Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Curso de Graduação em Engenharia Elétrica

BRUNA MARIA JUSTINO CRUZ

TRABALHO DE CONCLUSÃO DE CURSO
ESTIMAÇÃO RECURSIVA DO MODELO DE POTÊNCIA DE UM SISTEMA COMPUTACIONAL

Campina Grande, Paraíba
Agosto de 2012

BRUNA MARIA JUSTINO CRUZ

ESTIMAÇÃO RECURSIVA DO MODELO DE POTÊNCIA DE UM SISTEMA COMPUTACIONAL

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Engenheiro Eletricista.

Área de Concentração: Processamento de Energia

Orientador:
Professor Saulo Oliveira Dornellas Luiz, D. Sc.

Campina Grande, Paraíba
Agosto de 2012

BRUNA MARIA JUSTINO CRUZ

ESTIMAÇÃO RECURSIVA DE UM MODELO DE POTÊNCIA DE UM SISTEMA COMPUTACIONAL

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Engenheiro Eletricista.

Área de Concentração: Processamento de Energia

Aprovado em / /

Professor Avaliador
Universidade Federal de Campina Grande
Avaliador

Professor Saulo Oliveira Dornellas Luiz, D. Sc.
Universidade Federal de Campina Grande
Orientador, UFCG

AGRADECIMENTOS

A Deus, por sempre estar presente, iluminando meu caminho e me abençoando em todos os momentos da vida.

Aos meus pais e irmãos pela confiança em mim depositada, pela admiração, e acima de tudo pelo amor.

Ao professor Saulo Oliveira Dornellas Luiz pelas valiosas orientações e contribuições ao longo de minha formação acadêmica.

A Pablo por todo o incentivo, paciência e amor.

A todos os amigos do EMBEDDED.

"As pessoas parecem concentrar-se melhor
Quando o que lhes é exigido é um pouco mais que o habitual
E elas podem dar mais que o habitual.
Se as exigências são poucas demais, elas se entediam."

Daniel Goleman.

LISTA DE ILUSTRAÇÕES

Figura 1: Bloco lógico que relaciona as entradas e a saída de um modelo 4
Figura 2: Monitor do Sistema do Ubuntu 10.10 que permite ao usuário observar o histórico de utilização da CPU, memória, e rede 7
Figura 3: Fluxograma para a implementação do *software* GDE 13
Figura 4: Gráfico dos Parâmetros estimados para utilização de um núcleo lógico do processador 15
Figura 5: Variáveis de entrada do modelo para utilização de um núcleo lógico do processador 16
Figura 6: Gráfico com os estados de energia para cada um dos núcleos do processador, potência média e potência estimada, quando na utilização de um núcleo lógico do processador 17
Figura 7: Potência medida e potência estimada quando um núcleo lógico 100% utilizado..... 18
Figura 8: Gráfico dos Parâmetros estimados para utilização de dois núcleos lógicos do processador..... 19
Figura 9: Variáveis de entrada do modelo para utilização de dois núcleos lógicos do processador 20
Figura 10: Gráfico com os estados de energia para cada um dos núcleos do processador, potência média e potência estimada, quando na utilização dois núcleos lógicos do processador21
Figura 11: Potência medida e potência estimada quando dois núcleos lógicos 100% utilizados 22
Figura 12: Gráfico dos Parâmetros estimados para utilização de três núcleos lógicos do processador..... 23
Figura 13: Variáveis de entrada do modelo para utilização de três núcleos lógicos do processador..... 24
Figura 14: Gráfico com os estados de energia para cada um dos núcleos do processador, potência média e potência estimada, quando na utilização de três núcleos lógicos do processador 25
Figura 15: Potência medida e potência estimada quando três núcleos lógicos 100% utilizados 26
Figura 16: Gráfico dos Parâmetros estimados para utilização dos quatro núcleos lógicos do processador..... 27
Figura 17: Variáveis de entrada do modelo para utilização dos quatro núcleos lógicos do processador..... 28
Figura 18: Gráfico com os estados de energia para cada um dos núcleos do processador, potência média e potência estimada, quando na utilização dos quatro núcleos lógicos do processador..... 29
Figura 19: Potência medida e potência estimada quando quatro núcleos lógicos 100% utilizados 30

