UNIVERSIDADE FEDERAL FLUMINENSE

BRUNO ALVES NOVELLI

Economia de Energia no Escalonamento de Tarefas em Sistemas de Tempo Real

Niterói

2005

Bruno Alves Novelli

Economia de Energia no Escalonamento de Tarefas em Sistemas de Tempo Real

Dissertação apresentada ao Curso de Pós-Graduação em Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre em Computação. Área de concentração: Processamento Paralelo e Distribuído.

Orientador:
Prof. Julius C. B. Leite

UNIVERSIDADE FEDERAL FLUMINENSE

Niterói

Agosto de 2005

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

N938 Novelli, Bruno Alves.
Economia de energia no escalonamento de tarefas em sistemas de tempo real / Bruno Alves Novelli. - Niterói, RJ : [s.n.], 2005.
92 f.

Orientador: Julius C. B. Leite.
Dissertação (Mestrado em Computação) - Universidade Federal Fluminense, 2005.

1. Escalonamento de tarefas. 2. Energia - Consumo. 3. Sistemas em tempo real. 4. Dinâmica de voltagem - Regulagem. 5. Dispositivo móvel. I. Título.

CDD 005.136

Economia de Energia no Escalonamento de Tarefas em Sistemas de Tempo Real

Bruno Alves Novelli

Dissertação apresentada ao Curso de Pós-Graduação em Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre em Computação. Área de concentração: Processamento Paralelo e Distribuído.

Aprovada por:

Prof. Julius C. B. Leite / IC-UFF (Presidente)

Prof. Orlando G. Loques Filho / IC -UFF

Prof. Rômulo Silva de Oliveira / DAS - UFSC

Niterói, 26 de Agosto de 2005.

Dedico este trabalho à minha mãe Lina.

Agradecimentos

Inicialmente meu agradecimento é a Deus.

À minha mãe, pelo apoio incontestável em todos os passos e decisões de minha vida. Aos meus irmãos Bethânia, Luciano e Iara, pelo apoio de sempre na família.

Ao meu Orientador, Julius C. B. Leite. Além dos ensinamentos transmitidos durante todo o mestrado, seu caráter como profissional é um modelo para todos os alunos. Sua dedicação, respeito, amizade, compromisso e constante incentivo com os seus orientados colaboram para obtenção de excelentes resultados e para a formação de pesquisadores competentes.

A todos os professores e funcionários do Programa de Pós-Graduação em Computação da Universidade Federal Fluminense deixo minha gratidão, em especial aos professores com quem tive a oportunidade de fazer disciplinas e às funcionárias Angela, Izabela e Maria.

Ao CNPq pelo financiamento parcial deste trabalho.

Por fim, agradeço aos amigos do Mestrado: (i) àqueles que compartilharam da pesquisa diária como acorradi (companheiro de laboratório e de chopp), dvianna (amiga, divertida, compreensiva, uma pessoinha especial que apresenta um bom humor constante e que é capaz de ajudar a todos em qualquer momento), gfreitas (companheiro de mesa e chopp), jacques (pela amizade, salvamentos e chopp), rcapua (grande amiga, divertida, legal e conhecedora do Latex), vthome (grande amiga, loira, animada e prestativa) e holiveira (pela amizade e pelo chopp); (ii) aos uffanistas mineiros lbrugiolo, lciuffo, mribeiro e jsilva (por tornarem a vinda para Niterói possível e por todos os momentos de amizade); (iii) a tmartins (pela amizade, compreensão, carinho e lanchinhos noturnos) e a vka (pela amizade, respeito, bondade, compreensão e prestígio); e, (iv) a todos os outros amigos que, diretamente ou indiretamente me ajudaram nesse trabalho.

Resumo

Nos últimos anos os sistemas de tempo real têm incorporado novas características e, conseqüentemente, têm sido mais exigidos em termos de capacidade de processamento. Alguns desses sistemas são postos para operar em condições onde o suprimento de energia é restrito ou, ainda, fornecido por baterias. Infelizmente, o avanço da tecnologia de baterias tem sido lento em acompanhar as crescentes necessidades de consumo. Assim, restam duas opções: o uso de grandes baterias ou a realização de um gerenciamento de energia eficiente. Como o aumento físico da bateria nem sempre é possível ou desejável, técnicas de gerenciamento de energia que levem a um menor consumo tornam-se um importante requisito de projeto para esses sistemas.

Essa dissertação propõe métodos para serem utilizados em conjunto com o escalonador Taxa Monotônica para a economia de energia em sistemas de tempo real, através do uso de regulagem dinâmica de voltagem. Todos esses métodos utilizam uma etapa estática e uma dinâmica. Na etapa estática determina-se a menor freqüência de operação possível para o conjunto de tarefas, de tal forma que o escalonador ainda garanta todas as restrições temporais. A outra etapa, dinâmica, aproveita o tempo adicional gerado pela não utilização do tempo máximo de execução das tarefas para reduzir a freqüência de operação de cada uma delas. Através de simulações, foram realizados vários experimentos que demonstram significativa redução no consumo de energia.

Abstract

In the last years real-time systems have incorporated new characteristics and, consequently, they have been more demanded in terms of processing capacity. Some of these systems are put to operate in conditions where the energy supply is restricted or, still, came from batteries. Unfortunately, the advance of the technology of batteries has been slow in following the increasing necessities of consumption. Thus, two options are left: the use of bigger batteries or a more efficient energy management. As the physical increase of the battery is not always possible nor desirable, techniques of energy management that lead to a smaller consumption become an important project requirement for these systems.

This dissertation presents methods to be used with together a rate monotonic scheduler for power saving in real-time systems, through the use of dynamic voltage scaling. All the proposed methods use an off-line and an on-line phase. The off-line phase calculates the smaller frequency that still guarantees all timing constraints. The on-line phase takes into account the additional time produced by tasks that run for less than their worst-case execution time, and also the slack left in the first phase, to yet reduce the frequency of execution. Through simulation, some experiments had been realized that demonstrate significant reduction in power consumption.

Palavras-chave

1. Economia de energia
2. Sistemas de tempo real
3. Escalonamento de tarefas
4. Regulagem dinâmica de voltagem
5. Dispositivos móveis

Glossário

ACET	:	*Average Case Execution Time*
AGR	:	*Aggressive*
Aprox 1N	:	Aproximado de um nível
Aprox 2N	:	Aproximado de dois níveis
BCET	:	*Best Case Execution Time*
CNC	:	*Computerized Numerical Control*
ccEDF	:	*Cycle Conserving Earliest Deadline First*
ccRM	:	*Cycle Conserving Rate Monotonic*
ccRMG	:	*Cycle Conserving Rate Monotonic Greedy*
DASS	:	*Dynamic Approximate Slack Stealing*
DPM-Clock	:	*Dynamic PM-Clock*
DRA	:	*Dynamic Reclaiming Algorithm*
DVS	:	*Dynamic Voltage Scaling*
EDF	:	*Earliest Deadline First*
FUTURE	:	*Bounded-Delay Limited-Future*
GGT 1N	:	*Greedy Gain Time* de um nível
GGT 2N	:	*Greedy Gain Time* de dois níveis
IRIS	:	*Increased-Reward with Increased-Service*
LLF	:	*Least Laxity First*
lppsEDF	:	*Low Power Priority Scheduling Earliest Deadline First*
lppsRM	:	*Low Power Priority Scheduling Rate Monotonic*
lpSEH	:	*Low-Power Scheduling Algorithm Based on Slack Estimation*
lpWDA	:	*Low-Power Short-Term Work-Demand Analysis*
NTA	:	*Arrival Time of the Next Task*
OPT	:	*Unbounded-Delay Perfect-Future*
Opt-Clock	:	*Optimal Clock Frequency Assignment*

PASS	:	*Periodic Approximate Slack Stealing*
PAST	:	*Bounded-Delay Limited-Past*
PM-Clock	:	*Priority-Monotonic Clock Frequency Assignment*
RM	:	*Rate Monotonic*
RDV	:	Regulagem Dinâmica de Voltagem
SASS	:	*Static Approximate Slack Stealing*
Sys-clock	:	*System Clock Frequency Assignment*
STR	:	Sistemas de Tempo Real
WCET	:	*Worst Case Execution Time*

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Sistemas de tempo real, notadamente aqueles classificados como sistemas embutidos, têm incorporado novas características e, conseqüentemente, têm sido mais exigidos em termos de capacidade de processamento. Alguns desses sistemas são postos para operar em condições onde o suprimento de energia é limitado ou, ainda, fornecido por baterias. Infelizmente, o avanço da tecnologia de baterias tem sido lento em acompanhar as crescentes necessidades de consumo. Assim, restam duas opções: o uso de grandes baterias ou a realização de um gerenciamento de energia eficiente. Como o aumento físico da bateria nem sempre é possível ou desejável, técnicas de gerenciamento de energia que levem a um menor consumo tornam-se um importante requisito de projeto para esses sistemas. Além disso, como um sub-produto, deve ser notado que com um menor consumo a vida útil da bateria e dos dispositivos por ela alimentados é normalmente estendida. A redução da demanda energética tem ainda como vantagem a simplificação dos mecanismos de dissipação, o aumento da confiabilidade dos componentes e uma redução em seu custo.

Dentre os diversos tipos de componentes de *hardware* que constituem um dispositivo móvel destacam-se o processador, o sistema de memória e a interface de rede como os principais consumidores de energia. Para cada um deles existem técnicas de *software* que têm por objetivo minimizar esse consumo. O estudo aqui apresentado é focado no processador, pois é normalmente o componente que mais consome energia no sistema. Isto é verdadeiro não somente para pequenos dispositivos de mão como PDAs [Ellis 1999], que tem poucos componentes, mas também para computadores portáteis [Lorch e Smith 1998] que possuem muitos componentes, incluindo grandes telas de vídeo.

Para motivar essa apresentação é mostrado na tabela 1.1 um exemplo do consumo de energia medido em um computador móvel [Pillai e Shin 2001]. Nessa tabela, a coluna *Processador*

indica não somente o consumo desse componente mas também o da memória. É fácil notar nesse exemplo que, quando o vídeo está desligado e o disco está em *standby*, o fato de o processador passar do estado de carga máxima para ocioso leva a um substancial decréscimo de consumo. Obviamente, outros dispositivos, como memória, também são desativados, mas o processador pode ser responsável por até cerca de 60% desse decréscimo.

Tabela 1.1: Consumo de energia medido em um laptop HP N3350.

Vídeo	**Processador**	**Disco**	**Energia**
Ligado	Ocioso	Girando	13,5W
Ligado	Ocioso	*Standby*	13,0W
Desligado	Ocioso	*Standby*	7,1W
Desligado	Carga Máxima	*Standby*	27,3W

A principal área de interesse desse trabalho são os sistemas de tempo real (STR) que, por definição, são aqueles submetidos a requisitos de natureza temporal. Nesses sistemas os resultados devem estar corretos não somente do ponto de vista lógico-aritmético, mas também devem ser gerados no momento correto. Os STRs podem ser classificados como críticos (*hard real-time*) e não críticos (*soft real-time*). Nos sistemas críticos um atraso ou resposta incorreta é considerado um erro inaceitável, que pode resultar em uma catástrofe ou envolver perda de vidas (e.g., sistemas de controle de tráfego aéreo, de controle de linhas ferroviárias e de usinas nucleares). Já nos sistemas não críticos pode-se aceitar ocasionalmente uma resposta atrasada (e.g., sistemas de telefonia digital ou de vídeo conferência). Quando um sistema cumpre todas as restrições de tempo diz-se que ele é escalonável.

Uma tarefa em um STR pode ser periódica, esporádica ou aperiódica. A tarefa é dita periódica se ela está pronta para ser executada a cada *t* unidades de tempo. Uma tarefa é do tipo esporádica se ela não é periódica e pode ser invocada em intervalos irregulares. Tarefas esporádicas são caracterizadas por possuir um limite superior na taxa com que elas podem ser invocadas, ou seja, invocações sucessivas de tarefas esporádicas são separadas por um período mínimo. Tarefas aperiódicas são definidas como tarefas que não são periódicas e nem possuem um limite superior para a taxa de invocação.

Seguindo o modelo multiprocessos-monoprocessador do trabalho seminal de [Liu e Layland 1973], as tarefas de tempo real a serem consideradas nesse trabalho são críticas, periódicas, independentes e preemptáveis. O conjunto de tarefas é especificado como S(n) = {(C_1, T_1, D_1), (C_2, T_2, D_2), ... , (C_n, T_n, D_n)}, onde C_i, T_i e D_i denotam, respectivamente, o tempo de execução de pior caso (WCET - *Worst Case Execution Time*), o

período de ativação e o prazo para cada tarefa *i*. Uma suposição comum é que, ∀i, $D_i = T_i$. Quando as tarefas competirem pelo processador o conflito será resolvido de acordo com as regras definidas por uma política de prioridades. É chamado de instante crítico aquele no qual todas as tarefas, simultaneamente, irão competir pelo processador. O hiperperíodo de um conjunto de tarefas é definido como o mínimo múltiplo comum dos seus períodos.

Durante um hiperperíodo há momentos em que o processador estará ocupado executando tarefas e outros nos quais o processador se encontrará ocioso. Esses espaços vazios (ou seja, onde há ociosidade do processador) são chamados de folga (*slack*). Seja o conjunto de tarefas definido na tabela 1.2. Admitindo-se uma situação de instante crítico, a figura 1.1 apresenta a escala gerada pelo algoritmo *Rate Monotonic* (RM, [Liu e Layland 1973]) para o hiperperíodo desse conjunto de tarefas. Esse algoritmo atribui prioridades às tarefas de forma inversamente proporcional aos seus períodos. Ou seja, uma tarefa de menor período será mais prioritária que uma tarefa de maior período. Nessa figura, os números nos retângulos indicam as tarefas e *e* representa os períodos ociosos do processador (nesse caso a folga no hiperperíodo é igual a duas unidades de tempo).

Tabela 1.2: Conjunto de tarefas para exemplo de folga do hiperperíodo.

Tarefa	Período	WCET
1	2	1
2	5	1
3	10	1

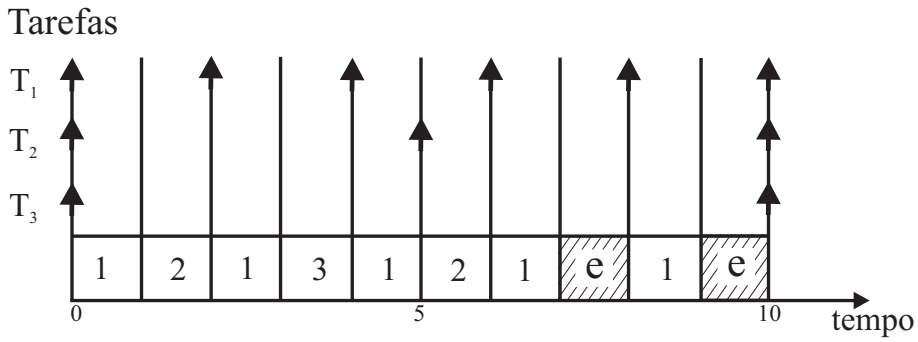


Figura 1.1: Escalonamento do conjunto de tarefas da tabela 1.2.

Em aplicações de tempo real, o tempo de execução de uma tarefa pode variar significativamente em relação ao seu WCET. Por exemplo, [Ernst e Ye 1997] relatam que o tempo de pior caso pode chegar a ser até 10 vezes maior que o melhor tempo de execução (BCET - *Best Case Execution Time*). O principal objetivo deste trabalho é a utilização de métodos para economia

de energia no escalonamento de tarefas de Sitemas de tempo real. Todos os métodos aqui propostos utilizam a técnica conhecida como RDV - Regulagem Dinâmica de Voltagem (DVS - *Dynamic Voltage Scaling* [Weiser et al. 1994, Bertini e Leite 2004]) para o escalonamento com o objetivo de economia de energia. Neles utiliza-se a folga e/ou *gain time* para poder diminuir a voltagem de alimentação e a freqüência de operação do processador, diminuindo o gasto energético mas ainda garantindo o atendimento aos prazos definidos pelas aplicações. Apresenta-se nesses métodos uma técnica que utiliza estaticamente (*off-line*) a folga presente no hiperperíodo e, são apresentados técnicas utilizadas dinamicamente que aproveitam, em tempo de execução, o tempo não utilizado por uma tarefa (denominado *gain time*). Adicionalmente, apresenta-se também um procedimento mais elaborado baseado em roubo de folga (*slack stealing*) para adiantar partes do tempo livre (folga mais *gain time*). Embora esse trabalho seja voltado para sistemas críticos, a mesma idéia pode ser utilizada, com algumas modificações, para sistemas não críticos, como os empregados em aplicações multimídia (e.g., [Yuan e Nahrstedt 2003]).

Esta dissertação está organizada em seis capítulos. No capítulo a seguir é explicado o funcionamento do mecanismo de RDV e são apresentados trabalhos relacionados. No capítulo 3 são explicados os métodos aqui propostos para economia de energia. No capítulo 4 são apresentados e discutidos alguns resultados. No capítulo 5 são apresentadas as conclusões e são discutidos possíveis trabalhos futuros. Finalizando, no capítulo 6 as referências bibliográficas desse trabalho são indicadas.

A partir das idéias aqui apresentadas foram elaborados dois artigos científicos que foram publicados no VI Workshop de Comunicação sem Fio e Computação Móvel [Urriza et al. 2004] e no XXXII Seminário Integrado de Software e Hardware [Novelli et al. 2005].

Capítulo 2

Trabalhos Relacionados

2.1 Regulagem Dinâmica de Voltagem

A tecnologia CMOS é hoje comumente utilizada em diversos processadores. Em circuitos integrados implementados com essa tecnologia o consumo de energia *(E)* é aproximadamente proporcional ao quadrado da voltagem de alimentação *(V)* e varia linearmente com a freqüência de operação $\frac{E}{clock} \propto V^2$. A redução da freqüência operacional do processador não implica em redução de voltagem e, embora acarrete uma diminuição do consumo, causa um aumento no tempo de execução das tarefas. Assim, o uso dessa técnica isoladamente não traz nenhum benefício. Para circuitos integrados reais, admitindo que para uma dada voltagem é sempre utilizada a máxima freqüência possível correspondente a esse nível, a redução na tensão de alimentação exige uma conseqüente redução na freqüência de operação. Logo, a economia de energia ocorre porque a redução no nível de voltagem tem um impacto de ordem quadrática na redução de energia. No que se segue, quando houver menção a redução de freqüência estará subentendido que haverá uma correspondente diminuição na tensão de alimentação.

Vários processadores comerciais (e.g., AMD *Mobile*, Intel *Centrino Mobile* e *XScale Technology*, Transmeta *Efficeon* e *Crusoe*) implementam o mecanismo de regulagem dinâmica de voltagem. Para todos os processadores citados há um número limitado de níveis de tensão que podem ser utilizados. Ou seja, a tecnologia atual ainda não permite o oferecimento de circuitos com variação contínua de seu ponto de operação. Com base no mecanismo de RDV diversos algoritmos de escalonamento têm sido desenvolvidos para obtenção de economia de energia, especialmente para sistemas de tempo real ([Weiser et al. 1994, Pillai e Shin 2001, Aydin et al. 2001a, Kim et al. 2002, Rusu et al. 2003b, Urriza et al. 2004]).

A maioria dos trabalhos que utilizam RDV assumem a hipótese de que o consumo de ener-

gia é diminuído quando há redução da tensão de alimentação. Contudo, isso nem sempre é verdadeiro. Em [Miyoshi et al. 2002] é demonstrado que, para alguns processadores comerciais, algumas das freqüências de operação são energeticamente ineficientes, ou seja, a freqüência imediatamente superior produz melhores resultados. Para que a hipótese anteriormente indicada seja verdadeira, o processador deve ter uma relação potência-freqüência convexa não decrescente. O trabalho desenvolvido por [Saewong e Rajkumar 2003] indica como obter uma grade ótima de freqüências de operação de forma a minimizar os erros de quantização causados pelo uso de freqüências discretas. Esse é um resultado importante, não só para a construção e avaliação de modelos teóricos, mas também para a construção de processadores reais.

A figura 2.1 apresenta um comparativo do consumo de energia e do desempenho para as arquiteturas StrongARM[1] e XScale[2]. Observa-se que um processador XScale operando com relógio de 1GHz, alimentado com 1,80V e consumindo 1,6W tem um desempenho de 1.200MIPS. O mesmo processador, operando a 800MHz e 1,60V, consome 0,9W com desempenho de 1.000MIPS. Verifica-se neste exemplo que, embora o desempenho tenha caído apenas 17%, a potência foi reduzida a quase metade (56%). Ou seja, a perda de desempenho quando se reduz a tensão e a freqüência é pequena comparada com o ganho na economia de energia.

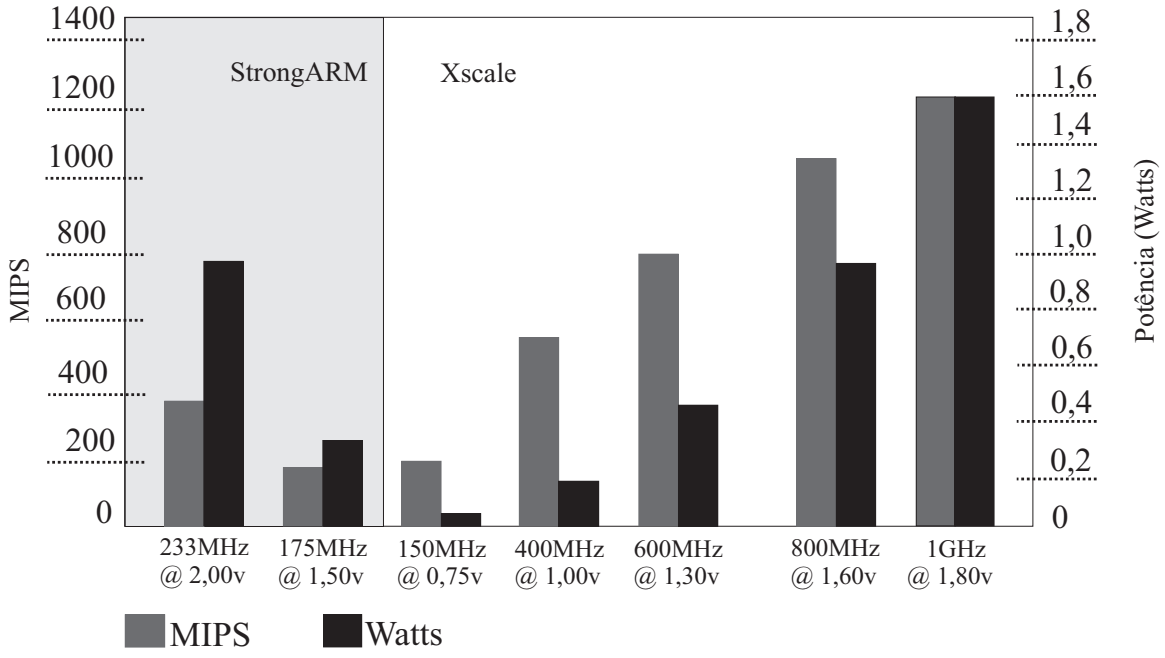


Figura 2.1: Comparação do consumo de energia e do desempenho para as arquiteturas StrongARM e XScale.

[1]http://www.arm.com
[2]http://developer.intel.com/design/intelxscale/

Em [Ishihara e Yasuura 1998] são apresentados alguns resultados teóricos para sistemas que usam RDV. Dentre os mais importantes resultados estão os teoremas a seguir:

i) *Teorema 1*: Se um processador somente pode utilizar um número discreto de voltagens, o escalonamento com o uso de no máximo dois níveis de tensão, para qualquer restrição temporal, minimiza o consumo de energia;

ii) *Teorema 2*: Se um processador somente pode utilizar um número discreto de voltagens, as duas voltagens que minimizam o consumo de energia, para uma dada restrição temporal, são aquelas imediatamente vizinhas à voltagem ideal (aquela que implicaria em um único nível para todas as tarefas).

A fonte dominante de dissipação de energia em um circuito CMOS é a dissipação dinâmica de energia [Ishihara e Yasuura 1998]. Em um sistema de tempo real onde a tarefa possui um tempo de execução, um período e um prazo, geralmente o prazo é igual ao período e, o tempo de execução é necessariamente menor ou igual ao período. Portanto, nesses sistemas é comum que o processador apresente intervalos de tempo ocioso. Nos trabalhos que utilizam a técnica de RDV em sistemas de tempo real os períodos de ociosidade são aproveitados com o objetivo de minimizar o consumo de energia no sistema, mas ainda garantindo os prazos de execução das tarefas ([Pering e Brodersen 1998, Sinha e Chandrakasan 2001, Pillai e Shin 2001, Luo e Jha 2002, Rusu et al. 2002, Urriza et al. 2004]). Pode-se associar a esses períodos de folga a execução de tarefas aperiódicas não críticas ([Sprunt et al. 1989, Lehoczky e Ramos-Thuel 1992, Strosnider et al. 1995]) ou ainda a execução de partes opcionais de tarefas críticas cujo processamento possa implicar em algum tipo de recompensa ([Dey et al. 1996, Aydin et al. 2001b, Santos et al. 2002, Rusu et al. 2003b]).

Para um melhor entendimento da RDV, apresenta-se nas figuras 2.2 e 2.3 o seu funcionamento. Nessas figuras, há apenas a execução de uma única tarefa e essa apresenta um WCET de $1000 * 10^6$ ciclos e um período/prazo de 25 segundos. A tabela 2.1 indica as características do processador utilizado. A figura 2.2 apresenta a execução sem considerar a utilização do RDV, nesse caso a tarefa executa à máxima voltagem (5V) e também à máxima freqüência (50Mhz). Então, a energia gasta nesse escalonamento é de 40J, assumindo que após a execução da tarefa não há gasto de energia, ou seja, entre os tempos 20s (término da execução) e 25s (prazo de execução) o consumo de energia é zero. Já a figura 2.3 indica a execução da mesma tarefa com utilização da RDV. Nesse caso, a tarefa pode ser executada em uma voltagem (4V) e freqüência (40Mhz) menores. Assim, consumiu-se menos energia (25J) para se executar a mesma quantidade de ciclos, e embora tenha-se atrasado a execução, não houve perda do prazo.

Tabela 2.1: Processador teórico do artigo [Ishihara e Yasuura 1998].

Nível	Freqüência(Mhz)	Voltagem(V)	Consumo(nJ/ciclo)
1	50	5,0	40
2	40	4,0	25
3	25	2,5	10

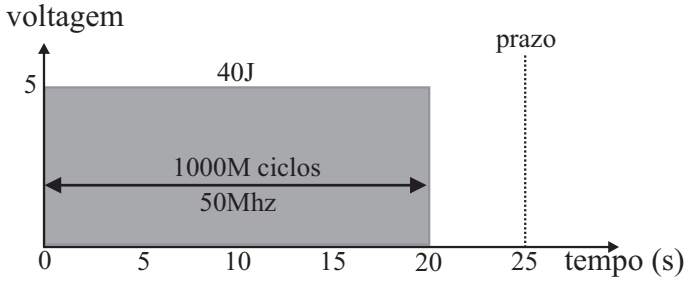


Figura 2.2: Exemplo de execução de uma tarefa sem utilização de RDV.

