#<user>                 <controllers>          <destination>
#<user>:<process name>  <controllers>          <destination>
*                       *                      os_default/
```

6. Agora o sistema operacional deverá ser reiniciado novamente. Sendo assim, a primeira parte da configuração do ambiente de experimentação está pronta para ser integrada com o RTAI.

A segunda parte de configuração do ambiente de experimentação é realizar a integração com o RTAI, permitindo que o *Kernel* seja capaz de executar aplicações em tempo real. Segue abaixo o processo de instalação do RTAI modificado, de acordo com a modelagem do método proposto feita na Seção 4.2:

1. Baixar o RTAI 3.9.1 direto do GitHub do projeto, disponível em Gonçalves e Barreto (2015b);

2. Seguir o processo de instalação normal do RTAI, disponível em: https://www.rtai.org/ userfiles/ downloads/ RTAICONTRIB/ RTAI_Installation _Guide.pdf. Vale ressaltar que o tutorial de instalação do RTAI deve fazer referência a versão ao *Kernel* 2.6.38.8, portanto qualquer parte do tutorial que venha a fazer referência a uma versão diferente, o texto deverá ser substituído pela versão correta compatível com a versão do *Kernel* instalado no primeiro passo de configuração do ambiente de experimentação.

3. Concluída a etapa de configuração do RTAI, o sistema operacional deverá ser reiniciado, selecionando o novo *Kernel* já integrado ao RTAI;

Concluída as etapas de configuração do *Kernel* e do RTAI temos um ambiente de experimentação baseado no método proposto por esta dissertação, contendo um processador exclusivo para aplicações de tempo real. A Figura A.1 mostra, através do comando *htop* do Linux, a lista de tarefas em execução em cada núcleo de processamento disponível no sistema, onde vale observar que no processador zero, exclusivo para tarefas de tempo real, estão executando apenas quatro processos.

```
0  [                                                                    0.0%]
1  [|||                                                                 2.0%]
Mem[|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||260/3706MB]
Swp[                                                                  0/7250MB]
```

PID	CPU	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
3	0	root	20	0	0	0	0	S	0.0	0.0	0:00.00	ksoftirqd/0
4	0	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kworker/0:0
6	0	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
10	0	root	20	0	0	0	0	S	0.0	0.0	0:00.23	kworker/0:1
1	1	root	20	0	10476	768	628	S	0.0	0.0	0:02.12	init [2]
2	1	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
7	1	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/1
9	1	root	20	0	0	0	0	S	0.0	0.0	0:00.02	ksoftirqd/1
11	1	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	cpuset
12	1	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	khelper
13	1	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	netns
14	1	root	20	0	0	0	0	S	0.0	0.0	0:00.00	sync_supers
15	1	root	20	0	0	0	0	S	0.0	0.0	0:00.00	bdi-default
16	1	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kintegrityd
17	1	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kblockd
18	1	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kacpid
19	1	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kacpi_notify
20	1	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kacpi_hotplug
21	1	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kondemand
43	1	root	20	0	0	0	0	S	0.0	0.0	0:00.25	kworker/1:1
44	1	root	20	0	0	0	0	S	0.0	0.0	0:00.00	khungtaskd
45	1	root	20	0	0	0	0	S	0.0	0.0	0:00.81	kswapd0
46	1	root	25	5	0	0	0	S	0.0	0.0	0:00.00	ksmd
47	1	root	20	0	0	0	0	S	0.0	0.0	0:00.00	fsnotify_mark
48	1	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	aio
49	1	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	crypto
172	1	root	20	0	0	0	0	S	0.0	0.0	0:00.00	khubd
173	1	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	ata_sff
177	1	root	20	0	0	0	0	S	0.0	0.0	0:00.00	scsi_eh_0
178	1	root	20	0	0	0	0	S	0.0	0.0	0:00.00	scsi_eh_1
182	1	root	20	0	0	0	0	S	0.0	0.0	0:00.00	scsi_eh_2
183	1	root	20	0	0	0	0	S	0.0	0.0	0:00.00	scsi_eh_3
186	1	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kworker/u:4

Figura A.1: Lista de tarefas em execução em cada núcleo do processador, durante o carregamento inicial do ambiente de experimentação.

Esses processos são essenciais para o funcionamento do sistema operacional, são eles: "*ksoftirqd*" é responsável por gerenciar as interrupções; "*kworker*" é responsável por gerenciar a ordem de execução das tarefas; e por fim, "*migration*" é responsável por realizar o balanceamento de carga de trabalho entre os processadores, mas com a atuação da política do *cgroups*, ele passa a ser ignorado pelo sistema. Vale salientar que as prioridades desses processos, exibidas na coluna "PRI", são definidas por padrão pelo sistema operacional e que as tarefas em tempo real criadas pelo usuário poderão ter prioridades de 0 a 99 ou "RT" (*Real Time*).

Quanto ao modo de execução das aplicações de tempo real dentro do ambiente de experimentação basta seguir o processo de execução de aplicações do RTAI, pois não foram feitas modificações nesse processo.

Os trechos de código a seguir mostram as principais modificações feitas no *Kernel* do Linux e no RTAI:

Código A.3: Os novos atributos adicionados na estrutura de dados da tarefa dentro do *Kernel* do Linux.

```
1 //Arquivo: linux-raw-gov-2.6.38.8/include/linux/sched.h
2 //...
3 struct task_struct {
4   //...
5
6   //TODO:RAWLINSON - Flag que indica ao RAW GOVERNOR que a tarefa acabou de voltar
7   // de uma preempcao.
8
9   // (1 - se a tarefa FOI preemptada dentro do RTAI e 0 - caso contrario)
10  unsigned int flagPreemption;
11
12  // (1 - se a tarefa voltou de preempcao dentro do RTAI e 0 - caso contrario)
13  unsigned int flagReturnPreemption;
14
15  // (1 - se a tarefa foi verificada pelo RAW MONITOR e 0 - caso contrario)
16  unsigned int flagCheckedRawMonitor;
17
18  /* TODO:RAWLINSON - VARIAVEIS DE CONTROLE E GERENCIAMENTO DO RAW GOVERNOR */
19  unsigned long tsk_wcec; // WCEC - Worst Case Execution Cycles - of the task
20  unsigned long rwcec; // RWCEC - Remaining Worst Case Execution Cycle
21
22  // Estado da tarefa durante o periodo...
23  // 0 - indefinida; 1 - tarefa rodando; 2 - tarefa finalizada
24  unsigned int state_task_period;
25
26  // Frequencia minima a ser utilizada pela tarefa...
27  // Se igual a zero ela eh desconsiderada.
28  unsigned int cpu_frequency_min;
29  unsigned int cpu_frequency;
30  unsigned int cpu_voltage;
31  unsigned int last_cpu_frequency;
32  unsigned int last_cpu_voltage;
33  /* TODO:RAWLINSON - FIM DAS DEFINICOES...*/
34  //...
35 }
```

Código A.4: Esse trecho de código direciona as tarefas do sistema operacional para o segundo núcleo do processador, deixando o primeiro núcleo exclusivo para as aplicações de tempo real.

```
1  //Arquivo: linux-raw-gov-2.6.38.8/init/main.c
2
3  //...
4
5
6  static int __init kernel_init(void * unused)
7  {
8    //TODO:RAWLINSON
9    struct cpumask cpu_padrao = cpumask_of_cpu(CPUID_PADRAO);
10
11   //...
12
13   //TODO:RAWLINSON
14   //set_cpus_allowed_ptr(current, cpu_all_mask); //TODO:RAWLINSON - CODIGO ORIGINAL
15   current->cpus_allowed = cpu_padrao;
16   set_cpus_allowed_ptr(current, &cpu_padrao);
17
18   //...
19 }
```

Código A.5: Esse trecho de código direciona as novas tarefas do sistema operacional, que não sejam de tempo real, para o segundo núcleo do processador garantindo que o primeiro núcleo seja usado apenas por aplicações de tempo real (criadas pelo RTAI).

```
1  //Arquivo: linux-raw-gov-2.6.38.8/kernel/kthread.c
2
3  //...
4
5  int kthreadd(void *unused)
6  {
7    struct task_struct *tsk = current;
8
9    //TODO:RAWLINSON
10   struct cpumask cpu_padrao = cpumask_of_cpu(CPUID_PADRAO);
11
12   //...
13
14   //TODO:RAWLINSON
15   //set_cpus_allowed_ptr(tsk, cpu_all_mask); //TODO:RAWLINSON - CODIGO ORIGINAL...
16   tsk->cpus_allowed = cpu_padrao;
17   set_cpus_allowed_ptr(tsk, &cpu_padrao);
18
19   //...
20 }
```

Código A.6: Função utilizada pelo *Raw Governor* para validar e obter frequências validas do processador.

```
1  //Arquivo: linux-raw-gov-2.6.38.8/drivers/cpufreq/cpufreq_raw.c
2
3  //...
4
```

```
5 unsigned int get_frequency_table_target(struct cpufreq_policy *policy,
6                                         unsigned int target_freq)
7 {
8   unsigned int new_freq;
9   unsigned int i;
10
11   if (!cpu_online(policy->cpu))
12     return -EINVAL;
13
14   //OBS.: as frequencias comecam do MAIOR para o MENOR.
15   new_freq = freq_table[0].frequency;
16   for (i = 0; (freq_table[i].frequency != CPUFREQ_TABLE_END); i++) {
17     unsigned int freq = freq_table[i].frequency;
18
19     if (freq == CPUFREQ_ENTRY_INVALID)
20       continue;
21
22     if ((freq < policy->min) || (freq > policy->max))
23       continue;
24
25     if (freq < target_freq) {
26       break;
27     }
28     new_freq = freq;
29   }
30
31   printk("DEBUG:RAWLINSON - RAW GOVERNOR - get_frequency_table_target(%u) kHz for
32             cpu %u => NOVA FREQ(%u kHz)\n", target_freq, policy->cpu, new_freq);
33
34   return new_freq;
35 }
36
37 //...
```

Código A.7: Função utilizada pelo *Raw Governor* para definir novas frequências no processador.

```
1 //Arquivo: linux-raw-gov-2.6.38.8/drivers/cpufreq/cpufreq_raw.c
2
3 //...
4
5 /**
6  * Sets the CPU frequency to freq.
7  */
8 static int set_frequency(struct cpufreq_policy *policy, struct task_struct *task,
9                          unsigned int freq)
10 {
11   unsigned int valid_freq = 0;
12   int ret = -EINVAL;
13
14   mutex_lock(&raw_mutex);
15
16   // Se alguma frequencia foi definida... entao o monitor nao precisa mais
17   // verificar a tarefa que foi sinalizada... \o/
18   if(task && task->pid > 0)
19   {
20     task->flagPreemption = 0;
21     task->flagReturnPreemption = 0;
22     task->flagCheckedRawMonitor = 0;
```

```
23
24      /*
25       * We're safe from concurrent calls to ->target() here
26       * as we hold the raw_mutex lock. If we were calling
27       * cpufreq_driver_target, a deadlock situation might occur:
28       * A: cpufreq_set (lock raw_mutex) ->
29       *      cpufreq_driver_target(lock policy->lock)
30       * B: cpufreq_set_policy(lock policy->lock) ->
31       *      __cpufreq_governor ->
32       *         cpufreq_governor_raw (lock raw_mutex)
33       */
34      valid_freq = get_frequency_table_target(policy, freq);
35      if(valid_freq >= task->cpu_frequency_min)
36      {
37        ret = __cpufreq_driver_target(policy, valid_freq, CPUFREQ_RELATION_H);
38
39        //Atualizando a frequencia da tarefa para uma frequencia valida.
40        task->cpu_frequency = policy->cur; // (KHz)
41
42        printk("DEBUG:RAWLINSON - RAW GOVERNOR - set_frequency(%u) for cpu %u -
43               %u KHz - GOV(%s) -> PID (%d)\n", freq, policy->cpu, policy->cur,
44               policy->governor->name, task->pid);
45      }
46      else
47      {
48        ret = __cpufreq_driver_target(policy, task->cpu_frequency_min,
49                                      CPUFREQ_RELATION_H);
50
51        //Atualizando a frequencia da tarefa para uma frequencia valida.
52        task->cpu_frequency = policy->cur; // (KHz)
53
54        printk("DEBUG:RAWLINSON - RAW GOVERNOR - set_frequency(%u) - OBS.: FREQUENCIA
55               INVALIDA! PID (%d) [FREQ_ALVO(%u KHz) < FREQ_MIN(%u KHz)] \n", freq,
56               task->pid, valid_freq, task->cpu_frequency_min);
57      }
58    }
59
60    mutex_unlock(&raw_mutex);
61    return ret;
62  }
63
64  //...
```

Código A.8: Essa função é uma extensão do *Raw Monitor* dentro do *Raw Governor* cujo objetivo é realizar o monitoramento das tarefas de tempo real que estão retornando de preempção de acordo com as instruções passadas pelo escalonador do sistema.

```
1  //Arquivo: linux-raw-gov-2.6.38.8/drivers/cpufreq/cpufreq_raw.c
2
3  //...
4
5  void raw_gov_work(struct kthread_work *work)
6  {
7    struct raw_gov_info_struct *info;
8    unsigned long target_freq = 0;
9
10   info = container_of(work, struct raw_gov_info_struct, work);
```

```
11
12   mutex_lock(&info->timer_mutex);
13   if(info->tarefa_sinalizada && info->tarefa_sinalizada->pid > 0)
14   {
15     if(info->tarefa_sinalizada->rwcec > 0)
16     {
17       target_freq = calc_freq(info);
18       if(target_freq < info->tarefa_sinalizada->cpu_frequency_min)
19         target_freq = info->tarefa_sinalizada->cpu_frequency_min;
20
21       __cpufreq_driver_target(info->policy, target_freq, CPUFREQ_RELATION_H);
22
23       // (KHz) Nova frequencia para a tarefa...
24       // visando diminuir o tempo de folga da tarefa.
25       info->tarefa_sinalizada->cpu_frequency = target_freq;
26
27       printk("------------------------[ RAW MONITOR ]-------------------\n");
28       printk("DEBUG:RAWLINSON - raw_gov_work(%lu) for cpu %u, freq %u kHz -
29              PID(%d)\n", target_freq, info->policy->cpu, info->policy->cur,
30              info->tarefa_sinalizada->pid);
31     }
32
33     info->tarefa_sinalizada->flagReturnPreemption = 0;
34     info->tarefa_sinalizada->flagCheckedRawMonitor = 1;
35
36     clear_task_monitor(info);
37   }
38   mutex_unlock(&info->timer_mutex);
39 }
40
41 //...
```

Código A.9: Esta função foi inserida dentro do escalonador do RTAI para alertar o *Raw Monitor* sobre as tarefas que estão retornando de preempção.