SUMÁRIO

1 Introdução 1
1.1 Objetivos 2
1.2 Metodologia 2
1.3 Organização do Texto 3
2 Fundamentação Teórica 3
2.1 Estimação dos Parâmetros de um Modelo 4
3 Plataforma Experimental 6
3.1 Políticas de Gerenciamento de Energia no Linux 7
3.1.1 Performance 8
3.1.2 Powersave 8
3.1.3 Userspace 8
3.1.4 Ondemand 9
3.1.5 Conservative 9
3.2 Medição e Utilização do Processador 9
3.3 Frequência do Processador 11
3.4 Potência da Bateria 11
4 Estimação Recursiva de um Modelo De Potência para um Sistema Computacional 12
4.1 Implementação de um *software* para estimação do Modelo de Potência do sistema computacional 12
4.2 Cargas de Trabalho 13
5 Resultados Obtidos 14
5.1 Estimação do Modelo com um núcleo lógico 100% utilizado 14
5.2 Estimação do Modelo com dois núcleos lógicos 100% utilizados 18
5.3 Estimação do Modelo com três núcleos lógicos 100% utilizados 22
5.4 Estimação do Modelo com quatro núcleos lógicos 100% utilizados 26
6 Conclusões e Trabalhos Futuros 31
Apêndice A: Código em Linguagem C do *Software* Livre GDE 32
Apêndice B: Biblioteca de Funções para Auxiliar o GDE 41
Apêndice C: Modelo do Sistema 53
REFERÊNCIAS BIBLIOGRÁFICAS 56

1 INTRODUÇÃO

Reduzir o consumo de energia de sistemas computacionais, além de uma necessidade, representa uma área de pesquisa desafiante. Por exemplo, num processador de um sistema computacional, é preciso conciliar velocidade de processamento e resfriamento de *hardware*. Quão maior for a velocidade de processamento, maior será a dissipação de calor e maiores serão os gastos com resfriamento [1].

Para dispositivos móveis, alimentados por baterias como, *notebooks*, *netbooks*, telefones celulares, *tablets*, etc. Existe uma demanda crescente de desempenho [4]. Processadores mais poderosos, maior quantidade de memória, interfaces de rede, etc, contribuem para tornar a carga do sistema cada vez maior. Assim, é preciso fornecer maior autonomia ao sistema. Contudo, a densidade de energia armazenada nas baterias não tem aumentado na mesma taxa que o desempenho requerido pelos sistemas computacionais. Assim, é necessário o desenvolvimento de técnicas de gerenciamento de energia, para reduzir o consumo de energia de sistemas computacionais, e assim permitir uma maior autonomia com o mesmo ou menor tamanho das baterias.

Nesse contexto insere-se o Gerenciamento Dinâmico de Energia (GDE), que é uma das técnicas mais utilizadas na indústria e que explora características de *hardware*, possibilitando assim mudanças no modo de operação de alguns componentes do sistema, sem prejudicar o desempenho, de forma a economizar energia e aumentar a autonomia do sistema [5].

Para atender à demanda crescente por energia, os fabricantes de semicondutores têm desenvolvido várias técnicas que visam a redução do consumo de potência, tais como estados de baixo consumo e escalonamento dinâmico de tensão e frequência (DVFS) [2]. Entretanto, usar bem esses recursos para obter redução no consumo e consequentemente otimização do tempo da bateria requer o desenvolvimento de uma política de gerenciamento de energia que estime com precisão a carga de trabalho do sistema e a potência consumida pelo sistema.

1.1 OBJETIVOS

O objetivo deste trabalho de conclusão de curso é a estimação de um modelo de potência, para um sistema computacional, a partir da utilização do processador e de sua frequência.