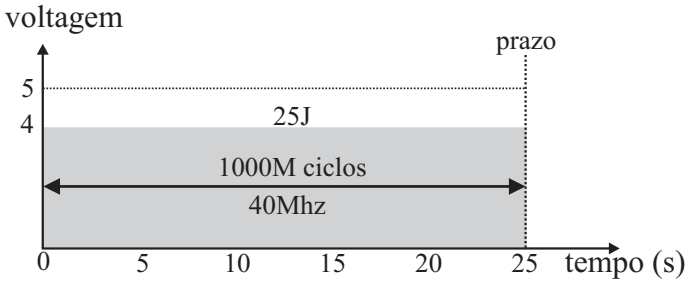


Figura 2.3: Exemplo de utilização de RDV.

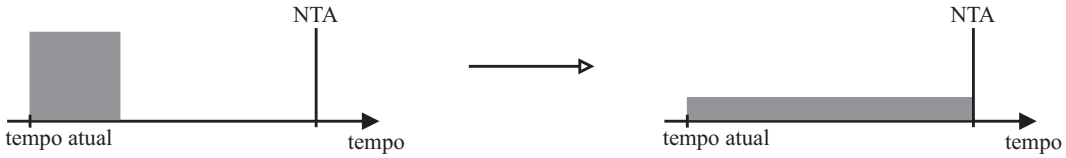
2.2 Algoritmos de Escalonamento

Um dos primeiros trabalhos que tratam do escalonamento de tarefas para redução de energia foi apresentado em 1994 por [Weiser et al. 1994]. O conjunto de tarefas considerado nesse trabalho consiste de tarefas críticas e não críticas e essas podem ser periódicas ou aperiódicas. São apresentados três algoritmos (OPT, FUTURE, PAST) para redução de tempo ocioso. Todos os algoritmos utilizam uma janela que indica as tarefas e suas ativações. O OPT (*unbounded-delay perfect-future*) "estica" as execuções das tarefas de forma a preencher todos os períodos ociosos (desconsiderando os prazos). Esse algoritmo é impraticável e indesejável. Impraticável porque ele requer um conhecimento do futuro da execução das tarefas (janela igual ao hiperperíodo). É indesejável pois produz grandes atrasos nas execuções das tarefas. A utilização do OPT se justifica como um importante parâmetro de comparação. O FUTURE (*bounded-delay limited-future*) é parecido com o OPT, porém ele considera que tamanho da janela é definido, ou seja,

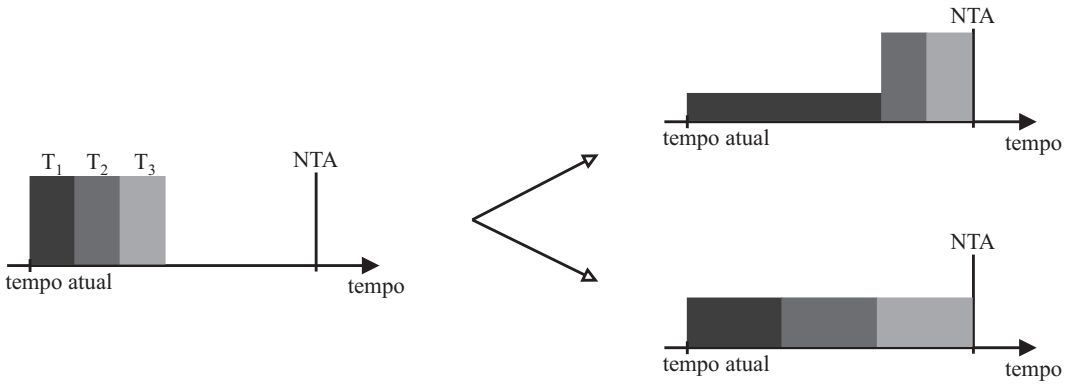
observa-se um certo número de ativações e não todo o hiperperíodo como ocorre no OPT. O FUTURE é impraticável, pois necessita de informações sobre o futuro das execuções. Ele é desejável visto que o atraso é conhecido em virtude do tamanho da janela. Já o algoritmo PAST (*bounded-delay limited-past*) é uma versão praticável do FUTURE uma vez que utiliza a mesma idéia de janela fixa, porém olhando para o passado. Ele assume que a próxima janela será igual a última, logo considera as execuções anteriores para os cálculos.

O trabalho desenvolvido por [Shin et al. 2000] apresenta algoritmos estáticos (*off-line*) e dinâmicos (*on-line*) para conjuntos de tarefas periódicas, em sistemas de tempo real críticos, que utilizam escalonador EDF - *Earliest Deadline First* [Liu e Layland 1973] e RM. São apresentados dois algoritmos (lppsRM e lppsEDF; lpps - *low power priority scheduling*). Em ambos os algoritmos utiliza-se uma fase estática que seleciona a menor freqüência operacional possível (única) que ainda garanta todos os prazos do conjunto de tarefas. A fase dinâmica trabalha com a folga remanescente do sistema e com o *gain time*. Nessa etapa, uma tarefa tem a voltagem e a freqüência ajustadas somente quando existe apenas uma tarefa ativa. A figura 2.4 (a) ilustra a execução dessa última etapa. Nessa figura NTA (*arrival time of the next task*) indica que outra tarefa estará pronta para ser executada. Caso mais de uma tarefa esteja ativa a tarefa executará na freqüência definida estaticamente. Quando não há nenhuma tarefa a executar o processador é ajustado para um modo de economia de energia, ou seja, a voltagem utilizada nesse período ocioso é menor. Embora esse trabalho alcance o objetivo de economizar energia, seus resultados são limitados por realizar, na etapa dinâmica, economia adicional de energia somente se uma única tarefa estiver ativa.

Um trabalho baseado em *gain time* e aproveitamento de folga é discutido em [Pillai e Shin 2001], onde eles apresentam abordagens estáticas e dinâmicas para sistemas de tempo real críticos. Na abordagem estática seleciona-se a menor freqüência operacional possível que ainda permita ao escalonador, no caso EDF ou RM, garantir todos os prazos do conjunto de tarefas. A redução é feita com base nos testes de escalonabilidade conhecidos para esses dois algoritmos, pela introdução de um fator de escala (relação entre a freqüência de operação pretendida e a freqüência máxima permitida pelo processador) para os tempos de execução máximos das tarefas. Na abordagem dinâmica, ou seja, para o caso em que as tarefas executam por menos tempo que o seu WCET, são apresentados dois algoritmos: o *Cycle Conserving*, desenvolvido para escalonadores RM e EDF, e o *Look Ahead*, desenvolvido somente para EDF. Ambos os algoritmos para EDF baseiam-se no cálculo da utilização do processador ao término da execução de cada tarefa, de forma a reclamar quaisquer ciclos (previstos no WCET) não utilizados. A diferença entre esses algoritmos é que no *Look Ahead* é assumida uma abordagem otimista,



(a) uma tarefa ativa



(b) várias tarefas ativas

Figura 2.4: Exemplos de escalonamento de tarefas.

mais agressiva, onde, no início da execução da tarefa ela é processada a uma velocidade inferior, podendo essa velocidade (freqüência-voltagem) ser aumentada para garantir o atendimento à restrição temporal. Já o algoritmo *Cycle Conserving* para RM (ccRM) estende o trabalho de [Shin et al. 2000] por permitir que várias tarefas estejam ativas simultaneamente. Quando múltiplas tarefas estão ativas existem várias formas de se distribuir a folga. Por exemplo, a folga pode ser utilizada por apenas uma tarefa ou pode ser distribuída igualmente entre todas as tarefas ativas, como mostrado na figura 2.4 (b). No ccRM a folga é distribuída igualmente entre todas as tarefas ativas.

O trabalho apresentado em [Aydin et al. 2001a] refere-se ao problema de escalonamento de tarefas periódicas de tempo real crítico. A solução apresenta uma versão estática e duas dinâmicas, denominadas DRA (*Dynamic Reclaiming Algorithm*) e AGR (*Aggressive*). A solução estática baseia-se em uma velocidade única que minimiza o total de energia consumido enquanto ainda mantém todos os prazos. Além disso, pode ser utilizada qualquer política que utilize totalmente o processador (e.g., EDF, *LLF - Least Laxity First* [Mok 1983]). Os algoritmos *on-line* utilizam um mecanismo de redução de velocidade para reclamar o tempo de processamento não utilizado pelas tarefas que completaram sua execução, reduzindo a energia gasta. O AGR é mais agressivo, pois utiliza a informação de carga de trabalho do caso médio para prever términos antecipados de execuções futuras e, com isso, consegue melhores resultados.

Em [Kim et al. 2002] é apresentado um algoritmo (denominado *Low-Power Scheduling Algorithm Based on Slack Estimation* - lpSEH) baseado no aproveitamento do *gain time* para um conjunto de tarefas periódicas, em sistemas de tempo real críticos, utilizando o escalonador EDF. A principal característica deste algoritmo está na determinação do *gain time*, que é feita de duas formas. A primeira determina o *gain time* para as tarefas de maior prioridade que completam sua execução antes do seu WCET. Já na segunda forma, o aproveitamento provém das tarefas de menor prioridade que durante sua execução utilizaram menos que o seu WCET. Ao serem preemptadas, elas permitem que tarefas de maior prioridade possam aproveitar esse tempo. Esse método acrescenta um pequeno *overhead* para a determinação do *gain time*, porém não há nenhuma fase *off-line*.

Para melhor entendimento do algoritmo, a figura 2.5 apresenta um exemplo ilustrativo baseado no conjunto de tarefas definido na tabela 2.2. Nessa tabela ACET denota o tempo de execução do caso médio (*Average Case Execution Time*). Para esse exemplo assume-se um processador com variação contínua de voltagem. Aqui também é considerado que $T_{i,j}$ representa a ativação j da tarefa i. Em (a) tem-se a execução das tarefas com seus tempos de execução de pior caso. Em (b), quando $T_{0,0}$ completa sua execução em $t = 0,5$, $T_{1,0}$ é iniciada. Uma vez que $T_{1,0}$ foi inicialmente prevista para iniciar sua execução em $t = 1,0$, esta pode ser prolongada em $0,5$ unidade de tempo, e o escalonamento restante pode ser ainda possível. Então, como mostrado em (b), $T_{1,0}$ pode realizar sua execução em uma menor velocidade ($\frac{1}{1+0,5} f_{max}$). Em (c), quando $T_{1,0}$ completa sua execução em $t = 1,25$, $T_{2,0}$ pode ser executada com velocidade de $\frac{1}{1+0,75} f_{max}$. Quando $T_{0,1}$ é ativada em $t = 2$, $T_{2,0}$ está ainda executando e será *preemptada* por $T_{0,1}$. Nesse ponto, uma vez que $T_{2,0}$ está parcialmente executada, o restante de sua execução não gastará seu WCET, assim $T_{0,1}$ poderá reclamar essa folga para a sua execução. Nesse caso, somente $0,57$ unidades de tempo estão faltando para a execução de $T_{2,0}$, assumindo que ela requeira seu WCET. Então $T_{0,1}$ pode realizar sua execução até $t = 3,43$ com a velocidade de $\frac{1}{1+0,43} f_{max}$, como mostrado em (d). As outras ativações $T_{2,0}$, $T_{1,1}$ e $T_{0,2}$ são escalonadas de maneira similar. O escalonamento final é mostrado em (e).

Tabela 2.2: Conjunto de tarefas do exemplo de [Kim et al. 2002].

Tarefa	Período	WCET	ACET
T_0	2	1	0,5
T_1	3	1	0,5
T_2	6	1	0,5

Em [Kim et al. 2002] os autores apresentam uma avaliação de vários algoritmos com base em RDV ([Pillai e Shin 2001], [Aydin et al. 2001a], [Kim et al. 2002], [Shin et al. 2000]) para

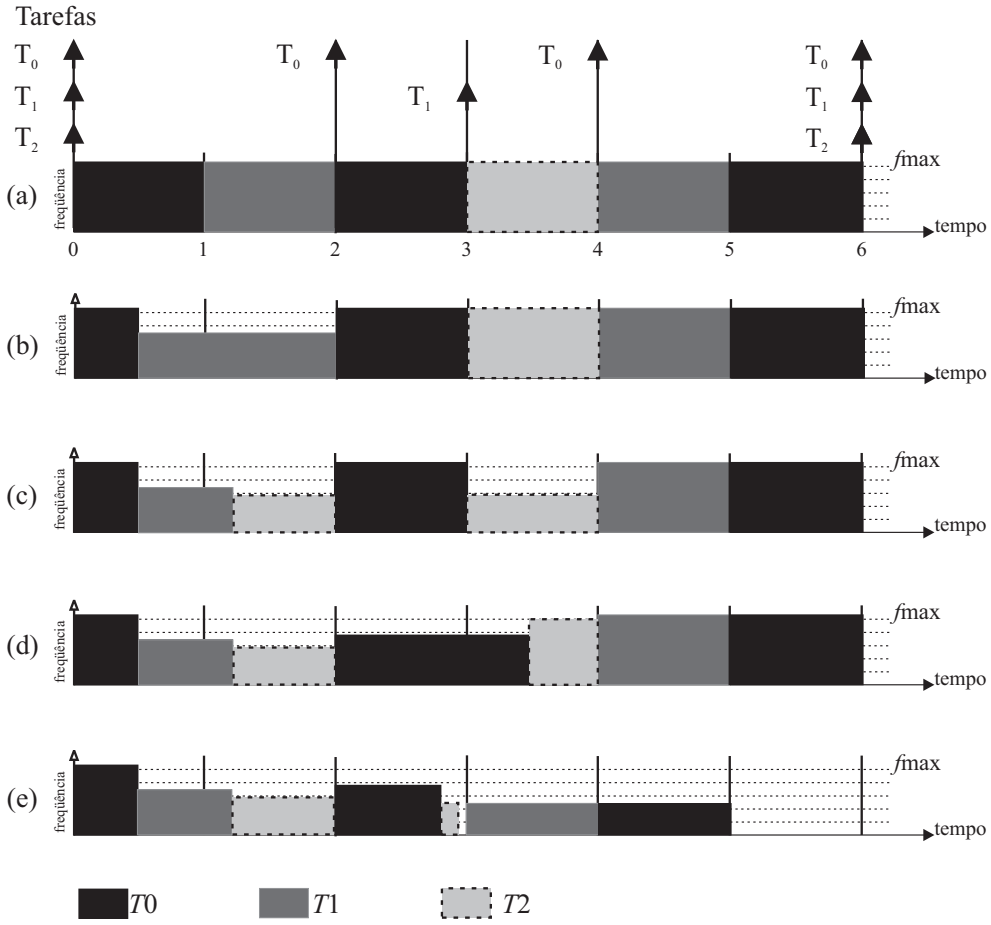


Figura 2.5: Escalonamento do conjunto de tarefas da tabela 2.2.

conjuntos de tarefas periódicas e premptáveis em sistemas de tempo real críticos. Na avaliação é utilizado um simulador que incorpora as características desses diferentes esquemas. Como resultado, eles apresentam uma comparação da eficiência na conservação de energia entre os algoritmos baseados em EDF ([Pillai e Shin 2001], [Aydin et al. 2001a], [Kim et al. 2002], [Shin et al. 2000]) e aqueles que usam RM ([Pillai e Shin 2001], [Shin et al. 2000]) em relação a um limite inferior obtido teoricamente. Os autores concluem o trabalho mostrando que alguns dos algoritmos que utilizam EDF atingem resultados próximos ao limite inferior (cerca de 9 a 12% piores). Para os algoritmos baseados em RM eles reportam resultados que ficam a uma distância significativa desse limite.

Quatro algoritmos (Opt-Clock, PM-clock, Sys-Clock e DPM-Clock) que utilizam a técnica de RDV para sistemas de tempo real críticos baseados em prioridades fixas são propostos por [Saewong e Rajkumar 2003]. O método Opt-Clock *Optimal Clock Frequency Assignment* determina uma velocidade ótima para cada tarefa em cada ativação. O método, porém, só deve ser utilizado estaticamente uma vez que apresenta um alto custo computacional. Sys-clock *System Clock Frequency Assignment* foi desenvolvido para sistemas onde o tempo de chaveamento de

voltagem e de freqüência representa um alto custo. Assim, apenas uma freqüência de relógio é determinada, na admissão das tarefas, e o relógio é mantido constante até que se mude o conjunto de tarefas. O algoritmo PM-Clock *Priority-Monotonic Clock Frequency Assignment* primeiro determina a menor freqüência para se executar todas as tarefas, de forma similar ao Sys-Clock. Depois ele determina uma freqüência para cada tarefa individualmente e, então, na etapa dinâmica cada tarefa executará na freqüência de relógio pré-determinada. Já o algoritmo DPM-Clock *Dynamic PM-Clock* é um esquema dinâmico simples de utilização de *gain time*. Nesse método, estaticamente utiliza-se o PM-Clock e na etapa dinâmica a folga gerada por uma tarefa que termina a execução antes do previsto é toda entregue à próxima tarefa (caso possua prioridade menor ou igual à atual).

Em [Kim et al. 2003] os autores apresentam um algoritmo (denominado *Low-Power Short-Term Work-Demand Analysis* - lpWDA) baseado em roubo de folga para um conjunto de tarefas periódicas, em sistemas de tempo real críticos, utilizando um escalonador RM. A principal característica desse algoritmo está na determinação do tempo que a tarefa pode utilizar, pois ele objetiva aumentar a folga disponível para uma tarefa através do atraso do escalonamento de tarefas de menor prioridade tanto quanto possível. Assim como no trabalho desenvolvido para EDF em [Kim et al. 2002] a determinação do *gain time* para a tarefa atual é feita de duas formas. Uma para determinação da folga proveniente das tarefas de maior prioridade que completam sua execução antes do seu WCET; outra, obtém a folga das tarefas de menor prioridade que durante sua execução utilizaram menos que o seu WCET. Ao serem preemptadas elas permitem que tarefas de maior prioridade possam aproveitar esse *gain time*. Nesse método não há fase *off-line* e os autores reportam que há um pequeno *overhead* para a determinação do *gain time*.

2.3 Classificação dos Algoritmos

Para sistemas de tempo real críticos, como os utilizados nesse trabalho, existem duas metodologias de algoritmos de RDV [Kim et al. 2002]: *Intra-task* DVS (IntraDVS) e *Inter-task* DVS (InterDVS). A principal diferença entre elas é que o IntraDVS utiliza a folga gerada pela tarefa atual para a própria tarefa, enquanto que o InterDVS distribui a folga gerada pela tarefa atual para as demais tarefas.

Algoritmos InterDVS utilizam a estratégia de "executa-calcula-atribui" para determinar a voltagem a ser utilizada na execução de uma tarefa. Esse cálculo pode ser resumido como se segue: (i) executa-se a tarefa atual; (ii) quando a tarefa termina, calcula-se o prazo máximo de

execução para a próxima tarefa; (iii) determina-se a menor voltagem possível para execução da próxima tarefa; e, (iv) executa-se a próxima tarefa. A maioria dos algoritmos InterDVS diferem durante o passo (ii), no cálculo do prazo máximo permitido para a próxima tarefa (t_{prox}), que será igual à soma do tempo de execução da próxima tarefa($C_{t_{prox}}$) e da folga disponível.

Assim como nos algoritmo InterDVS, nos algoritmos IntraDVS calcula-se a freqüência de operação considerando-se o WCET para cada tarefa. Porém, uma vez que a tarefa possui muitos caminhos possíveis de execução, pode existir uma grande variabilidade no tempo de execução. Então, quando o caminho de execução da tarefa é diferente do caminho de execução de pior caso, a tarefa terminará sua execução antes do WCET, resultando em *gain time*. Nesse caso, o IntraDVS explora esse *gain time* para ajustar a velocidade do processador para a tarefa que se encontra executando (tarefa atual). Resumidamente, tem-se: (i) executa-se parte do código; (ii) calcula-se o tempo máximo de execução do restante a ser executado; (iii) determina-se a menor voltagem possível para o término da execução da tarefa e, (iv) executa-se o restante da tarefa. Exemplos de algoritmos IntraDVS podem ser obtidos em [Gruian 2001, Lee e Sakurai 2000, Shin et al. 2001].

Os estudos efetuados para essa dissertação foram direcionados unicamente para métodos InterDVS.

2.3.1 InterDVS

Um algoritmo InterDVS genérico consiste em determinar e distribuir a folga para as próximas tarefas a serem executadas. Na determinação da folga o objetivo é calcular a quantidade máxima da folga que poderá ser utilizada por uma tarefa. Já na distribuição da folga o objetivo é distribuí-la para que haja minimização do consumo de energia.

2.3.1.1 Métodos de Determinação de Folga

Os métodos de determinação de folga podem ser divididos em duas etapas: uma estática, onde os cálculos são realizados antes da execução do conjunto de tarefas, através do uso dos tempos ociosos existentes no hiperperíodo, e outra dinâmica, onde os cálculos são realizados durante o processamento, através do aproveitamento das variações dos tempos de execução das tarefas (alguns métodos também aproveitam a folga remanescente da etapa estática). Os procedimentos explicados adiante diferem entre si principalmente na etapa dinâmica, na determinação de *gain time* e da folga remanescente.

A) Estática

O método de utilização de folga na etapa estática mais comumente utilizado é escolher a menor freqüência (única) de operação possível para o conjunto de tarefas que ainda garanta todas as restrições de tempo. Nesse caso, a folga utilizada é somente parte dos períodos ociosos existentes em um hiperperíodo. Os métodos descritos em [Shin et al. 2000, Pillai e Shin 2001, Aydin et al. 2001a, Kim et al. 2002] utilizam essa fase estática.

B) Dinâmica

Os métodos para a fase dinâmica, ou seja, aqueles que são utilizados durante a execução do conjunto de tarefas, precisam calcular de forma eficiente o tempo que poderá ser utilizado por uma tarefa, incorporando tempos ociosos do hiperperíodo e *gain time*.

B.1) Esticar até que a próxima tarefa esteja pronta para ser executada: Um método simples de utilização dinâmica da folga é utilizar a informação de quando a próxima tarefa estará pronta para executar (NTA), assumindo que a tarefa atual (*ta*) é escalonada no tempo t. Se NTA é maior que t + WCET(*ta*), a tarefa *ta* pode executar em uma freqüência menor para terminar exatamente no NTA. Os algoritmos FUTURE, PAST [Weiser et al. 1994], lppsEDF, lppsRM [Shin et al. 2000], DRA [Aydin et al. 2001a] e ccRM [Pillai e Shin 2001] são exemplos da utilização desse método.

A figura 2.4 ilustra essa técnica. Quando apenas uma tarefa está ativa (pronta para ser executada) como mostrado em 2.4 (a), a execução da tarefa *ta* pode ser esticada até NTA. Quando várias tarefas estão ativas a folga pode ser utilizada por apenas uma tarefa ou pode ser distribuída igualmente entre todas as tarefas ativas, como mostrado, respectivamente, na figura 2.4 (b).

B.2) Atualização da utilização: A utilização real do processador é menor que a utilização de pior caso. Na técnica de atualização de utilização essa é recalculada, no momento do escalonamento de uma tarefa, com base na diferença entre o WCET da tarefa que termina e o tempo que ela utilizou efetivamente (e.g., ccEDF [Pillai e Shin 2001]). Quando a utilização do processador é atualizada, a freqüência do relógio pode ser ajustada para o novo ponto de operação. O mérito principal desta técnica é a simplicidade de implementação, uma vez que somente a utilização da tarefa que termina a sua execução deve ser recalculada em cada ponto de escalonamento.

No algoritmo *Cycle Conserving* EDF (ccEDF), ao término de cada tarefa a freqüência de relógio é ajustada para o menor valor exigido pela carga sistema, calculada com base nos tempos de execução do pior caso das tarefas ainda não executadas e nos tempos efetivamente ocorridos das tarefas já executadas. A equação 2.1 apresenta o teste de escalonabilidade utilizado pelo ccEDF, onde α é tal que $0 \leq \alpha \leq 1$.

$$\sum_{i=1}^{n} C_i/P_i \leq \alpha \tag{2.1}$$

Seja o conjunto de tarefas definido na tabela 2.3, onde estão indicados duas ativações sucessivas para cada tarefa. A figura 2.6 mostra um exemplo simples de aplicação do ccEDF, assumindo-se um processador que só opere com freqüências normalizadas de $0,50$, $0,75$ e $1,00$. No início da execução, considerando-se todas as tarefas ativas, a carga do sistema é $0,746$, admitindo-se o WCET para todas as tarefas. Assim, o relógio é ajustado para $0,75$. Ao término da primeira tarefa, o valor do fator de carga se altera, pois a tarefa não executou por todo o seu WCET. Nessa nova situação, a utilização prevista é de 0,621. Como para o processador considerado a freqüência que garante a escalonabilidade é de 0,75, o relógio não é alterado. Ao término da execução da tarefa 2 o fator de carga passa para $0,421$, podendo então a freqüência ser ajustada para $0,5$.

Tabela 2.3: Conjunto de tarefas para exemplo do ccEDF.

Tarefa	Período	WCET	Ativação 1	Ativação 2
1	8	3	2	1
2	10	3	1	1
3	14	1	1	1

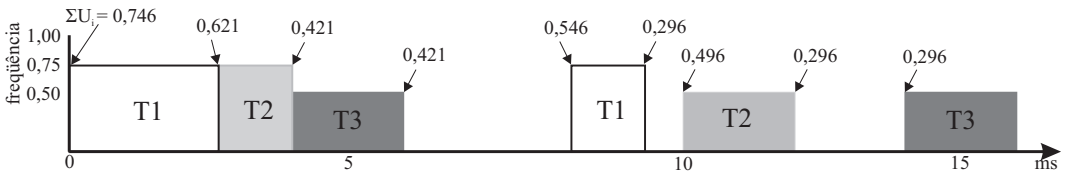


Figura 2.6: Grade gerada pelo algoritmo ccEDF.

B.3) Roubo de folga: Para sistemas de tempo real que consideram tarefas periódicas e aperiódicas, e utilizam escalonadores de prioridades fixas, existem diferentes algoritmos de roubo de folga ([Lehoczky e Ramos-Thuel 1992, Davis et al. 1993, Davis 1993, Tia et al. 1996]). Todos os algoritmos objetivam determinar a quantidade máxima de folga que pode ser roubada

das tarefas periódicas, de tal forma que minimize o tempo de resposta no atendimento de tarefas aperiódicas e que o escalonador ainda garanta todas as restrições de tempo das tarefas periódicas.

Um trabalho seminal de roubo de folga foi proposto por [Lehoczky e Ramos-Thuel 1992]. Nesse trabalho é reportada uma técnica estática, que consiste em armazenar em uma tabela todas as informações (folgas que podem ser roubadas para cada tarefa em cada ativação) do hiperperíodo. O problema desse algoritmo é que ele é estático, e não leva em consideração as variações do tempo de execução das tarefas e pode precisar de uma grande tabela para armazenar todos os dados.