```
1  //Arquivo: rtai-raw-gov-3.9.1/base/sched/sched.c
2
3  //...
4
5  //TODO:RAWLINSON... MONITORA A EXECUCAO DAS TAREFAS QUE RETORNARAM DE PREEMPCAO.
6  int preemption_monitor(void)
7  {
8    struct task_struct *current_task_linux;
9    RT_TASK *rt_task;
10   struct cpufreq_policy *policy;
11   unsigned long long deadline_ns;
12
13   // possui o timer do processador RTAI atualizado...
14   unsigned long long tick_timer_atual_ns;
15
16   // Verifica se a tarefa em execucao retornou de uma preempcao...
17   current_task_linux = get_current_task(CPUID_RTAI);
18   rt_task = pid2rttask(current_task_linux->pid);
19
20   if(rt_task && current_task_linux && current_task_linux->flagReturnPreemption &&
21      current_task_linux->pid > 0 && current_task_linux->state == TASK_RUNNING &&
22      current_task_linux->state_task_period == TASK_PERIOD_RUNNING)
23   {
24     printk("@@@@@@@@@@@@@@@@@@@@ RETORNANDO DE PREEMPCAO @@@@@@@@@@@@@@@@@@@@\n");
```

```
25      printk("[RAWLINSON_SCHEDULE - TASK_TO_SCHEDULE]: C_PID(%d) C_STATE(%ld)
26              C_FRP(%d)\n", current_task_linux->pid, current_task_linux->state,
27              current_task_linux->flagReturnPreemption);
28
29      policy = cpufreq_cpu_get(CPUID_RTAI);
30      if(policy && policy->governor && policy->governor->wake_up_kworker)
31      {
32        tick_timer_atual_ns = rt_get_cpu_time_ns();
33
34        // Verifica se o deadline foi informado pelo usuario...
35        if(rt_task->deadline > 0 && rt_task->deadline != rt_task->period)
36        {
37          // Deve ser utilizado o deadline, pois ele foi especificado pelo usuario.
38          deadline_ns = count2nano(rt_task->periodic_resume_time+rt_task->deadline);
39        }
40        else
41        {
42          // Significa que o deadline nao foi informado pelo usuario ou que o
43          // deadline eh igual ao periodo.
44          deadline_ns = count2nano(rt_task->periodic_resume_time + rt_task->period);
45        }
46
47        policy->governor->wake_up_kworker(policy, current_task_linux,
48                                          tick_timer_atual_ns, deadline_ns);
49      }
50      //TODO:RAWLINSON - FIM
51    }
52
53    return 0;
54  }
55
56  //...
```

Código A.10: Trecho de código é responsável por aplicar o *Raw Governor* ao primeiro núcleo do processador durante a chamada da função de inicialização do RTAI.

```
1  //Arquivo: rtai-raw-gov-3.9.1/base/sched/sched.c
2  //...
3  static int lxrt_init(void)
4  {
5    //...
6
7    //TODO:RAWLINSON - APLICANDO O RAW GOVERNOR NO CPUID DO RTAI...
8    char str_raw_governor[16] = "raw";
9    struct cpufreq_policy *policy;
10   policy = cpufreq_cpu_get(CPUID_RTAI);
11   if(policy)
12   {
13     store_scaling_governor(policy, str_raw_governor, 1);
14   }
15   printk("*******DEBUG:RAWLINSON - RAW GOVERNOR - store_scaling_governor for cpu %u -
16          %u - %s\n", policy->cpu, policy->cur, policy->governor->name);
17   //TODO:RAWLINSON - FIM
18   //...
19 }
```

Apêndice B

Chamadas de Sistema do *Raw Governor*

Este apêndice irá detalhar as chamadas de sistema utilizadas pelas aplicações de tempo real para se comunicar com o *Raw Governor*, permitindo que elas possam enviar e obter as informações necessárias para que o gerenciamento das tensões e frequências a serem utilizadas no processador possa ser realizar no núcleo do sistema operacional de acordo com a tarefa que venha a entrar em execução.

A seguir serão listadas as assinaturas das funções que estarão disponíveis no sistema operacional:

- **init_info(*tsk_struct, WCEC, freq_ideal, freq_curr, volt_curr):** esta função informa para o sistema operacional os dados inicias da tarefa τ_i. Vale observar que a tarefa inicialmente executará os valores de tensão e frequência, respectivamente, *freq_curr* e *volt_curr*, enquanto que a frequência ideal (*freq_ideal*) somente será utilizada nos retornos de preempção;
- **get_tsk_wcec(*tsk_struct, WCEC):** esta função é utilizada para se obter o valor do WCEC de uma tarefa τ_i;
- **set_rwcec(*tsk_struct, RWCEC):** esta função é utilizada para informar ao sistema operacional o RWCEC atual de uma tarefa τ_i;
- **get_rwcec(*tsk_struct):** esta função é utilizada para obter o RWCEC de uma tarefa τ_i informada na função **set_rwcec()**;
- **set_cpu_frequency(*tsk_struct, new_freq):** esta função é utilizada para definir a frequência a ser aplicada sobre o processador. Vale ressaltar que o *Governor* irá aplicar apenas valores de frequências válidas (como por exemplo os valores exibidos na Tabela 2.1), caso o valor seja inválido então será atribuída a frequências maior e mais próxima do valor informado;

- **get_cpu_frequency(*tsk_struct):** esta função é utilizada para obter a frequência atual do processador;
- **set_cpu_voltage(*tsk_struct, new_volt):** esta função é utilizada para definir a tensão que deve ser aplicada sobre o processador. Vale ressaltar que o *Governor* irá aplicar apenas valores de tensões válidos, caso o valor seja inválido então será atribuída a tensão maior e mais próxima do valor informado;
- **get_cpu_voltage(*tsk_struct):** esta função é utilizada para obter a tensão atual do processador;
- **get_timer():** esta função é utilizada para se obter o *tick time* atual do processador, ou seja, são as unidades de tempo utilizadas pelos temporizadores do sistema operacional. Essa informação é útil para que a tarefa possa obter o tempo restante de processamento até o final do período corrente;
- **get_periodic_resume_time(*tsk_struct):** esta função é utilizada pelas tarefas de tempo real para obter a fatia de tempo do período atual de uma determinada tarefa τ_i.

Apêndice C

Ferramenta *Smartenum*

A ferramenta *Smartenum* tem como objetivo encontrar as tensões e frequências ideais de um dado conjunto de tarefas de tempo real, de forma que o consumo de energia dinâmica do processador seja minimizado. Portanto, segue abaixo um breve passo-a-passo de como obter as tensões e frequências ideias:

1. Baixar a ferramenta *Smartenum* direto do repositório do desenvolvedor, disponível em: http://repo.or.cz/w/smartenum.git;
2. Seguir o processo de instalação descrito no arquivo *INSTALL*, dentro do projeto;
3. O próximo passo é definir os dados de entrada da ferramenta, que basicamente são compostos por um conjunto de tarefas de tempo real e suas informações (ver exemplo no Código C.1);

Código C.1: Exemplo de dados de entrada da ferramenta *Smartenum* (*layout* do arquivo de entrada está descrito no manual da ferramenta).

```
15  4 1
3000000000  2300000000  1800000000  800000000
1.3500  1.2500  1.1500  1.1250
3920000 10  0 0
1170200000  16  0 0
416160000 16  0 0
1170200000  16  0 0
416070000 16  0 0
1170200000  20  0 0
416070000 20  0 0
1328000000  20  0 0
5658200000  30  0 0
419220000 30  0 0
7226100000  30  0 0
1328000000  30  0 0
7226100000  30  0 0
416070000 30  0 0
5658200000  30  0 0
```

4. Uma vez criado os dados de entrada da ferramenta, basta abrir o terminal na pasta do projeto e executar o seguinte comando "**cat [caminho_arquivo_com_dados_entrada].txt | se -s**" para dar início ao processamento dos dados informados.

5. Por fim, a ferramenta irá mostrar as tensões e frequências ideais de cada tarefa de tempo real (ver exemplo no Código C.2).

Código C.2: Exemplo dos dados de saída gerados pela ferramenta *Smartenum*.

```
1 Sumario
2 Numero de Configuracoes: 1073741824
3 Configuracoes Avaliadas: 1073741824
4 Configuracoes Viaveis: 24811432
5 Tempo de processamento: 13575s and 227598 us
6 Melhor espalhamento 129.85 com as seguintes frequencias
7 (800000000.00; 1.12)
8 (1800000000.00; 1.15)
9 (800000000.00; 1.12)
10 (1800000000.00; 1.15)
11 (2300000000.00; 1.25)
12 (1800000000.00; 1.15)
13 (2300000000.00; 1.25)
14 (2300000000.00; 1.25)
15 (2300000000.00; 1.25)
16 (2300000000.00; 1.25)
17 (3000000000.00; 1.35)
18 (3000000000.00; 1.35)
19 (3000000000.00; 1.35)
20 (3000000000.00; 1.35)
21 (800000000.00; 1.12)
22 Utilizacao total do sistema eh 120.09%
23 Energia gasta pelo sistema eh 78508331976.93 x C
24 Energia gasta pelo sistema eh 90665240842.82 x C se usar apenas a maior frequencia
25 Reducao de energia:  13.41%
```

O único ponto negativo da ferramenta é que ela não leva em consideração as características de *software* e *hardware* na obtenção das tensões e frequências ideais das tarefas. No entanto, os resultados obtidos com o uso dessa ferramenta facilitam significativamente na obtenção dos valores reais da plataforma, como já foi comentando e exemplificado na Seção 4.3.

O Apêndice D mostra como utilizar as tensões e frequências calculadas pela ferramenta *Smartenum* como base para a geração de casos de teste mais críticos e com maior utilização do processador, levando em consideração o tempo de computação real de cada tarefa dentro do ambiente de experimentação.

Apêndice D

Teste de escalonabilidade feito no MatLab

O *script* implementado no MatLab foi baseado no teste de escalonabilidade 4 (ver Equação 2.25 da Seção 2.2.4) e tem como objetivo automatizar o teste de escalonabilidade das 15 tarefas de tempo real definidas na Tabela 5.1, levando em consideração o tempo de computação real de cada tarefa dentro do ambiente de experimentação. Dessa forma, é possível validar a utilização do processador, levando em consideração as características intrínsecas da arquitetura.

Os objetivos dos experimentos criados foi propor o pior caso de utilização do processador, para então aplicar e validar a metodologia proposta, mas isso só foi possível utilizando o tempo de computação real das tarefas. Logo, para obter o tempo de computação real, cada tarefa foi executada isoladamente dentro do ambiente de experimentação, utilizando as tensões e frequências ideais calculadas pela ferramenta *Smartenum* (sem nenhuma interferência das demais tarefas). Em seguida, esses dados foram aplicados no teste de escalonabilidade feito no MatLab para se obter a utilização real do processador.

Através desse *script* foi possível constatá que a utilização do processador com base nos dados computados pela ferramenta *Smartenum* foi cerca de $79,36\%$ (ver Código D.1). Então para alcançar taxas de utilização ainda maiores foram reduzidas as frequências de algumas tarefas, sem que houvesse violações de premissas temporais. Dessa forma, foi possível aumentar a utilização real do processador para 96,82%, criando assim um estudo de caso crítico para validação da metodologia proposta.

Código D.1: Resultado da utilização do processador com base nos dados calculados pela ferramenta *Smartenum*.

```
1 >> calc_tempo_resposta_15_tarefas
2
3 W1_1 = 0.0049
4
5 W2_1 = 0.6550
```

```
6  W2_2 = 0.6550
7
8  W3_2 = 1.1752
9  W3_3 = 1.1752
10
11 W4_3 = 1.8253
12 W4_4 = 1.8253
13
14 W5_4 = 2.0062
15 W5_5 = 2.0062
16
17 W6_5 = 2.6563
18 W6_6 = 2.6563
19
20 W7_6 = 2.8372
21 W7_7 = 2.8372
22
23 W8_7 = 3.4146
24 W8_8 = 3.4146
25
26 W9_8 = 7.0311
27 W9_9 = 7.0311
28
29 W10_9  = 7.2120
30 W10_10 = 7.2120
31
32 W11_10 = 10.4740
33 W11_11 = 10.4740
34
35 W12_12 = 10.8776
36 W12_13 = 10.8776
37
38 W13_16 = 14.1347
39 W13_17 = 14.1347
40
41 W14_16 = 14.2743
42 W14_17 = 14.2743
43
44 W15_24 = 24.7616
45 W15_25 = 24.7616
46
47 Utilizacao = 0.7936
```

O Código D.2 mostra o código fonte completo do *script* implementado no MatLab para obter as novas tensões e frequência ideais levando em consideração o tempo de computação real das tarefas e o Código D.3 mostra os tempos de resposta e a utilização do processador, após a execução do *Script*.

Código D.2: *Script* implementado no MatLab para realizar o teste de escalonabilidade interativo baseado na Equação 2.25.

```
1 %OBS.: AS TAREFAS ESTAO ORDENADAS POR PRIORIDADE E NAO POR ID.
2 C1 = 0.0049; % 0.8 GHz %(OBS.: Freq. obtida pela ferramenta Smartenum)
3 P1 = 10;
4
5 C2 = 0.6501; % 1.8 GHz %(OBS.: Freq. obtida pela ferramenta Smartenum)
6 P2 = 16;
```

```
7
8  C3 = 0.5202; % 0.8 GHz %(OBS.: Freq. obtida pela ferramenta Smartenum)
9  P3 = 16;
10
11 C4 = 0.6501; % 1.8 GHz %(OBS.: Freq. obtida pela ferramenta Smartenum)
12 P4 = 16;
13
14 C5 = 0.1809; % 2.3 GHz %(OBS.: Freq. obtida pela ferramenta Smartenum)
15 P5 = 16;
16
17 C6 = 0.6501; % 1.8 GHz %(OBS.: Freq. obtida pela ferramenta Smartenum)
18 P6 = 20;
19
20 C7 = 0.1809; % 2.3 GHz %(OBS.: Freq. obtida pela ferramenta Smartenum)
21 P7 = 20;
22
23 C8 = 0.5774; % 2.3 GHz %(OBS.: Freq. obtida pela ferramenta Smartenum)
24 P8 = 20;
25
26 %C9 = 3.6165; % 2.3 GHz %(OBS.: Freq. obtida pela ferramenta Smartenum)
27 C9 = 7.0727; % 0.8 GHz
28 P9 = 30;
29
30 %C10 = 0.1809; % 2.3 GHz %(OBS.: Freq. obtida pela ferramenta Smartenum)
31 C10 = 0.2329; % 1.8 GHz
32 P10 = 30;
33
34 %C11 = 3.2571; % 3.0 GHz %(OBS.: Freq. obtida pela ferramenta Smartenum)
35 C11 = 4.0145; % 1.8 GHz
36 P11 = 30;
37
38 %C12 = 0.4036; % 3.0 GHz %(OBS.: Freq. obtida pela ferramenta Smartenum)
39 C12 = 0.5774; % 2.3 GHz
40 P12 = 30;
41
42 %C13 = 3.2571; % 3.0 Ghz %(OBS.: Freq. obtida pela ferramenta Smartenum)
43 C13 = 4.0145; % 1.8Gghz
44 P13 = 30;
45
46 %C14 = 0.1396; % 3.0 GHz %(OBS.: Freq. obtida pela ferramenta Smartenum)
47 C14 = 0.1809; % 2.3 GHz
48 P14 = 30;
49
50 C15 = 7.0727; % 800Mghz %(OBS.: Freq. obtida pela ferramenta Smartenum)
51 P15 = 30;
52
53 %TEMPO DE RESPOSTA DA TAREFA DE PRIORIDADE 1
54 W1_1 = C1
55
56 %TEMPO DE RESPOSTA DA TAREFA DE PRIORIDADE 2
57 W2_0 = C2;
58 W2_1 = C2 + ceil(W2_0/P1)*C1
59 W2_2 = C2 + ceil(W2_1/P1)*C1
60
61 %TEMPO DE RESPOSTA DA TAREFA DE PRIORIDADE 3
62 W3_0 = C3;
63 W3_1 = C3 + ceil(W3_0/P1)*C1 + ceil(W3_0/P2)*C2;
64 W3_2 = C3 + ceil(W3_1/P1)*C1 + ceil(W3_1/P2)*C2
65 W3_3 = C3 + ceil(W3_2/P1)*C1 + ceil(W3_2/P2)*C2
```