O sistema computacional estudado, é um *notebook*, com processador Intel® Core™ i5-2410M. O Sistema Operacional (SO) instalado no *notebook* é o Linux, por ser um *software* livre, licenciado pela *General Public License* (GNU), e de código aberto, o que permite ao usuário acesso à informações como utilização e frequência da *Central Processing Unit* (CPU).

Além disso para a estimação do modelo de potência do sistema computacional, são necessárias medições de potência, o que pode ser realizado por meio de medições da tensão e corrente que a bateria fornece ao sistema [6].

1.2 METODOLOGIA

Para a realização deste trabalho de conclusão de curso, foram realizadas as seguintes atividades:

- Consulta às referências bibliográficas sobre gerenciamento dinâmico de energia disponíveis;
- Implementação de um *software,* na linguagem de programação C, a ser executado em espaço de usuário do sistema operacional Linux, na plataforma alvo, para aquisição dos estados dos dispositivos e da potência consumida pela bateria;
- Implementação de um algoritmo de Mínimos Quadrados Recursivos para estimação do modelo de potência em função dos estados da plataforma alvo;
- Realização de experimentos para avaliação do *software* e o algoritmo de estimação do modelo de potência.

1.3 ORGANIZAÇÃO DO TEXTO

Na Seção 2 é apresentada a fundamentação sobre a necessidade dos modelos de potência, e é apresentado o método dos mínimos quadrados recursivos que é uma ferramenta matemática para a estimação de modelos. Em seguida, na Seção 3, é apresentada a plataforma alvo e o sistema operacional Linux. São discutidos os conceitos de frequência e utilização de um processador, e é mostrado como é possível ter acesso a essas informações no Linux. Na Seção 4 são apresentados os experimentos realizados para avaliação do *software* e do algoritmo de estimação do modelo de potência. E por fim, na Seção 5, são realizadas as considerações finais.

2 FUNDAMENTAÇÃO TEÓRICA

Modelos de potência são importantes para a otimização e o gerenciamento de energia em sistemas alimentados por baterias. Existem métodos para estimação de modelos de potência que são construídos em laboratórios, com o auxílio de um segundo computador dedicado à aquisição de dados, e que requerem grande quantidade de experimentos. No entanto, esses métodos não se adaptam à grande variedade de *hardwares* utilizados em sistemas móveis, porque são estimados para uma plataforma alvo específica. Outros métodos estimam o consumo de energia para intervalos de tempo maiores que um segundo, o que não permite uma maior precisão da carga. Um terceiro método é aquele em que o sistema móvel gera automaticamente seu modelo de potência sem qualquer dispositivo externo [3].

Definir qual método matemático será empregado no processo de modelagem é um passo importante para a construção de um modelo. Dentre as metodologias utilizadas para estimação dos parâmetros de um modelo, uma das mais tradicionais é o método dos mínimos quadrados.

O algoritmo dos mínimos quadrados (MQ) processa todas as observações da entrada e da saída de um sistema simultaneamente e produz uma estimativa única do vetor de parâmetros. Este é chamado de algoritmo em batelada e é usado quando as estimativas são necessárias somente uma vez e processadas *off-line*. Por outro lado, os métodos recursivos processam as observações e adaptam os parâmetros estimados a

cada intervalo de amostragem (*online*). Estes últimos são utilizados, preferencialmente, em sistemas variantes no tempo e em aplicações de monitoração, controle e predição em tempo real [7]

Para a estimação do modelo de potência proposto para o sistema computacional, foi utilizado o método dos mínimos quadrados recursivos (MQR).

2.1 ESTIMAÇÃO DOS PARÂMETROS DE UM MODELO

O processo a ser modelado é o consumo de potência fornecida ao *notebook* pela bateria, tendo como entradas funções da utilização e frequência de cada núcleo do processador. O bloco lógico que representa o modelo de potência pode ser visualizado na Fig.1.

Figura 1: Bloco lógico que relaciona as entradas e a saída de um modelo

A carga de trabalho em um sistema computacional varia com o tempo, devido à própria utilização pelo usuário, e até por atividades que independem do mesmo, como atualizações de antivírus. Dessa forma a carga de trabalho, ou seja, a carga a qual o sistema computacional está submetido é dita não estacionária.