O trabalho de [Davis et al. 1993] apresenta um método eficiente (na determinação da quantidade máxima de folga que pode ser roubada) e que pode ser usado tanto estaticamente quanto dinamicamente. Embora este algoritmo seja eficiente, ele apresenta uma alta complexidade $O(m * n^2)$ para que seja utilizado dinamicamente, o que o torna inviável na prática. Em um trabalho posterior [Davis 1993] apresenta três algoritmos aproximados (SASS, DASS e PASS) para determinação de folga que têm como objetivo principal diminuir a complexidade do método original. O SASS (*Satic Approximate Slack Stealing* - $O(n^2)$) é um algoritmo utilizado somente na fase estática que apresenta uma boa aproximação da quantidade de folga que pode ser roubada ao se comparar com o método proposto em [Davis et al. 1993]. O DASS (*Dynamic Approximate Slack Stealing* - O(n)) é um algoritmo que possui somente a etapa dinâmica e apresenta uma complexidade menor que o SASS (o DASS não utiliza o método *Effective Deadline Enhacement*), porém apresenta uma determinação menos eficiente. Já o PASS (*Periodic Approximate Slack Stealing*) utiliza os algoritmos SASS na etapa estática e o DASS na dinâmica.

Embora os métodos aproximados de Davis reduzam significativamente a complexidade para que viabilizem a sua utilização na fase dinâmica, em alguns casos o valor calculado não apresenta bons resultados. Por exemplo, seja o conjunto de tarefas definido na tabela 2.4. Considere que as tarefas sempre executam seu WCET à máxima voltagem e freqüência. Ao término da execução da tarefa 3 (instante de tempo igual a 22 como indicado na figura 2.7) será calculada a folga para sua próxima ativação. Aplicando-se o algoritmo DASS obtém-se o valor de 8 unidades de folga, enquanto que o valor real é de 11.

O trabalho [Tia et al. 1996] também apresenta um algoritmo de roubo de folga. Nele é apresentada uma variação dos algoritmos [Davis et al. 1993] e de [Lehoczky e Ramos-Thuel 1992]. Enquanto que esses métodos utilizam a heurística gulosa no atendimento das requisições ape-

Tabela 2.4: Conjunto de tarefas para o exemplo do algoritmo DASS.

Tarefa	Período	WCET
1	25	17
2	25	3
3	40	2

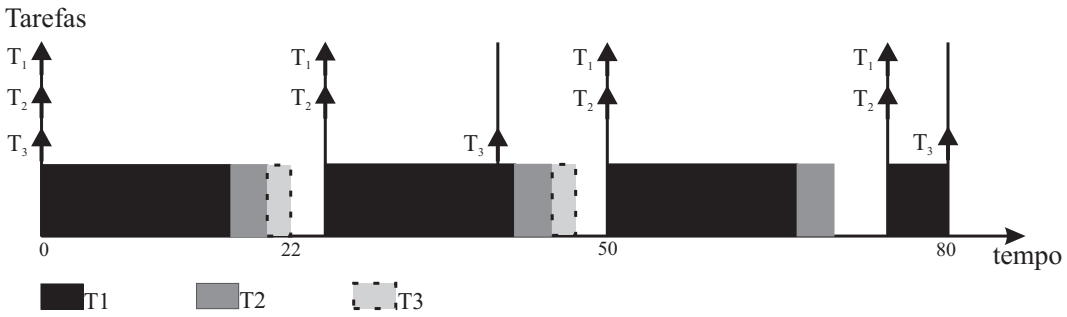


Figura 2.7: Grade de execução do conjunto de tarefas da Tabela 2.4.

riódicas, o algoritmo de [Tia et al. 1996] não necessariamente irá atender a uma requisição aperiódica imediatamente. Eles justificam a política não gulosa por terem demonstrado que o total de folga disponível, em um intervalo específico, pode ser maior para o algoritmo de roubo de folga se certas tarefas de prioridades menores forem atendidas.

Embora os algoritmos de roubo de folga apresentados tenham sido inicialmente desenvolvidos para minimizar o atendimento de tarefas aperiódicas, pode-se utilizar a mesma idéia com pequenas modificações, em conjunto com RDV. Por exemplo, os trabalhos de [Pillai e Shin 2001, Aydin et al. 2001a, Kim et al. 2002, Kim et al. 2003, Urriza et al. 2004] apresentam heurísticas que incorporam o roubo de folga.

2.3.1.2 Distribuição de Folga

Para a distribuição de folga a maioria dos métodos adota a metodologia gulosa, onde toda a folga será repassada à tarefa seguinte. Essa não é uma solução ótima, mas a metodologia gulosa é utilizada devido a sua simplicidade.

2.3.1.3 Resumo dos Métodos Apresentados

Na tabela 2.5 é apresentada uma classificação dos métodos e na tabela 2.6 é fornecido um resumo da classificação dos algoritmos InterDVS discutidos.

Tabela 2.5: Classificação das técnicas que utilizam InterDVS.

Método	Técnica	Escala de decisão
1	Freqüência máxima constante	Estático
2	Esticar até NTA	Dinâmico
3	Atualização da utilização	
4	Roubo de folga	

Tabela 2.6: Classificação dos algoritmos InterDVS.

Escalonador	Algoritmo	Método utilizado	Trabalho
EDF	OPT	2 + 3	[Weiser et al. 1994]
	FUTURE	2 + 3	
	PAST	2 + 3	
	lppsEDF	1 + 2	[Shin et al. 2000]
	DRA	1 + 2 + 3	[Aydin et al. 2001b]
	AGR	3 + 4	
	lpSEH	4	[Kim et al. 2002]
	ccEDF	3	[Pillai e Shin 2001]
	laEDF	4	
RM	ccRM	1 + 2	
	lppsRM	1 + 2	[Shin et al. 2000]
	lpDWA	4	[Kim et al. 2003]
	Opt-Clock	1[1]	[Saewong e Rajkumar 2003]
	Sys-Clock	1[2]	
	PM-Clock	1	
	DPM-Clock	1 + 3	

[1]calculado para cada ativação de cada tarefa
[2]calculado para cada tarefa individualmente, somente no instante crítico

2.4 Outros Trabalhos de Pesquisa Relacionados

Essa seção apresenta outros trabalhos de pesquisa em sistemas de tempo real relacionados com a redução do consumo de energia. Esses trabalhos envolvem conceitos de computação imprecisa e sistemas baseados em recompensa, sistemas multiversão e qualidade de serviço (QoS).

A computação imprecisa está relacionada com o conceito de escalonamento baseado em recompensa (*Reward Based*), que se refere ao problema no qual existe uma recompensa associada com a execução de uma tarefa. Esse problema é conhecido também na literatura como IRIS (*Increased-Reward with Increased-Service*), onde cada tarefa de tempo real apresenta uma componente obrigatória e outra opcional. A parte obrigatória deve ser executada necessariamente antes do prazo da tarefa e a parte opcional pode ser interrompida a qualquer momento, com uma função de recompensa crescente associada.

Alguns trabalhos importantes de computação imprecisa com restrições de energia foram apresentados em [Rusu et al. 2002], com enfoque no modelo de prazo único, ou seja, considera que existe apenas um único prazo (*deadline*) para todo o sistema e não um para cada tarefa. Em [Rusu et al. 2003a] considera-se que cada tarefa executa em um nível de voltagem/freqüência e para cada uma existe uma função de energia associada. Nesse trabalho são apresentados algoritmos para aplicações com consumo homogêneo de energia (onde todas funções de energia são idênticas) e também algoritmos para aplicações onde as funções são heterogêneas no consumo de energia, com garantia de limite superior. No trabalho de [Aydin et al. 2001a] é estabelecida uma equivalência entre o problema do escalonamento estático de tarefas com redução do consumo e o problema do escalonamento baseado em recompensa.

O estudo apresentado por [Rusu et al. 2003b] preocupa-se com a questão de oferecer uma variedade de níveis de QoS para a aplicação de tempo real, através da idéia de sistemas multiversão. Trata-se de sistemas capazes de contar com múltiplas versões do programa de aplicação, cada uma com diferentes níveis de recompensa, restrições de prazo e consumo de energia. A técnica de múltiplas versões tem sido usada no contexto de sistemas tolerantes a falhas para alcançar um certo nível de redundância de *software*. O objetivo de se ter um escalonamento multiversão em um sistema de tempo real é o de prover diferentes níveis de QoS, em função de restrições de prazo como, por exemplo, no caso de aplicação de processamento de vídeo. Ainda relacionado com QoS, o trabalho apresentado por [Quan et al. 2004] relaciona otimização de energia com os requisitos quantificados através de restrições (m, k), onde pelo menos m de qualquer seqüência de k ativações de uma tarefa devem cumprir os seus prazos.

O trabalho de [Melhem et al. 2004] apresenta duas políticas de colocação de *checkpoints* que permitem o sistema recuperar de uma falha e ainda reduzir o consumo de energia. As políticas reduzem a velocidade do processador durante a operação normal e, quando ocorre uma falha, o processador re-executa todo o código após o último *checkpoint* na velocidade máxima, para garantir o cumprimento do prazo da tarefa. A primeira política coloca os *checkpoints* uniformemente, de forma periódica, reservando uma parte da folga para a execução da recuperação. Assim, se o número de *checkpoints* for grande, menor será o tempo necessário para a recuperação. Por outro lado, quanto maior o número de *checkpoints*, maior também será o *overhead* causado pelos mesmos, que é também descontado da folga. Nessa técnica, se nenhuma falha ocorre, o tempo reservado para a recuperação é perdido. A segunda política pressupõe que a ocorrência de falhas é um evento raro, com a distância entre *checkpoints* diminuindo à medida que se aproxima do prazo. Caso ocorra uma falha antes do primeiro *checkpoint*, apesar do có-

digo a ser re-executado ser maior, existe tempo suficiente para executar tudo novamente antes do prazo.

O trabalho desenvolvido por [Mossé et al. 2004] propõe um compilador *power-aware*. Esse método é voltado para aplicações executando em sistemas embutidos, onde o WCET e o tempo médio de execução de seções de código de uma tarefa (e.g., *loops*, *procedures*) sejam conhecidos. O compilador insere pontos de gerenciamento de energia no início e no fim das seções para monitorar o tempo de execução, verificar a velocidade do processador e, eventualmente, alterá-la. A comparação entre o tempo de execução e o WCET permite ou incrementar ou diminuir a velocidade do processador. Outros métodos baseados nessa mesma técnica procuram, principalmente, otimizar a ocupação de memória (um problema importante em sistemas embutidos) e, assim, economizar energia [Unsal e Koren 2003].

O trabalho apresentado por [Sharma et al. 2003] aborda técnicas para sistemas de tempo real não crítico, onde o objetivo é minimizar o consumo e garantir QoS em sistemas com carga não determinística, como, por exemplo, uma composição de servidores *back-ends* (*server farms*), onde as requisições dos clientes são distribuídas entre eles. Cada servidor recebe aperiodicamente as requisições dos clientes e essas são divididas em classes, com diferentes requisitos de QoS. O problema consiste em reduzir o consumo e garantir os requisitos de tempo de resposta. A base da técnica é reduzir a velocidade do processador até que seja observado o não cumprimento de alguns prazos. A técnica apresentada pelo artigo é baseada em um controle realimentado onde o ajuste, ou *set-point*, é um limite de utilização. Quando a utilização do servidor é inferior ao limite a velocidade é diminuída, e quando a utilização excede o limite a velocidade aumenta.

O artigo [Yuan e Nahrstedt 2003] apresenta um esquema de escalonamento, GRACE-OS, para sistemas móveis multimídia. GRACE-OS utiliza o mecanismo de RDV em um escalonador voltado para aplicações não críticas de tempo real para decidir qual a velocidade de processamento a ser empregada e também para determinar por quanto tempo uma tarefa será executada. As decisões de escalonamento são tomadas com base na distribuição de probabilidades das demandas das aplicações, que são obtidas através da estimação *on-line* do perfil dessas aplicações. Os autores indicam economias de energia variando entre 7% e 72%, oferecendo garantias probabilísticas de desempenho.

2.5 Comentários

Economizar energia no escalonamento de tarefas é uma ação que explora os períodos ociosos do processador. A forma com que cada algoritmo determina quanto tempo cada tarefa poderá executar é que determina sua eficiência na economia de energia. Nos trabalhos aqui discutidos e também nos métodos propostos nessa dissertação considera-se que o *overhead* de chaveamento para troca de voltagem e para troca de freqüência é nulo, visto que esse *overhead* pode ser incorporado ao tempo de execução das tarefas.

A eficiência dos algoritmos baseados em RM é menor quando comparada aos algoritmos que utilizam EDF. Isto ocorre devido a dois fatores (além, é claro, do limite no fator de utilização). Primeiro, os métodos baseados em RM têm sido muito menos investigados. Foram aqui identificados e discutidos quatro métodos dinâmicos de economia de energia que utilizam RDV em conjunto com o escalonador RM. Segundo, é mais difícil desenvolver algoritmos eficientes RM que utilizem RDV ao se comparar com algoritmos baseados em EDF. Por exemplo, o método de roubo de folga baseado em prioridades, comumente utilizado em algoritmos EDF, como em [Aydin et al. 2001a, Kim et al. 2002], apresentam bons resultados como indicado em [Kim et al. 2002]. Esse método é definido por uma simples heurística: os tempos não utilizados por tarefas de maiores prioridades que terminaram sua execução são utilizados pelas tarefas seguintes (que possuem prioridade menor ou igual). Contudo, uma vez que as tarefas nos sistemas de prioridades fixas do tipo RM possuem sempre a mesma prioridade, essa técnica não funciona tão bem quanto no escalonamento EDF. No escalonamento EDF as mudanças dinâmicas de prioridades servem como um eficiente distribuidor de *gain time* entre as tarefas. Então, no escalonamento RM, as tarefas de maior prioridade tendem a ter menos folga que as tarefas de menor prioridade. Quanto maior for a prioridade de uma tarefa, maior tende a ser sua freqüência. Esse desbalanceamento, geralmente, resulta em um menor desempenho na economia de energia.

O objetivo principal deste trabalho é desenvolver um método dinâmico para determinação e utilização de folga em sistemas de tempo real. Esse método utilizará regulagem dinâmica de voltagem, segundo o escalonador *Rate Monotonic*, e terá de ser eficiente quanto à economia de energia no escalonamento de tarefas. No próximo capítulo serão apresentados diferentes métodos para determinar quanto tempo (folga mais WCET) uma tarefa poderá utilizar.

Capítulo 3

Métodos para Economia de Energia

O escalonamento com prioridades fixas é o método mais utilizado atualmente para a implementação de aplicações sujeitas a restrições de tempo real. No entanto, nos diversos métodos de economia de energia encontrados na literatura o escalonamento é majoritariamente realizado usando-se prioridades dinâmicas, mais especificamente, o algoritmo EDF. Algoritmos baseados em prioridades dinâmicas, como EDF e LLF, são mais eficientes no aproveitamento do processador que aqueles baseados em prioridades fixas, como o RM, sendo o seu limite teórico de utilização igual a 100%. Esses algoritmos são capazes de tratar tanto tarefas aperiódicas quanto periódicas, podendo ser usados em aplicações de tempo real críticas e não críticas. Contudo, se as aplicações necessitam somente de tarefas periódicas, o algoritmo RM, por sua implementação eficiente, simples e intuitiva, mostra-se uma opção interessante. Adicionalmente, RM é um algoritmo muito utilizado em sistemas industriais e foi indicado pelo DoD dos EUA para aplicações de tempo real críticas [Obenza 1993]. Uma razão para isso é o fato do EDF ter um comportamento instável em circunstâncias especiais, como, por exemplo, em condições de sobrecarga [Buttazzo 2003]. A grande aceitação de RM como base para os sistemas operacionais de tempo real atuais foi um importante motivador para a sua utilização nesse trabalho. Nesse capítulo são apresentados os diferentes métodos de determinação de folga propostos nessa dissertação. O objetivo de cada um deles é determinar a quantidade máxima de tempo que poderá ser utilizado por uma tarefa entre o início de sua execução e o seu prazo.

3.1 Fase 1: Escala Estática de Freqüências

Dado um conjunto de tarefas, definido para uma determinada fase de operação do sistema, a pior situação de carga que pode ser apresentada para o escalonador é quando ocorre o instante crítico e quando todas as tarefas executam o WCET. Nessa situação, o objetivo é escolher a me-

nor freqüência de operação possível para o conjunto de tarefas, de tal forma que o escalonador RM ainda garanta todas as restrições de tempo. Nessa fase, o método aqui apresentado funciona como uma variante ao proposto para o algoritmo RM em [Pillai e Shin 2001], procurando utilizar a folga disponível que ocorre para o sistema quando opera em freqüência máxima. A escalonabilidade do conjunto de tarefas é garantida pelo teste definido em [Joseph e Pandya 1986] e é apresentado na equação 3.1. Nessa equação, hp_i denota o conjunto das tarefas de maior prioridade que a tarefa i e R_i significa o tempo de resposta. Logo, o sistema é escalonável se $R_i \leq T_i$.

$$R_i = C_i + \sum_{j \in hp_i} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{3.1}$$

A primeira fase é utilizada em todos os métodos dinâmicos e já oferece uma substancial melhoria na utilização da energia. Contudo, a economia é limitada pela quantidade de níveis de voltagem e freqüência do processador e pelo fator de utilização do sistema. Como os modelos de processadores reais ainda não funcionam com variações contínuas no mecanismo de RDV, nem sempre é possível utilizar uma freqüência ótima e é, portanto, selecionado o nível discreto imediatamente acima ao que seria o ótimo para o RM.

A título de exemplo, sejam as tarefas da tabela 3.1. Admitindo-se uma situação de instante crítico, a figura 3.1(a) apresenta a escala gerada pelo algoritmo RM para o hiperperíodo desse conjunto de tarefas, assumindo-se freqüência máxima de operação e WCET para todas as tarefas. O modelo de processador considerado nesse caso é ideal e apresenta variação contínua de voltagem e freqüência. Nessa figura, os números nos retângulos indicam as tarefas e *e* representa os períodos ociosos do processador.

É possível notar que, sem a utilização de RDV, as tarefas executam à máxima freqüência e por 75% do tempo do hiperperíodo. Existirá, então, 25% de folga a cada hiperperíodo. Em 3.1(b) está representada a escala ótima em termos de ocupação de períodos ociosos, desde que seja possível ao processador assumir qualquer freqüência e voltagem. Em 3.1(c) está representada a escala gerada pela aplicação da fase estática do algoritmo, onde todas as tarefas executam à mesma freqüência (menor que a máxima). Como pode ser observado, mesmo após essa fase, ainda restam períodos de folga. Comparando-se as figuras 3.1(b) e 3.1(c) vê-se que há possibilidade de otimização do mecanismo, o que será feito em sua fase dinâmica. Graficamente, dado que a energia é calculada como o produto da potência pelo tempo, pode-se visualizar o ganho obtido como a área limitada entre a linha horizontal de f_{max} e o topo dos

retângulos indicativos de execução das tarefas.

Tabela 3.1: Conjunto de tarefas para exemplo da fase estática.

Tarefa	Período	WCET
1	3	1
2	4	1
3	6	1

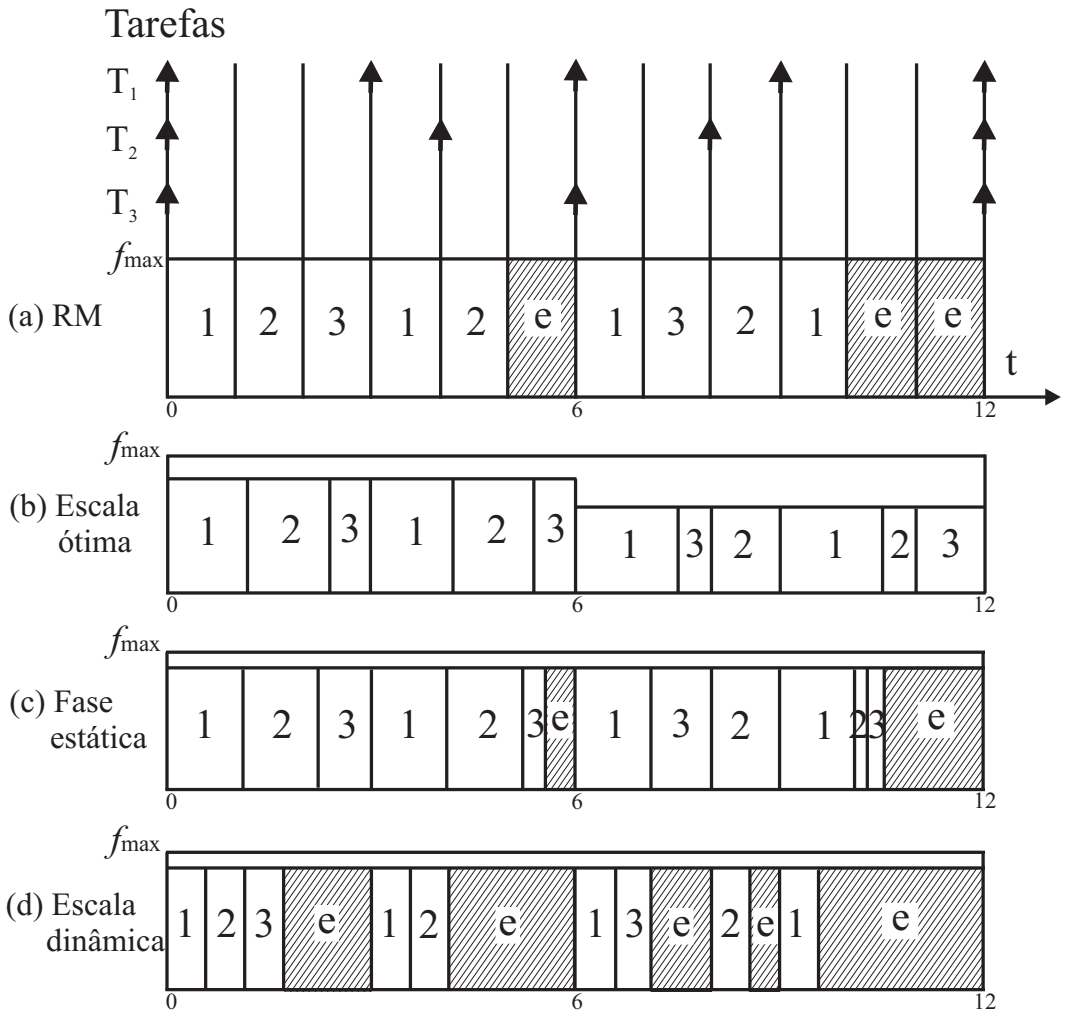


Figura 3.1: Escalonamento do conjunto de tarefas da Tabela 3.1.

Em relação ao fator de utilização, essa fase não aproveita de forma ótima os tempos ociosos do processador (como visto no exemplo anterior), sendo, portanto, utilizada uma fase dinâmica que pode aproveitar a folga remanescente. Além disso, a fase dinâmica se faz necessária visto que, embora os conjuntos de tarefas de tempo real crítico sejam especificados com base em seu WCET, essas tarefas, durante sua execução, utilizam, em média, um tempo menor. Cada vez que uma tarefa está para ser executada não há como saber quanto tempo ela gastará e, para manter a escalonabilidade do conjunto, deve-se considerar que ela utilizará o seu tempo

máximo. Ao fim da execução de uma tarefa pode-se observar quanto tempo ela utilizou e caso ocorra um *gain time* este poderá ser alocado pelos métodos dinâmicos.

Na figura 3.1(d) apresentamos a escala dinâmica gerada pelo algoritmo RM considerando que há somente utilização da fase estática e que as tarefas executam utilizando 50% do seu WCET. Nesse caso é interessante notar a quantidade de folga que surge no sistema, que totaliza mais de 50% do hiperperíodo.

3.2 Fase 2: Escala Dinâmica de Freqüências

Nessa seção são descritos os métodos dinâmicos para economia de energia no escalonamento de tarefas. O objetivo principal desses métodos é determinar e distribuir a folga do sistema para minimizar o consumo de energia do processador através da diminuição dos tempos ociosos do hiperperíodo.

Imediatamente antes de se executar uma tarefa utiliza-se um dos métodos de determinação de folga para determinar o tempo máximo que a tarefa poderá utilizar. Para a distribuição dessa folga é utilizada a heurística gulosa. Essa escolha foi feita em virtude de sua baixa complexidade de implementação e também pelo fato de que, em muitas aplicações, o tempo de execução de uma tarefa pode ser bem menor que o seu WCET [Ernst e Ye 1997]. Deve ser notado que, em sistemas que apresentam muito *gain time*, a heurística gulosa apresenta melhores resultados, pois, desde o início da execução, de forma otimista, todo o *gain time* que vai surgindo é aproveitado.

3.2.1 Método Aproximado de Determinação de Folga

O trabalho apresentado por [Santos et al. 2004] trata do problema de escalonamento de tarefas de tempo real crítico, onde o sistema considerado é monoprocessado, com tarefas periódicas, independentes e preemptáveis. Adicionalmente, as liberações das tarefas e os seus tempos de execução são iguais em cada hiperperíodo. Uma certa fração do hiperperíodo é dedicada à execução de tarefas periódicas críticas, com política de escalonamento RM, sendo o tempo ocioso disponibilizado para outros usos (e.g., tarefas aperiódicas não críticas).

Assim, o problema é como redistribuir a folga de tal forma que nenhum prazo crítico seja perdido e, ainda, que a qualidade de serviço associada às tarefas não críticas seja melhorada. Nesse trabalho, eles utilizaram uma política que atrasa a execução de tarefas críticas, sem perda

de prazo, e adianta o "tempo de folga" para a execução de tarefas não críticas.

A política de [Santos et al. 2004] foi baseada no método de *slots* vazios de [Santos e Orozco 1993] no qual o tempo é considerado como dividido (*sloted*), e onde a duração de um *slot* é denotada como uma unidade de tempo. Esse trabalho admite um conjunto de n tarefas periódicas, preemptáveis e independentes, $S_n = (C_1, T_1, D_1), (C_2, T_2, D_2), ..., (C_n, T_n, D_n)$, onde C_i é tempo de execução de pior caso, T_i é o período e D_i é o prazo. É também assumido que, $\forall i$, $D_i = T_i$. Para simplificar, assumiu-se que as tarefas são liberadas no início dos *slots* e que os tempos de execução, períodos e prazos são múltiplos desses *slots* de tempo.

[Santos e Orozco 1993] formalmente provaram que S_n é RM escalonável se:

$$\forall i \in (1, 2, ..., n)\ T_i \geq menor\ t |\ t \leftarrow C_i + \sum_{h=1}^{i-1} C_h \left\lceil \frac{t}{T_h} \right\rceil \tag{3.2}$$

O escalonador RM não deixa *slots* vazios caso existam tarefas pendentes para execução. Contudo, *slots* vazios existem somente quando todas as tarefas liberadas no intervalo [1, *slot* vazio] tiverem sido executadas. O último termo do lado direito da expressão (3.2) é chamado função de trabalho, denotada $W_{i-1}(t)$. Se M denota o hiperperíodo, a folga (ou número de *slots* vazios no hiperperíodo) será $M - W_n(M)$.