```
66
67  %TEMPO DE RESPOSTA DA TAREFA DE PRIORIDADE 4
68  W4_0 = C4;
69  W4_1 = C4 + ceil(W4_0/P1)*C1 + ceil(W4_0/P2)*C2 + ceil(W4_0/P3)*C3;
70  W4_2 = C4 + ceil(W4_1/P1)*C1 + ceil(W4_1/P2)*C2 + ceil(W4_1/P3)*C3;
71  W4_3 = C4 + ceil(W4_2/P1)*C1 + ceil(W4_2/P2)*C2 + ceil(W4_2/P3)*C3
72  W4_4 = C4 + ceil(W4_3/P1)*C1 + ceil(W4_3/P2)*C2 + ceil(W4_3/P3)*C3
73
74  %TEMPO DE RESPOSTA DA TAREFA DE PRIORIDADE 5
75  W5_0 = C5;
76  W5_1 = C5 + ceil(W5_0/P1)*C1 + ceil(W5_0/P2)*C2 + ceil(W5_0/P3)*C3 ...
77         + ceil(W5_0/P4)*C4;
78  W5_2 = C5 + ceil(W5_1/P1)*C1 + ceil(W5_1/P2)*C2 + ceil(W5_1/P3)*C3 ...
79         + ceil(W5_1/P4)*C4;
80  W5_3 = C5 + ceil(W5_2/P1)*C1 + ceil(W5_2/P2)*C2 + ceil(W5_2/P3)*C3 ...
81         + ceil(W5_2/P4)*C4;
82  W5_4 = C5 + ceil(W5_3/P1)*C1 + ceil(W5_3/P2)*C2 + ceil(W5_3/P3)*C3 ...
83         + ceil(W5_3/P4)*C4
84  W5_5 = C5 + ceil(W5_4/P1)*C1 + ceil(W5_4/P2)*C2 + ceil(W5_4/P3)*C3 ...
85         + ceil(W5_4/P4)*C4
86
87  %TEMPO DE RESPOSTA DA TAREFA DE PRIORIDADE 6
88  W6_0 = C6;
89  W6_1 = C6 + ceil(W6_0/P1)*C1 + ceil(W6_0/P2)*C2 + ceil(W6_0/P3)*C3 ...
90         + ceil(W6_0/P4)*C4 + ceil(W6_0/P5)*C5;
91  W6_2 = C6 + ceil(W6_1/P1)*C1 + ceil(W6_1/P2)*C2 + ceil(W6_1/P3)*C3 ...
92         + ceil(W6_1/P4)*C4 + ceil(W6_1/P5)*C5;
93  W6_3 = C6 + ceil(W6_2/P1)*C1 + ceil(W6_2/P2)*C2 + ceil(W6_2/P3)*C3 ...
94         + ceil(W6_2/P4)*C4 + ceil(W6_2/P5)*C5;
95  W6_4 = C6 + ceil(W6_3/P1)*C1 + ceil(W6_3/P2)*C2 + ceil(W6_3/P3)*C3 ...
96         + ceil(W6_3/P4)*C4 + ceil(W6_3/P5)*C5;
97  W6_5 = C6 + ceil(W6_4/P1)*C1 + ceil(W6_4/P2)*C2 + ceil(W6_4/P3)*C3 ...
98         + ceil(W6_4/P4)*C4 + ceil(W6_4/P5)*C5
99  W6_6 = C6 + ceil(W6_5/P1)*C1 + ceil(W6_5/P2)*C2 + ceil(W6_5/P3)*C3 ...
100        + ceil(W6_5/P4)*C4 + ceil(W6_5/P5)*C5
101
102 %TEMPO DE RESPOSTA DA TAREFA DE PRIORIDADE 7
103 W7_0 = C7;
104 W7_1 = C7 + ceil(W7_0/P1)*C1 + ceil(W7_0/P2)*C2 + ceil(W7_0/P3)*C3 ...
105        + ceil(W7_0/P4)*C4 + ceil(W7_0/P5)*C5 + ceil(W7_0/P6)*C6;
106 W7_2 = C7 + ceil(W7_1/P1)*C1 + ceil(W7_1/P2)*C2 + ceil(W7_1/P3)*C3 ...
107        + ceil(W7_1/P4)*C4 + ceil(W7_1/P5)*C5 + ceil(W7_1/P6)*C6;
108 W7_3 = C7 + ceil(W7_2/P1)*C1 + ceil(W7_2/P2)*C2 + ceil(W7_2/P3)*C3 ...
109        + ceil(W7_2/P4)*C4 + ceil(W7_2/P5)*C5 + ceil(W7_2/P6)*C6;
110 W7_4 = C7 + ceil(W7_3/P1)*C1 + ceil(W7_3/P2)*C2 + ceil(W7_3/P3)*C3 ...
111        + ceil(W7_3/P4)*C4 + ceil(W7_3/P5)*C5 + ceil(W7_3/P6)*C6;
112 W7_5 = C7 + ceil(W7_4/P1)*C1 + ceil(W7_4/P2)*C2 + ceil(W7_4/P3)*C3 ...
113        + ceil(W7_4/P4)*C4 + ceil(W7_4/P5)*C5 + ceil(W7_4/P6)*C6;
114 W7_6 = C7 + ceil(W7_5/P1)*C1 + ceil(W7_5/P2)*C2 + ceil(W7_5/P3)*C3 ...
115        + ceil(W7_5/P4)*C4 + ceil(W7_5/P5)*C5 + ceil(W7_5/P6)*C6
116 W7_7 = C7 + ceil(W7_6/P1)*C1 + ceil(W7_6/P2)*C2 + ceil(W7_6/P3)*C3 ...
117        + ceil(W7_6/P4)*C4 + ceil(W7_6/P5)*C5 + ceil(W7_6/P6)*C6
118
119 %TEMPO DE RESPOSTA DA TAREFA DE PRIORIDADE 8
120 W8_0 = C8;
121 W8_1 = C8 + ceil(W8_0/P1)*C1 + ceil(W8_0/P2)*C2 + ceil(W8_0/P3)*C3 ...
122        + ceil(W8_0/P4)*C4 + ceil(W8_0/P5)*C5 + ceil(W8_0/P6)*C6 ...
123        + ceil(W8_0/P7)*C7;
124 W8_2 = C8 + ceil(W8_1/P1)*C1 + ceil(W8_1/P2)*C2 + ceil(W8_1/P3)*C3 ...
```

```
125         + ceil(W8_1/P4)*C4 + ceil(W8_1/P5)*C5 + ceil(W8_1/P6)*C6 ...
126         + ceil(W8_1/P7)*C7;
127   W8_3 = C8 + ceil(W8_2/P1)*C1 + ceil(W8_2/P2)*C2 + ceil(W8_2/P3)*C3 ...
128         + ceil(W8_2/P4)*C4 + ceil(W8_2/P5)*C5 + ceil(W8_2/P6)*C6 ...
129         + ceil(W8_2/P7)*C7;
130   W8_4 = C8 + ceil(W8_3/P1)*C1 + ceil(W8_3/P2)*C2 + ceil(W8_3/P3)*C3 ...
131         + ceil(W8_3/P4)*C4 + ceil(W8_3/P5)*C5 + ceil(W8_3/P6)*C6 ...
132         + ceil(W8_3/P7)*C7;
133   W8_5 = C8 + ceil(W8_4/P1)*C1 + ceil(W8_4/P2)*C2 + ceil(W8_4/P3)*C3 ...
134         + ceil(W8_4/P4)*C4 + ceil(W8_4/P5)*C5 + ceil(W8_4/P6)*C6 ...
135         + ceil(W8_4/P7)*C7;
136   W8_6 = C8 + ceil(W8_5/P1)*C1 + ceil(W8_5/P2)*C2 + ceil(W8_5/P3)*C3 ...
137         + ceil(W8_5/P4)*C4 + ceil(W8_5/P5)*C5 + ceil(W8_5/P6)*C6 ...
138         + ceil(W8_5/P7)*C7;
139   W8_7 = C8 + ceil(W8_6/P1)*C1 + ceil(W8_6/P2)*C2 + ceil(W8_6/P3)*C3 ...
140         + ceil(W8_6/P4)*C4 + ceil(W8_6/P5)*C5 + ceil(W8_6/P6)*C6 ...
141         + ceil(W8_6/P7)*C7
142   W8_8 = C8 + ceil(W8_7/P1)*C1 + ceil(W8_7/P2)*C2 + ceil(W8_7/P3)*C3 ...
143         + ceil(W8_7/P4)*C4 + ceil(W8_7/P5)*C5 + ceil(W8_7/P6)*C6 ...
144         + ceil(W8_7/P7)*C7
145
146   %TEMPO DE RESPOSTA DA TAREFA DE PRIORIDADE 9
147   W9_0 = C9;
148   W9_1 = C9 + ceil(W9_0/P1)*C1 + ceil(W9_0/P2)*C2 + ceil(W9_0/P3)*C3 ...
149         + ceil(W9_0/P4)*C4 + ceil(W9_0/P5)*C5 + ceil(W9_0/P6)*C6 ...
150         + ceil(W9_0/P7)*C7 + ceil(W9_0/P8)*C8;
151   W9_2 = C9 + ceil(W9_1/P1)*C1 + ceil(W9_1/P2)*C2 + ceil(W9_1/P3)*C3 ...
152         + ceil(W9_1/P4)*C4 + ceil(W9_1/P5)*C5 + ceil(W9_1/P6)*C6 ...
153         + ceil(W9_1/P7)*C7 + ceil(W9_1/P8)*C8;
154   W9_3 = C9 + ceil(W9_2/P1)*C1 + ceil(W9_2/P2)*C2 + ceil(W9_2/P3)*C3 ...
155         + ceil(W9_2/P4)*C4 + ceil(W9_2/P5)*C5 + ceil(W9_2/P6)*C6 ...
156         + ceil(W9_2/P7)*C7 + ceil(W9_2/P8)*C8;
157   W9_4 = C9 + ceil(W9_3/P1)*C1 + ceil(W9_3/P2)*C2 + ceil(W9_3/P3)*C3 ...
158         + ceil(W9_3/P4)*C4 + ceil(W9_3/P5)*C5 + ceil(W9_3/P6)*C6 ...
159         + ceil(W9_3/P7)*C7 + ceil(W9_3/P8)*C8;
160   W9_5 = C9 + ceil(W9_4/P1)*C1 + ceil(W9_4/P2)*C2 + ceil(W9_4/P3)*C3 ...
161         + ceil(W9_4/P4)*C4 + ceil(W9_4/P5)*C5 + ceil(W9_4/P6)*C6 ...
162         + ceil(W9_4/P7)*C7 + ceil(W9_4/P8)*C8;
163   W9_6 = C9 + ceil(W9_5/P1)*C1 + ceil(W9_5/P2)*C2 + ceil(W9_5/P3)*C3 ...
164         + ceil(W9_5/P4)*C4 + ceil(W9_5/P5)*C5 + ceil(W9_5/P6)*C6 ...
165         + ceil(W9_5/P7)*C7 + ceil(W9_5/P8)*C8;
166   W9_7 = C9 + ceil(W9_6/P1)*C1 + ceil(W9_6/P2)*C2 + ceil(W9_6/P3)*C3 ...
167         + ceil(W9_6/P4)*C4 + ceil(W9_6/P5)*C5 + ceil(W9_6/P6)*C6 ...
168         + ceil(W9_6/P7)*C7 + ceil(W9_6/P8)*C8;
169   W9_8 = C9 + ceil(W9_7/P1)*C1 + ceil(W9_7/P2)*C2 + ceil(W9_7/P3)*C3 ...
170         + ceil(W9_7/P4)*C4 + ceil(W9_7/P5)*C5 + ceil(W9_7/P6)*C6 ...
171         + ceil(W9_7/P7)*C7 + ceil(W9_7/P8)*C8
172   W9_9 = C9 + ceil(W9_8/P1)*C1 + ceil(W9_8/P2)*C2 + ceil(W9_8/P3)*C3 ...
173         + ceil(W9_8/P4)*C4 + ceil(W9_8/P5)*C5 + ceil(W9_8/P6)*C6 ...
174         + ceil(W9_8/P7)*C7 + ceil(W9_8/P8)*C8
175
176   %TEMPO DE RESPOSTA DA TAREFA DE PRIORIDADE 10
177   W10_0  = C10;
178   W10_1  = C10 + ceil(W10_0/P1)*C1 + ceil(W10_0/P2)*C2 + ceil(W10_0/P3)*C3 ...
179         + ceil(W10_0/P4)*C4 + ceil(W10_0/P5)*C5 + ceil(W10_0/P6)*C6 ...
180         + ceil(W10_0/P7)*C7 + ceil(W10_0/P8)*C8 + ceil(W10_0/P9)*C9;
181   W10_2  = C10 + ceil(W10_1/P1)*C1 + ceil(W10_1/P2)*C2 + ceil(W10_1/P3)*C3 ...
182         + ceil(W10_1/P4)*C4 + ceil(W10_1/P5)*C5 + ceil(W10_1/P6)*C6 ...
183         + ceil(W10_1/P7)*C7 + ceil(W10_1/P8)*C8 + ceil(W10_1/P9)*C9;
```