Dessa forma, a modelagem de tal processo e com tal carga de trabalho necessita ser feita em tempo de execução, pois os parâmetros desse processo estão variando em tempo real, juntamente com a carga. É necessário então, que os parâmetros desse modelo de potência, sejam obtidos por meio de estimação recursiva.

A técnica utilizada para realizar essa estimação foi a implementação do algoritmo de *mínimos quadrados recursivos* (MQR).

Uma relação básica entre as entradas u(t) com a saída y(t) de um sistema, semelhante ao apresentado na Fig. 1, é a equação de diferenças linear, apresentada em (1):

$$y(t) + a_1 y(t-1) + \cdots + a_n y(t-\mathrm{n}) = b_1 u(t-1) + \cdots + b_m u(t-\mathrm{m}) \quad (1)$$

Os valores de entrada e saída que são utilizados para o modelo de potência do sistema computacional, são obtidos por amostragem, ou seja, tem-se um modelo discreto no tempo. Para melhor manipulação, a relação em (1) é colocada na forma em (2). De forma que é possível observar que o valor da saída atual do sistema depende dos valores passados de entrada e saída observadas [8].

$$y(t) = -a_1 y(t-1) - \cdots - a_n y(t-\mathrm{n}) + b_1 u(t-1) + \cdots + b_m u(t-\mathrm{m}) \quad (2)$$

Para uma notação mais compacta, são introduzidos os vetores:

$$\theta = [a_1 \ldots a_n \; b_1 \ldots b_m]^T \quad (3)$$

$$\varphi(t) = [-y(t-1) \ldots -y(t-n) \;\; u(t-1) \ldots u(t-m)]^T \quad (4)$$

Onde θ é o vetor de parâmetros que se deseja estimar, e $\varphi(t)$ é o vetor de entradas e saídas passadas do sistema.

E assim, a relação em (4), pode ser reescrita da forma:

$$y(t) = \varphi^T(t)\theta \quad (5)$$

Para enfatizar que a estimativa de $y(t)$ depende das entradas passadas, saídas passadas e também de θ, ela é denotada como:

$$\hat{y}(t \mid \theta) \;=\; \varphi^T(t)\theta \quad (6)$$

É importante enfatizar que o modelo em (2) é variante no tempo. Então é necessário identificar esse modelo recursivamente em tempo de execução, após cada novo período de amostragem. Foram implementadas as expressões de (7) à (10), onde I é a matriz identidade e o parâmetro $\lambda(t)$ é o fator de esquecimento que é usado para reduzir a dependência das estimativas das medidas mais distantes no tempo isto é, ele cria uma memória que cai exponencialmente para o estimador, λ é escolhido de tal forma que a constante de tempo da variação dos parâmetros é maior que a memória do algoritmo [9].

Nesse trabalho λ foi escolhido empiricamente, com valor $\lambda(t) = 0{,}98$. Para a implementação do algoritmo dos mínimos quadrados, são utilizadas as equações de (7)

à (9) [8]. É necessário calcular as matrizes L(t) e P(t) antes de calcular a estimativa dos parâmetros $\hat{\theta}$(t).

$$\epsilon(t) = y(t) - \varphi^T(t)\hat{\theta}(t-1) \tag{7}$$

$$\hat{\theta}(t) = \hat{\theta}(t-1) + L(t)\epsilon(t) \tag{8}$$

$$L(t) = \frac{P(t-1)\varphi(t)}{\lambda(t) + \varphi^T P(t-1)\varphi(t)} \tag{9}$$

$$P(t) = \frac{1}{\lambda(t)}[P(t-1) - \frac{P(t-1)\varphi(t)\varphi^T(t)P(t-1)}{\lambda(t) + \varphi^T(t)P(t-1)\varphi^T} \tag{10}$$

$$= \frac{1}{\lambda(t)}[I - L(t)\varphi^T(t)]P(t-1)$$

$$\beta(t,k) = \lambda(t)\beta(t-1,k) \qquad 0 \le k \le t-1$$
$$\beta(t,t) = 1 \tag{11}$$

3 PLATAFORMA EXPERIMENTAL

A plataforma utilizada na estimação do modelo de potência foi um *notebook*, com processador Intel® Core™ i5-2410M, com 2,3GHz de processamento, e 4 GB de memória RAM.