A utilização associada à tarefa T_i é definida como C_i/T_i. Assim, a utilização total é $\sum_{i=1}^{n} C_i/T_i$. Se ela é menor que 1, *slots* vazios serão disponibilizados para expandir o sistema e incorporar outras tarefas. Se ela é igual a 1, o sistema está saturado e não se pode adicionar outras tarefas. Se for maior que 1 o sistema está supersaturado e não é possível escalonar nem as tarefas críticas.

Nessa dissertação é apresentado um algoritmo de escalonamento de tarefas de tempo real críticas periódicas, que utilizará como base o método dos *slots*. A idéia principal do algoritmo é utilizar os *slots* livres para diminuir a velocidade de execução de tarefas críticas, ao invés de utilizá-los para atender tarefas aperiódicas e ainda incorporar outras funções (e.g., variação do tempo de execução das tarefas). A folga a ser determinada por este método consiste na folga remanescente da etapa estática mais o *gain time*.

3.2.1.1 Determinação da Folga

É aqui assumido que, em um tempo t qualquer, os seguintes dados estão disponíveis para o sistema operacional:

$l_i(t) \Rightarrow$ tempo no qual a tarefa i teve sua última liberação para ser executada
$x_i(t) \Rightarrow$ próxima liberação da tarefa i; $x_i(t) = l_i(t) + p_i$
$d_i(t) \Rightarrow$ vencimento da próxima invocação da tarefa i; $d_i(t) = x_i(t) + p_i$
$ta \Rightarrow$ tempo atual
$wa(t) \Rightarrow$ tempo total de execução em relação ao início do hiperperíodo
$S_i(t) \Rightarrow$ folga disponível para a tarefa i (Contador da tarefa i)
$c_i \Rightarrow$ tempo de execução da tarefa i
$tresc_i \Rightarrow$ tempo restante de execução da tarefa i no instante atual
$p_i \Rightarrow$ período da tarefa i
$\epsilon \Rightarrow$ variável auxiliar
$hp_i \Rightarrow$ conjunto das tarefas de maior prioridade que a tarefa i

Notar que $l_i(t)$, $x_i(t)$, $d_i(t)$ são calculados no tempo t. Caso t seja 0, $\forall_i$, $x_i(t) = 0$.

O método Aproximado de determinação de folga é baseado no trabalho de [Urriza e Orozco 2004] e objetiva determinar a quantidade máxima de folga, $S_i(t)$, que pode ser roubada para tarefa i durante o intervalo $[ta, t_{at\_i} + d_i(t)]$, de tal forma que o escalonador RM ainda garanta todas as restrições de tempo. Esse método utiliza contadores de folga individuais para cada tarefa. Qualquer período de tempo ocioso entre o término da execução da tarefa i e o seu prazo poderá ser trocado com a execução de i sem que esta perca seu prazo. Contudo, a quantidade máxima de folga que pode ser roubada é igual ao total do período ocioso do intervalo acima indicado. Para garantir que a tarefa i manterá seu prazo, considera-se o tempo de execução de pior caso da tarefa atual e das demais tarefas a serem executadas. Assume-se também que todas tarefas do conjunto hp_i serão re-invocadas tão cedo quanto possível na próxima liberação da tarefa i e, subseqüentemente, a cada período. Para esse método, ao término da fase estática, são conhecidas as folgas de todas as tarefas ao início do hiperperíodo. O algoritmo para cálculo de folga está indicado através do pseudo-código apresentado adiante.

Ao fim da execução de uma tarefa (ou seja, quando o escalonador é chamado): (i) a folga por ela utilizada deve ser subtraída das folgas de todas as outras tarefas (há somente atualização de contadores - $O(n)$); (ii) caso essa tarefa tenha produzido um *gain time*, ele é acrescentado à folga de todas as tarefas de menor prioridade (também aqui há somente atualização de contadores); e,

(iii) a folga para a próxima ativação dessa tarefa é então calculada (é somente nesse momento que se utiliza o algoritmo de determinação de folga - $O(n^2)$). Caso ao fim dessa execução haja um período ocioso, o valor desse período é subtraído de todas as folgas (novamente, só atualização de contadores).

Como o algoritmo utilizado é $O(n^2)$ utiliza-se contadores para manter a informação da folga disponível para cada tarefa. Assim, somente quando uma tarefa terminar de executar é que utiliza-se o algoritmo para se determinar a folga disponível para a próxima ativação da mesma. Em todos os outros casos ocorre apenas uma atualização dos contadores o que simplifica a utilização do método.

Algoritmo 3.1 - Aproximado.

1: **Se** Primeira tarefa **Então**
2: $S_i(t) \leftarrow d_i(t) - ta - c_1$
3: **Senão**
4: $\epsilon \leftarrow 0,00000001$;
5: $S_i(t) \leftarrow 0$;
6: $wa(t) \leftarrow \sum_{\forall\ j\ \in\ hp(i)\ \cup\ i} \left(\left\lfloor \frac{ta}{p_j} \right\rfloor * c_j - tresc_j \right)$
7: **Para** j $\leftarrow 1$ **até** i **Faça**
8: $aux \leftarrow \left\lfloor \frac{d_i(t)}{p_j} \right\rfloor * p_j$;
9: **Se** $(Abs(aux - (d_i(t))) < \epsilon)$ **e** $(j \neq i)$ **Então**
10: $aux \leftarrow aux - p_j$
11: **Fim Se**
12: $auxfolga \leftarrow aux - ta - \left(\sum_{j=1}^{i} \left\lceil \frac{aux}{p_j} \right\rceil * c_j \right) + \ wa(t)$
13: **Se** $(Abs(S_i(t) - auxfolga) > \epsilon)$ **e** $(S_i(t) - auxfolga < \epsilon)$ **Então**
14: $S_i(t) \leftarrow auxfolga$
15: **Fim Se**
16: **Fim Para**
17: **Fim Se**
18: Retorne $S_i(t)$

Seja o conjunto de tarefas definido na tabela 3.1. Admitindo-se uma situação de instante crítico e que todas tarefas executam o WCET à máxima freqüência, a figura 3.2 apresenta a escala dinâmica gerada pelo algoritmo RM para o hiperperíodo desse conjunto de tarefas.

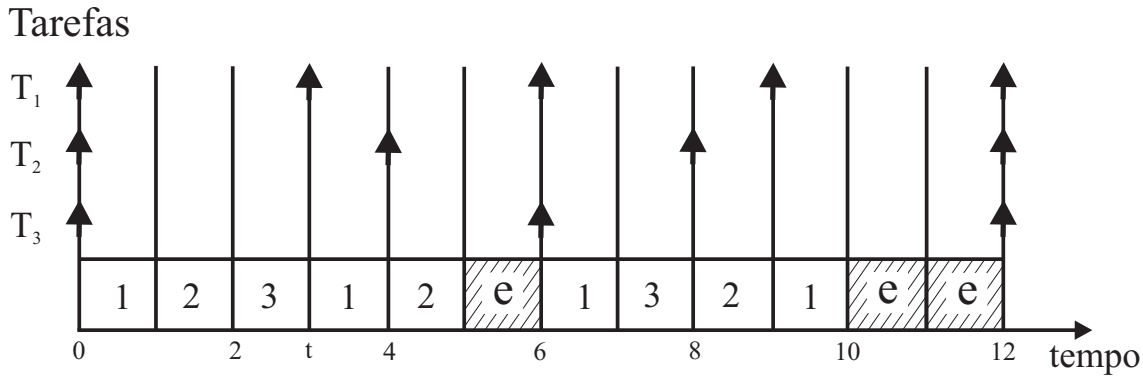


Figura 3.2: Grade de execução do conjunto de tarefas I.

A tabela 3.2 apresenta a quantidade de folga disponível, em relação à figura 3.2, imediatamente antes da execução de cada tarefa. Nessa tabela indica-se os contadores individuais para cada tarefa (denominados contadores 1 a 3; S_i no algoritmo 3.1). Como a folga disponível para a tarefa é igual ao contador de menor valor, esse é apresentado em negrito no instante considerado. Por exemplo, seja o intervalo de tempo 0-1, no qual a tarefa 1 será executada. Com base nos valores considerados dos contadores 1 à 3 (valores 2, 1 e 1 respectivamente) vê-se que a folga disponível será de 1. Esse valor é a quantidade de tempo que a tarefa 1 poderá utilizar, além do WCET, na aplicação do mecanismo de RDV para economia da energia. Pode ser observado que na tabela que a linha correspondente ao intervalo de tempo 12-13 apresenta os mesmos valores da linha 0-1. Isso ocorre porque iniciou-se um novo hiperperíodo e, como as tarefas nesse caso sempre executam o WCET, os valores de folga nos instantes (intervalos) correspondentes em cada hiperperíodo serão sempre iguais.

3.2.1.2 Distribuição da Folga

No início da execução de uma tarefa é verificada a quantidade de folga para ela disponível, cujo valor será igual à menor folga existente entre os contadores de todas as tarefas. A seguir são descritos refinamentos para distribuição de folga, que permitem melhorias suplementares na economia de energia.

Tabela 3.2: Folgas disponíveis.

Intervalo	Tarefa	Contador 1	Contador 2	Contador 3	Folga
0-1	1	2	**1**	1	**1**
1-2	2	4	**1**	1	**1**
2-3	3	3	4	**1**	**1**
3-4	1	**2**	3	3	**2**
4-5	2	4	**2**	3	**2**
5-6	*e*	3	4	3	**-**
6-7	1	**2**	3	2	**2**
7-8	3	4	3	**2**	**2**
8-9	2	3	**2**	3	**2**
9-10	1	**2**	3	3	**2**
10-11	*e*	4	3	3	**-**
11-12	*e*	3	2	2	**-**
12-13	1	2	**1**	1	**1**

A) Execução em Um Nível:

Esse método utiliza toda a folga (folga mais *gain time*) disponível para verificar se é possível baixar a freqüência de operação da tarefa atual. Caso seja possível, ela executará nessa nova freqüência. Como os processadores comerciais que permitem RDV possuem um número discreto de níveis de voltagem, à tarefa atual, na maioria das vezes, será atribuído um nível imediatamente acima daquele que seria o ideal. Caso o *gain time* disponível não seja suficiente para se baixar ao menos um nível a tarefa executará normalmente na freqüência em que se encontra e a folga poderá ser alocada para a próxima tarefa, uma vez que a folga é calculada imediatamente antes da execução da tarefa. Notar que esse método representa a simples implementação da heurística gulosa.

As figuras 3.3, 3.4 e 3.5 ilustram o funcionamento da técnica de RDV. Nelas, as setas claras delimitam o período da tarefa e *D* representa o prazo máximo de execução. A figura 3.3 apresenta a execução de uma tarefa sem utilização de RDV. Nessa figura a tarefa executa à voltagem e freqüência máximas, requerendo *C* unidades de tempo e tendo uma certa quantidade de folga. Na figura 3.4 mostra-se a execução da mesma tarefa com utilização de RDV. Nela, a tarefa executa em uma voltagem *v'*, menor que a voltagem máxima, requerendo, assim, mais unidades de tempo para executar o mesmo número de ciclos, diminuindo a folga e também a quantidade de energia gasta na sua execução.

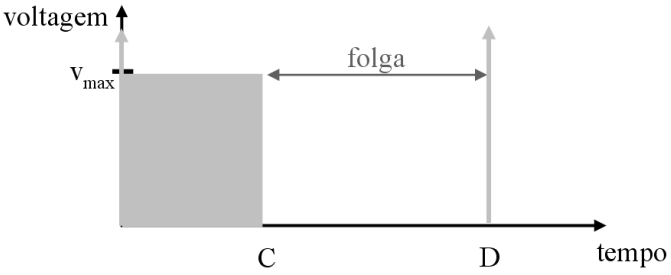


Figura 3.3: Execução de uma tarefa sem utilização de RDV.

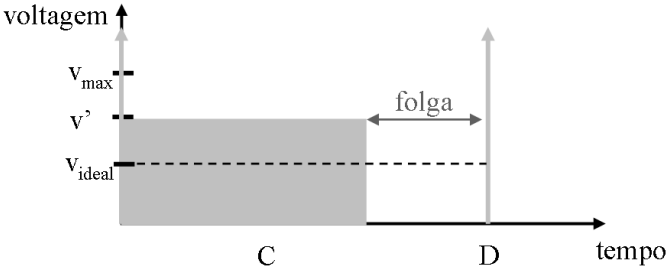


Figura 3.4: Execução de uma tarefa com utilização de RDV.

B) Execução em Dois Níveis:

Um refinamento pode ser feito com base nos teoremas apresentados em [Ishihara e Yasuura 1998] e citados no capítulo 2. Agora, ao invés da tarefa executar em um nível de voltagem e freqüência ela será executada em dois níveis. Para se executar em dois níveis deve-se primeiro determinar quais serão os níveis e depois quanto tempo ela executará em cada um deles. Então, na determinação dos níveis esse método utiliza toda a folga (folga mais *gain time*) disponível para determinar o nível ótimo (f_{opt}) de operação, conforme descrito na expressão 3.3. Nessa expressão $nivel\_atual$ é o nível de freqüência da tarefa atual. Uma vez determinado o nível ótimo, a tarefa será executada utilizando-se os níveis imediatamente abaixo (f_{baixo}) e acima (f_{alto}). Em seguida, calcula-se o instante de tempo da execução da tarefa no qual ocorre a transição da execução do nível baixo (f_{baixo}) para o nível alto (f_{alto}) conforme a expressão 3.4. Com estas duas expressões determina-se a execução da tarefa atual considerando o WCET do $tresc_i$. Afim de aproveitar variações no tempo de execução das tarefas, e de forma otimista, as tarefas serão inicialmente executadas no menor dos dois níveis. A figura 3.5 ilustra a utilização de dois níveis de voltagem/freqüência na execução de uma tarefa.

$$f_{opt} \leftarrow \frac{tresc_i}{tresc_i + S_i(t)} * nivel\_atual \tag{3.3}$$

$$instante\_transicao \leftarrow \frac{f_{alto} - f_{opt}}{f_{alto} - f_{baixo}} * (tresc_i + S_i(t)) \tag{3.4}$$

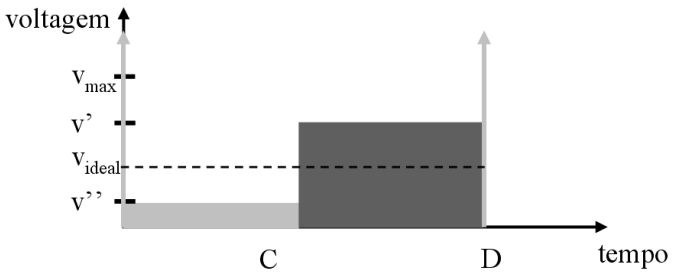


Figura 3.5: Execução de tarefa com utilização de RDV e dois níveis de voltagem.

O método Aproximado para cálculo de folga aqui descrito serviu de base para o estudo inicial. Serão apresentados a seguir outros métodos (variação de DPM-Clock e ccRM) propostos por diferentes autores e as propostas de modificações desses feitas nessa dissertação. Esses outros algoritmos são importantes para comparação dos resultados.

3.2.2 *Greedy Gain Time*

Apresenta-se nesse instante um método simples denominado *Greedy Gain Time* (GGT) que é composto de duas etapas. A primeira, estática, que utiliza a fase estática descrita anteriormente. A segunda, dinâmica, que não utiliza a folga remanescente do sistema sendo considerado e que, portanto, somente utiliza o *gain time*. Ao término da execução de uma tarefa determina-se o valor do *gain time* que poderá ser utilizado pela próxima tarefa (caso a tarefa possua prioridade igual ou menor que a tarefa atual). Esse pode ser diminuído caso exista espaço ocioso entre o término da uma tarefa e a ativação da próxima. Esse método funciona como uma variação ao DPM-Clock proposto por [Saewong e Rajkumar 2003]. Uma diferença entre os métodos está na fase estática, já que o GGT utiliza uma única velocidade de relógio para todas tarefas enquanto que o DPM-Clock determina uma velocidade para cada tarefa. Na etapa dinâmica ambos utilizam a mesma idéia de determinação e distribuição de *gain time*. Outra diferença é que na etapa dinâmica o GGT também pode executar a tarefa em um nível de voltagem (igual ao DPM-Clock) ou em dois níveis de voltagem. Embora a determinação do *gain time* no GGT (aproveita somente *gain time*) a ser utilizado seja diferente do método Aproximado (através do algoritmo 3.1 que aproveita o *gain time* mais a folga remanescente) utiliza-se, com pequenas modificações, a mesma idéia de um nível e dois níveis para a distribuição de folga conforme descrito naquele método. O algoritmo 3.2 apresenta o pseudo-código da determinação do *gain time*.

Algoritmo 3.2 - *Greedy Gain Time.*

assume-se que f_j é a freqüência definida na fase estática

$folga \leftarrow 0$

Laço

  Se $folga > 0$ **Então**

    utiliza menor freqüência $f_i \in \{f_1, ..., fm | f_1 < ... < f_m\}$

    $C_i/(C_i + folga) \leq f_i/fm$

    após executar a tarefa diminua a folga

    termino_execucao(T_i)

  Senão

    executa a tarefa na freqüência f_j

    termino_execucao(T_i)

  Fim Se

  termino_execucao(T_i) :

  Se $tempo\_de\_execucao\_T_i < WCET_i$ **Então**

    $folga \leftarrow WCET_i - tempo\_de\_execucao\_T_i$

  Senão

    $folga \leftarrow 0$

  Fim Se

Fim Laço Fim do hiperperíodo

3.2.3 *Cycle Conserving RM Greedy- ccRMG*

Será apresentado agora, simplificadamente, o método desenvolvido por [Pillai e Shin 2001] e, em seguida, uma modificação aqui proposta para a distribuição de folga. No algoritmo ccRM, toda vez que uma tarefa vai ser executada, determina-se a folga disponível considerando o instante em que a tarefa inicia execução e o próximo prazo (dentre todas tarefas ativas). Em um segundo momento o ccRM utiliza essa folga para verificar se é possível baixar a freqüência de operação de todas as tarefas ativas. O algoritmo 3.3 apresenta o pseudo-código do ccRM.

Algoritmo 3.3 - *Cycle Conserving* RM.

assume-se que f_j é a freqüência definida na fase estática

liberacao_tarefa(T_i) :
    $ciclos\_restantesT_i \leftarrow C_i$
    $s_m \leftarrow maximo\_ciclos\_ate\_proximo\_prazo$
    $s_j \leftarrow s_m * f_i/f_m$
    aloca_ciclos(s_j)
    seleciona_frequencia()

aloca_ciclos(k) :
Para i ← 1 até n **Faça**
    Se $ciclos\_restantesT_i < k$ **Então**
        $d_i \leftarrow ciclos\_restantesT_i$
        $k \leftarrow k - ciclos\_restantesT_i$
    Senão
        $d_i \leftarrow k$
        $k \leftarrow 0$
    Fim Se
Fim Para

seleciona_frequencia():
    $s_m \leftarrow maximo\_ciclos\_ate\_proximo\_prazo$
    utilize a menor freqüência $f_i \in \{f_1, ..., fm | f_1 < ... < f_m\}$
    tal que $(d1 + ... + dm)/sm \leq f_i/fm$

durante execucao_tarefa(T_i) :
    diminua $ciclos\_restantesT_i$
    diminua d_i

termino_execucao(T_i) :
    $ciclos\_restantesT_i \leftarrow 0$
    $d_i \leftarrow 0$
    $s_m = maximo\_ciclos\_ate\_proximo\_prazo$

Uma modificação aqui proposta é a utilização do mesmo mecanismo de determinação de folgas do ccRM, porém com a heurística gulosa na distribuição. Então, nessa modificação, ao invés de distribuir a folga entre as tarefas ativas ela é toda alocada à tarefa atual. No algoritmo 3.4 descreve-se o pseudo-código do ccRMG. Ao comparar esse algoritmo com o anterior observam-se simplificações, principalmente nos procedimentos de alocação de folga e determinação da nova freqüência. No capítulo 4 é apresentado o impacto dessas modificações.

Algoritmo 3.4 - *Cycle Conserving* RM *Greedy* - ccRMG.

assume-se que f_j é a freqüência definida na fase estática

apos liberacao_tarefa(T_i) **:**
    $ciclos\_restantesT_i \leftarrow C_i$
    $s_m \leftarrow maximo\_ciclos\_ate\_proximo\_prazo$
    $s_j \leftarrow s_m * f_i/f_m$
    aloca_ciclos(s_j)
    seleciona_frequencia()

aloca_ciclos(k) **:**
    $tarefa\_atual \leftarrow k$
seleciona_frequencia():
    $s_m \leftarrow maximo\_ciclos\_ate\_proximo\_prazo$
    utiliza menor freqüência $f_i \in \{f_1, ..., fm | f_1 < ... < f_m\}$
    tal que $ciclos\_restantesT_i \; tarefa\_atual \leq f_i/fm$

durante execucao_tarefa(T_i) **:**
    diminua $ciclos\_restantesT_i$
    diminua d_i

apos termino_execucao$T_i)$ **:**
    $ciclos\_restantesT_i \leftarrow 0$
    $d_i \leftarrow 0$
    $s_m = maximo\_ciclos\_ate\_proximo\_prazo$

Durante sua execução, para qualquer um dos métodos dinâmicos, uma tarefa pode ser preemptada. Antes disso, ao início de sua execução, um determinado ponto de operação do processador já havia sido determinado. Durante a pre-empção novas folgas podem surgir e serão, automaticamente, incorporadas à folga dessa tarefa. Assim, quando ela retorna a executar, um novo ponto de operação será calculado, permitindo, eventualmente, uma economia de energia adicional.

A tabela 3.3 apresenta um resumo do funcionamento dos métodos aqui descritos. Nessa tabela tem-se: (i) Método utilizado; (ii) se há ou não etapa estática; (iii) se há ou não etapa dinâmica; (iv) qual a política de distribuição de folga; e, (v) em quantos níveis de voltagem a tarefa pode executar.

Nos métodos da tabela 3.3 e da figura 3.6 tem-se:

i) *RM*: *Rate Monotonic* puro sem utilização de regulagem dinâmica de voltagem;
ii) *Estático*: utiliza somente a fase estática;
iii) *GGT 1N*: *Greedy Gain Time* de um nível;
iv) *GGT 2N*: *Greedy Gain Time* de dois níveis;
v) *ccRM*: *Cycle Conserving Rate Monotonic*;
vi) *ccRMG*: *Cycle Conserving Rate Monotonic Greedy*;
vii) *Aprox 1N*: Aproximado de um nível;
viii) *Aprox 2N*: Aproximado de dois níveis;
ix) *Ótimo*: Ótimo aqui considerado.

Tabela 3.3: Resumo de funcionamento dos métodos.

Método	Etapa Estática	Etapa Dinâmica	Distr. de Folga	Níveis de Volt.
RM	Não	Não	Não	1
Estático	Sim	Não	Não	1
GGT 1N	Sim	Sim	Gulosa	1
GGT 2N	Sim	Sim	Gulosa	2
ccRM	Sim	Sim	ccRM	1
ccRMG	Sim	Sim	Gulosa	1
Aprox 1N	Sim	Sim	Gulosa	1
Aprox 2N	Sim	Sim	Gulosa	2
Ótimo	-	-	Ótimo	1

3.3 Exemplo

Para efeito de ilustração é apresentada uma aplicação dos métodos. O conjunto de tarefas e os tempos de execução de duas ativações consecutivas das tarefas está descrito na tabela 3.4. O modelo de processador utilizado é o teórico indicado na tabela 3.5 e é assumido que o consumo de energia no estado ocioso é zero.

Considerando os tempo de ativação da tabela 3.4, a figura 3.6 apresenta a grade de execução, dos diferentes métodos, para duas ativações consecutivas de cada tarefa. Admitindo a situação

Tabela 3.4: Conjunto de tarefas para o exemplo do métodos.

Tarefa	Período	WCET	Ativação 1	Ativação 2
1	8	3	1	1
2	10	3	2	1
3	14	1	1	1

Tabela 3.5: Modelo de processador para o exemplo dos métodos.

Nível	Voltagem	Freqüência
1	1,00	1,00
2	0,80	0,75
3	0,60	0,50

de instante crítico, a figura 3.6(a) apresenta a escala gerada pelo algoritmo RM assumindo-se o WCET para todas as tarefas. Nota-se que, sem utilização de RDV, as tarefas executam à máxima freqüência e ocupam 87,5% do tempo considerado. Nesse exemplo, considera-se essa execução como sendo 100% do gasto de energia. Para os demais métodos adota-se a variação do tempo de execução das tarefas como indicado na tabela 3.4, colunas Ativação 1 e Ativação 2. Então, conforme indicado na figura 3.6(b), existirá 56,25% de tempo ocioso que poderá ser utilizado através de RDV. Nas figuras 3.6(c) e 3.6(d) estão apresentadas as escalas geradas pelo método de execução GGT 1N e GGT 2N. Nesses casos, respectivamente, houve redução para cerca de 43,75% e 31,25% do tempo ocioso do processador, e o consumo de energia foi de 44,85% e 42,49% do máximo. O método de execução de dois níveis utiliza melhor a folga disponível pois: (i) pode ser que a tarefa, por começar executando no menor dos dois níveis, não precise executar no nível maior e, (ii) a política utilizada na execução em dois níveis tenta utilizar todo o *gain time* disponível. A figura 3.6(e) apresenta a escala gerada pelo algoritmo ccRM e a figura 3.6(f) apresenta a escala da modificação, aqui proposta, desse algoritmo. Nesses casos, respectivamente, houve redução para cerca de 37,5% e 35,43% do tempo ocioso do processador e o consumo de energia foi de 42,85% e de 41,79%. É interessante notar que a utilização da heurística gulosa se mostra promissora pois, quando o sistema não possui muita folga disponível (e.g., no instante inicial) ela consegue diminuir a tensão por considerar somente uma tarefa. As figuras 3.6(g) e 3.6(h) apresentam as escalas geradas pelos métodos Aproximado de um nível e de dois níveis. Nesses casos, respectivamente, houve redução do tempo ocioso para cerca de 25% e 14,56% e o consumo de energia foi de 38,85% e 36,49%. A redução do tempo ocioso nesse método se explica pela método Aproximado ser mais eficiente em determinar a quantidade máxima de folga (folga mais *gain time*) que pode ser roubada para a tarefa atual.

Na tabela 3.6 são apresentados as folgas disponíveis em cada ativação de cada tarefa. Nessa tabela, T_iA_j indica a ativação *j* da tarefa *i.* As folgas indicadas nessa tabela são as folgas disponíveis imediatamente antes da execução de cada ativação de cada tarefa e sua utilização está exemplificada na figura 3.6. Nessa tabela, para os métodos *Greed Gain Time* de um nível e de dois níveis (GGT 1N e GGT 2N), os valores indicam a quantidade de *gain time* disponível para a tarefa atual. Já para os métodos *Cycle Conserving* e *Cycle Conserving Greedy* (ccRM e ccRMG) os valores indicam a quantidade de folga (folga mais *gain time*) disponível. Finalmente, para os métodos Aproximado de um nível e de dois níveis (Aprox 1N e Aprox 2N) são apresentados os contadores individuais de folga (folga mais *gain time*) e a folga disponível para a tarefa ativa, que é igual ao contador de menor valor em cada instante de tempo.