```
W10_3  = C10 + ceil(W10_2/P1)*C1 + ceil(W10_2/P2)*C2 + ceil(W10_2/P3)*C3 ...
       + ceil(W10_2/P4)*C4 + ceil(W10_2/P5)*C5 + ceil(W10_2/P6)*C6 ...
       + ceil(W10_2/P7)*C7 + ceil(W10_2/P8)*C8 + ceil(W10_2/P9)*C9;
W10_4  = C10 + ceil(W10_3/P1)*C1 + ceil(W10_3/P2)*C2 + ceil(W10_3/P3)*C3 ...
       + ceil(W10_3/P4)*C4 + ceil(W10_3/P5)*C5 + ceil(W10_3/P6)*C6 ...
       + ceil(W10_3/P7)*C7 + ceil(W10_3/P8)*C8 + ceil(W10_3/P9)*C9;
W10_5  = C10 + ceil(W10_4/P1)*C1 + ceil(W10_4/P2)*C2 + ceil(W10_4/P3)*C3 ...
       + ceil(W10_4/P4)*C4 + ceil(W10_4/P5)*C5 + ceil(W10_4/P6)*C6 ...
       + ceil(W10_4/P7)*C7 + ceil(W10_4/P8)*C8 + ceil(W10_4/P9)*C9;
W10_6  = C10 + ceil(W10_5/P1)*C1 + ceil(W10_5/P2)*C2 + ceil(W10_5/P3)*C3 ...
       + ceil(W10_5/P4)*C4 + ceil(W10_5/P5)*C5 + ceil(W10_5/P6)*C6 ...
       + ceil(W10_5/P7)*C7 + ceil(W10_5/P8)*C8 + ceil(W10_5/P9)*C9;
W10_7  = C10 + ceil(W10_6/P1)*C1 + ceil(W10_6/P2)*C2 + ceil(W10_6/P3)*C3 ...
       + ceil(W10_6/P4)*C4 + ceil(W10_6/P5)*C5 + ceil(W10_6/P6)*C6 ...
       + ceil(W10_6/P7)*C7 + ceil(W10_6/P8)*C8 + ceil(W10_6/P9)*C9;
W10_8  = C10 + ceil(W10_7/P1)*C1 + ceil(W10_7/P2)*C2 + ceil(W10_7/P3)*C3 ...
       + ceil(W10_7/P4)*C4 + ceil(W10_7/P5)*C5 + ceil(W10_7/P6)*C6 ...
       + ceil(W10_7/P7)*C7 + ceil(W10_7/P8)*C8 + ceil(W10_7/P9)*C9;
W10_9  = C10 + ceil(W10_8/P1)*C1 + ceil(W10_8/P2)*C2 + ceil(W10_8/P3)*C3 ...
       + ceil(W10_8/P4)*C4 + ceil(W10_8/P5)*C5 + ceil(W10_8/P6)*C6 ...
       + ceil(W10_8/P7)*C7 + ceil(W10_8/P8)*C8 + ceil(W10_8/P9)*C9
W10_10 = C10 + ceil(W10_9/P1)*C1 + ceil(W10_9/P2)*C2 + ceil(W10_9/P3)*C3 ...
       + ceil(W10_9/P4)*C4 + ceil(W10_9/P5)*C5 + ceil(W10_9/P6)*C6 ...
       + ceil(W10_9/P7)*C7 + ceil(W10_9/P8)*C8 + ceil(W10_9/P9)*C9

%TEMPO DE RESPOSTA DA TAREFA DE PRIORIDADE 11
W11_0  = C11;
W11_1  = C11 + ceil(W11_0/P1)*C1  + ceil(W11_0/P2)*C2  + ceil(W11_0/P3)*C3  ...
       + ceil(W11_0/P4)*C4  + ceil(W11_0/P5)*C5  + ceil(W11_0/P6)*C6  ...
       + ceil(W11_0/P7)*C7  + ceil(W11_0/P8)*C8  + ceil(W11_0/P9)*C9  ...
       + ceil(W11_0/P10)*C10;
W11_2  = C11 + ceil(W11_1/P1)*C1  + ceil(W11_1/P2)*C2  + ceil(W11_1/P3)*C3  ...
       + ceil(W11_1/P4)*C4  + ceil(W11_1/P5)*C5  + ceil(W11_1/P6)*C6  ...
       + ceil(W11_1/P7)*C7  + ceil(W11_1/P8)*C8  + ceil(W11_1/P9)*C9  ...
       + ceil(W11_1/P10)*C10;
W11_3  = C11 + ceil(W11_2/P1)*C1  + ceil(W11_2/P2)*C2  + ceil(W11_2/P3)*C3  ...
       + ceil(W11_2/P4)*C4  + ceil(W11_2/P5)*C5  + ceil(W11_2/P6)*C6  ...
       + ceil(W11_2/P7)*C7  + ceil(W11_2/P8)*C8  + ceil(W11_2/P9)*C9  ...
       + ceil(W11_2/P10)*C10;
W11_4  = C11 + ceil(W11_3/P1)*C1  + ceil(W11_3/P2)*C2  + ceil(W11_3/P3)*C3  ...
       + ceil(W11_3/P4)*C4  + ceil(W11_3/P5)*C5  + ceil(W11_3/P6)*C6  ...
       + ceil(W11_3/P7)*C7  + ceil(W11_3/P8)*C8  + ceil(W11_3/P9)*C9  ...
       + ceil(W11_3/P10)*C10;
W11_5  = C11 + ceil(W11_4/P1)*C1  + ceil(W11_4/P2)*C2  + ceil(W11_4/P3)*C3  ...
       + ceil(W11_4/P4)*C4  + ceil(W11_4/P5)*C5  + ceil(W11_4/P6)*C6  ...
       + ceil(W11_4/P7)*C7  + ceil(W11_4/P8)*C8  + ceil(W11_4/P9)*C9  ...
       + ceil(W11_4/P10)*C10;
W11_6  = C11 + ceil(W11_5/P1)*C1  + ceil(W11_5/P2)*C2  + ceil(W11_5/P3)*C3  ...
       + ceil(W11_5/P4)*C4  + ceil(W11_5/P5)*C5  + ceil(W11_5/P6)*C6  ...
       + ceil(W11_5/P7)*C7  + ceil(W11_5/P8)*C8  + ceil(W11_5/P9)*C9  ...
       + ceil(W11_5/P10)*C10;
W11_7  = C11 + ceil(W11_6/P1)*C1  + ceil(W11_6/P2)*C2  + ceil(W11_6/P3)*C3  ...
       + ceil(W11_6/P4)*C4  + ceil(W11_6/P5)*C5  + ceil(W11_6/P6)*C6  ...
       + ceil(W11_6/P7)*C7  + ceil(W11_6/P8)*C8  + ceil(W11_6/P9)*C9  ...
       + ceil(W11_6/P10)*C10;
W11_8  = C11 + ceil(W11_7/P1)*C1  + ceil(W11_7/P2)*C2  + ceil(W11_7/P3)*C3  ...
       + ceil(W11_7/P4)*C4  + ceil(W11_7/P5)*C5  + ceil(W11_7/P6)*C6  ...
       + ceil(W11_7/P7)*C7  + ceil(W11_7/P8)*C8  + ceil(W11_7/P9)*C9  ...
       + ceil(W11_7/P10)*C10;
```

```
W11_9  = C11 + ceil(W11_8/P1)*C1  + ceil(W11_8/P2)*C2  + ceil(W11_8/P3)*C3  ...
       + ceil(W11_8/P4)*C4  + ceil(W11_8/P5)*C5  + ceil(W11_8/P6)*C6  ...
       + ceil(W11_8/P7)*C7  + ceil(W11_8/P8)*C8  + ceil(W11_8/P9)*C9  ...
       + ceil(W11_8/P10)*C10;
W11_10 = C11 + ceil(W11_9/P1)*C1  + ceil(W11_9/P2)*C2  + ceil(W11_9/P3)*C3  ...
       + ceil(W11_9/P4)*C4  + ceil(W11_9/P5)*C5  + ceil(W11_9/P6)*C6  ...
       + ceil(W11_9/P7)*C7  + ceil(W11_9/P8)*C8  + ceil(W11_9/P9)*C9  ...
       + ceil(W11_9/P10)*C10
W11_11 = C11 + ceil(W11_10/P1)*C1 + ceil(W11_10/P2)*C2 + ceil(W11_10/P3)*C3 ...
       + ceil(W11_10/P4)*C4 + ceil(W11_10/P5)*C5 + ceil(W11_10/P6)*C6 ...
       + ceil(W11_10/P7)*C7 + ceil(W11_10/P8)*C8 + ceil(W11_10/P9)*C9 ...
       + ceil(W11_10/P10)*C10

%TEMPO DE RESPOSTA DA TAREFA DE PRIORIDADE 12
W12_0  = C12;
W12_1  = C12 + ceil(W12_0/P1)*C1  + ceil(W12_0/P2)*C2  + ceil(W12_0/P3)*C3  ...
       + ceil(W12_0/P4)*C4  + ceil(W12_0/P5)*C5  + ceil(W12_0/P6)*C6  ...
       + ceil(W12_0/P7)*C7  + ceil(W12_0/P8)*C8  + ceil(W12_0/P9)*C9  ...
       + ceil(W12_0/P10)*C10  + ceil(W12_0/P11)*C11;
W12_2  = C12 + ceil(W12_1/P1)*C1  + ceil(W12_1/P2)*C2  + ceil(W12_1/P3)*C3  ...
       + ceil(W12_1/P4)*C4  + ceil(W12_1/P5)*C5  + ceil(W12_1/P6)*C6  ...
       + ceil(W12_1/P7)*C7  + ceil(W12_1/P8)*C8  + ceil(W12_1/P9)*C9  ...
       + ceil(W12_1/P10)*C10  + ceil(W12_1/P11)*C11;
W12_3  = C12 + ceil(W12_2/P1)*C1  + ceil(W12_2/P2)*C2  + ceil(W12_2/P3)*C3  ...
       + ceil(W12_2/P4)*C4  + ceil(W12_2/P5)*C5  + ceil(W12_2/P6)*C6  ...
       + ceil(W12_2/P7)*C7  + ceil(W12_2/P8)*C8  + ceil(W12_2/P9)*C9  ...
       + ceil(W12_2/P10)*C10  + ceil(W12_2/P11)*C11;
W12_4  = C12 + ceil(W12_3/P1)*C1  + ceil(W12_3/P2)*C2  + ceil(W12_3/P3)*C3  ...
       + ceil(W12_3/P4)*C4  + ceil(W12_3/P5)*C5  + ceil(W12_3/P6)*C6  ...
       + ceil(W12_3/P7)*C7  + ceil(W12_3/P8)*C8  + ceil(W12_3/P9)*C9  ...
       + ceil(W12_3/P10)*C10  + ceil(W12_3/P11)*C11;
W12_5  = C12 + ceil(W12_4/P1)*C1  + ceil(W12_4/P2)*C2  + ceil(W12_4/P3)*C3  ...
       + ceil(W12_4/P4)*C4  + ceil(W12_4/P5)*C5  + ceil(W12_4/P6)*C6  ...
       + ceil(W12_4/P7)*C7  + ceil(W12_4/P8)*C8  + ceil(W12_4/P9)*C9  ...
       + ceil(W12_4/P10)*C10  + ceil(W12_4/P11)*C11;
W12_6  = C12 + ceil(W12_5/P1)*C1  + ceil(W12_5/P2)*C2  + ceil(W12_5/P3)*C3  ...
       + ceil(W12_5/P4)*C4  + ceil(W12_5/P5)*C5  + ceil(W12_5/P6)*C6  ...
       + ceil(W12_5/P7)*C7  + ceil(W12_5/P8)*C8  + ceil(W12_5/P9)*C9  ...
       + ceil(W12_5/P10)*C10  + ceil(W12_5/P11)*C11;
W12_7  = C12 + ceil(W12_6/P1)*C1  + ceil(W12_6/P2)*C2  + ceil(W12_6/P3)*C3  ...
       + ceil(W12_6/P4)*C4  + ceil(W12_6/P5)*C5  + ceil(W12_6/P6)*C6  ...
       + ceil(W12_6/P7)*C7  + ceil(W12_6/P8)*C8  + ceil(W12_6/P9)*C9  ...
       + ceil(W12_6/P10)*C10  + ceil(W12_6/P11)*C11;
W12_8  = C12 + ceil(W12_7/P1)*C1  + ceil(W12_7/P2)*C2  + ceil(W12_7/P3)*C3  ...
       + ceil(W12_7/P4)*C4  + ceil(W12_7/P5)*C5  + ceil(W12_7/P6)*C6  ...
       + ceil(W12_7/P7)*C7  + ceil(W12_7/P8)*C8  + ceil(W12_7/P9)*C9  ...
       + ceil(W12_7/P10)*C10  + ceil(W12_7/P11)*C11;
W12_9  = C12 + ceil(W12_8/P1)*C1  + ceil(W12_8/P2)*C2  + ceil(W12_8/P3)*C3  ...
       + ceil(W12_8/P4)*C4  + ceil(W12_8/P5)*C5  + ceil(W12_8/P6)*C6  ...
       + ceil(W12_8/P7)*C7  + ceil(W12_8/P8)*C8  + ceil(W12_8/P9)*C9  ...
       + ceil(W12_8/P10)*C10  + ceil(W12_8/P11)*C11;
W12_10 = C12 + ceil(W12_9/P1)*C1  + ceil(W12_9/P2)*C2  + ceil(W12_9/P3)*C3  ...
       + ceil(W12_9/P4)*C4  + ceil(W12_9/P5)*C5  + ceil(W12_9/P6)*C6  ...
       + ceil(W12_9/P7)*C7  + ceil(W12_9/P8)*C8  + ceil(W12_9/P9)*C9  ...
       + ceil(W12_9/P10)*C10  + ceil(W12_9/P11)*C11;
W12_11 = C12 + ceil(W12_10/P1)*C1 + ceil(W12_10/P2)*C2 + ceil(W12_10/P3)*C3 ...
       + ceil(W12_10/P4)*C4 + ceil(W12_10/P5)*C5 + ceil(W12_10/P6)*C6 ...
       + ceil(W12_10/P7)*C7 + ceil(W12_10/P8)*C8 + ceil(W12_10/P9)*C9 ...
       + ceil(W12_10/P10)*C10 + ceil(W12_10/P11)*C11;
```

```
W12_12 = C12 + ceil(W12_11/P1)*C1 + ceil(W12_11/P2)*C2 + ceil(W12_11/P3)*C3 ...
       + ceil(W12_11/P4)*C4 + ceil(W12_11/P5)*C5 + ceil(W12_11/P6)*C6 ...
       + ceil(W12_11/P7)*C7 + ceil(W12_11/P8)*C8 + ceil(W12_11/P9)*C9 ...
       + ceil(W12_11/P10)*C10 + ceil(W12_11/P11)*C11
W12_13 = C12 + ceil(W12_12/P1)*C1 + ceil(W12_12/P2)*C2 + ceil(W12_12/P3)*C3 ...
       + ceil(W12_12/P4)*C4 + ceil(W12_12/P5)*C5 + ceil(W12_12/P6)*C6 ...
       + ceil(W12_12/P7)*C7 + ceil(W12_12/P8)*C8 + ceil(W12_12/P9)*C9 ...
       + ceil(W12_12/P10)*C10 + ceil(W12_12/P11)*C11