O Processador Intel® Core™ i5-2410M possui dois núcleos físicos, cada um com *multithread*, oferecendo dessa forma, quatro núcleos lógicos. Memória *cache* de 3MB. A frequência básica é de 2,3 GHz, mas devido ao *Turbo Boost* pode atingir 2,6 GHz, com dois núcleos ativos, e 2,9 GHz quando apenas um núcleo está ativo [10].

O sistema operacional Linux com distribuição Ubuntu 10.10 e versão do núcleo 2.6.35-22-*generic* (i686), foi escolhido por fornecer suporte à medição, pelo próprio sistema operacional, da potência que a bateria fornece à plataforma experimental, assim como a frequência e utilização de cada núcleo do processador. Fornece também, uma aplicação que permite ao usuário observar o histórico de uso da CPU, como apresentado na Fig. 2.

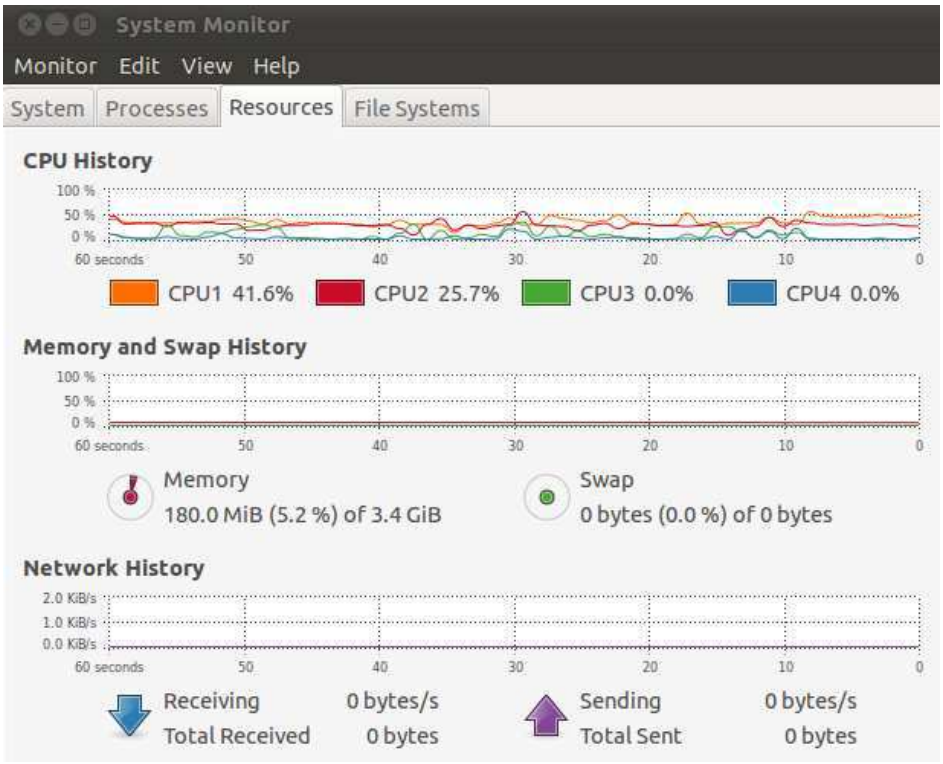

Figura 2: Monitor do Sistema do Ubuntu 10.10 que permite ao usuário observar o histórico de utilização da CPU, memória, e rede

3.1 POLÍTICAS DE GERENCIAMENTO DE ENERGIA NO LINUX

No Sistema Operacional Linux, as frequências do processador podem ser escaladas dinamicamente através do *driver* de gerenciamento de energia `CPUfreq`. Quando os processadores operam a uma frequência mais baixa, eles consomem menos energia e dissipam menos calor. Essa escala dinâmica de frequência oferece algum controle, regulando o sistema para consumir menos energia quando o processador estiver inativo.