Analisando-se somente as duas primeiras ativações para cada tarefa, o exemplo demonstra que os métodos mostram-se promissores no que diz respeito ao aproveitamento dos tempos ociosos e à economia de energia. Comparando-se a figura 3.6(b) com a figura 3.6(h) observa-se uma diminuição de até 41,68% do tempo ocioso e uma redução de até 13,51% no consumo de energia. Apresenta-se, no próximo capítulo, as simulações realizadas que permitem uma melhor comparação dos métodos.

Tabela 3.6: Folgas para o conjunto de tarefas da Tabela 3.4.

Método	T_1A_1	T_2A_1	T_3A_1	T_1A_2	T_2A_2	T_3A_2
GGT 1N	0	2	2,33	0	1	0
GGT 2N	0	2	2	0	1	0
ccRM	1	3	3,33	1	1	1
ccRMG	1	2,67	3	1	1	1
Aprox 1N	-	-	-	-	-	-
Contador 1	5	11,66	9	5	11	9
Contador 2	**1**	**2,66**	7	**3**	**4**	7
Contador 3	1	2,66	**2**	4	5	**4**
Disponível	**1**	**2,66**	**2**	**3**	**4**	**4**
Aprox 2N	-	-	-	-	-	-
Contador 1	5	11	7,33	5	11	9
Contador 2	**1**	**2**	5,33	**3**	**4**	7
Contador 3	1	2	**1,33**	4	5	**4**
Disponível	**1**	**2**	**1,33**	**3**	**4**	**4**

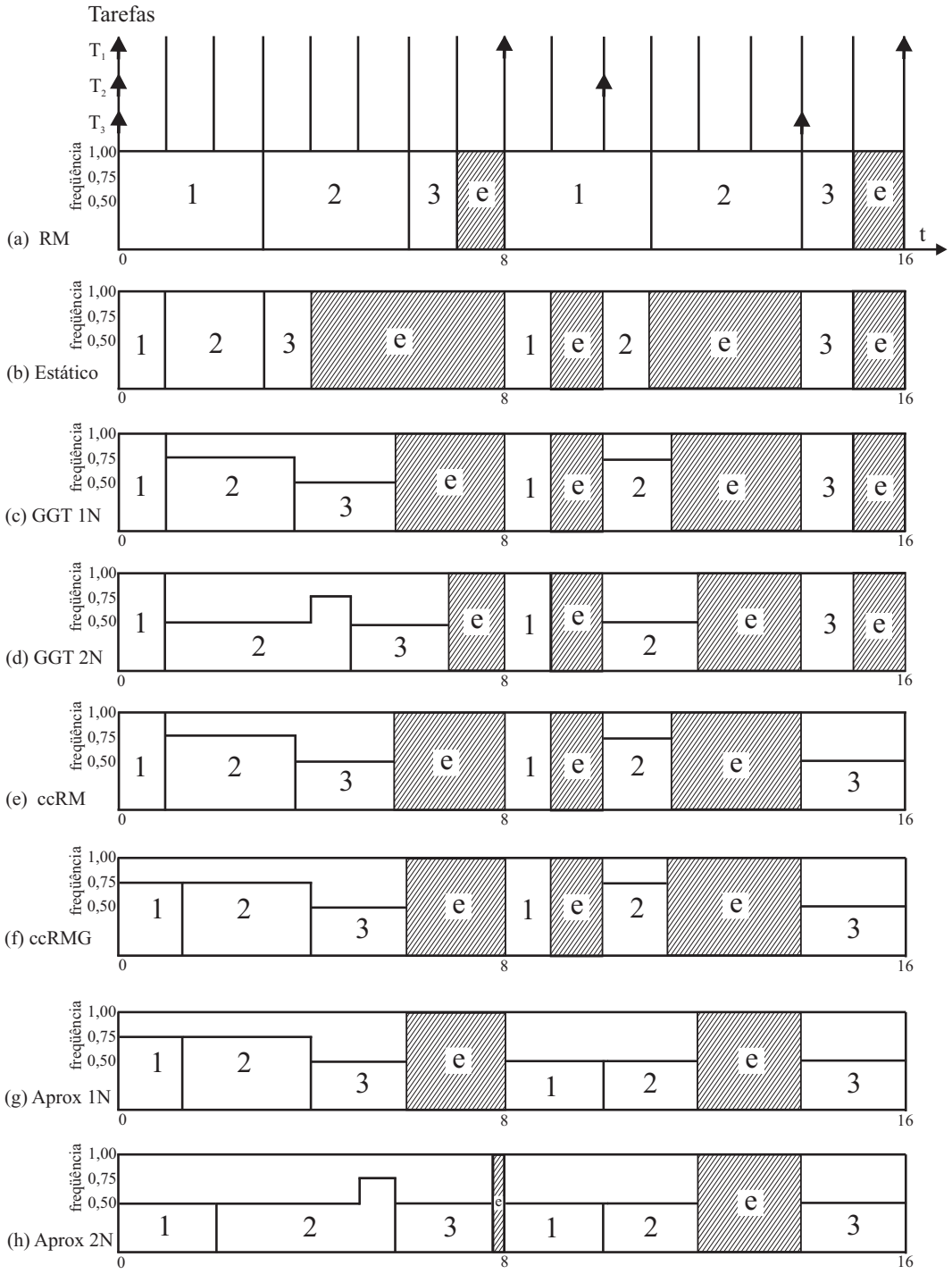


Figura 3.6: Escalonamento do conjunto de tarefas da Tabela 3.4.

3.4 Análise dos Métodos

Nesse capítulo discutiu-se os diferentes métodos para economia de energia no escalonamento de tarefas. Esses métodos diferem entre si ou pela determinação da folga ou pela distribuição/utilização da folga. Todos eles têm por objetivo aproveitar os tempos ociosos do processador pela aplicação do mecanismo de RDV para economia de energia.

Na distribuição e utilização da folga, os métodos aqui propostos ou utilizam 1 nível ou utilizam 2 níveis de voltagem/freqüência na execução de uma tarefa. Embora o método de 2 níveis exija mais cálculos, ambos possuem a mesma complexidade $O(n)$. Já para os métodos de determinação da folga tem-se diferentes complexidades: o Aproximado é $O(n^2)$ [Urriza e Orozco 2004]; o ccRM de [Pillai e Shin 2001] é $O(n^2)$ e o GGT é O(1).

Capítulo 4

Resultados

Nesse capítulo são apresentados e discutidos resultados obtidos através de simulação para permitir avaliar a eficiência dos métodos desenvolvidos nessa dissertação. As simulações foram realizadas considerando fatores como o número de tarefas, o fator de utilização de processador, o consumo do processador quando se encontra no estado ocioso, o número de níveis de voltagem e o intervalo entre eles. Todos os resultados foram comparados com o valor ótimo. O cálculo desse valor é feito admitindo-se que o processador possa assumir qualquer freqüência, e calcula-se então qual o valor de freqüência que leva a 100% de utilização do processador. Essa consideração é equivalente ao escalonamento com EDF utilizando um esquema contínuo de freqüências.

4.1 Diferentes Características em um STR

Nessa seção as simulações foram realizadas com conjuntos de 4, 8, 12 e 16 tarefas que estão definidos, respectivamente, nas tabelas 4.1 a 4.4. Nessas tabelas, FU indica o fator de utilização do processador. Esses conjuntos foram gerados de tal forma a não serem harmônicos e também a garantir que todas as tarefas possuam períodos distintos. Cada tarefa teve uma probabilidade uniforme discreta de possuir um período pequeno, médio ou grande. Nos experimentos realizados as faixas desses períodos são, respectivamente, $(2 - 10)$, $(11 - 30)$ e $(31 - 100)$, e cada período foi gerado com base em uma distribuição uniforme no intervalo escolhido. O WCET de uma tarefa foi gerado aleatoriamente, também segundo uma distribuição uniforme, entre $0,1$ e o período da tarefa. De forma a obter o fator de utilização desejado, os tempos de execução foram modificados por um fator de escala. Em todos os casos, os conjuntos de tarefas são escalonáveis segundo RM.

Tabela 4.1: Conjunto com 4 tarefas e 80% FU.

Tarefa	**1**	**2**	**3**	**4**
Período	2	9	10	29
WCET	0,74	1,48	1,11	4,45

Tabela 4.2: Conjunto com 8 tarefas e 80% FU.

Tarefa	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Período	7	14	16	21	24	36	42	45
WCET	0,98	0,24	2,20	0,73	0,73	0,24	8,33	10,53

Nas simulações, para efeito de avaliação da fase dinâmica, o tempo de execução das tarefas foi gerado aleatoriamente. Para escolha desse tempo foram utilizadas duas distribuições, uma uniforme e outra exponencial, e ainda um valor de c_i fixado como percentual do WCET. Nas figuras 4.1 a 4.18, a marca de 0,2 no eixo das abscissas indica que os tempos de execução das tarefas variam entre 20% e 100% do WCET, para o caso da distribuição uniforme. Para os gráficos com distribuição exponencial, a marca de 0,2 indica que o valor médio do tempo de execução das tarefas é 20% do WCET. Para os gráficos com fixação do c_i, a marca de 0,2 representa o tempo de execução das tarefas fixado em 20% do WCET. Nos gráficos, os pontos foram gerados executando-se o hiperperíodo 10 vezes, para o mesmo conjunto de tarefas, e são fornecidos com intervalo de confiança de 95%. No eixo das ordenadas, em todas as figuras referidas, é fornecida a energia consumida como um percentual do pior caso (ou seja, 100%, que corresponderia à utilização de voltagem/freqüência máximas em todo o hiperperíodo). Na legenda das figuras tem-se:

i) *RM*: *Rate Monotonic* puro sem utilização de regulagem dinâmica de voltagem;
ii) *Estático*: utiliza somente a fase estática;
iii) *GGT 1N*: *Greedy Gain Time* de um nível;
iv) *GGT 2N*: *Greedy Gain Time* de dois níveis;
v) *ccRM*: *Cycle Conserving Rate Monotonic*;

Tabela 4.3: Conjunto com 12 tarefas e 80% FU.

Tarefa	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Período	6	7	8	9	15	28	33	45
WCET	0,50	0,10	0,20	0,79	1,29	1,59	1,49	3,97
Tarefa	**9**	**10**	**11**	**12**	**-**	**-**	**-**	**-**
Período	56	60	63	88	-	-	-	-
WCET	5,55	5,95	3,27	5,65	-	-	-	-

Tabela 4.4: Conjunto com 16 tarefas e 80% FU.

Tarefa	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Período	8	9	12	14	18	20	28	30
WCET	0,93	0,47	0,70	0,12	1,40	1,40	1,74	2,21
Tarefa	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Período	32	40	48	50	56	60	63	64
WCET	0,35	1,05	4,07	0,93	5,35	0,47	1,74	0,70

vi) *ccRMG*: *Cycle Conserving Rate Monotonic Greedy*;
vii) *Aprox 1N*: Aproximado de um nível;
viii) *Aprox 2N*: Aproximado de dois níveis;
ix) *Ótimo*: Ótimo aqui considerado.

4.1.1 Número de Tarefas

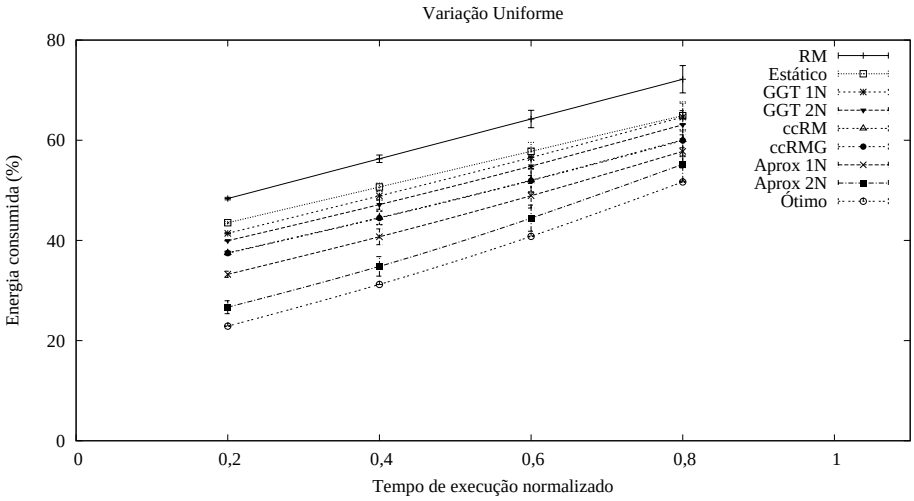
No primeiro conjunto de simulações é avaliado o efeito da variação do número de tarefas. Para esses resultados utilizou-se um processador teórico que está definido na tabela 4.5. Essa tabela é utilizada para calcular a quantidade de energia gasta pelos métodos. As figuras 4.1 a 4.4 apresentam o consumo de energia para os conjuntos definidos nas tabelas 4.1 a 4.4. Admitiu-se que o consumo de energia no estado ocioso (*idle*) é igual ao consumo no menor nível de voltagem/freqüência. A utilização desse processador teórico e o consumo de energia no estado ocioso serão justificados posteriormente.

Embora as figuras 4.1 a 4.4 diferenciem-se pelas quantidades de tarefas, as curvas mostradas nos gráficos são muito semelhantes. Em todos eles, o fator de utilização (igual a 80%) afeta substancialmente o consumo de energia, como esperado, mas o número de tarefas não tem quase nenhum impacto. Esse resultado é consistente com o obtido por [Pillai e Shin 2001]. Na tabela 4.6 apresenta-se um comparativo do gasto energético dos métodos para os quatros conjuntos de tarefas, segundo a distribuição uniforme dos tempos de execução e com valor mínimo de 0,4 do WCET para esses tempos. Observa-se que a diferença no consumo de energia entre os diferentes conjuntos de tarefas (8, 12 e 16 tarefas), para um mesmo método, não é muito significativa. Adicionalmente, deve ser notado que a distribuição do tempo de execução das tarefas pode ter influência nos resultados. Como dito anteriormente, no caso da distribuição exponencial, um número no eixo das abscissas indica o valor médio do tempo de execução, no caso da distribuição uniforme indica o valor mínimo do intervalo, e no c_i fixo informa o valor exato. Assim, uma carga com a primeira distribuição produzirá muito mais *gain time* e, conseqüentemente, maior economia de energia. Contudo, é interessante notar que isso é

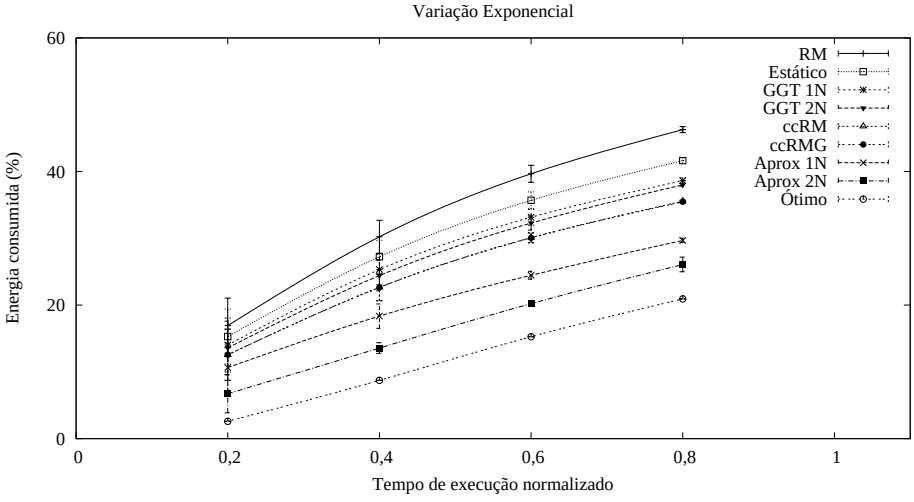
mantido mesmo quando compara-se um ponto com abscissa 0,6 no primeiro caso com um de abscissa 0,2 e distribuição uniforme. Essa observação é importante para a proposição de novas heurísticas para a distribuição de folga, talvez mais agressivas, quando se conhece a distribuição do tempo de execução. Pelos motivos acima expostos, na seqüência somente serão apresentados resultados para o conjunto de oito tarefas e com a distribuição uniforme (já que ela apresenta uma situação de pior caso).

Tabela 4.5: Processador teórico.

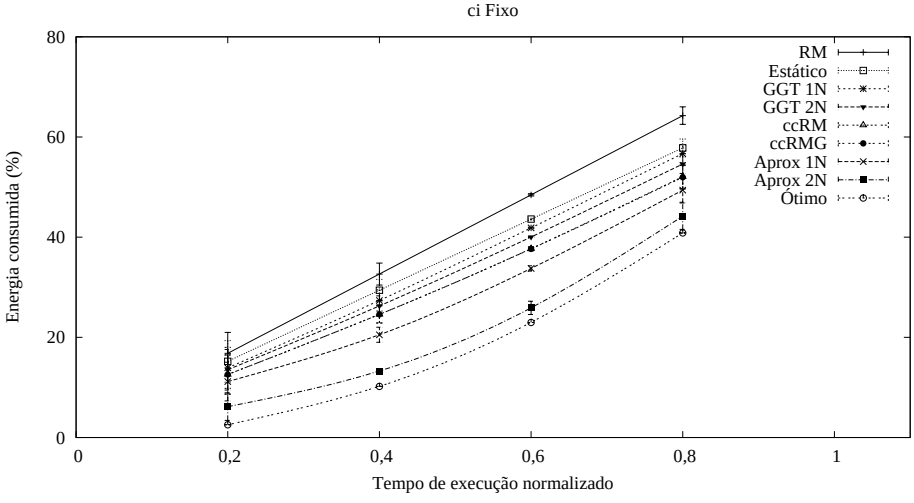
Nível	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Freqüência	100	200	300	400	500	600	700	800	900	1000
Voltagem	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1,0
Corrente	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1,0



(a)

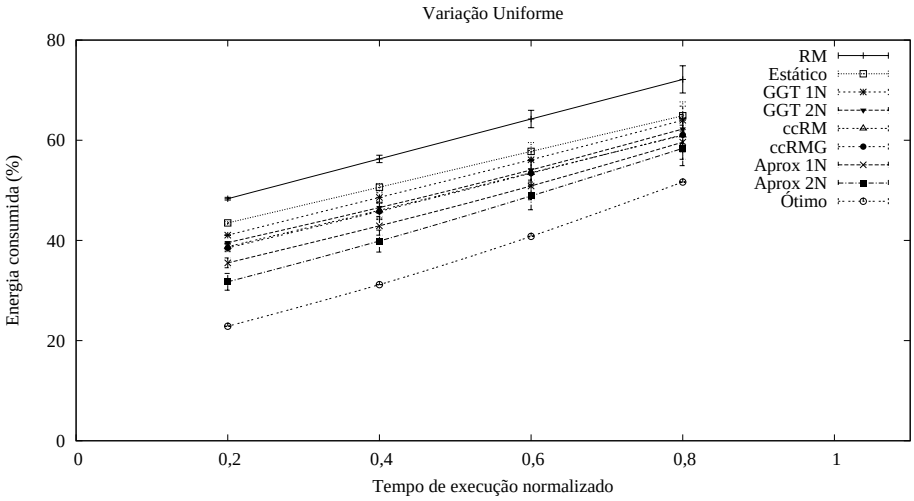


(b)

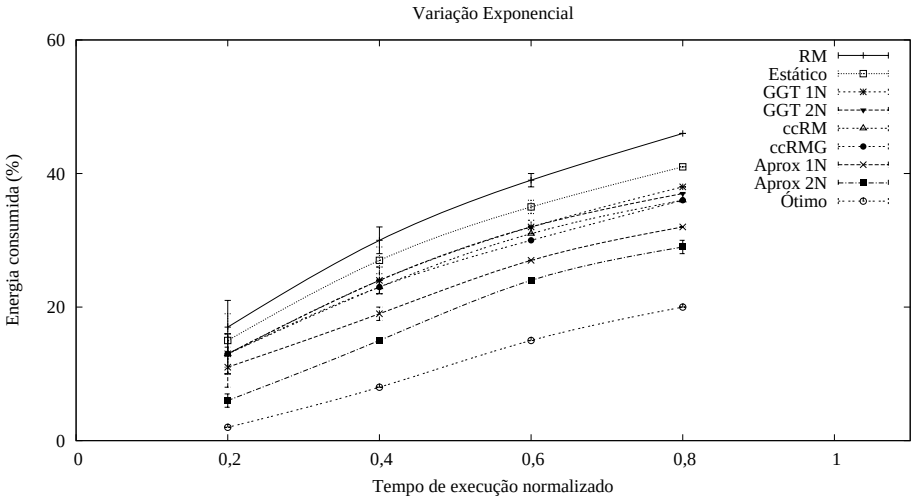


(c)

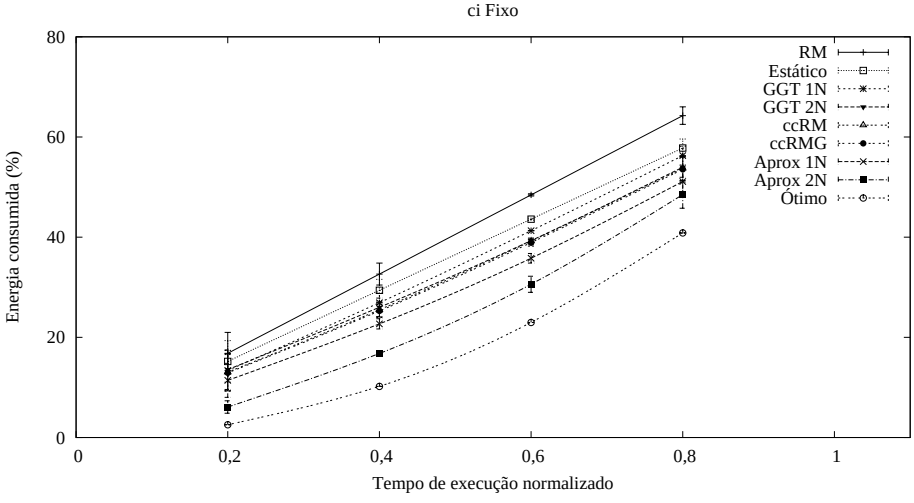
Figura 4.1: Conjunto de 4 tarefas.



(a)

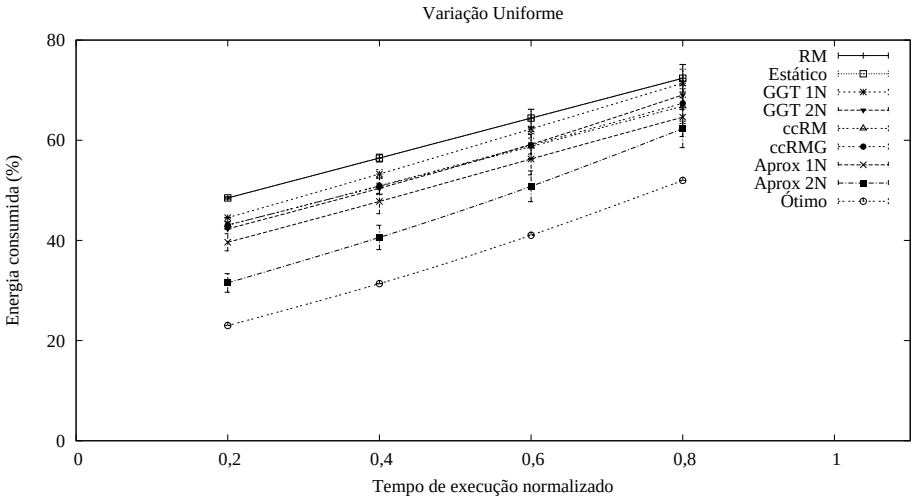


(b)

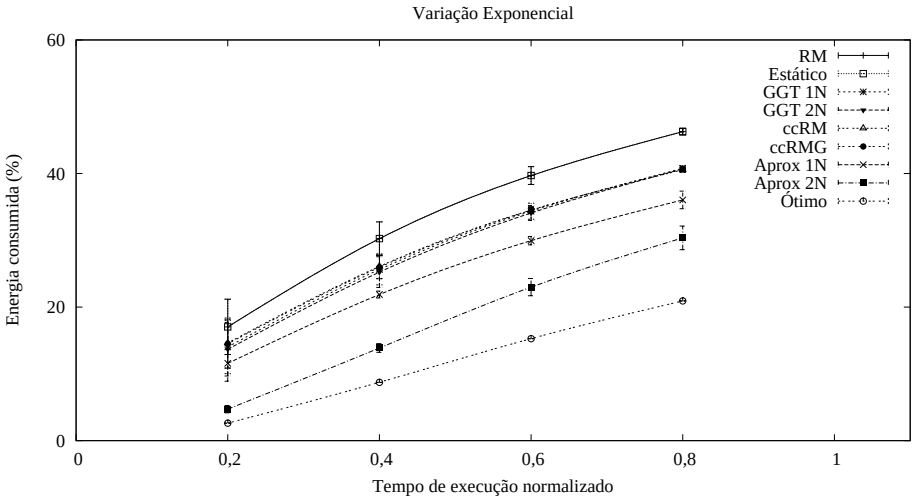


(c)

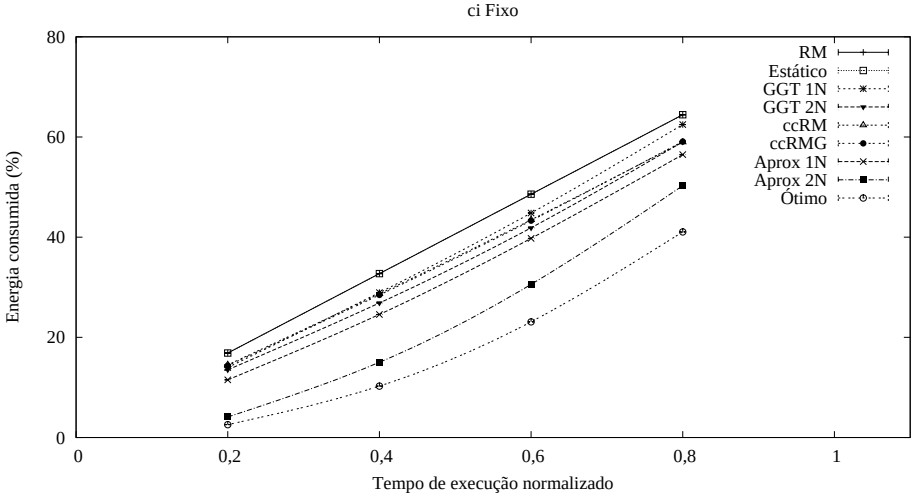
Figura 4.2: Conjunto de 8 tarefas.