%TEMPO DE RESPOSTA DA TAREFA DE PRIORIDADE 13
W13_0  = C13;
W13_1  = C13 + ceil(W13_0/P1)*C1  + ceil(W13_0/P2)*C2  + ceil(W13_0/P3)*C3  ...
       + ceil(W13_0/P4)*C4  + ceil(W13_0/P5)*C5  + ceil(W13_0/P6)*C6  ...
       + ceil(W13_0/P7)*C7  + ceil(W13_0/P8)*C8  + ceil(W13_0/P9)*C9  ...
       + ceil(W13_0/P10)*C10  + ceil(W13_0/P11)*C11  + ceil(W13_0/P12)*C12;
W13_2  = C13 + ceil(W13_1/P1)*C1  + ceil(W13_1/P2)*C2  + ceil(W13_1/P3)*C3  ...
       + ceil(W13_1/P4)*C4  + ceil(W13_1/P5)*C5  + ceil(W13_1/P6)*C6  ...
       + ceil(W13_1/P7)*C7  + ceil(W13_1/P8)*C8  + ceil(W13_1/P9)*C9  ...
       + ceil(W13_1/P10)*C10  + ceil(W13_1/P11)*C11  + ceil(W13_1/P12)*C12;
W13_3  = C13 + ceil(W13_2/P1)*C1  + ceil(W13_2/P2)*C2  + ceil(W13_2/P3)*C3  ...
       + ceil(W13_2/P4)*C4  + ceil(W13_2/P5)*C5  + ceil(W13_2/P6)*C6  ...
       + ceil(W13_2/P7)*C7  + ceil(W13_2/P8)*C8  + ceil(W13_2/P9)*C9  ...
       + ceil(W13_2/P10)*C10  + ceil(W13_2/P11)*C11  + ceil(W13_2/P12)*C12;
W13_4  = C13 + ceil(W13_3/P1)*C1  + ceil(W13_3/P2)*C2  + ceil(W13_3/P3)*C3  ...
       + ceil(W13_3/P4)*C4  + ceil(W13_3/P5)*C5  + ceil(W13_3/P6)*C6  ...
       + ceil(W13_3/P7)*C7  + ceil(W13_3/P8)*C8  + ceil(W13_3/P9)*C9  ...
       + ceil(W13_3/P10)*C10  + ceil(W13_3/P11)*C11  + ceil(W13_3/P12)*C12;
W13_5  = C13 + ceil(W13_4/P1)*C1  + ceil(W13_4/P2)*C2  + ceil(W13_4/P3)*C3  ...
       + ceil(W13_4/P4)*C4  + ceil(W13_4/P5)*C5  + ceil(W13_4/P6)*C6  ...
       + ceil(W13_4/P7)*C7  + ceil(W13_4/P8)*C8  + ceil(W13_4/P9)*C9  ...
       + ceil(W13_4/P10)*C10  + ceil(W13_4/P11)*C11  + ceil(W13_4/P12)*C12;
W13_6  = C13 + ceil(W13_5/P1)*C1  + ceil(W13_5/P2)*C2  + ceil(W13_5/P3)*C3  ...
       + ceil(W13_5/P4)*C4  + ceil(W13_5/P5)*C5  + ceil(W13_5/P6)*C6  ...
       + ceil(W13_5/P7)*C7  + ceil(W13_5/P8)*C8  + ceil(W13_5/P9)*C9  ...
       + ceil(W13_5/P10)*C10  + ceil(W13_5/P11)*C11  + ceil(W13_5/P12)*C12;
W13_7  = C13 + ceil(W13_6/P1)*C1  + ceil(W13_6/P2)*C2  + ceil(W13_6/P3)*C3  ...
       + ceil(W13_6/P4)*C4  + ceil(W13_6/P5)*C5  + ceil(W13_6/P6)*C6  ...
       + ceil(W13_6/P7)*C7  + ceil(W13_6/P8)*C8  + ceil(W13_6/P9)*C9  ...
       + ceil(W13_6/P10)*C10  + ceil(W13_6/P11)*C11  + ceil(W13_6/P12)*C12;
W13_8  = C13 + ceil(W13_7/P1)*C1  + ceil(W13_7/P2)*C2  + ceil(W13_7/P3)*C3  ...
       + ceil(W13_7/P4)*C4  + ceil(W13_7/P5)*C5  + ceil(W13_7/P6)*C6  ...
       + ceil(W13_7/P7)*C7  + ceil(W13_7/P8)*C8  + ceil(W13_7/P9)*C9  ...
       + ceil(W13_7/P10)*C10  + ceil(W13_7/P11)*C11  + ceil(W13_7/P12)*C12;
W13_9  = C13 + ceil(W13_8/P1)*C1  + ceil(W13_8/P2)*C2  + ceil(W13_8/P3)*C3  ...
       + ceil(W13_8/P4)*C4  + ceil(W13_8/P5)*C5  + ceil(W13_8/P6)*C6  ...
       + ceil(W13_8/P7)*C7  + ceil(W13_8/P8)*C8  + ceil(W13_8/P9)*C9  ...
       + ceil(W13_8/P10)*C10  + ceil(W13_8/P11)*C11  + ceil(W13_8/P12)*C12;
W13_10 = C13 + ceil(W13_9/P1)*C1  + ceil(W13_9/P2)*C2  + ceil(W13_9/P3)*C3  ...
       + ceil(W13_9/P4)*C4  + ceil(W13_9/P5)*C5  + ceil(W13_9/P6)*C6  ...
       + ceil(W13_9/P7)*C7  + ceil(W13_9/P8)*C8  + ceil(W13_9/P9)*C9  ...
       + ceil(W13_9/P10)*C10  + ceil(W13_9/P11)*C11  + ceil(W13_9/P12)*C12;
W13_11 = C13 + ceil(W13_10/P1)*C1 + ceil(W13_10/P2)*C2 + ceil(W13_10/P3)*C3 ...
       + ceil(W13_10/P4)*C4 + ceil(W13_10/P5)*C5 + ceil(W13_10/P6)*C6 ...
       + ceil(W13_10/P7)*C7 + ceil(W13_10/P8)*C8 + ceil(W13_10/P9)*C9 ...
       + ceil(W13_10/P10)*C10 + ceil(W13_10/P11)*C11 + ceil(W13_10/P12)*C12;
W13_12 = C13 + ceil(W13_11/P1)*C1 + ceil(W13_11/P2)*C2 + ceil(W13_11/P3)*C3 ...
       + ceil(W13_11/P4)*C4 + ceil(W13_11/P5)*C5 + ceil(W13_11/P6)*C6 ...
       + ceil(W13_11/P7)*C7 + ceil(W13_11/P8)*C8 + ceil(W13_11/P9)*C9 ...
       + ceil(W13_11/P10)*C10 + ceil(W13_11/P11)*C11 + ceil(W13_11/P12)*C12;
```

```
W13_13 = C13 + ceil(W13_12/P1)*C1 + ceil(W13_12/P2)*C2 + ceil(W13_12/P3)*C3 ...
       + ceil(W13_12/P4)*C4 + ceil(W13_12/P5)*C5 + ceil(W13_12/P6)*C6 ...
       + ceil(W13_12/P7)*C7 + ceil(W13_12/P8)*C8 + ceil(W13_12/P9)*C9 ...
       + ceil(W13_12/P10)*C10 + ceil(W13_12/P11)*C11 + ceil(W13_12/P12)*C12;
W13_14 = C13 + ceil(W13_13/P1)*C1 + ceil(W13_13/P2)*C2 + ceil(W13_13/P3)*C3 ...
       + ceil(W13_13/P4)*C4 + ceil(W13_13/P5)*C5 + ceil(W13_13/P6)*C6 ...
       + ceil(W13_13/P7)*C7 + ceil(W13_13/P8)*C8 + ceil(W13_13/P9)*C9 ...
       + ceil(W13_13/P10)*C10 + ceil(W13_13/P11)*C11 + ceil(W13_13/P12)*C12;
W13_15 = C13 + ceil(W13_14/P1)*C1 + ceil(W13_14/P2)*C2 + ceil(W13_14/P3)*C3 ...
       + ceil(W13_14/P4)*C4 + ceil(W13_14/P5)*C5 + ceil(W13_14/P6)*C6 ...
       + ceil(W13_14/P7)*C7 + ceil(W13_14/P8)*C8 + ceil(W13_14/P9)*C9 ...
       + ceil(W13_14/P10)*C10 + ceil(W13_14/P11)*C11 + ceil(W13_14/P12)*C12;
W13_16 = C13 + ceil(W13_15/P1)*C1 + ceil(W13_15/P2)*C2 + ceil(W13_15/P3)*C3 ...
       + ceil(W13_15/P4)*C4 + ceil(W13_15/P5)*C5 + ceil(W13_15/P6)*C6 ...
       + ceil(W13_15/P7)*C7 + ceil(W13_15/P8)*C8 + ceil(W13_15/P9)*C9 ...
       + ceil(W13_15/P10)*C10 + ceil(W13_15/P11)*C11 + ceil(W13_15/P12)*C12
W13_17 = C13 + ceil(W13_16/P1)*C1 + ceil(W13_16/P2)*C2 + ceil(W13_16/P3)*C3 ...
       + ceil(W13_16/P4)*C4 + ceil(W13_16/P5)*C5 + ceil(W13_16/P6)*C6 ...
       + ceil(W13_16/P7)*C7 + ceil(W13_16/P8)*C8 + ceil(W13_16/P9)*C9 ...
       + ceil(W13_16/P10)*C10 + ceil(W13_16/P11)*C11 + ceil(W13_16/P12)*C12

%TEMPO DE RESPOSTA DA TAREFA DE PRIORIDADE 14
W14_0  = C14;
W14_1  = C14 + ceil(W14_0/P1)*C1  + ceil(W14_0/P2)*C2  + ceil(W14_0/P3)*C3  ...
       + ceil(W14_0/P4)*C4  + ceil(W14_0/P5)*C5  + ceil(W14_0/P6)*C6  ...
       + ceil(W14_0/P7)*C7  + ceil(W14_0/P8)*C8  + ceil(W14_0/P9)*C9  ...
       + ceil(W14_0/P10)*C10  + ceil(W14_0/P11)*C11  + ceil(W14_0/P12)*C12  ...
       + ceil(W14_0/P13)*C13;
W14_2  = C14 + ceil(W14_1/P1)*C1  + ceil(W14_1/P2)*C2  + ceil(W14_1/P3)*C3  ...
       + ceil(W14_1/P4)*C4  + ceil(W14_1/P5)*C5  + ceil(W14_1/P6)*C6  ...
       + ceil(W14_1/P7)*C7  + ceil(W14_1/P8)*C8  + ceil(W14_1/P9)*C9  ...
       + ceil(W14_1/P10)*C10  + ceil(W14_1/P11)*C11  + ceil(W14_1/P12)*C12  ...
       + ceil(W14_1/P13)*C13;
W14_3  = C14 + ceil(W14_2/P1)*C1  + ceil(W14_2/P2)*C2  + ceil(W14_2/P3)*C3  ...
       + ceil(W14_2/P4)*C4  + ceil(W14_2/P5)*C5  + ceil(W14_2/P6)*C6  ...
       + ceil(W14_2/P7)*C7  + ceil(W14_2/P8)*C8  + ceil(W14_2/P9)*C9  ...
       + ceil(W14_2/P10)*C10  + ceil(W14_2/P11)*C11  + ceil(W14_2/P12)*C12  ...
       + ceil(W14_2/P13)*C13;
W14_4  = C14 + ceil(W14_3/P1)*C1  + ceil(W14_3/P2)*C2  + ceil(W14_3/P3)*C3  ...
       + ceil(W14_3/P4)*C4  + ceil(W14_3/P5)*C5  + ceil(W14_3/P6)*C6  ...
       + ceil(W14_3/P7)*C7  + ceil(W14_3/P8)*C8  + ceil(W14_3/P9)*C9  ...
       + ceil(W14_3/P10)*C10  + ceil(W14_3/P11)*C11  + ceil(W14_3/P12)*C12  ...
       + ceil(W14_3/P13)*C13;
W14_5  = C14 + ceil(W14_4/P1)*C1  + ceil(W14_4/P2)*C2  + ceil(W14_4/P3)*C3  ...
       + ceil(W14_4/P4)*C4  + ceil(W14_4/P5)*C5  + ceil(W14_4/P6)*C6  ...
       + ceil(W14_4/P7)*C7  + ceil(W14_4/P8)*C8  + ceil(W14_4/P9)*C9  ...
       + ceil(W14_4/P10)*C10  + ceil(W14_4/P11)*C11  + ceil(W14_4/P12)*C12  ...
       + ceil(W14_4/P13)*C13;
W14_6  = C14 + ceil(W14_5/P1)*C1  + ceil(W14_5/P2)*C2  + ceil(W14_5/P3)*C3  ...
       + ceil(W14_5/P4)*C4  + ceil(W14_5/P5)*C5  + ceil(W14_5/P6)*C6  ...
       + ceil(W14_5/P7)*C7  + ceil(W14_5/P8)*C8  + ceil(W14_5/P9)*C9  ...
       + ceil(W14_5/P10)*C10  + ceil(W14_5/P11)*C11  + ceil(W14_5/P12)*C12  ...
       + ceil(W14_5/P13)*C13;
W14_7  = C14 + ceil(W14_6/P1)*C1  + ceil(W14_6/P2)*C2  + ceil(W14_6/P3)*C3  ...
       + ceil(W14_6/P4)*C4  + ceil(W14_6/P5)*C5  + ceil(W14_6/P6)*C6  ...
       + ceil(W14_6/P7)*C7  + ceil(W14_6/P8)*C8  + ceil(W14_6/P9)*C9  ...
       + ceil(W14_6/P10)*C10  + ceil(W14_6/P11)*C11  + ceil(W14_6/P12)*C12  ...
       + ceil(W14_6/P13)*C13;
W14_8  = C14 + ceil(W14_7/P1)*C1  + ceil(W14_7/P2)*C2  + ceil(W14_7/P3)*C3  ...
```