A estrutura `CPUfreq` dispõem de gerenciadores de energia cada um definido por uma política de gerenciamento de energia para escolha estática ou dinâmica da

frequência do processador. Os gerenciadores de energia dinâmicos podem alternar entre as frequências da CPU com base na utilização da CPU para permitir economia de energia sem comprometer o desempenho. Esses gerenciadores de energia também permitem configurações pelo usuário para que seja possível alterar a escala de frequências [11].

No Linux há cinco gerenciadores de energia integrados no núcleo *(kernel)* disponíveis para uso com o driver `CPUfreq` [12]. Os gerenciadores estáticos são o *performance* e *powersave*. Já o *userspace*, *ondemand* e *conservative* são gerenciadores de energia dinâmicos. As características de cada um dos gerenciadores são apresentadas a seguir:

3.1.1 PERFORMANCE

A política *performance* configura o processador para a mais alta frequência disponível. É possível ajustar a faixa de frequências disponível para esta política. O objetivo desta política é obter o máximo desempenho de um sistema configurando a velocidade de *clock* do processador para o nível máximo e deixando-o nesse nível.

3.1.2 POWERSAVE

A política *powersave* configura o processador para a mínima frequência disponível. É possível ajustar a frequência mínima disponível para esta política. O objetivo dessa política é executar a carga de trabalho do sistema na menor velocidade possível. Consequentemente, isso pode afetar o desempenho do sistema, que nunca ultrapassará essa frequência mínima definida, não importando quão ocupado o processado esteja.

3.1.3 USERSPACE

A política *userspace* permite selecionar e definir uma frequência manualmente. A política *userspace* não altera dinamicamente a frequência. Ela permite que o usuário ou um programa de espaço de usuário selecione dinamicamente a frequência do processador.

3.1.4 ONDEMAND

A política *ondemand* foi a primeira política integrada no *núcleo do Linux* a alterar dinamicamente a frequência baseada na utilização do processador. A política *ondemand* verifica a utilização do processador e se ela exceder o limite, a política definirá a frequência para a mais elevada disponível. Se for verificado na política que a utilização está abaixo do limite, a frequência será reduzida para a próxima disponível. Se o sistema continuar a ser subutilizado, a política continuará a diminuir a frequência até a menor disponível.

3.1.5 CONSERVATIVE

A política *conservative* é semelhante à *ondemand*, pois ajusta dinamicamente as frequências baseadas na utilização do processador; entretanto, a política *conservative* se comporta de forma um pouco diferente e permite um aumento mais gradual dos estados de energia. Verifica-se a utilização do processador e se esta estiver acima ou abaixo dos limites de utilização, o gerenciador, respectivamente, aumenta ou diminui a frequência para a próxima disponível.

3.2 MEDIÇÃO E UTILIZAÇÃO DO PROCESSADOR

A utilização do processador é a fração de tempo durante o qual o processador está executando instruções. Não se tem acesso a utilização diretamente, mas pode ser calculada a partir da leitura do conteúdo do arquivo `/proc/stat`, que é formado por um conjunto de contadores que são incrementados sempre que na CPU é realizada alguma atividade. Um exemplo das primeiras linhas desse arquivo é apresentado em seguida:

cpu 5452 23 2180 473379 4409 0 11 0 0 0
cpu0 2625 3 854 114302 3144 0 7 0 0 0
cpu1 935 12 478 120019 360 0 3 0 0 0
cpu2 1650 0 645 118684 590 0 0 0 0 0
cpu3 241 7 202 120373 314 0 0 0 0 0

A linha cpu contêm contadores associados a todo o processador, enquanto as linhas seguintes contêm contadores específicos de cada núcleo do processador. Cada coluna corresponde a um contador que é incrementado quando é realizada uma determinada atividade no processador [13]. A cada 10 ms apenas um contador de cada núcleo é atualizado.