(a)

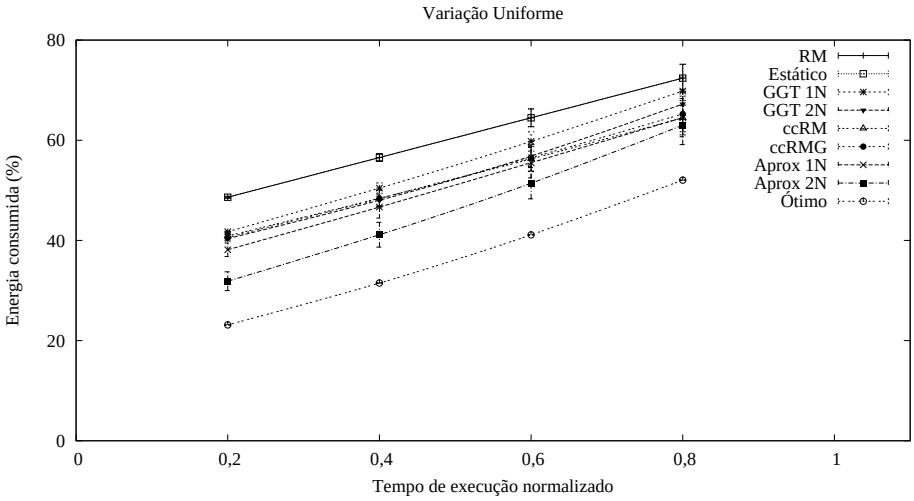


(b)

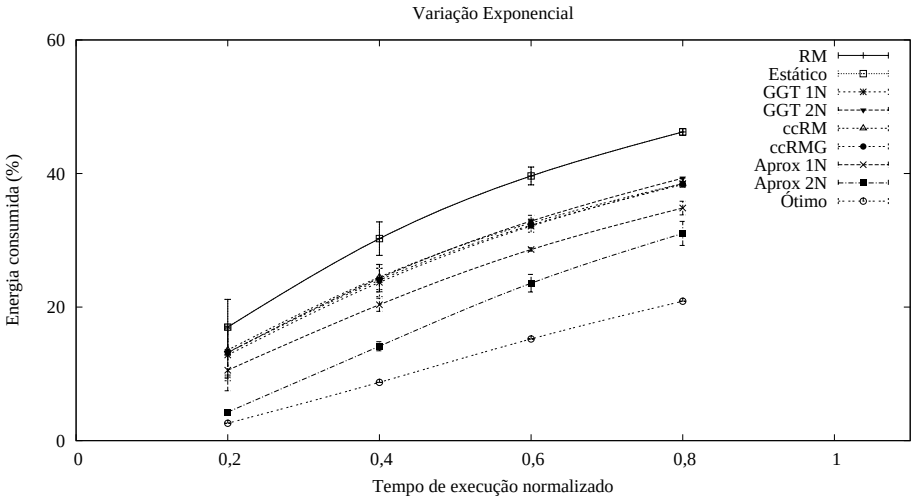


(c)

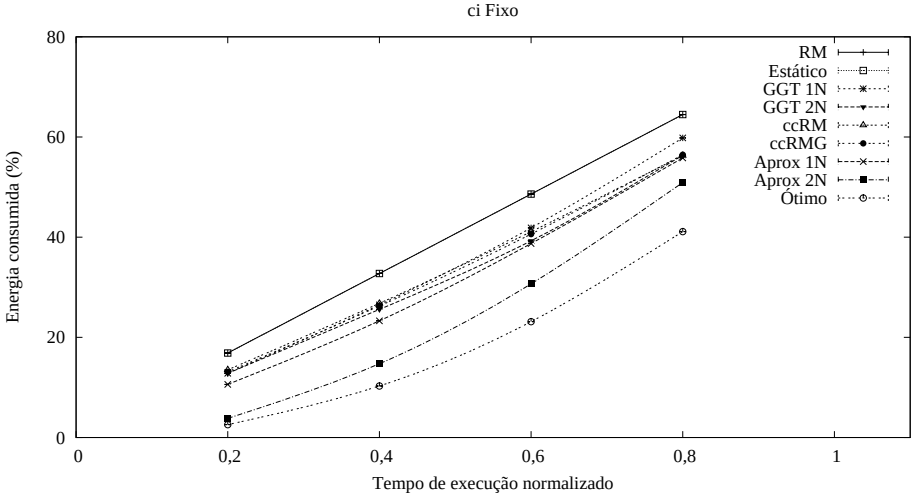
Figura 4.3: Conjunto de 12 tarefas.



(a)



(b)



(c)

Figura 4.4: Conjunto de 16 tarefas.

Tabela 4.6: Comparação do consumo de energia entre os conjuntos de tarefas.

Método	**Quantidade de tarefas**			
-	**4**	**8**	**12**	**16**
RM	56,3093	56,2801	56,4461	56,5512
Estático	50,6665	50,6205	56,4461	56,5512
GGT 1N	48,8836	48,5823	53,2561	50,4527
GGT 2N	47,1561	46,5643	50,4169	48,0239
ccRM	44,5442	46,0026	50,8159	48,4213
ccRMG	44,4621	45,8015	50,9077	48,3487
Aprox 1N	40,7334	42,9179	47,7889	46,6630
Aprox 2N	34,8146	39,8758	40,5975	41,1353
Ótimo	31,2140	31,1804	31,3669	31,4859

4.1.2 Fator de Utilização

Nessa seção utiliza-se, para um mesmo conjunto de tarefas (especificado na tabela 4.7), diferentes valores de fatores de utilização (20%, 40%, 60% e 80%). A tabela 4.8 indica o consumo de energia dos métodos para esses diferentes fatores de utilização e com tempo de execução (uniformemente distribuído) normalizado igual a 0,6. Considerando as figuras 4.5 a 4.8, nota-se que o esquema Aprox 2N, na pior situação, ficará a 9% (80% de FU) acima do ótimo e no melhor caso, 0,8% superior (20% de FU). Observa-se que os métodos afastam-se do ótimo à medida que se aumenta o fator de utilização, o que é natural, já que há diminuição da folga. Por exemplo, o consumo de energia do método Aprox 1N para 20% de FU é 0,9% pior que o ótimo e nos demais fatores de utilização têm-se 40% de FU – 3,2% ; 60% de FU – 6,4% e 80% de FU – 10%. Nota-se, para um mesmo fator de utilização, como esperado, que os resultados produzidos pelos métodos de 2 níveis convergem para aqueles dos métodos de 1 nível à medida que diminui a variabilidade do tempo de execução, já que há uma conseqüente diminuição das folgas. Da mesma forma, para qualquer método, a energia consumida aumenta quando a variabilidade do tempo de execução em relação ao WCET diminui.

Tabela 4.7: Conjunto com 8 tarefas - tempo de execução.

Tarefa	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Período	7	14	16	21	24	36	42	45
20% do FU	0,24	0,06	0,55	0,18	0,18	0,06	2,08	2,63
40% do FU	0,49	0,12	1,10	0,37	0,37	0,12	4,16	5,27
60% do FU	0,73	0,18	1,65	0,55	0,55	0,18	6,25	7,90
80%do FU	0,98	0,24	2,20	0,73	0,73	0,24	8,33	10,53

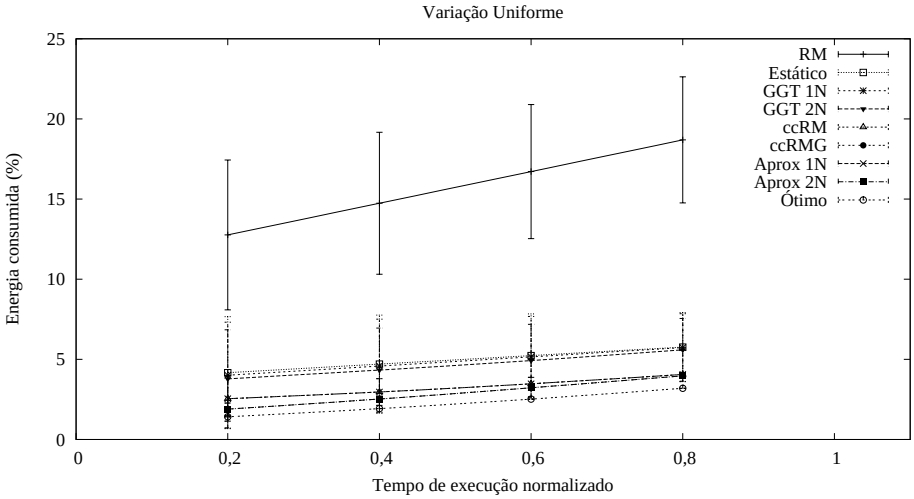


Figura 4.5: 20% de fator de utilização.

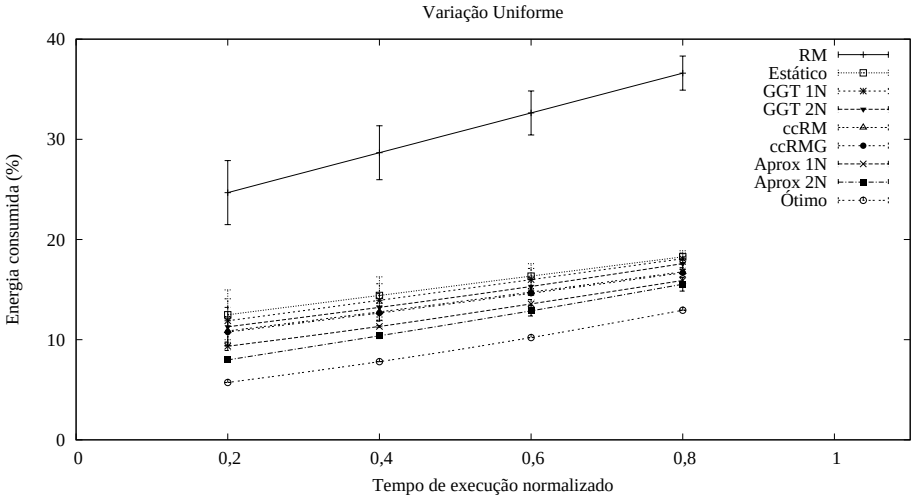


Figura 4.6: 40% de fator de utilização.

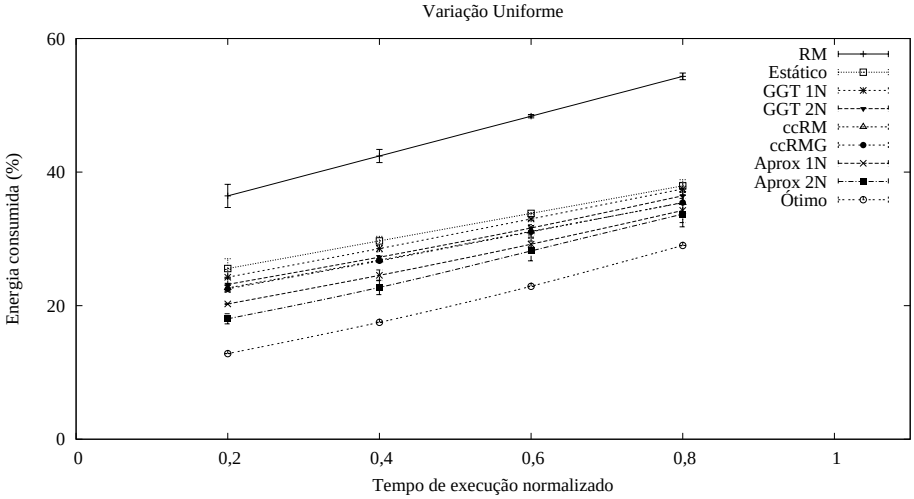


Figura 4.7: 60% de fator de utilização.

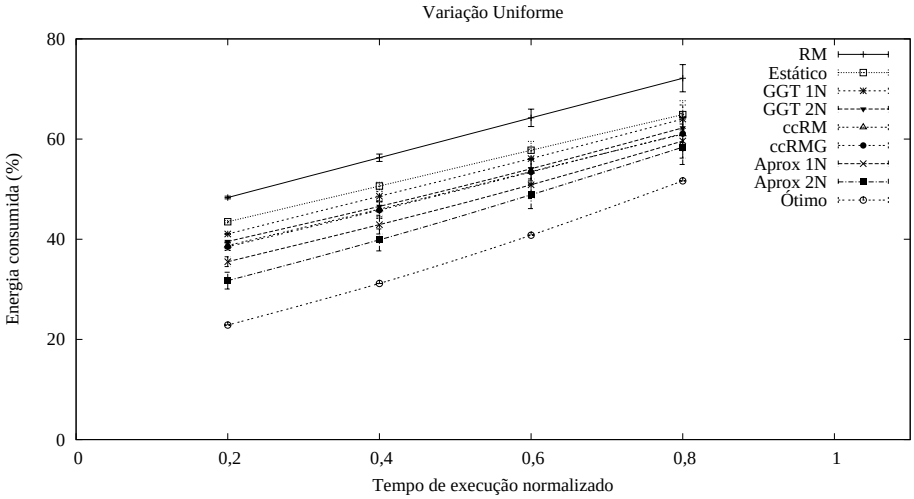


Figura 4.8: 80% de fator de utilização.

Tabela 4.8: Comparação do consumo de energia - diferentes FU.

Método	**Fator de utilização**			
-	**20**%	**40**%	**60**%	**80**%
RM	16,7148	32,6359	48,3726	64,2412
Estático	5,2334	16,3480	33,8250	57,7743
GGT 1N	5,1561	15,9955	32,9922	56,1110
GGT 2N	4,9247	15,3130	31,6391	54,0790
ccRM	4,5234	14,7523	31,1202	53,5098
ccRMG	4,4978	14,6337	31,0533	53,4116
Aprox 1N	3,4739	13,5714	29,2030	50,8780
Aprox 2N	3,2269	12,8790	28,1956	48,8847
Ótimo	2,5197	10,2118	22,8976	40,8073

4.1.3 Processador

A quantidade de níveis e a disposição (próximos ou afastados) desses em um processador influenciam diretamente nos resultados. Além disso, a energia consumida no estado ocioso também afeta os resultados e por isso ela será discutida a parte.

Geralmente, os processadores reais apresentam certas dificuldades em sua utilização, pois: (i) não oferecem variação contínua na voltagem; (ii) nos *datasheets* nem sempre estão disponíveis todas as informações necessárias; (iii) em geral oferecem poucos níveis de voltagem; e, (iv) em alguns casos apresentam níveis de energia ineficientes do ponto de vista energético. Em virtude desses fatores, simulações foram realizadas para analisar o impacto de se utilizar diferentes processadores.

Em todas as simulações realizadas para investigar os processadores considerou-se o conjunto de 8 tarefas definido na tabela 4.2. O consumo de energia do estado ocioso, para cada processador, é igual ao consumo do menor nível de voltagem em cada um deles.

Considere os modelos de processador definidos nas tabelas 4.9 e 4.10 que se caracterizam por possuir poucos níveis de voltagem (3 e 4, respectivamente) e pelo fato de o menor nível de voltagem estar a 50% ou mais do nível máximo (os níveis estão próximos uns dos outros). Nas figuras 4.9 e 4.10 observa-se que, embora o uso do processador da tabela 4.10 apresente melhores resultados que o processador de três níveis, em ambos os modelos de processador os distintos métodos não economizam tanta energia quando se compara com um modelo de processador que apresente mais níveis (como se verá com o uso do processador indicado na tabela 4.13).

A utilização de modelos de processadores com poucos níveis diminui a qualidade dos resultados, tanto na etapa estática, quanto na dinâmica. Por exemplo, na figura 4.9 nota-se que não há economia de energia na etapa estática, uma vez que o modelo de processador escolhido não possui um nível intermediário que possa ser utilizado e que ainda garanta o prazo de execução das tarefas. Outra conseqüência é uma redução da economia de energia na fase dinâmica, principalmente para os métodos GGT que, como não utilizam a folga remanescente, só aproveitam o *gain time*. Assim, os métodos ao utilizarem o RDV não conseguem aproveitar de maneira eficiente a folga disponível. Portanto, os processadores, já que não oferecem variação contínua de voltagem, devem possuir uma quantidade mínima de níveis para que a utilização do mecanismo de RDV não seja prejudicada. Em [Saewong e Rajkumar 2003] é sugerido que os processadores tenham um mínimo de 10 níveis de modo a manter o erro de quantização abaixo de 10%.

Tabela 4.9: Processador teórico de 3 níveis.

Nível	**1**	**2**	**3**
Freqüência	500	750	1000
Voltagem	0,6	0,8	1,0
Corrente	0,6	0,8	1,0

Tabela 4.10: Processador teórico de 4 níveis próximos.

Nível	**1**	**2**	**3**	**4**
Freqüência	500	700	900	1000
Voltagem	0,5	0,7	0,9	1,0
Corrente	0,5	0,7	0,9	1,0

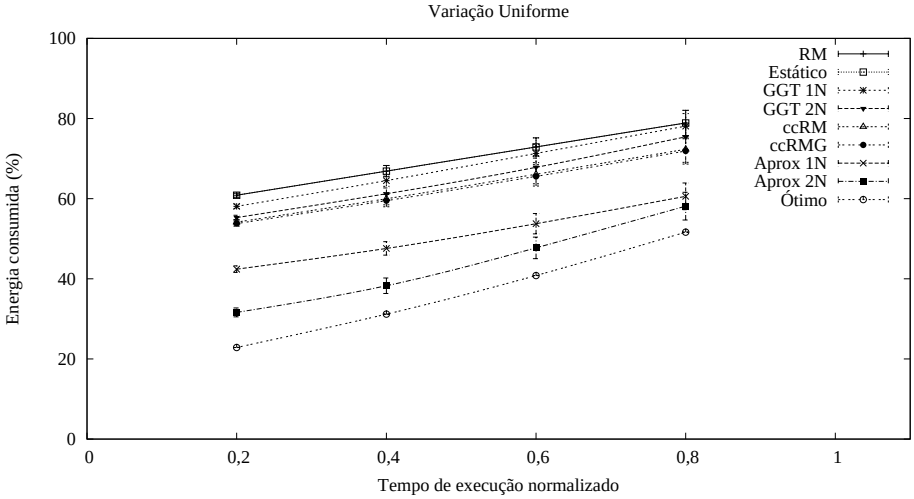


Figura 4.9: Processador teórico de 3 níveis.

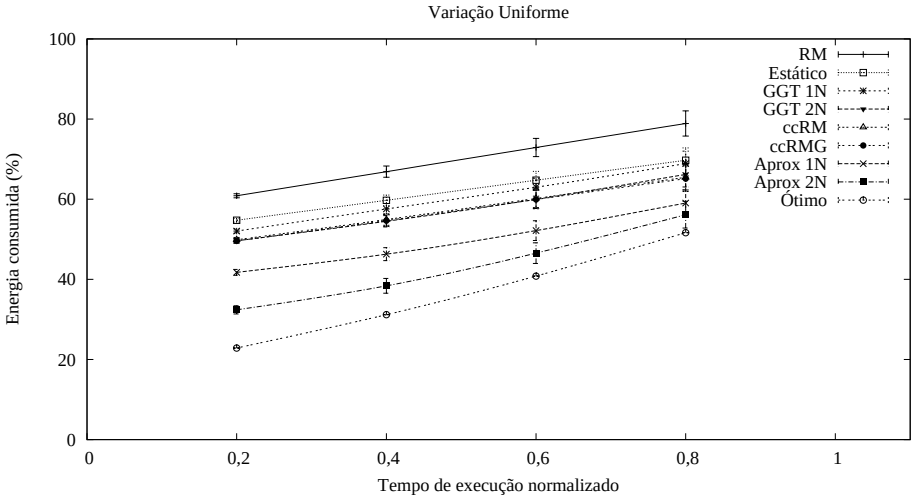


Figura 4.10: Processador teórico de 4 níveis próximos.

Sejam os modelos de processadores definidos nas tabelas 4.11 e 4.12, que se caracterizam por possuírem mais níveis e níveis mais espaçados que os anteriores (ver tabelas 4.9 e 4.10). As curvas dos gráficos 4.11 e 4.12 apresentam um menor consumo de energia ao se comparar com as figuras 4.9 e 4.10. Essa diminuição se explica pelo fato dos modelos de processadores possuirem mais níveis e, principalmente, por apresentarem níveis de voltagens mais baixos e que consomem pouca energia. Os níveis de baixo consumo são interessantes, sobretudo, quando há bastante *gain time* (eixo das abscissas em 0,2 e 0,4), pois permitem que os tempos ociosos sejam melhor aproveitados pelo mecanismo de RDV. Por exemplo, considerando o eixo das abscissas em 0,2 observa-se uma diferença significativa nas curvas dos diferentes métodos, principalmente no Estático, GGT 1N e GGT 2N, ao se comparar as figuras 4.9 e 4.12.

Tabela 4.11: Processador teórico de 4 níveis mais espaçados.

Nível	**1**	**2**	**3**	**4**
Freqüência	250	500	750	1000
Voltagem	0,25	0,50	0,75	1,00
Corrente	0,25	0,50	0,75	1,00

Tabela 4.12: Processador teórico de 5 Níveis.

Nível	**1**	**2**	**3**	**4**	**5**
Freqüência	200	400	600	800	1000
Voltagem	0,2	0,4	0,6	0,8	1,0
Corrente	0,2	0,4	0,6	0,8	1,0

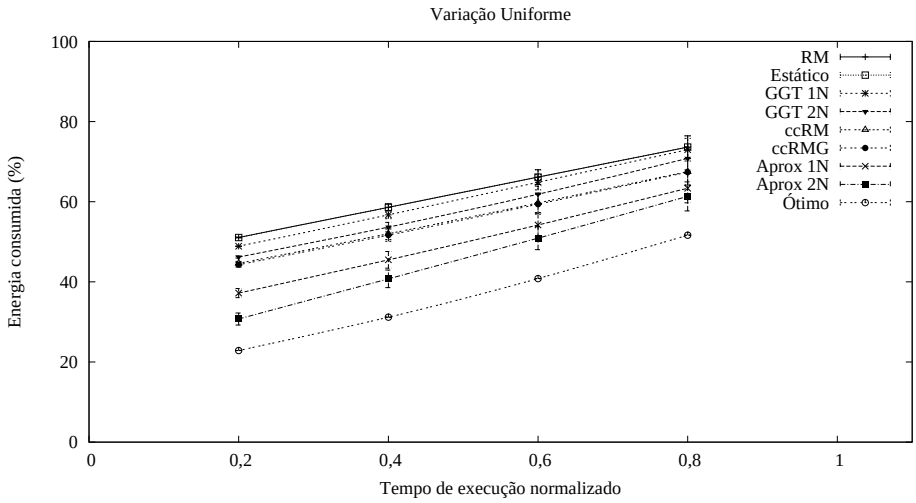


Figura 4.11: Processador teórico de 4 níveis mais espaçados.

As tabelas 4.13 e 4.14 apresentam modelos de processadores com dez e vinte níveis de voltagem/freqüência, que estão igualmente espaçados entre si. Analisando as figuras 4.13 e 4.14,

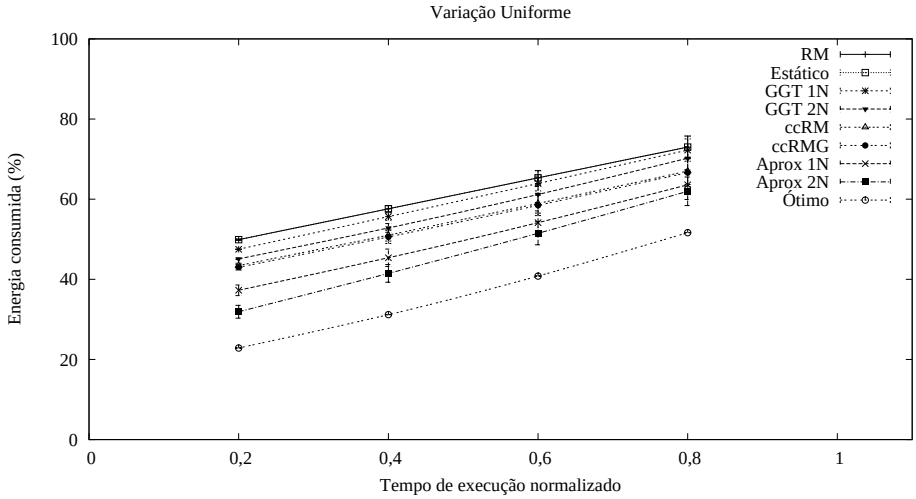


Figura 4.12: Processador teórico de 5 níveis.

que apresentam os resultados para os modelos de dez e vinte níveis, respectivamente, observa-se uma melhoria na economia de energia. De todos os modelos de processadores considerados nessa subseção, o gráfico 4.14 (vinte níveis de voltagem) é o que apresenta as curvas com os melhores resultados. Isso se justifica por apresentar mais níveis, o que garante um melhor aproveitamento dos tempos ociosos pelo mecanismo de RDV, tanto na etapa estática, quanto na dinâmica, além do nível mais baixo corresponder a uma situação de menor consumo que no caso de 10 níveis.

Contudo, a quantidade de níveis de voltagem afeta tanto a etapa estática quanto a dinâmica. Os métodos que não utilizam a folga remanescente são os mais prejudicados na fase estática. Isso ocorre quando há uma perda considerável por se utilizar o nível de voltagem/freqüência imediatamente acima ao que seria o ideal, uma vez que esses métodos não utilizam a folga remanescente. Já na fase dinâmica os métodos que empregam um único nível de voltagem é que são os mais prejudicados, já que utilizam o nível discreto acima ao que seria o ótimo.

Tabela 4.13: Processador teórico de 10 níveis.

Nível	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Freqüência	100	200	300	400	500	600	700	800	900	1000
Voltagem	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1,0
Corrente	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1,0

Tabela 4.14: Processador teórico de 20 níveis.

Nível	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Freqüência	50	100	150	200	250	300	350	400	450	500
Voltagem	0,05	0,10	0,15	0,20	0,25	0,30	0,35	0,40	0,45	0,50
Corrente	0,05	0,10	0,15	0,20	0,25	0,30	0,35	0,40	0,45	0,50
Nível	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Freqüência	550	600	650	700	750	800	850	900	950	1000
Voltagem	0,55	0,60	0,65	0,70	0,75	0,80	0,85	0,90	0,95	1,00
Corrente	0,55	0,60	0,65	0,70	0,75	0,80	0,85	0,90	0,95	1,00

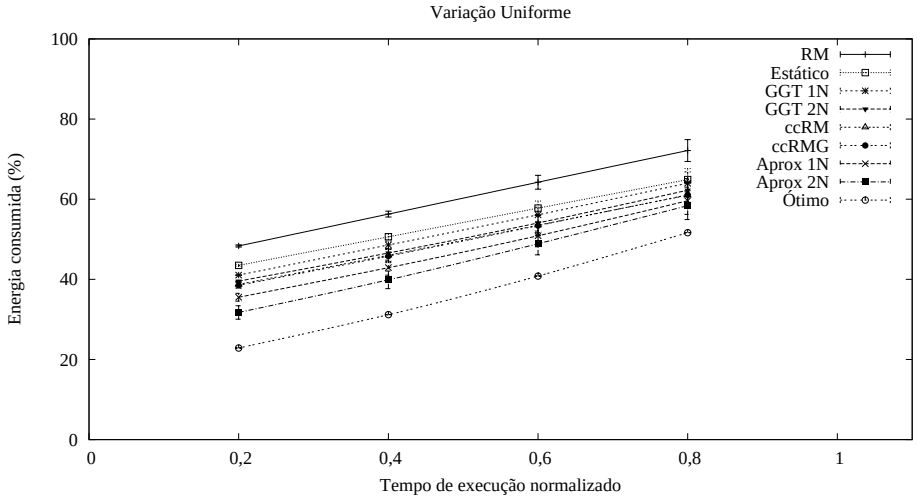


Figura 4.13: Processador teórico de 10 níveis.