```
420         + ceil(W14_7/P4)*C4   + ceil(W14_7/P5)*C5   + ceil(W14_7/P6)*C6   ...
421         + ceil(W14_7/P7)*C7   + ceil(W14_7/P8)*C8   + ceil(W14_7/P9)*C9   ...
422         + ceil(W14_7/P10)*C10   + ceil(W14_7/P11)*C11   + ceil(W14_7/P12)*C12   ...
423         + ceil(W14_7/P13)*C13;
424 W14_9   = C14 + ceil(W14_8/P1)*C1   + ceil(W14_8/P2)*C2   + ceil(W14_8/P3)*C3   ...
425         + ceil(W14_8/P4)*C4   + ceil(W14_8/P5)*C5   + ceil(W14_8/P6)*C6   ...
426         + ceil(W14_8/P7)*C7   + ceil(W14_8/P8)*C8   + ceil(W14_8/P9)*C9   ...
427         + ceil(W14_8/P10)*C10   + ceil(W14_8/P11)*C11   + ceil(W14_8/P12)*C12   ...
428         + ceil(W14_8/P13)*C13;
429 W14_10 = C14 + ceil(W14_9/P1)*C1   + ceil(W14_9/P2)*C2   + ceil(W14_9/P3)*C3   ...
430         + ceil(W14_9/P4)*C4   + ceil(W14_9/P5)*C5   + ceil(W14_9/P6)*C6   ...
431         + ceil(W14_9/P7)*C7   + ceil(W14_9/P8)*C8   + ceil(W14_9/P9)*C9   ...
432         + ceil(W14_9/P10)*C10   + ceil(W14_9/P11)*C11   + ceil(W14_9/P12)*C12   ...
433         + ceil(W14_9/P13)*C13;
434 W14_11 = C14 + ceil(W14_10/P1)*C1 + ceil(W14_10/P2)*C2 + ceil(W14_10/P3)*C3 ...
435         + ceil(W14_10/P4)*C4 + ceil(W14_10/P5)*C5 + ceil(W14_10/P6)*C6 ...
436         + ceil(W14_10/P7)*C7 + ceil(W14_10/P8)*C8 + ceil(W14_10/P9)*C9 ...
437         + ceil(W14_10/P10)*C10 + ceil(W14_10/P11)*C11 + ceil(W14_10/P12)*C12 ...
438         + ceil(W14_10/P13)*C13;
439 W14_12 = C14 + ceil(W14_11/P1)*C1 + ceil(W14_11/P2)*C2 + ceil(W14_11/P3)*C3 ...
440         + ceil(W14_11/P4)*C4 + ceil(W14_11/P5)*C5 + ceil(W14_11/P6)*C6 ...
441         + ceil(W14_11/P7)*C7 + ceil(W14_11/P8)*C8 + ceil(W14_11/P9)*C9 ...
442         + ceil(W14_11/P10)*C10 + ceil(W14_11/P11)*C11 + ceil(W14_11/P12)*C12 ...
443         + ceil(W14_11/P13)*C13;
444 W14_13 = C14 + ceil(W14_12/P1)*C1 + ceil(W14_12/P2)*C2 + ceil(W14_12/P3)*C3 ...
445         + ceil(W14_12/P4)*C4 + ceil(W14_12/P5)*C5 + ceil(W14_12/P6)*C6 ...
446         + ceil(W14_12/P7)*C7 + ceil(W14_12/P8)*C8 + ceil(W14_12/P9)*C9 ...
447         + ceil(W14_12/P10)*C10 + ceil(W14_12/P11)*C11 + ceil(W14_12/P12)*C12 ...
448         + ceil(W14_12/P13)*C13;
449 W14_14 = C14 + ceil(W14_13/P1)*C1 + ceil(W14_13/P2)*C2 + ceil(W14_13/P3)*C3 ...
450         + ceil(W14_13/P4)*C4 + ceil(W14_13/P5)*C5 + ceil(W14_13/P6)*C6 ...
451         + ceil(W14_13/P7)*C7 + ceil(W14_13/P8)*C8 + ceil(W14_13/P9)*C9 ...
452         + ceil(W14_13/P10)*C10 + ceil(W14_13/P11)*C11 + ceil(W14_13/P12)*C12 ...
453         + ceil(W14_13/P13)*C13;
454 W14_15 = C14 + ceil(W14_14/P1)*C1 + ceil(W14_14/P2)*C2 + ceil(W14_14/P3)*C3 ...
455         + ceil(W14_14/P4)*C4 + ceil(W14_14/P5)*C5 + ceil(W14_14/P6)*C6 ...
456         + ceil(W14_14/P7)*C7 + ceil(W14_14/P8)*C8 + ceil(W14_14/P9)*C9 ...
457         + ceil(W14_14/P10)*C10 + ceil(W14_14/P11)*C11 + ceil(W14_14/P12)*C12 ...
458         + ceil(W14_14/P13)*C13;
459 W14_16 = C14 + ceil(W14_15/P1)*C1 + ceil(W14_15/P2)*C2 + ceil(W14_15/P3)*C3 ...
460         + ceil(W14_15/P4)*C4 + ceil(W14_15/P5)*C5 + ceil(W14_15/P6)*C6 ...
461         + ceil(W14_15/P7)*C7 + ceil(W14_15/P8)*C8 + ceil(W14_15/P9)*C9 ...
462         + ceil(W14_15/P10)*C10 + ceil(W14_15/P11)*C11 + ceil(W14_15/P12)*C12 ...
463         + ceil(W14_15/P13)*C13
464 W14_17 = C14 + ceil(W14_16/P1)*C1 + ceil(W14_16/P2)*C2 + ceil(W14_16/P3)*C3 ...
465         + ceil(W14_16/P4)*C4 + ceil(W14_16/P5)*C5 + ceil(W14_16/P6)*C6 ...
466         + ceil(W14_16/P7)*C7 + ceil(W14_16/P8)*C8 + ceil(W14_16/P9)*C9 ...
467         + ceil(W14_16/P10)*C10 + ceil(W14_16/P11)*C11 + ceil(W14_16/P12)*C12 ...
468         + ceil(W14_16/P13)*C13
469
470 %TEMPO DE RESPOSTA DA TAREFA DE PRIORIDADE 15
471 W15_0   = C15;
472 W15_1   = C15 + ceil(W15_0/P1)*C1   + ceil(W15_0/P2)*C2   + ceil(W15_0/P3)*C3   ...
473         + ceil(W15_0/P4)*C4   + ceil(W15_0/P5)*C5   + ceil(W15_0/P6)*C6   ...
474         + ceil(W15_0/P7)*C7   + ceil(W15_0/P8)*C8   + ceil(W15_0/P9)*C9   ...
475         + ceil(W15_0/P10)*C10   + ceil(W15_0/P11)*C11   + ceil(W15_0/P12)*C12   ...
476         + ceil(W15_0/P13)*C13   + ceil(W15_0/P14)*C14;
477 W15_2   = C15 + ceil(W15_1/P1)*C1   + ceil(W15_1/P2)*C2   + ceil(W15_1/P3)*C3   ...
478         + ceil(W15_1/P4)*C4   + ceil(W15_1/P5)*C5   + ceil(W15_1/P6)*C6   ...
```

```
479         + ceil(W15_1/P7)*C7   + ceil(W15_1/P8)*C8   + ceil(W15_1/P9)*C9   ...
480         + ceil(W15_1/P10)*C10   + ceil(W15_1/P11)*C11   + ceil(W15_1/P12)*C12   ...
481         + ceil(W15_1/P13)*C13   + ceil(W15_1/P14)*C14;
482   W15_3   = C15 + ceil(W15_2/P1)*C1   + ceil(W15_2/P2)*C2   + ceil(W15_2/P3)*C3   ...
483         + ceil(W15_2/P4)*C4   + ceil(W15_2/P5)*C5   + ceil(W15_2/P6)*C6   ...
484         + ceil(W15_2/P7)*C7   + ceil(W15_2/P8)*C8   + ceil(W15_2/P9)*C9   ...
485         + ceil(W15_2/P10)*C10   + ceil(W15_2/P11)*C11   + ceil(W15_2/P12)*C12   ...
486         + ceil(W15_2/P13)*C13   + ceil(W15_2/P14)*C14;
487   W15_4   = C15 + ceil(W15_3/P1)*C1   + ceil(W15_3/P2)*C2   + ceil(W15_3/P3)*C3   ...
488         + ceil(W15_3/P4)*C4   + ceil(W15_3/P5)*C5   + ceil(W15_3/P6)*C6   ...
489         + ceil(W15_3/P7)*C7   + ceil(W15_3/P8)*C8   + ceil(W15_3/P9)*C9   ...
490         + ceil(W15_3/P10)*C10   + ceil(W15_3/P11)*C11   + ceil(W15_3/P12)*C12   ...
491         + ceil(W15_3/P13)*C13   + ceil(W15_3/P14)*C14;
492   W15_5   = C15 + ceil(W15_4/P1)*C1   + ceil(W15_4/P2)*C2   + ceil(W15_4/P3)*C3   ...
493         + ceil(W15_4/P4)*C4   + ceil(W15_4/P5)*C5   + ceil(W15_4/P6)*C6   ...
494         + ceil(W15_4/P7)*C7   + ceil(W15_4/P8)*C8   + ceil(W15_4/P9)*C9   ...
495         + ceil(W15_4/P10)*C10   + ceil(W15_4/P11)*C11   + ceil(W15_4/P12)*C12   ...
496         + ceil(W15_4/P13)*C13   + ceil(W15_4/P14)*C14;
497   W15_6   = C15 + ceil(W15_5/P1)*C1   + ceil(W15_5/P2)*C2   + ceil(W15_5/P3)*C3   ...
498         + ceil(W15_5/P4)*C4   + ceil(W15_5/P5)*C5   + ceil(W15_5/P6)*C6   ...
499         + ceil(W15_5/P7)*C7   + ceil(W15_5/P8)*C8   + ceil(W15_5/P9)*C9   ...
500         + ceil(W15_5/P10)*C10   + ceil(W15_5/P11)*C11   + ceil(W15_5/P12)*C12   ...
501         + ceil(W15_5/P13)*C13   + ceil(W15_5/P14)*C14;
502   W15_7   = C15 + ceil(W15_6/P1)*C1   + ceil(W15_6/P2)*C2   + ceil(W15_6/P3)*C3   ...
503         + ceil(W15_6/P4)*C4   + ceil(W15_6/P5)*C5   + ceil(W15_6/P6)*C6   ...
504         + ceil(W15_6/P7)*C7   + ceil(W15_6/P8)*C8   + ceil(W15_6/P9)*C9   ...
505         + ceil(W15_6/P10)*C10   + ceil(W15_6/P11)*C11   + ceil(W15_6/P12)*C12   ...
506         + ceil(W15_6/P13)*C13   + ceil(W15_6/P14)*C14;
507   W15_8   = C15 + ceil(W15_7/P1)*C1   + ceil(W15_7/P2)*C2   + ceil(W15_7/P3)*C3   ...
508         + ceil(W15_7/P4)*C4   + ceil(W15_7/P5)*C5   + ceil(W15_7/P6)*C6   ...
509         + ceil(W15_7/P7)*C7   + ceil(W15_7/P8)*C8   + ceil(W15_7/P9)*C9   ...
510         + ceil(W15_7/P10)*C10   + ceil(W15_7/P11)*C11   + ceil(W15_7/P12)*C12   ...
511         + ceil(W15_7/P13)*C13   + ceil(W15_7/P14)*C14;
512   W15_9   = C15 + ceil(W15_8/P1)*C1   + ceil(W15_8/P2)*C2   + ceil(W15_8/P3)*C3   ...
513         + ceil(W15_8/P4)*C4   + ceil(W15_8/P5)*C5   + ceil(W15_8/P6)*C6   ...
514         + ceil(W15_8/P7)*C7   + ceil(W15_8/P8)*C8   + ceil(W15_8/P9)*C9   ...
515         + ceil(W15_8/P10)*C10   + ceil(W15_8/P11)*C11   + ceil(W15_8/P12)*C12   ...
516         + ceil(W15_8/P13)*C13   + ceil(W15_8/P14)*C14;
517   W15_10 = C15 + ceil(W15_9/P1)*C1   + ceil(W15_9/P2)*C2   + ceil(W15_9/P3)*C3   ...
518         + ceil(W15_9/P4)*C4   + ceil(W15_9/P5)*C5   + ceil(W15_9/P6)*C6   ...
519         + ceil(W15_9/P7)*C7   + ceil(W15_9/P8)*C8   + ceil(W15_9/P9)*C9   ...
520         + ceil(W15_9/P10)*C10   + ceil(W15_9/P11)*C11   + ceil(W15_9/P12)*C12   ...
521         + ceil(W15_9/P13)*C13   + ceil(W15_9/P14)*C14;
522   W15_11 = C15 + ceil(W15_10/P1)*C1 + ceil(W15_10/P2)*C2 + ceil(W15_10/P3)*C3 ...
523         + ceil(W15_10/P4)*C4 + ceil(W15_10/P5)*C5 + ceil(W15_10/P6)*C6 ...
524         + ceil(W15_10/P7)*C7 + ceil(W15_10/P8)*C8 + ceil(W15_10/P9)*C9 ...
525         + ceil(W15_10/P10)*C10 + ceil(W15_10/P11)*C11 + ceil(W15_10/P12)*C12 ...
526         + ceil(W15_10/P13)*C13 + ceil(W15_10/P14)*C14;
527   W15_12 = C15 + ceil(W15_11/P1)*C1 + ceil(W15_11/P2)*C2 + ceil(W15_11/P3)*C3 ...
528         + ceil(W15_11/P4)*C4 + ceil(W15_11/P5)*C5 + ceil(W15_11/P6)*C6 ...
529         + ceil(W15_11/P7)*C7 + ceil(W15_11/P8)*C8 + ceil(W15_11/P9)*C9 ...
530         + ceil(W15_11/P10)*C10 + ceil(W15_11/P11)*C11 + ceil(W15_11/P12)*C12 ...
531         + ceil(W15_11/P13)*C13 + ceil(W15_11/P14)*C14;
532   W15_13 = C15 + ceil(W15_12/P1)*C1 + ceil(W15_12/P2)*C2 + ceil(W15_12/P3)*C3 ...
533         + ceil(W15_12/P4)*C4 + ceil(W15_12/P5)*C5 + ceil(W15_12/P6)*C6 ...
534         + ceil(W15_12/P7)*C7 + ceil(W15_12/P8)*C8 + ceil(W15_12/P9)*C9 ...
535         + ceil(W15_12/P10)*C10 + ceil(W15_12/P11)*C11 + ceil(W15_12/P12)*C12 ...
536         + ceil(W15_12/P13)*C13 + ceil(W15_12/P14)*C14;
537   W15_14 = C15 + ceil(W15_13/P1)*C1 + ceil(W15_13/P2)*C2 + ceil(W15_13/P3)*C3 ...
```