A sequência da esquerda para a direita das colunas da linha "**cpu**"corresponde às seguintes atividades [14]:

<user>: tempo de processos executando em espaço de usuário;

<nice>: tempo de processos executado em espaço de usuário e com baixa prioridade;

<system>: processos executados em espaço de núcleo;

<idle>: contador incrementado quando o processador está inativo;

<iowait>: tempo de espera por uma operação de entrada ou saída ser completada;

<irq>: tempo de interrupções;

<softirq>: tempo de interrupções por *software*;

<steal>: "tempo roubado", ou seja, tempo utilizado na execução de outros sistemas operacionais num ambiente virtualizado;

<guest>: tempo utilizado executando uma CPU virtual para outro sistema operacional sob controle do núcleo do sistema operacional Linux.

Para medir a utilização do processador o arquivo /proc/stat é lido periodicamente segundo um período de amostragem.

Para medir a razão entre o tempo inativo do processador e o período de amostragem é necessário considerar que os contadores no arquivo /proc/stat estão sendo sempre incrementados. Portanto é necessário medir as diferenças. Seja S(t) a soma dos contadores da linha '**cpu**', no instante de tempo t, e I(t) o valor da coluna '*idle*'. As diferenças S(t)-S(t-1) e I(t)-I(t-1), correspondem aos incrementos de S(t) e I(t), respectivamente durante o período de amostragem. Então a razão entre o tempo inativo do processador e o período de amostragem equivale à razão em (12). E a variável y(t), ou seja, a utilização do processador é o complemento dessa razão, i(t), como apresentado em (13).

$$i(t) = \frac{I(t) - I(t-1)}{S(t) - S(t-1)} \quad (12)$$

$$\gamma(t) = 1 - i(t) \quad (13)$$

Onde:

i(t) = tempo de CPU inativa;

γ(t) = tempo de CPU ativa;

S(t) = soma dos contares na linha '**cpu**';

I(t) = contador referente ao tempo ocioso de CPU, '*idle*'.

3.3 FREQUÊNCIA DO PROCESSADOR

A frequência do processador é obtida a partir da leitura da interface *cpufreq* a partir do diretório: /sys/devices/system/cpu/cpuX/cpufreq/scaling_cur_freq. O conjunto de estados de energia do processador Intel® Core™ i5-2410M, é: $F = \{2{,}301GHz, 2{,}3GHz, 1{,}8GHz, 1{,}6GHz, 1{,}4GHz, 1{,}2GHz, 1{,}0GHz, 0{,}8GHz\}$. A partir de chamadas do *software* GDE, o valor de frequência utilizado por cada núcleo é armazenado em um vetor.

3.4 POTÊNCIA DA BATERIA

No sistema operacional Linux é possível acessar informações sobre a corrente e a tensão fornecidas pela bateria ao sistema computacional. Ao acessar o arquivo: /proc/acpi/battery/BAT0/state, são obtidas as seguintes informações:

present: yes
capacity state: ok
charging state: discharging
present rate: 1229 mW
remaining capacity: 1140 mWh
present voltage: 14302 mV

Como dito na seção 2.1, estimou-se um modelo de potência, e para tanto, ter acesso às informações de corrente e tensão são fundamentais.

4 ESTIMAÇÃO RECURSIVA DE UM MODELO DE POTÊNCIA PARA UM SISTEMA COMPUTACIONAL

A potência de chaveamento de um processador é proporcional à atividade de chaveamento A, onde $0 \leq A \leq 1$, à carga capacitiva C, ao quadrado da tensão v e à frequência f, ou seja, a relação da potência é do tipo : $P_{chaveamento} = ACv^2f$. Sabendo que a frequência de um processador é proporcional à tensão e que a constante adimensional A pode ser aproximada pela utilização do processador, a potência pode ser simplificada para $P_{chaveamento} = \gamma f^3$, onde γ é a utilização e f a frequência de cada núcleo do processador.

Para estimar um modelo de potência para o processador, utilizou-se como variáveis de entrada 'u_i' os produtos $\gamma_i f_i^3$ de cada núcleo. A saída do sistema é a potência fornecida pela bateria ao sistema computacional 'y'. Como apresentado no diagrama de blocos apresentado na Fig. 1.