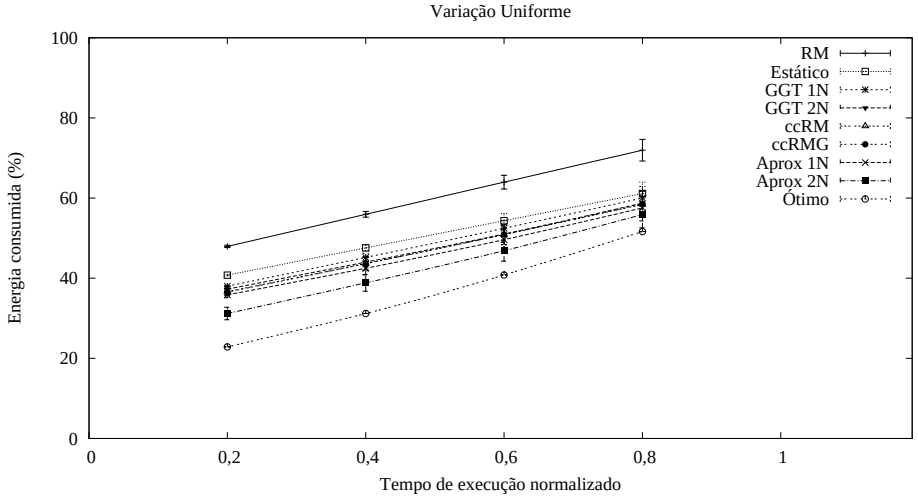


Figura 4.14: Processador teórico de 20 níveis.

4.1.4 Consumo no Estado Ocioso

Analisa-se nesse instante o impacto do consumo de energia quando o processador entra em estado ocioso. Em um hiperperíodo, o processador entrará em estado ocioso sempre que uma tarefa terminar de executar (utilizando o WCET ou não) antes de seu prazo e não houver tarefas aguardando para serem executadas. Obviamente, quando uma tarefa chegar a fila de prontos (quando uma tarefa estiver pronta para ser executada) o processador sairá do estado ocioso. Nos trabalhos que utilizam RDV (e.g., [Pillai e Shin 2001, Kim et al. 2002, Kim et al. 2003]) geralmente o consumo de energia no estado ocioso ou é zero ou se utiliza o consumo como sendo idêntico ao do menor nível de voltagem. Nessa dissertação, utilizou-se normalmente o consumo de energia no estado ocioso como sendo igual ao consumo do menor nível de voltagem do processador. As simulações indicam que é importante que esse gasto seja pequeno porque em sistemas de tempo real crítico sempre haverá momentos em que o processador se encontrará ocioso.

Para os testes dessa subseção utilizou-se o processador teórico de 10 níveis definido na tabela 4.13, assumindo diferentes gastos de energia no estado ocioso (0%, 10%, 50% e 100% do máximo). O conjunto de tarefas utilizados é o de oito tarefas e 80% de FU, definido na tabela 4.2. Os gráficos 4.15 e 4.16 indicam resultados considerando, respectivamente, o consumo de energia no estado ocioso igual a 0% e 10% do consumo máximo. Na tabela 4.15 apresenta-se uma comparação entre os diferentes valores do consumo de energia no estado ocioso para o eixo das abscissas igual a 0,4. Analisando-se as colunas de 0% e 10% dessa tabela, observa-se um aumento de consumo de 3% para o método GGT 1N. Já o método Aprox 2N apresenta um aumento de somente 0,4%. De uma maneira geral, observa-se um pequeno aumento em todos os métodos dinâmicos à medida que se aumenta o consumo de energia no estado ocioso. Isso ocorre porque quanto menos eficiente for o método na utilização do período ocioso maior será o consumo de energia.

Com o objetivo de avaliar a eficiência da utilização da folga, apresenta-se nas figuras 4.17 e 4.18 os resultados considerando que o gasto de energia no estado ocioso é alto, 50% e 100% do consumo máximo, respectivamente. Esses resultados indicam um aumento considerável no gasto de energia para todos os métodos (ao se comparar com 4.15 ou 4.16), com exceção do Aprox 2N que apresenta um pequeno consumo adicional de energia. Por exemplo, na tabela 4.15 colunas 50% e 100% indicam para o GGT 1N um aumento de 16% e 23% em relação a coluna de 0%. Comparando-se somente os métodos Aprox 1N e Aprox 2N, que possuem o mesmo mecanismo de determinação de folga, têm-se que o Aprox 1N apresenta uma menor

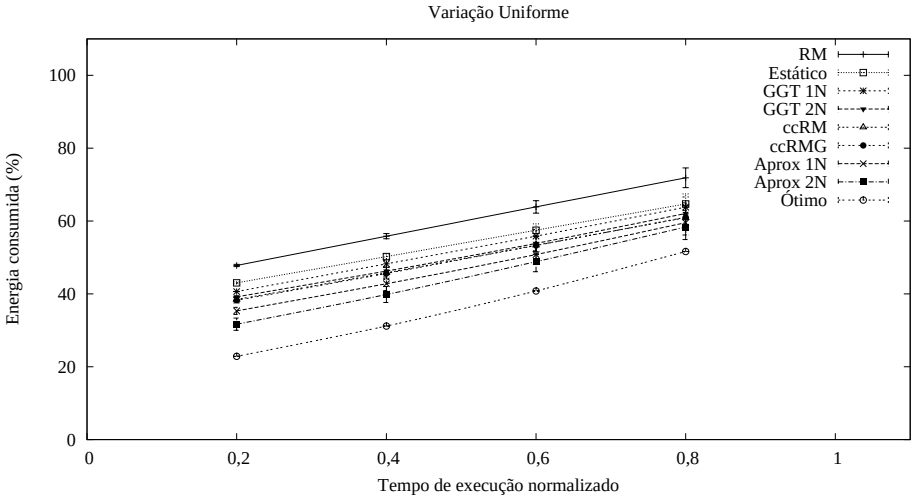


Figura 4.15: Consumo no estado ocioso igual a zero.

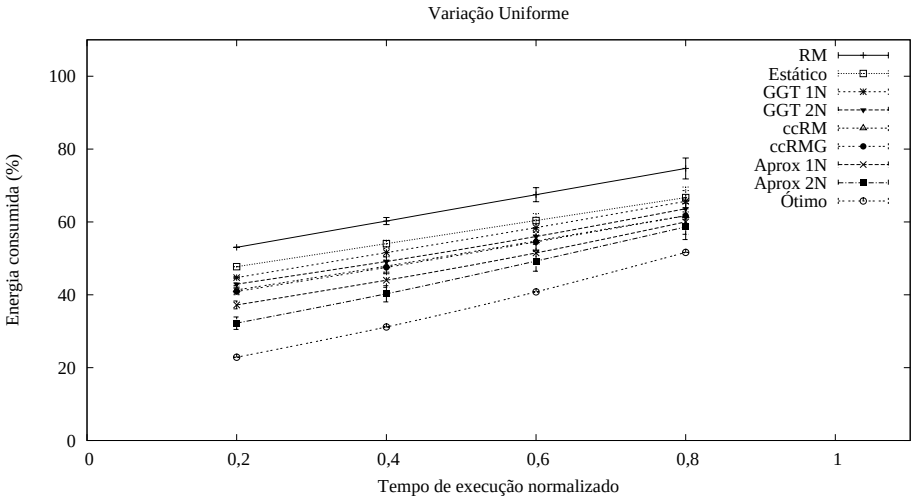


Figura 4.16: Consumo no estado ocioso igual 10% do máximo.

eficiência na economia de energia principalmente nos casos em que há um maior *gain time* no sistema (eixo das abscissas em 0,4 e principalmente em 0,2). Segundo a tabela 4.15, o Aprox 1N apresenta um consumo maior do que o Aprox 2N de 3% (coluna 0%) e de até 10% (coluna 100%). Isso ocorre porque o método de um nível geralmente utiliza o nível imediatamente acima ao que seria o ótimo, fazendo com que parte da folga não seja utilizada e, portanto, ocorre um maior consumo de energia.

Comparando-se as figuras 4.15 e 4.18 verifica-se o quanto os métodos conseguem utilizar da folga disponível no sistema e o quanto o sistema economiza por entrar em um estado de menor consumo. Analisando-se somente o RM a economia de energia pode ser de até 50% (eixo das abscissas igual a 0,2). É interessante notar que esse valor refere-se apenas ao consumo de energia no estado ocioso do processador, uma vez que o método RM não utiliza a RDV.

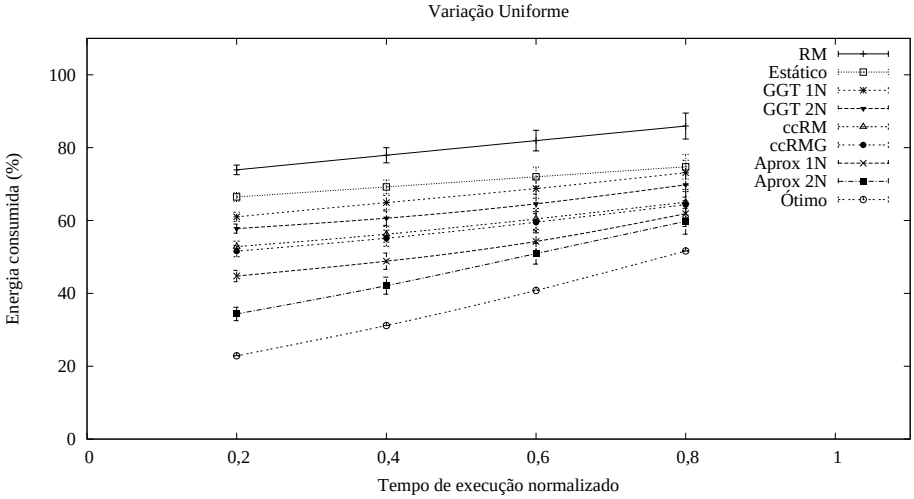


Figura 4.17: Consumo no estado ocioso igual 50% do máximo.

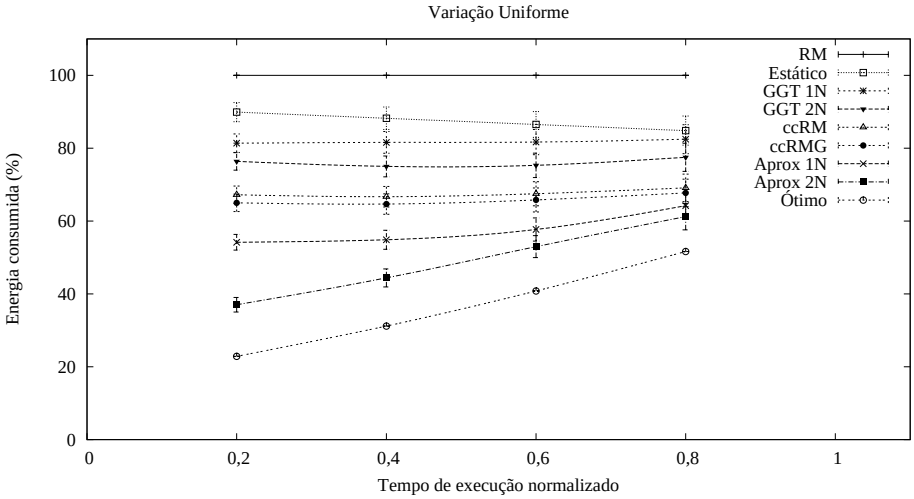


Figura 4.18: Consumo no estado ocioso igual a 100% do máximo.

Tabela 4.15: Comparação da influência do consumo de energia no estado ocioso.

Método	**Consumo no estado ocioso**			
-	**0%**	**10%**	**50%**	**100%**
RM	55,8384	60,2546	77,9192	100
Estático	50,2407	54,0382	69,2279	88,2151
GGT 1N	48,2489	51,5836	64,9227	81,5965
GGT 2N	46,2770	49,1502	60,6431	75,0092
ccRM	45,7937	47,8825	56,2376	66,6815
ccRMG	45,6109	47,5168	55,1404	64,6700
Aprox 1N	42,7972	44,0041	48,8317	54,8663
Aprox 2N	39,8301	40,2869	42,1142	44,3983
Ótimo	31,1804	31,1804	31,1804	31,1804

As colunas 0% e 100% da tabela 4.15 indicam um aumento substancial no consumo de energia para todos os métodos. O método Aprox 2N é uma exceção, pois o aumento foi menor que 5%. Isso ocorre porque o Aprox 2N é o método que utiliza o mecanismo mais elaborado para a determinação e distribuição de folga. Na utilização do RDV pelo Aprox 2N, na figura 4.15, observa-se uma economia suplementar de até 22% em relação ao RM, ficando a apenas 7% do consumo ótimo. Esse último valor demonstra a eficiência da utilização do mecanismo de RDV, uma vez que o consumo no estado ocioso não interfere nos resultados.

Na etapa dinâmica, o sistema sempre apresentará folga remanescente da etapa estática, pois utiliza-se modelos de processadores que apresentam níveis discretos de voltagem e, principalmente, pelo cálculo da voltagem/freqüência a ser utilizada ser realizado considerando-se o instante crítico. A única exceção ocorre quando os períodos das tarefas são harmônicos, o que não é o caso dos experimentos aqui realizados. Em todas as simulações discutidas, os métodos GGT (GGT 1N e GGT 2N), ccRM e ccRMG apresentam resultados semelhantes. Porém, os GGT, por não aproveitarem a folga remanescente do sistema, apresentam resultados ligeiramente piores. Isso ocorre porque sempre haverá folga remanescente da etapa estática e, principalmente, quando a variabilidade do tempo de execução diminui, já que no caso tem-se pouco *gain time*. Por outro lado, o mecanismo do GGT é mais simples do que o empregado no ccRM ou no ccRMG. O método Aproximado mostrou-se eficiente na utilização da folga remanescente mais o *gain time*. O Aprox 2N, que utiliza o mesmo mecanismo de determinação de folga do Aprox 1N e uma maneira mais refinada na distribuição (aqui denominado dois níveis), apresenta, como esperado, os melhores resultados em todos os gráficos. A seguir apresentam-se simulações com aplicações reais.

4.2 Simulações em Aplicações Reais

Nessa seção apresenta-se os resultados de simulações considerando modelos reais de aplicações. Diversos trabalhos (e.g., [Kim et al. 2002, Shin et al. 2000, Kim et al. 2003]) utilizam essas aplicações para validarem seus experimentos.

Nessas simulações, para efeito de avaliação da fase dinâmica, o tempo de execução das tarefas é distribuído uniformemente e é obtido através do cálculo de $(random[BCET, WCET] + WCET)/2$. Nas figuras 4.19 a 4.22, no eixo das abscissas a marca de 0,2 que BCET vale 20% do WCET. Em todos os gráficos, cada ponto foi gerado executando-se o hiperperíodo 10 vezes e os pontos são obtidos com intervalo de confiança de 95%. No eixo das ordenadas, em todas as figuras referidas, é fornecida a energia consumida como um percentual do pior caso (ou seja, 100%, que corresponderia à utilização de voltagem/freqüência máximas em todo o hiperperíodo).

Os experimentos foram realizadas considerando um processador real ARM-8[1] e suas características estão definidas na tabela 4.16. O consumo de energia no estado ocioso foi considerado zero. A utilização desse modelo de processador de 1997 se justifica por ter sido utilizado em outros trabalhos.

[1] As informações deste processador não foram encontradas no site da empresa, mas ele está descrito no trabalho [Im et al. 2004].

Tabela 4.16: Processador ARM 8.

Nível	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Freq.	5	6	7	8	9	10	11	12	13	14
Volt.	1,53	1,72	1,91	2,09	2,26	2,44	2,61	2,78	2,95	3,12
Corr.	1,53	1,72	1,91	2,09	2,26	2,44	2,61	2,78	2,95	3,12
Nível	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Freq.	15	16	17	18	19	20	21	22	23	24
Volt.	3,29	3,46	3,64	3,81	3,99	4,16	4,34	4,52	4,70	4,88
Corr.	3,29	3,46	3,64	3,81	3,99	4,16	4,34	4,52	4,70	4,88
Nível	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
Freq.	25	26	27	28	29	30	31	32	33	34
Volt.	5,06	5,25	5,44	5,63	5,82	6,01	6,21	6,41	6,61	6,81
Corr.	5,06	5,25	5,44	5,63	5,82	6,01	6,21	6,41	6,61	6,81
Nível	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**
Freq.	35	36	37	38	39	40	41	42	43	44
Volt.	7,02	7,23	7,44	7,65	7,87	8,09	8,31	8,53	8,76	8,99
Corr.	7,02	7,23	7,44	7,65	7,87	8,09	8,31	8,53	8,76	8,99
Nível	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**50**
Freq.	45	46	47	48	49	50	51	52	53	54
Volt.	9,22	9,46	9,70	9,94	10,19	10,44	10,69	10,94	11,20	11,46
Corr.	9,22	9,46	9,70	9,94	10,19	10,44	10,69	10,94	11,20	11,46
Nível	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**
Freq.	55	56	57	58	59	60	61	62	63	64
Volt.	11,73	11,99	12,27	12,54	12,82	13,10	13,39	13,68	13,97	14,27
Corr.	11,73	11,99	12,27	12,54	12,82	13,10	13,39	13,68	13,97	14,27
Nível	**61**	**62**	**63**	**64**	**65**	**66**	**67**	**68**	**69**	**70**
Freq.	65	66	67	68	69	70	71	72	73	74
Volt.	14,57	14,87	15,18	15,49	15,80	16,12	16,45	16,77	17,11	17,44
Corr.	14,57	14,87	15,18	15,49	15,80	16,12	16,45	16,77	17,11	17,44
Nível	**71**	**72**	**73**	**74**	**75**	**76**	**-**	**-**	**-**	**-**
Freq.	75	76	77	78	79	80	-	-	-	-
Volt.	17,78	18,12	18,47	18,82	19,18	19,54	-	-	-	-
Corr.	17,78	18,12	18,47	18,82	19,18	19,54	-	-	-	-

4.2.1 CNC

O primeiro caso real é uma máquina de controle numérico computadorizada (*computerized numerical control* - CNC) que está definida em [Kim et al. 1996] e suas características se encontram na tabela 4.17. É interessante notar que nesse conjunto algumas tarefas possuem o mesmo período. Para efeito de desempate pelo algoritmo RM adotou-se a ordem indicada na tabela 4.17 (números menores indicam prioridades mais altas).

O conjunto de tarefas CNC possui um fator de utilização de aproximadamente 48,9%. Assim, o sistema apresenta, inicialmente, uma grande quantidade de folga. A utilização do ARM 8, que apresenta 76 níveis de voltagem/freqüência, faz com que a fase estática apresente uma substancial economia de energia, como pode ser observado na curva do método Estático na figura 4.19. Para facilitar as comparações, a figura 4.20 apresenta os mesmos resultados da figura 4.19, porém sem a curva do RM.

Observa-se na figura 4.20 que, para valores do eixo das abcissas a partir de aproximadamente 0,7, pela primeira vez, o método ccRM apresenta resultados melhores que o Aprox 2N. Isso ocorre devido a dois fatores. Primeiro, devido ao modelo de processador utilizado que permite um escala estática muito boa, o que resulta em uma pequena folga remanescente para a fase dinâmica. Embora o método Aprox 2N seja mais eficiente que o ccRM na determinação da folga remanescente, no caso da faixa citada essa folga não é tão expressiva, o que diminui o ganho relativo desse método em relação ao ccRM. Segundo, quando há uma pequena variação do tempo de execução, o método ccRM apresenta uma política de distribuição de folga melhor que a heurística gulosa empregada no Aprox 2N. Nessa situação, há uma maior economia de energia quando se executa duas ou mais tarefas em um nível de voltagem intermediário do que quando executa-se uma tarefa em um nível baixo e as outras em um nível mais alto. Esse resultado, como será visto, também ocorreu para as outras aplicações reais.

Tabela 4.17: Conjunto de 8 tarefas - CNC.

Tarefa	**Tampl**	**Tcalc**	**Tdist**	**Tatta**	**Txref**	**Tyref**	**Txctrl**	**Tyctrl**
Período (μs)	2400	2400	2400	2400	4800	4800	7800	9600
WCET (μs)	35	40	165	165	180	720	570	570
Prioridade	1	2	3	4	5	6	7	8

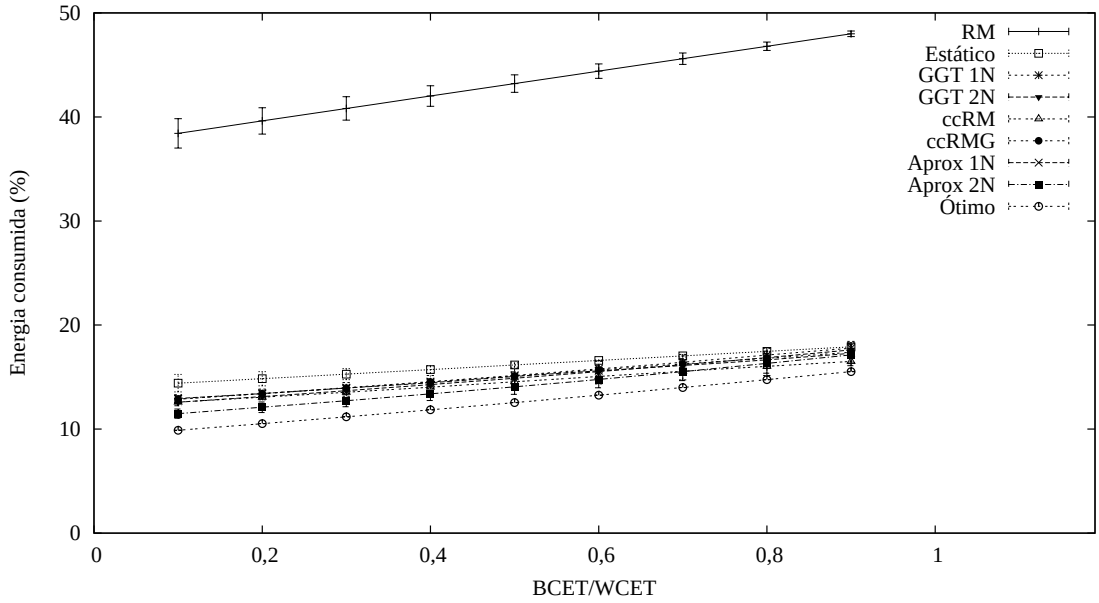


Figura 4.19: CNC.

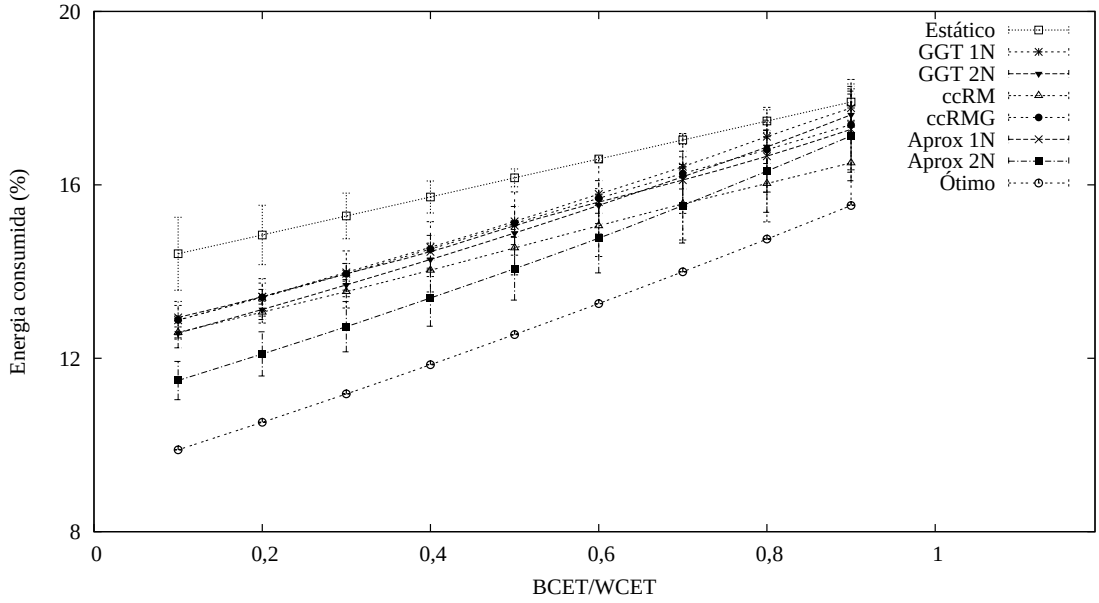


Figura 4.20: CNC sem RM.

4.2.2 Aviônica

O segundo experimento real é uma aplicação em aviônica (eletrônica aplicada à aviação, como por exemplo, o piloto automático). As informações podem ser encontradas em [Locke et al. 1991] e o conjunto de tarefas[2] está definido na tabela 4.18. Esse conjunto de tarefas apresenta um fator de utilização de 84,8%. Novamente, essa aplicação também possui tarefas com períodos iguais e as suas prioridades foram escolhidas segundo a ordem indicada na tabela 4.18. Na figura 4.21 apresenta-se os resultados. Observa-se que quanto maior a variabilidade do tempo de execução das tarefas, maior é a diferença entre os métodos Aprox 2N e os demais. Isso é explicado pela diferença da eficiência dos métodos na determinação/aproveitamento do *gain time*. Novamente, a partir do valor 0,85 no eixo das abcissas ocorre a inversão das curvas do ccRM e do Aprox 2N, pelos mesmos motivos explicados anteriormente.

Tabela 4.18: Conjunto de 17 tarefas - Aviônica.

Tarefa	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Período (ms)	25	25	40	50	50	59	80	80	100
WCET (ms)	2	5	1	3	5	8	2	9	5
Prioridade	1	2	3	4	5	6	7	8	9
Tarefa	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**-**
Período (ms)	200	200	200	200	200	200	1000	1000	-
WCET (ms)	1	1	1	3	3	3	1	1	-
Prioridade	10	11	12	13	14	15	16	17	-

4.2.3 Videofone

A última simulação é uma aplicação de videofone que está definida no trabalho [Shin et al. 2001] e cujas características[3] estão indicadas na tabela 4.19. É interessante observar que embora esse sistema apresente apenas quatro tarefas, o fator de utilização é alto, cerca de 98%, e ele é RM escalonável. O tempo médio de execução dessas tarefas, conforme indicado no artigo original, é de 25% do WCET [Shin et al. 2001]. Apresenta-se na tabela 4.20 o consumo de energia para valores de BCET iguais a 20% e 30% do WCET. O gráfico 4.22 apresenta os resultados desta simulação. Como o sistema possui um fator de utilização muito

[2]No conjunto de tarefas original existem três tarefas aperiódicas que, nesse exemplo, foram transformadas em tarefas periódicas.