```
538         + ceil(W15_13/P4)*C4 + ceil(W15_13/P5)*C5 + ceil(W15_13/P6)*C6 ...
539         + ceil(W15_13/P7)*C7 + ceil(W15_13/P8)*C8 + ceil(W15_13/P9)*C9 ...
540         + ceil(W15_13/P10)*C10 + ceil(W15_13/P11)*C11 + ceil(W15_13/P12)*C12 ...
541         + ceil(W15_13/P13)*C13 + ceil(W15_13/P14)*C14;
542 W15_15 = C15 + ceil(W15_14/P1)*C1 + ceil(W15_14/P2)*C2 + ceil(W15_14/P3)*C3 ...
543         + ceil(W15_14/P4)*C4 + ceil(W15_14/P5)*C5 + ceil(W15_14/P6)*C6 ...
544         + ceil(W15_14/P7)*C7 + ceil(W15_14/P8)*C8 + ceil(W15_14/P9)*C9 ...
545         + ceil(W15_14/P10)*C10 + ceil(W15_14/P11)*C11 + ceil(W15_14/P12)*C12 ...
546         + ceil(W15_14/P13)*C13 + ceil(W15_14/P14)*C14;
547 W15_16 = C15 + ceil(W15_15/P1)*C1 + ceil(W15_15/P2)*C2 + ceil(W15_15/P3)*C3 ...
548         + ceil(W15_15/P4)*C4 + ceil(W15_15/P5)*C5 + ceil(W15_15/P6)*C6 ...
549         + ceil(W15_15/P7)*C7 + ceil(W15_15/P8)*C8 + ceil(W15_15/P9)*C9 ...
550         + ceil(W15_15/P10)*C10 + ceil(W15_15/P11)*C11 + ceil(W15_15/P12)*C12 ...
551         + ceil(W15_15/P13)*C13 + ceil(W15_15/P14)*C14;
552 W15_17 = C15 + ceil(W15_16/P1)*C1 + ceil(W15_16/P2)*C2 + ceil(W15_16/P3)*C3 ...
553         + ceil(W15_16/P4)*C4 + ceil(W15_16/P5)*C5 + ceil(W15_16/P6)*C6 ...
554         + ceil(W15_16/P7)*C7 + ceil(W15_16/P8)*C8 + ceil(W15_16/P9)*C9 ...
555         + ceil(W15_16/P10)*C10 + ceil(W15_16/P11)*C11 + ceil(W15_16/P12)*C12 ...
556         + ceil(W15_16/P13)*C13 + ceil(W15_16/P14)*C14;
557 W15_18 = C15 + ceil(W15_17/P1)*C1 + ceil(W15_17/P2)*C2 + ceil(W15_17/P3)*C3 ...
558         + ceil(W15_17/P4)*C4 + ceil(W15_17/P5)*C5 + ceil(W15_17/P6)*C6 ...
559         + ceil(W15_17/P7)*C7 + ceil(W15_17/P8)*C8 + ceil(W15_17/P9)*C9 ...
560         + ceil(W15_17/P10)*C10 + ceil(W15_17/P11)*C11 + ceil(W15_17/P12)*C12 ...
561         + ceil(W15_17/P13)*C13 + ceil(W15_17/P14)*C14;
562 W15_19 = C15 + ceil(W15_18/P1)*C1 + ceil(W15_18/P2)*C2 + ceil(W15_18/P3)*C3 ...
563         + ceil(W15_18/P4)*C4 + ceil(W15_18/P5)*C5 + ceil(W15_18/P6)*C6 ...
564         + ceil(W15_18/P7)*C7 + ceil(W15_18/P8)*C8 + ceil(W15_18/P9)*C9 ...
565         + ceil(W15_18/P10)*C10 + ceil(W15_18/P11)*C11 + ceil(W15_18/P12)*C12 ...
566         + ceil(W15_18/P13)*C13 + ceil(W15_18/P14)*C14;
567 W15_20 = C15 + ceil(W15_19/P1)*C1 + ceil(W15_19/P2)*C2 + ceil(W15_19/P3)*C3 ...
568         + ceil(W15_19/P4)*C4 + ceil(W15_19/P5)*C5 + ceil(W15_19/P6)*C6 ...
569         + ceil(W15_19/P7)*C7 + ceil(W15_19/P8)*C8 + ceil(W15_19/P9)*C9 ...
570         + ceil(W15_19/P10)*C10 + ceil(W15_19/P11)*C11 + ceil(W15_19/P12)*C12 ...
571         + ceil(W15_19/P13)*C13 + ceil(W15_19/P14)*C14;
572 W15_21 = C15 + ceil(W15_20/P1)*C1 + ceil(W15_20/P2)*C2 + ceil(W15_20/P3)*C3 ...
573         + ceil(W15_20/P4)*C4 + ceil(W15_20/P5)*C5 + ceil(W15_20/P6)*C6 ...
574         + ceil(W15_20/P7)*C7 + ceil(W15_20/P8)*C8 + ceil(W15_20/P9)*C9 ...
575         + ceil(W15_20/P10)*C10 + ceil(W15_20/P11)*C11 + ceil(W15_20/P12)*C12 ...
576         + ceil(W15_20/P13)*C13 + ceil(W15_20/P14)*C14;
577 W15_22 = C15 + ceil(W15_21/P1)*C1 + ceil(W15_21/P2)*C2 + ceil(W15_21/P3)*C3 ...
578         + ceil(W15_21/P4)*C4 + ceil(W15_21/P5)*C5 + ceil(W15_21/P6)*C6 ...
579         + ceil(W15_21/P7)*C7 + ceil(W15_21/P8)*C8 + ceil(W15_21/P9)*C9 ...
580         + ceil(W15_21/P10)*C10 + ceil(W15_21/P11)*C11 + ceil(W15_21/P12)*C12 ...
581         + ceil(W15_21/P13)*C13 + ceil(W15_21/P14)*C14;
582 W15_23 = C15 + ceil(W15_22/P1)*C1 + ceil(W15_22/P2)*C2 + ceil(W15_22/P3)*C3 ...
583         + ceil(W15_22/P4)*C4 + ceil(W15_22/P5)*C5 + ceil(W15_22/P6)*C6 ...
584         + ceil(W15_22/P7)*C7 + ceil(W15_22/P8)*C8 + ceil(W15_22/P9)*C9 ...
585         + ceil(W15_22/P10)*C10 + ceil(W15_22/P11)*C11 + ceil(W15_22/P12)*C12 ...
586         + ceil(W15_22/P13)*C13 + ceil(W15_22/P14)*C14;
587 W15_24 = C15 + ceil(W15_23/P1)*C1 + ceil(W15_23/P2)*C2 + ceil(W15_23/P3)*C3 ...
588         + ceil(W15_23/P4)*C4 + ceil(W15_23/P5)*C5 + ceil(W15_23/P6)*C6 ...
589         + ceil(W15_23/P7)*C7 + ceil(W15_23/P8)*C8 + ceil(W15_23/P9)*C9 ...
590         + ceil(W15_23/P10)*C10 + ceil(W15_23/P11)*C11 + ceil(W15_23/P12)*C12 ...
591         + ceil(W15_23/P13)*C13 + ceil(W15_23/P14)*C14
592 W15_25 = C15 + ceil(W15_24/P1)*C1 + ceil(W15_24/P2)*C2 + ceil(W15_24/P3)*C3 ...
593         + ceil(W15_24/P4)*C4 + ceil(W15_24/P5)*C5 + ceil(W15_24/P6)*C6 ...
594         + ceil(W15_24/P7)*C7 + ceil(W15_24/P8)*C8 + ceil(W15_24/P9)*C9 ...
595         + ceil(W15_24/P10)*C10 + ceil(W15_24/P11)*C11 + ceil(W15_24/P12)*C12 ...
596         + ceil(W15_24/P13)*C13 + ceil(W15_24/P14)*C14
```

```
597
598 %TEMPO DE UTILIZACAO DO PROCESSADOR PARA A ESCALA INFORMADA...
599 Utilizacao = C1/P1 + C2/P2 + C3/P3 + C4/P4 + C5/P5 + C6/P6 + C7/P7 + C8/P8 ...
600            + C9/P9 + C10/P10 + C11/P11 + C12/P12 + C13/P13 + C14/P14 ...
601            + C15/P15
```

Código D.3: Resultado gerado após a execução do Código D.2.

```
1  >> calc_tempo_resposta_15_tarefas
2
3  W1_1 = 0.0049
4
5  W2_1 = 0.6550
6  W2_2 = 0.6550
7
8  W3_2 = 1.1752
9  W3_3 = 1.1752
10
11 W4_3 = 1.8253
12 W4_4 = 1.8253
13
14 W5_4 = 2.0062
15 W5_5 = 2.0062
16
17 W6_5 = 2.6563
18 W6_6 = 2.6563
19
20 W7_6 = 2.8372
21 W7_7 = 2.8372
22
23 W8_7 = 3.4146
24 W8_8 = 3.4146
25
26 W9_8 = 10.4922
27 W9_9 = 10.4922
28
29 W10_9  = 10.7251
30 W10_10 = 10.7251
31
32 W11_10 = 14.7396
33 W11_11 = 14.7396
34
35 W12_12 = 15.3170
36 W12_13 = 15.3170
37
38 W13_16 = 22.7461
39 W13_17 = 22.7461
40
41 W14_16 = 22.9270
42 W14_17 = 22.9270
43
44 W15_24 = 29.9997
45 W15_25 = 29.9997
46
47 Utilizacao = 0.9682
```

Os casos de teste criados para validação desta dissertação são tão críticos que a tarefa 15, por exemplo: possui o *deadline* igual a 30 segundos e um tempo de resposta no pior caso

(segundo o Código D.3; variáveis $W15_24$ e $W15_25$) igual a $29,9997$ segundos. Em outras palavras, o menor custo com *overheads*, seja no gerenciamento do *Raw Governor* ou do *Raw Monitor*, levariam a violação de *deadline* da tarefa 15, porém isso não aconteceu. Isso reafirma ainda mais a eficácia da metodologia proposta por esta dissertação.

Apêndice E

Processo de geração dos resultados experimentais

A geração dos resultados experimentais foram obtidas a partir da execução dos 41 casos de teste definidos na Tabela 5.2, onde a quantidade de VSPs estão associados ao eixo X e o consumo de energia e o tempo ocioso do processador estão associados ao eixo Y, ambos retornados no relatório gerado pela tarefa CPUSTAT (ver Figura F.1). Dessa forma, que foram feitas as gerações dos gráficos das Figuras 5.2 e 5.5.

Para automatizar a geração dos resultados experimentais dos 41 casos de teste foi feito um *shell script* [1] no Linux (ver Código E.1).

Código E.1: *Shell Script* feito para automatizar a geração dos resultados experimentais.

```
1 #!/bin/bash
2 #
3 # Script de execucao do experimento final
4 # Copyright rawlinson.goncalves@gmail.com
5
6 TARGET=multiple_tasks
7 RUNINFO=.runinfo
8
9 make clean && make
10
11 printf "\n\n"
12 printf "****************************************\n"
13 printf "* Rawlinson Goncalves - Mestrado - iComp  *\n"
14 printf "* Copyright rawlinson.goncalves@gmail.com *\n"
15 printf "****************************************\n"
16 printf "PROCESSANDO EXPERIMENTO \n"
17
18 #-------------------------------------------------------------------------------
19 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
20 ARGV="1 0 0 0 0"
```

[1] O *Shell Script* de geração dos resultados experimentais está disponível no *GitHub* em *rtai-raw-gov-3.9.1/EXPERIMENTOS/Experimento_ Final_ 15_ Tasks_ Valentin/script_ experimento.sh*

```
21 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
22 printf "Experimento 01 -> RAW_MONITOR[X] - PC[ ] - QTD_PC[00] - Processando..."
23 rtai-load > logs_finais/log_COM_MONITOR_SEM_PC_01.txt
24 printf " - Concluido. \n"
25 sleep 5
26
27 #-------------------------------------------------------------------------------
28 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
29 ARGV="1 1 0 0 1"
30 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
31 printf "Experimento 02 -> RAW_MONITOR[X] - PC[X] - QTD_PC[01] - Processando..."
32 rtai-load > logs_finais/log_COM_MONITOR_COM_01_PC_02.txt
33 printf " - Concluido. \n"
34 sleep 5
35
36 #-------------------------------------------------------------------------------
37 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
38 ARGV="0 1 0 0 1"
39 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
40 printf "Experimento 03 -> RAW_MONITOR[ ] - PC[X] - QTD_PC[01] - Processando..."
41 rtai-load > logs_finais/log_SEM_MONITOR_COM_01_PC_03.txt
42 printf " - Concluido. \n"
43 sleep 5
44
45 #-------------------------------------------------------------------------------
46 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
47 ARGV="1 1 0 0 2"
48 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
49 printf "Experimento 04 -> RAW_MONITOR[X] - PC[X] - QTD_PC[02] - Processando..."
50 rtai-load > logs_finais/log_COM_MONITOR_COM_02_PC_04.txt
51 printf " - Concluido. \n"
52 sleep 5
53
54 #-------------------------------------------------------------------------------
55 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
56 ARGV="0 1 0 0 2"
57 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
58 printf "Experimento 05 -> RAW_MONITOR[ ] - PC[X] - QTD_PC[02] - Processando..."
59 rtai-load > logs_finais/log_SEM_MONITOR_COM_02_PC_05.txt
60 printf " - Concluido. \n"
61 sleep 5
62
63 #-------------------------------------------------------------------------------
64 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
65 ARGV="1 1 0 0 3"
66 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
67 printf "Experimento 06 -> RAW_MONITOR[X] - PC[X] - QTD_PC[03] - Processando..."
68 rtai-load > logs_finais/log_COM_MONITOR_COM_03_PC_06.txt
69 printf " - Concluido. \n"
70 sleep 5
71
72 #-------------------------------------------------------------------------------
73 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
74 ARGV="0 1 0 0 3"
75 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
76 printf "Experimento 07 -> RAW_MONITOR[ ] - PC[X] - QTD_PC[03] - Processando..."
77 rtai-load > logs_finais/log_SEM_MONITOR_COM_03_PC_07.txt
78 printf " - Concluido. \n"
79 sleep 5
```

```
80
81 #-------------------------------------------------------------------------------
82 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
83 ARGV="1 1 0 0 4"
84 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
85 printf "Experimento 08 -> RAW_MONITOR[X] - PC[X] - QTD_PC[04] - Processando..."
86 rtai-load > logs_finais/log_COM_MONITOR_COM_04_PC_08.txt
87 printf " - Concluido. \n"
88 sleep 5
89
90 #-------------------------------------------------------------------------------
91 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
92 ARGV="0 1 0 0 4"
93 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
94 printf "Experimento 09 -> RAW_MONITOR[ ] - PC[X] - QTD_PC[04] - Processando..."
95 rtai-load > logs_finais/log_SEM_MONITOR_COM_04_PC_09.txt
96 printf " - Concluido. \n"
97 sleep 5
98
99 #-------------------------------------------------------------------------------
100 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
101 ARGV="1 1 0 0 5"
102 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
103 printf "Experimento 10 -> RAW_MONITOR[X] - PC[X] - QTD_PC[05] - Processando..."
104 rtai-load > logs_finais/log_COM_MONITOR_COM_05_PC_10.txt
105 printf " - Concluido. \n"
106 sleep 5
107
108 #-------------------------------------------------------------------------------
109 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
110 ARGV="0 1 0 0 5"
111 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
112 printf "Experimento 11 -> RAW_MONITOR[ ] - PC[X] - QTD_PC[05] - Processando..."
113 rtai-load > logs_finais/log_SEM_MONITOR_COM_05_PC_11.txt
114 printf " - Concluido. \n"
115 sleep 5
116
117 #-------------------------------------------------------------------------------
118 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
119 ARGV="1 1 0 0 6"
120 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
121 printf "Experimento 12 -> RAW_MONITOR[X] - PC[X] - QTD_PC[06] - Processando..."
122 rtai-load > logs_finais/log_COM_MONITOR_COM_06_PC_12.txt
123 printf " - Concluido. \n"
124 sleep 5
125
126 #-------------------------------------------------------------------------------
127 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
128 ARGV="0 1 0 0 6"
129 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
130 printf "Experimento 13 -> RAW_MONITOR[ ] - PC[X] - QTD_PC[06] - Processando..."
131 rtai-load > logs_finais/log_SEM_MONITOR_COM_06_PC_13.txt
132 printf " - Concluido. \n"
133 sleep 5
134
135 #-------------------------------------------------------------------------------
136 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
137 ARGV="1 1 0 0 7"
138 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
```