[3]No conjunto de tarefas original os períodos de codificação e decodificação do vídeo são representados como números reais (ambos 66,667). Como no simulador desenvolvido utiliza-se números inteiros (o hiperperíodo é o mínimo múltiplo comum dos períodos das tarefas) os valores destes períodos foram arredondados para cima (67).

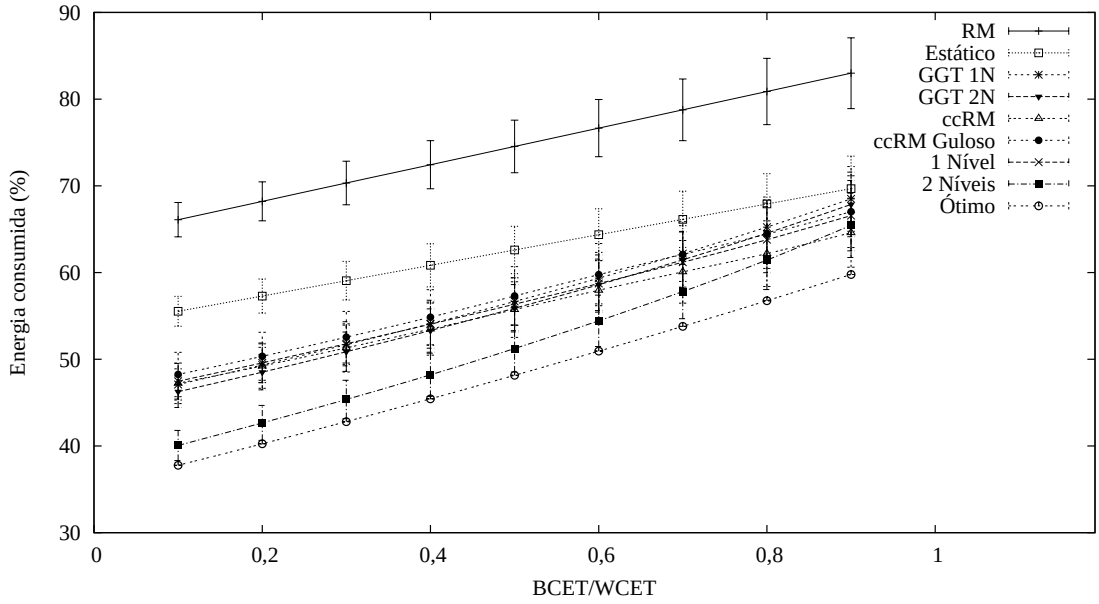


Figura 4.21: Aviônica.

alto a curva do método estático praticamente se sobrepõe a do escalonamento RM, indicando que não há ganho de energia na fase estática, como mostrado na tabela 4.20, que representa os pontos BCET/WCET iguais a 0,2 e 0,3 na figura 4.22. Nessa tabela, os valores indicam que todos os métodos encontram-se mais afastados do ótimo quando comparados aos valores obtidos nas duas aplicações reais descritas anteriormente. Isso ocorre porque nessa aplicação há uma maior diferença entre a carga inicial do sistema e a variação do tempo médio de execução das tarefas (ou seja, aqui a utilização assumindo-se WCET é muito alta). Porém, como a aplicação é aqui considerada como um STR crítico, não há como ser agressivo o suficiente para se aproveitar essa variabilidade. Nesse caso, isso justifica o porquê do método Aprox 2N se encontrar tão afastado do ótimo, principalmente quando a variabilidade do tempo de execução é grande. Novamente, a partir do valor 0,75 no eixo das abcissas ocorre a inversão das curvas do ccRM e do Aprox 2N, pelos mesmos motivos explicados previamente.

Tabela 4.19: Conjunto de 4 tarefas - Videofone.

Tarefa	**Decod. de som**	**Cod. de som**	**Decod. de vídeo**	**Cod. de vídeo**
Período (s)	40	40	67	67
WCET (s)	1,383	1,844	9,826	50,386

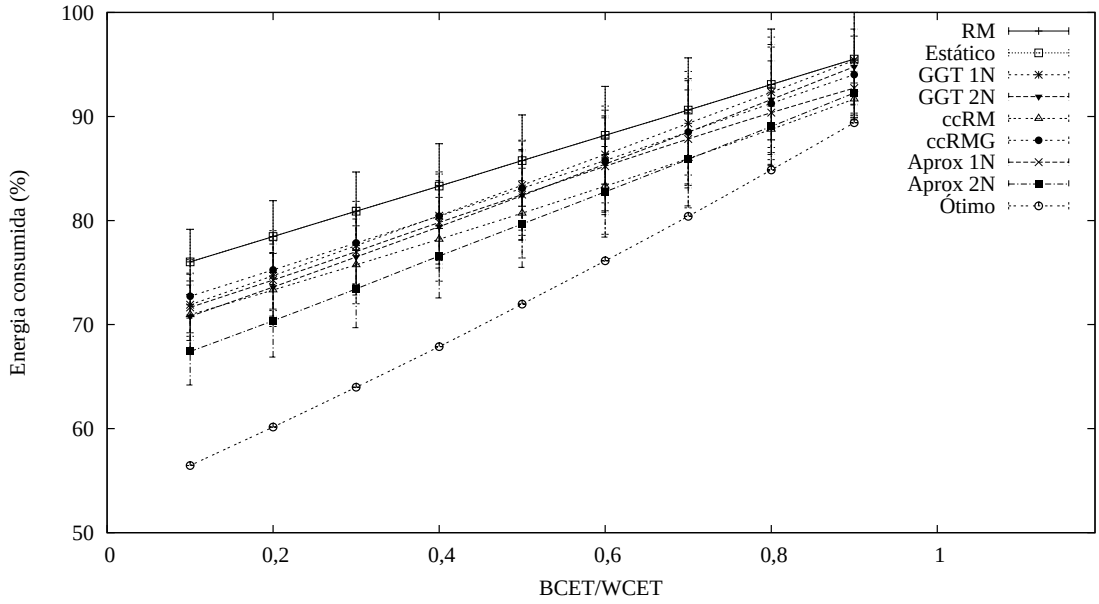


Figura 4.22: Videofone.

Tabela 4.20: Consumo de energia na aplicação de Videofone.

Método	**Fator de utilização**	
-	20%	30% %
RM	78,4616	80,8957
Estático	78,4616	80,8957
GGT 1N	74,6969	77,5961
GGT 2N	73,6005	76,495
ccRM	73,3258	75,7477
ccRMG	75,2534	77,8449
Aprox 1N	74,2788	77,0123
Aprox 2N	70,3684	73,4324
Ótimo	60,1588	63,9768

4.3 Análise dos Resultados

Nesse capítulo foram apresentados alguns experimentos realizados para analisar o desempenho dos métodos sob as diferentes características que um sistema de tempo real pode apresentar. A análise do número de tarefas com conjuntos de 4 até 16 tarefas mostrou não exercer influência nos resultados, sendo mais importante o fator de utilização. Já o número de níveis do processador afeta diretamente os resultados. Uma vez que os processadores reais possuem poucos níveis discretos de voltagem, utiliza-se para os diversos métodos de um nível aquele imediatamente acima ao que seria o ótimo. Quanto mais níveis o processador possuir, menor será a perda por utilizar o nível acima. Outro fator importante é o consumo de energia no estado ocioso do pro-

cessador, pois alguns processadores reais de hoje em dia apresentam o consumo ocioso como sendo igual ao consumo no menor nível de voltagem. Como no hiperperíodo sempre haverá folga é importante que o processador tenha um baixo consumo de energia quando no estado ocioso.

Todos os métodos utilizados apresentam resultados interessantes uma vez que são mais econômicos ao serem comparados com o RM ou o Estático. Os métodos GGT 1N e GGT 2N mostraram bons resultados, principalmente quando o sistema não apresenta, em sua fase dinâmica, muita folga remanescente. Esses métodos embora tenham apresentado, quase sempre, resultados piores que os demais métodos dinâmicos, a sua utilização é interessante devido a sua baixa complexidade. O GGT praticamente não causa um *overhead* na determinação da folga disponível. A utilização de dois níveis mostrou-se, como esperado, ser mais eficiente que o de um nível. O método ccRM mostrou-se ligeiramente melhor que o ccRMG, principalmente em situações com pequena variabilidade do tempo de execução como, por exemplo, os gráficos apresentados para as aplicações reais. Ambos os métodos utilizam mecanismo idêntico na determinação de folga, porém na distribuição de folga o ccRMG é menos complexo. O método Aproximado mostrou-se eficiente na determinação da folga remanescente do sistema mais o *gain time*. O Aprox 2N, geralmente, é o que apresenta melhores resultados. Isso era esperado, pois ele utiliza um mecanismo mais avançado na determinação de folga e também por utilizar o mecanismo de dois níveis. Tanto o método Aprox quanto o ccRM apresentam a mesma complexidade. A utilização da heurística gulosa, pelo método Aprox na distribuição da folga, só não apresenta resultados melhores que a heurística utilizada pelo ccRM quando há uma pequena variabilidade do tempo de execução de uma tarefa em relação ao seu WCET. Em todos os outros casos apresentados o método Aprox 2N mostrou-se mais eficiente na economia de energia do que os demais métodos. A figura 4.18 é a que melhor representa a eficiência da utilização deste método considerando o aproveitamento do tempo ocioso para a economia de energia.

Capítulo 5

Conclusão

Dispositivos portáteis têm se tornado cada vez mais populares. Uma de suas principais características é a de depender de alimentação por bateria para a sua operação. Adicionalmente, em várias situações, o tipo de carga a ser processada apresenta requisitos de tempo real. Dessa forma, técnicas que permitam economia de energia vêm sendo pesquisadas. Um dos maiores consumidores de energia em sistemas como os aqui citados é o processador. Neste trabalho foram apresentados métodos de escalonamento de tarefas de tempo real que procuram ocupar os tempos livres (folga e *gain time*) da UCP, colocando-a a operar em nível de voltagem (e também de freqüência de relógio) mais baixo, de forma a diminuir o consumo de energia.

Os métodos propostos nessa dissertação forneceram bons resultados nos diversos experimentos realizados e, em particular, quando comparados a um dos principais métodos para escalonadores RM publicados na literatura, o ccRM [Pillai e Shin 2001]. Vários testes foram realizados com o intuito de abranger as diferentes características que um sistema de tempo real pode apresentar. Em todos os testes os algoritmos propostos demonstraram significativa redução no consumo de energia.

Em diversas simulações, e principalmente quando o sistema apresenta em sua fase dinâmica muita folga remanescente da etapa estática, os métodos Aprox 1N e 2N apresentaram resultados melhores por determinarem mais eficientemente a folga remanescente. Isso se justifica porque determinar o *gain time* é fácil, porém o quanto há de folga remanescente em cada instante de tempo é um diferencial para os métodos dinâmicos.

Embora o Aprox 2N tenha apresentado uma melhor utilização da folga disponível, ainda ocorrem vários momentos em que o processador se encontra ocioso. Trabalhos futuros podem utilizar esta ociosidade para o atendimento de tarefas aperiódicas não críticas. Pode-se consi-

derar, nesses STR que utilizam tarefas periódicas e aperiódicas, dois casos: (i) o atendimento de tarefas aperiódicas ocorreria somente quando o método de economia de energia utilizado em conjunto com o RDV não utilizar toda a folga disponível e, (ii) pode-se também aplicar uma função com uma recompensa associada. Nesse último caso, o objetivo seria maximizar a recompensa balanceando entre economizar energia ou atender tarefas aperiódicas.

Ainda que se tenha realizado vários testes com o intuito de abranger as diferentes características que um sistema de tempo real pode apresentar, esses podem ser estendidos com um outro enfoque. Por exemplo, utilizar outros métodos de roubo de folga (como o descrito por [Davis 1993]) , analisando-os não somente em função da energia a ser economizada, mas também do ponto de vista do impacto do *overhead* na determinação da folga.

Em todos os experimentos realizados, nenhum método obteve resultados tão bons quanto o Ótimo. Isso era esperado, pois: (i) não há uma utilização de 100% da CPU pelo RM uma vez que os períodos gerados não são harmônicos; e, (ii) no caso do método ótimo considerou-se a utilização de um esquema contínuo de freqüências para o processador. Estudos futuros poderão ser realizados para se determinar o quão distante os resultados obtidos se encontram do ótimo, considerando que o ótimo seria determinado em função dessas restrições.

Finalmente, deve-se acrescentar que o uso de mecanismos de diminuição de gasto energético pode implicar em intervenção adicional do sistema operacional. Isso ocorre por que: (i) uma tarefa ao utilizar a folga disponível, durante sua execução, atrasa tanto quanto possível as outras, de menor prioridade que a tarefa atual; (ii) tarefas podem ser preemptadas mais vezes. Essas preempções causam um maior *overhead* de troca de contexto; e, (iii) a utilização de dois níveis acarreta uma intervenção adicional do sistema. Esses pontos também poderão ser objeto de estudos futuros.

De qualquer forma, os trabalhos realizados até o momento comprovam que uma quantidade considerável de energia pode ser economizada no escalonamento de tarefas em sistemas de tempo real. Para isso, utilizou-se diferentes formas de determinar a ociosidade da UCP que, geralmente, é ocasionada pela folga disponível no sistema e pelo *gain time*, e diferentes métodos na utilização desse tempo ocioso. Ainda que alguns dos métodos apresentem resultados melhores que outros, conforme discutido anteriormente, todos demonstraram uma significativa redução no consumo de energia, o que pôde ser comprovado pelos experimentos apresentados nesse trabalho.

Referências

[Aydin et al. 2001a] Aydin, H.; Mejía-Alvarez, P.; Mossé, D. e Melhem, R. **Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems**. In: *22nd IEEE Real-Time Systems Symposium*, Londres, Reino Unido, p. 95–105, Dezembro, 2001.

[Aydin et al. 2001b] Aydin, H.; Mejía-Alvarez, P.; Mossé, D. e Melhem, R. **Optimal Reward-Based Scheduling for Periodic Real-Time Tasks**. *IEEE Transactions on Computers*, v. 50, n. 1, p. 111–130, 2001.

[Bertini e Leite 2004] Bertini, L. e Leite, J. C. B. **Um Breve *Survey*: Escalonamento em Sistemas de Tempo Real com Otimização do Consumo de Energia**. In: *VI Workshop de Tempo Real*, Gramado, RS, Brasil, p. 288–297, Maio, 2004.

[Buttazzo 2003] Buttazzo, G. C. **Rate Monotonic vs. EDF: Judgment Day**. In: *3rd International Conference on Embedded Software*, Filadélfia, Pensilvânia, EUA, p. 67–83, Outubro, 2003.

[Davis 1993] Davis, R. **Approximate Slack Stealing Algorithms for Fixed Priority Preemptive Systems**. Relatório Técnico YCS, Department of Computer Science, University of York, Inglaterra, 1993.

[Davis et al. 1993] Davis, R. I.; Tindell, K. W. e Burns, A. **Scheduling Slack-Time in Fixed Priority Preemptive Systems**. In: *14th IEEE Real Time Systems Symposium*, Raleigh-Durham, Carolina do Norte, EUA, p. 222–231, Dezembro, 1993.

[Dey et al. 1996] Dey, J. K.; Kurose, J. e Towsley, D. **On-line Scheduling Policies for a Class of IRIS (Increasing Reward with Increasing Service) Real-Time Tasks**. *IEEE Transactions on Computers*, v. 45, n. 7, p. 802–813, 1996.

[Ellis 1999] Ellis, C. S. **The Case for Higher-Level Power Management**. In: *7th IEEE Workshop on Hot Topics in Operating Systems*, Rio Rico, Arizona, EUA, p. 162–167, Março, 1999.

[Ernst e Ye 1997] Ernst, R. e Ye, W. **Embedded Program Timing Analysis Based on Path Clustering and Architecture Classification**. In: *International Conference on Computer-Aided Design*, San Jose, Califórnia, EUA, p. 598–604, Novembro, 1997.

[Gruian 2001] Gruian, F. **Hard Real-Time Scheduling for Low-Energy Using Stochastic Data and DVS Processors**. In: *International Symposium on Low Power Electronics and Design*, Nova York, Nova York, EUA, p. 46–51, Agosto, 2001.

[Im et al. 2004] Im, C.; Ha, S. e Kim, H. **Dynamic Voltage Scheduling with Buffers in Low-Power Multimedia Applications**. *Transactions on Embedded Computing Systems*, v. 3, n. 4, p. 686–705, 2004.

[Ishihara e Yasuura 1998] Ishihara, T. e Yasuura, H. **Voltage Scheduling Problem for Dynamically Variable Voltage Processors**. In: *International Symposium on Low-Power Electronics and Desing*, Monterey, Califórnia, EUA, p. 197–201, Janeiro, 1998.

[Joseph e Pandya 1986] Joseph, M. e Pandya, P. **Finding Response Times in a Real-Time System**. *Computer Journal*, v. 29, n. 5, p. 390–395, 1986.

[Kim et al. 1996] Kim, N.; Ryu, M.; Hong, S.; Saksena, M.; Choi, C.-H. e Shin, H. **Visual Assessment of a Real-Time System Design: A Case Study on a CNC Controller**. In: *17th IEEE Real-Time Systems Symposium*, Washington, Distrito de Columbia, EUA, p. 300 – 310, Dezembro, 1996.

[Kim et al. 2002] Kim, W.; Kim, J. e Min, S. **A Dynamic Voltage Scaling Algorithm for Dynamic-Priority Hard Real-Time Systems Using Slack Time Analysis**. In: *Conference on Design, Automation and Test in Europe*, Washington, Distrito de Columbia, EUA, p. 788–794, Março, 2002.

[Kim et al. 2003] Kim, W.; Kim, J. e Min, S. **Dynamic Voltage Scaling Algorithm for Fixed-Priority Real-Time Systems Using Work-Demand Analysis**. In: *International Symposium on Low Power Electronics and Design*, Seul, Coréia, p. 396–401, Agosto, 2003.

[Kim et al. 2002] Kim, W.; Shin, D.; Yun, H.; Kim, J. e Min, S. **Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems**. In: *8th IEEE Real-Time and Embedded Technology and Applications Symposium*, São José, Califórnia, EUA, p. 219–228, Setembro, 2002.

[Lee e Sakurai 2000] Lee, S. e Sakurai, T. **Run-Time Voltage Hopping for Low-Power Real-Time Systems**. In: *37th Conference on Design Automation*, Los Angeles, Califórnia, EUA, p. 806–809, Junho, 2000.

[Lehoczky e Ramos-Thuel 1992] Lehoczky, J. P. e Ramos-Thuel, S. **An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems**. In: *13th IEEE Real-Time Systems Symposium*, Fênix, Arizona, EUA, p. 110–123, Dezembro, 1992.

[Liu e Layland 1973] Liu, C. L. e Layland, J. W. **Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment**. *Journal of the ACM*, v. 20, n. 1, p. 46–61, 1973.

[Locke et al. 1991] Locke, C.; Vogel, D. e Mesler, T. **Building a Predictable Avionics Platform in Ada: A Case Study**. In: *12th IEEE Real-Time Systems Symposium*, San Antonio, Texas, EUA, p. 181–189, Dezembro, 1991.

[Lorch e Smith 1998] Lorch, J. R. e Smith, A. J. **Apple Macintosh Energy Consumption**. *IEEE Micro*, v. 18, n. 6, p. 54–63, 1998.

[Luo e Jha 2002] Luo, J. e Jha, N. K. **Static and Dynamic Variable Voltage Scheduling Algorithms for Real-Time Heterogeneous Distributed Embedded Systems**. In: *ASP-DAC/VLSI Design*, Washington, Distrito de Columbia, EUA, p. 719–726, Janeiro, 2002.

[Melhem et al. 2004] Melhem, R.; Mossé, D. e Elnozahy, E. **The Interplay of Power Management and Fault Recovery in Real-Time Systems**. *IEEE Transactions on Computers*, v. 53, n. 2, p. 217–231, 2004.

[Miyoshi et al. 2002] Miyoshi, A.; Lefurgy, C.; Hensbergen, E. V.; Rajamony, R. e Rajkumar, R. **Critical Power Slope: Understanding the Runtime Effects of Frequency Scaling**. In: *16th ACM International Conference on Supercomputing*, Nova York, Nova York, EUA, p. 35–44, Junho, 2002.

[Mok 1983] Mok, A. K.-L. **Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment**. Tese (Doutorado), MIT, Cambridge, Massachusetts, EUA, 1983.

[Mossé et al. 2004] Mossé, D.; Aydin, H.; Childers, B. e Melhem, R. **Compiler-Assisted Dynamic Power-Aware Scheduling for Real-Time Applications**. In: *VI Workshop de Tempo Real*, Gramado, RS, Brasil, p. 54–63, Maio, 2004.

[Novelli et al. 2005] Novelli, B. A.; Leite, J.; Urriza, J. M. e Orozco, J. D. **Regulagem Dinâmica de Voltagem em Sistemas de Tempo Real**. In: *XXXII Seminário Integrado de Software e Hardware*, São Leopoldo, RS, Brasil, Julho, 2005.

[Obenza 1993] Obenza, R. **Rate Monotonic Analysis for Real-Time Systems**. *IEEE Computer*, v. 26, n. 3, p. 73–74, 1993.

[Pering e Brodersen 1998] Pering, T. e Brodersen, R. **Energy Efficient Voltage Scheduling for Real-Time Operating Systems**. In: *4th IEEE Real-Time Technology and Applications Symposium*, Denver, Colorado, EUA, Junho, 1998. Work-in-progress session.

[Pillai e Shin 2001] Pillai, P. e Shin, K. G. **Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems**. In: *18th Symposium on Operating Systems Principles*, Banff, Alberta, Canadá, p. 89–102, Dezembro, 2001.

[Quan et al. 2004] Quan, G.; Niu, L. e Davis, J. P. **Power Aware Scheduling for Real-Time Systems with (m,k)-Guarantee**. In: *Communication Networks and Distributed Systems Modeling and Simulation*, San Diego, Califórnia, EUA, p. 10–12, Janeiro, 2004.

[Rusu et al. 2002] Rusu, C.; Mossé, D. e Melhem, R. **Maximizing the System Value while Satisfying Time and Energy Constraints**. In: *23rt IEEE Real-Time Systems Symposium*, Austin, Texas, EUA, p. 246 – 255, Dezembro, 2002.

[Rusu et al. 2003a] Rusu, C.; Mossé, D. e Melhem, R. **Maximizing Rewards for Real-Time Applications With Energy Constraints**. *ACM Transactions on Embedded Computing Systems*, v. 2, n. 4, p. 537–559, 2003.

[Rusu et al. 2003b] Rusu, C.; Mossé, D. e Melhem, R. **Multi-Version Scheduling in Rechargeable Energy-Aware Real-Time Systems**. In: *15th Euromicro Conference on Real-Time Systems*, Porto, Portugal, p. 95 – 104, Julho, 2003.

[Saewong e Rajkumar 2003] Saewong, S. e Rajkumar, R. **Practical Voltage-Scaling for Fixed-Priority RT-Systems**. In: *9th IEEE Real-Time and Embedded Technology and Applications Symposium*, Toronto, Canadá, p. 106–115, Maio, 2003.

[Santos e Orozco 1993] Santos, J. e Orozco, J. **Rate Monotonic Scheduling in Hard Real-Time Systems**. *Information Processing Letters*, v. 48, p. 39–45, 1993.

[Santos et al. 2002] Santos, R. M.; Urriza, J.; Santos, J. e Orozco, J. **Heuristic use of Singularities for On-Line Scheduling of Real-Time Mandatory/Reward-Based Optional Systems**. In: *14th Euromicro Conference on Real-Time Systems*, Vienna, Áustria, p. 103–110, Junho, 2002.

[Santos et al. 2004] Santos, R. M.; Urriza, J.; Santos, J. e Orozco, J. **New Methods for Redistributing Slack Time in Real-Time Systems: Applications and Comparative Evaluations**. *Journal of Systems and Software*, v. 69, n. 1-2, p. 115–128, 2004.

[Sharma et al. 2003] Sharma, V.; Thomas, A.; Abdelzaher, T.; Skadron, K. e Lu, Z. **Power-aware QoS Management in Web Servers**. In: *24th IEEE International Real-Time Systems Symposium*, Cancun, México, p. 63–72, Dezembro, 2003.

[Shin et al. 2001] Shin, D.; Kim, J. e Lee, S. **Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications**. *IEEE Design and Test*, v. 18, n. 2, p. 20–30, 2001.

[Shin et al. 2000] Shin, Y.; Choi, K. e Sakurai, T. **Power Optimization of Real-Time Embedded Systems on Variable Speed Processors**. In: *International Conference on Computer-Aided Design*, San Jose, Califórnia, EUA, p. 365–368, Novembro, 2000.

[Sinha e Chandrakasan 2001] Sinha, A. e Chandrakasan, A. **Dynamic Voltage Scheduling Using Adaptive Filtering of Workload Traces**. In: *14th International Conference on VLSI Design*, Bangalore, India, p. 221–226, Janeiro, 2001.

[Sprunt et al. 1989] Sprunt, B.; Sha, L. e Lehoczky, J. P. **Aperiodic Task Scheduling for Hard Real-Time Systems**. *The Journal of Real-Time Systems*, v. 1, n. 1, p. 27–60, 1989.

[Strosnider et al. 1995] Strosnider, J. K.; Lehoczky, J. P. e Sha, L. **The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments**. *IEEE Transactions on Computers*, v. 44, n. 1, p. 73–91, 1995.

[Tia et al. 1996] Tia, T.-S.; Liu, J. W.-S. e Shankar, M. **Algorithms and Optimality of Scheduling Soft Aperiodic Requests in Fixed-Priority Preemptive Systems**. *Real-Time Systems*, Norwell, Massachusetts, EUA, v. 10, n. 1, p. 23–43, 1996.

[Unsal e Koren 2003] Unsal, O. S. e Koren, I. **System-Level Power-Aware Design Techniques in Real-Time Systems**. *Proceedings of the IEEE*, v. 91, n. 7, p. 1055–1069, 2003.

[Urriza et al. 2004] Urriza, J. M.; Novelli, B. A.; Leite, J. C. B. e Orozco, J. D. **Economia de Energia em Dispositivos Móveis**. In: *VI Workshop de Comunicação sem Fio e Computação Móvel*, Fortaleza, CE, Brasil, p. 48–56, Outubro, 2004.

[Urriza e Orozco 2004] Urriza, J. M. e Orozco, J. D. **Métodos Rápidos para el Cálculo del Slack Stealing Exacto y Aproximado para Aplicaciones en Sistemas Embebidos**. Relatório Técnico, Dep. de Ing. Eléctrica y de Computadoras, Universidad Nacional del Sur, Bahía Blanca, Argentina, Agosto, 2004.

[Weiser et al. 1994] Weiser, M.; Welch, B.; Demers, A. e Shenker, S. **Scheduling for Reduced CPU Energy**. In: *1st Symposium on Operating Systems Design and Implementation*, Monterey, Califórnia, EUA, p. 13–23, Novembro, 1994.

[Yuan e Nahrstedt 2003] Yuan, W. e Nahrstedt, K. **Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems**. In: *20th Symposium on Operating Systems Principles*, Bolton Landing, Nova York, EUA, p. 149–163, Outubro, 2003.