```
139 printf "Experimento 14 -> RAW_MONITOR[X] - PC[X] - QTD_PC[07] - Processando..."
140 rtai-load > logs_finais/log_COM_MONITOR_COM_07_PC_14.txt
141 printf " - Concluido. \n"
142 sleep 5
143
144 #-------------------------------------------------------------------------------
145 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
146 ARGV="0 1 0 0 7"
147 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
148 printf "Experimento 15 -> RAW_MONITOR[ ] - PC[X] - QTD_PC[07] - Processando..."
149 rtai-load > logs_finais/log_SEM_MONITOR_COM_07_PC_15.txt
150 printf " - Concluido. \n"
151 sleep 5
152
153 #-------------------------------------------------------------------------------
154 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
155 ARGV="1 1 0 0 8"
156 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
157 printf "Experimento 16 -> RAW_MONITOR[X] - PC[X] - QTD_PC[08] - Processando..."
158 rtai-load > logs_finais/log_COM_MONITOR_COM_08_PC_16.txt
159 printf " - Concluido. \n"
160 sleep 5
161
162 #-------------------------------------------------------------------------------
163 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
164 ARGV="0 1 0 0 8"
165 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
166 printf "Experimento 17 -> RAW_MONITOR[ ] - PC[X] - QTD_PC[08] - Processando..."
167 rtai-load > logs_finais/log_SEM_MONITOR_COM_08_PC_17.txt
168 printf " - Concluido. \n"
169 sleep 5
170
171 #-------------------------------------------------------------------------------
172 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
173 ARGV="1 1 0 0 9"
174 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
175 printf "Experimento 18 -> RAW_MONITOR[X] - PC[X] - QTD_PC[09] - Processando..."
176 rtai-load > logs_finais/log_COM_MONITOR_COM_09_PC_18.txt
177 printf " - Concluido. \n"
178 sleep 5
179
180 #-------------------------------------------------------------------------------
181 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
182 ARGV="0 1 0 0 9"
183 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
184 printf "Experimento 19 -> RAW_MONITOR[ ] - PC[X] - QTD_PC[09] - Processando..."
185 rtai-load > logs_finais/log_SEM_MONITOR_COM_09_PC_19.txt
186 printf " - Concluido. \n"
187 sleep 5
188
189 #-------------------------------------------------------------------------------
190 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
191 ARGV="1 1 0 0 10"
192 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
193 printf "Experimento 20 -> RAW_MONITOR[X] - PC[X] - QTD_PC[10] - Processando..."
194 rtai-load > logs_finais/log_COM_MONITOR_COM_10_PC_20.txt
195 printf " - Concluido. \n"
196 sleep 5
197
```

```
198 #-------------------------------------------------------------------------------
199 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
200 ARGV="0 1 0 0 10"
201 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
202 printf "Experimento 21 -> RAW_MONITOR[ ] - PC[X] - QTD_PC[10] - Processando..."
203 rtai-load > logs_finais/log_SEM_MONITOR_COM_10_PC_21.txt
204 printf " - Concluido. \n"
205 sleep 5
206
207 #-------------------------------------------------------------------------------
208 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
209 ARGV="1 1 0 0 11"
210 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
211 printf "Experimento 22 -> RAW_MONITOR[X] - PC[X] - QTD_PC[11] - Processando..."
212 rtai-load > logs_finais/log_COM_MONITOR_COM_11_PC_22.txt
213 printf " - Concluido. \n"
214 sleep 5
215
216 #-------------------------------------------------------------------------------
217 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
218 ARGV="0 1 0 0 11"
219 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
220 printf "Experimento 23 -> RAW_MONITOR[ ] - PC[X] - QTD_PC[11] - Processando..."
221 rtai-load > logs_finais/log_SEM_MONITOR_COM_11_PC_23.txt
222 printf " - Concluido. \n"
223 sleep 5
224
225 #-------------------------------------------------------------------------------
226 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
227 ARGV="1 1 0 0 12"
228 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
229 printf "Experimento 24 -> RAW_MONITOR[X] - PC[X] - QTD_PC[12] - Processando..."
230 rtai-load > logs_finais/log_COM_MONITOR_COM_12_PC_24.txt
231 printf " - Concluido. \n"
232 sleep 5
233
234 #-------------------------------------------------------------------------------
235 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
236 ARGV="0 1 0 0 12"
237 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
238 printf "Experimento 25 -> RAW_MONITOR[ ] - PC[X] - QTD_PC[12] - Processando..."
239 rtai-load > logs_finais/log_SEM_MONITOR_COM_12_PC_25.txt
240 printf " - Concluido. \n"
241 sleep 5
242
243 #-------------------------------------------------------------------------------
244 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
245 ARGV="1 1 0 0 13"
246 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
247 printf "Experimento 26 -> RAW_MONITOR[X] - PC[X] - QTD_PC[13] - Processando..."
248 rtai-load > logs_finais/log_COM_MONITOR_COM_13_PC_26.txt
249 printf " - Concluido. \n"
250 sleep 5
251
252 #-------------------------------------------------------------------------------
253 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
254 ARGV="0 1 0 0 13"
255 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
256 printf "Experimento 27 -> RAW_MONITOR[ ] - PC[X] - QTD_PC[13] - Processando..."
```

```
257 rtai-load > logs_finais/log_SEM_MONITOR_COM_13_PC_27.txt
258 printf " - Concluido. \n"
259 sleep 5
260
261 #-------------------------------------------------------------------------------
262 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
263 ARGV="1 1 0 0 14"
264 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
265 printf "Experimento 28 -> RAW_MONITOR[X] - PC[X] - QTD_PC[14] - Processando..."
266 rtai-load > logs_finais/log_COM_MONITOR_COM_14_PC_28.txt
267 printf " - Concluido. \n"
268 sleep 5
269
270 #-------------------------------------------------------------------------------
271 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
272 ARGV="0 1 0 0 14"
273 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
274 printf "Experimento 29 -> RAW_MONITOR[ ] - PC[X] - QTD_PC[14] - Processando..."
275 rtai-load > logs_finais/log_SEM_MONITOR_COM_14_PC_29.txt
276 printf " - Concluido. \n"
277 sleep 5
278
279 #-------------------------------------------------------------------------------
280 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
281 ARGV="1 1 0 0 15"
282 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
283 printf "Experimento 30 -> RAW_MONITOR[X] - PC[X] - QTD_PC[15] - Processando..."
284 rtai-load > logs_finais/log_COM_MONITOR_COM_15_PC_30.txt
285 printf " - Concluido. \n"
286 sleep 5
287
288 #-------------------------------------------------------------------------------
289 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
290 ARGV="0 1 0 0 15"
291 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
292 printf "Experimento 31 -> RAW_MONITOR[ ] - PC[X] - QTD_PC[15] - Processando..."
293 rtai-load > logs_finais/log_SEM_MONITOR_COM_15_PC_31.txt
294 printf " - Concluido. \n"
295 sleep 5
296
297 #-------------------------------------------------------------------------------
298 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
299 ARGV="1 1 0 0 16"
300 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
301 printf "Experimento 32 -> RAW_MONITOR[X] - PC[X] - QTD_PC[16] - Processando..."
302 rtai-load > logs_finais/log_COM_MONITOR_COM_16_PC_32.txt
303 printf " - Concluido. \n"
304 sleep 5
305
306 #-------------------------------------------------------------------------------
307 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
308 ARGV="0 1 0 0 16"
309 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
310 printf "Experimento 33 -> RAW_MONITOR[ ] - PC[X] - QTD_PC[16] - Processando..."
311 rtai-load > logs_finais/log_SEM_MONITOR_COM_16_PC_33.txt
312 printf " - Concluido. \n"
313 sleep 5
314
315 #-------------------------------------------------------------------------------
```

```
316 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
317 ARGV="1 1 0 0 17"
318 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
319 printf "Experimento 34 -> RAW_MONITOR[X] - PC[X] - QTD_PC[17] - Processando..."
320 rtai-load > logs_finais/log_COM_MONITOR_COM_17_PC_34.txt
321 printf " - Concluido. \n"
322 sleep 5
323
324 #-------------------------------------------------------------------------------
325 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
326 ARGV="0 1 0 0 17"
327 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
328 printf "Experimento 35 -> RAW_MONITOR[ ] - PC[X] - QTD_PC[17] - Processando..."
329 rtai-load > logs_finais/log_SEM_MONITOR_COM_17_PC_35.txt
330 printf " - Concluido. \n"
331 sleep 5
332
333 #-------------------------------------------------------------------------------
334 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
335 ARGV="1 1 0 0 18"
336 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
337 printf "Experimento 36 -> RAW_MONITOR[X] - PC[X] - QTD_PC[18] - Processando..."
338 rtai-load > logs_finais/log_COM_MONITOR_COM_18_PC_36.txt
339 printf " - Concluido. \n"
340 sleep 5
341
342 #-------------------------------------------------------------------------------
343 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
344 ARGV="0 1 0 0 18"
345 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
346 printf "Experimento 37 -> RAW_MONITOR[ ] - PC[X] - QTD_PC[18] - Processando..."
347 rtai-load > logs_finais/log_SEM_MONITOR_COM_18_PC_37.txt
348 printf " - Concluido. \n"
349 sleep 5
350
351 #-------------------------------------------------------------------------------
352 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
353 ARGV="1 1 0 0 19"
354 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
355 printf "Experimento 38 -> RAW_MONITOR[X] - PC[X] - QTD_PC[19] - Processando..."
356 rtai-load > logs_finais/log_COM_MONITOR_COM_19_PC_38.txt
357 printf " - Concluido. \n"
358 sleep 5
359
360 #-------------------------------------------------------------------------------
361 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
362 ARGV="0 1 0 0 19"
363 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
364 printf "Experimento 39 -> RAW_MONITOR[ ] - PC[X] - QTD_PC[19] - Processando..."
365 rtai-load > logs_finais/log_SEM_MONITOR_COM_19_PC_39.txt
366 printf " - Concluido. \n"
367 sleep 5
368
369 #-------------------------------------------------------------------------------
370 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
371 ARGV="1 1 0 0 20"
372 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
373 printf "Experimento 40 -> RAW_MONITOR[X] - PC[X] - QTD_PC[20] - Processando..."
374 rtai-load > logs_finais/log_COM_MONITOR_COM_20_PC_40.txt
```

```
375 printf " - Concluido. \n"
376 sleep 5
377
378 #------------------------------------------------------------------------------
379 #LAYOUT: |RAW_MONITOR|PC|SEC|FREQ_IDEAL_INICIAL|QTD_PC|
380 ARGV="0 1 0 0 20"
381 echo "$TARGET:lxrt:./$TARGET $ARGV;popall:Control+C para cancelar a execucao! ;)" > $RUNINFO
382 printf "Experimento 41 -> RAW_MONITOR[ ] - PC[X] - QTD_PC[20] - Processando..."
383 rtai-load > logs_finais/log_SEM_MONITOR_COM_20_PC_41.txt
384 printf " - Concluido. \n"
385 sleep 5
386
387 #------------------------------------------------------------------------------
388 printf "\n\nEXPERIMENTO FINALIZADO! \n\n"
```

Vale ressaltar que os argumentos 3 e 4, passados pela variável *ARGV* as tarefas de tempo real, devem ser desconsiderados, pois os experimentos realizados não estão levando em consideração SECs (em inglês, *Saved Execution Cycles*) e nem o cálculo automatizado das frequências ideais, essas são melhorias que estão sendo implementadas para a realização de experimentos futuros.

Apêndice F

Exemplos dos relatórios gerados pelo CPUSTAT

A tarefa CPUSTAT foi criada com o objetivo de monitorar e coletar os dados estatísticos do processador durante a realização dos experimentos. Dentre os dados estatísticos processados pelo CPUSTAT estão: o tempo de uso de cada uma das frequências disponíveis, o tempo total que o processador ficou ocioso durante a realização do experimento, a duração total do experimento, o número total de chaveamentos de contexto do processador e o consumo de energia dinâmica aproximado em *Joules*. A Figura F.1 mostra um exemplo dessa parte do relatório gerado pela tarefa CPUSTAT.

```
************** CONFIGURACOES INICIAIS DO EXPERIMENTO **************
=> [SIM] DEBUG
=> [SIM] RAW MONITOR
=> [NAO] PONTOS DE CONTROLE

************** ESTATISTICAS FINAIS **************

-> Estatísticas do Processador...

** cpufreq stats: **
->       3.00 GHz: 1.3007440639%
->       2.30 GHz: 5.5968924390%
->       1.80 GHz: 42.5032826349%
->        800 MHz: 50.5990808622%

Tempo Total CPU IDLE: (57446462) microsegundos  -> em segundos: (57.44646200)
Tempo Total Experimento: (73112) usertime units -> (USERTIME_UNIT * 10 = 731120 ms)
                                                -> em segundos: (731.12000000)
Num. Total de Transições: (687)

CONSUMO TOTAL DE ENERGIA: 7185524.88750 J (joules)
```

Figura F.1: Exemplo do relatório estatístico do processador gerado pelo CPUSTAT.

A segunda parte dos dados estatísticos exibidos pelo CPUSTAT são referentes ao número total de períodos executados por cada tarefa de tempo real, o número total de *deadlines*

violados por cada uma delas e qual foi o maior e o menor tempo de violação de *deadline* identificado durante a realização do experimento. A Figura F.2 mostra um exemplo dessa segunda parte do relatório gerado pela tarefa CPUSTAT, onde as tarefas na cor vermelha (as de número 0 a 4) são do tipo CNT, na cor verde (as de número 5 a 8) são do tipo MATMULT, na cor ciano (as de número 9 a 13) são do tipo BSORT e a tarefa 14 é a CPUSTAT.

```
***************** ESTATISTICA DAS TAREFAS *****************
[TASK  0] Total Periodos Executados:   45 -> Total Deadlines Violados:    0
[TASK  1] Total Periodos Executados:   36 -> Total Deadlines Violados:    0
[TASK  2] Total Periodos Executados:   24 -> Total Deadlines Violados:    0
[TASK  3] Total Periodos Executados:   45 -> Total Deadlines Violados:    0
[TASK  4] Total Periodos Executados:   36 -> Total Deadlines Violados:    0
[TASK  5] Total Periodos Executados:   24 -> Total Deadlines Violados:    0
[TASK  6] Total Periodos Executados:   24 -> Total Deadlines Violados:    0
[TASK  7] Total Periodos Executados:   24 -> Total Deadlines Violados:    0
[TASK  8] Total Periodos Executados:   24 -> Total Deadlines Violados:    0
[TASK  9] Total Periodos Executados:   45 -> Total Deadlines Violados:    0
[TASK 10] Total Periodos Executados:   45 -> Total Deadlines Violados:    0
[TASK 11] Total Periodos Executados:   24 -> Total Deadlines Violados:    0
[TASK 12] Total Periodos Executados:   36 -> Total Deadlines Violados:    0
[TASK 13] Total Periodos Executados:   24 -> Total Deadlines Violados:    0
[TASK 14] Total Periodos Executados:   74 -> Total Deadlines Violados:    0


***************** DETALHAMENTO *****************
-> Maior Deadline Ultrapassado:
* Nenhuma tarefa violou seu deadline!

-> Menor Deadline Ultrapassado:
* Nenhuma tarefa violou seu deadline!


Fim do Escalonamento
```

Figura F.2: Exemplo do relatório estatístico das tarefas de tempo real gerado pelo CPUSTAT.