4.1 IMPLEMENTAÇÃO DE UM *SOFTWARE* PARA ESTIMAÇÃO DO MODELO DE POTÊNCIA DO SISTEMA COMPUTACIONAL

O modelo escolhido possui uma saída $y(t)$ que é a potência consumida pelo notebook, e quatro entradas, $u_i(t) = \gamma_i f_i^3$, com i = 0,1,2 e 3.

Com a utilização e frequência de cada núcleo lógico, compõe-se a matriz $\varphi(t)$, que é matriz formada por todas as entradas, e pelas relações de (7) à (11) estima-se os parâmetros da matriz θ, como definida na seção 2.1.

$$y(t) = \varphi^T(t)\theta \tag{14}$$

A potência fornecida pela bateria ao sistema é uma função da utilização e frequência de cada núcleo do processador, o que pode ser visto em (15).

$$\hat{y}(t) = \sum_{i=0}^{3} b_i \gamma_i f_i^3 + b_4, \qquad i = 0, \ldots, 3 \tag{15}$$

onde o índice ‘i’ se refere a cada um dos núcleos do processador.

A obtenção e cálculo de todos os parâmetros utilizados para a modelagem é realizada pelo *software* livre para Gerenciamento Dinâmico de Energia (GDE).

Na Fig. 3 é apresentado o fluxograma para a implementação do *software* GDE.

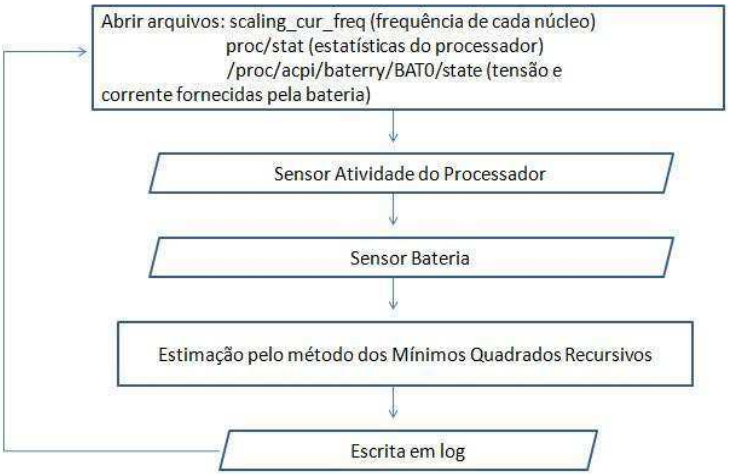

Figura 3: Fluxograma para a implementação do *software* GDE

4.2 CARGAS DE TRABALHO

Para uma correta estimação dos parâmetros com o método dos mínimos quadrados recursivos, é necessário que as variáveis de entrada estejam variando. Para auxiliar nessa tarefa optou-se pela utilização de uma carga de trabalho, o *Ebizzy*. E o escalonamento das frequências do processador segundo uma variável aleatória discreta com distribuição de probabilidade uniforme.

O *Ebizzy* foi projetado para gerar uma carga de trabalho representando a de uma aplicação de *web* comum. É multi *treaded*, aloca e desaloca memória constantemente [15].

5 RESULTADOS OBTIDOS

Para obter os parâmetros do modelo foram realizados testes, com a carga de trabalho *Ebizzy*, período de amostragem de 100 ms, com duração de 50 s.

5.1 ESTIMAÇÃO DO MODELO COM UM NÚCLEO LÓGICO 100% UTILIZADO

Para a realização desse teste, definiu-se o período de amostragem de 100 ms. E na carga de trabalho *Ebizzy* definiu-se que um núcleo deveria estar atuando com 100% de sua capacidade. Os demais núcleos também apresentam atividade, devido às atividades do próprio sistema operacional.

Na Fig. 4 estão apresentados os gráficos com os parâmetros da variável θ, a potência fornecida pela bateria ao sistema, medida no GDE, e a potência estimada pelos métodos dos *mínimos quadrados recursivos*.

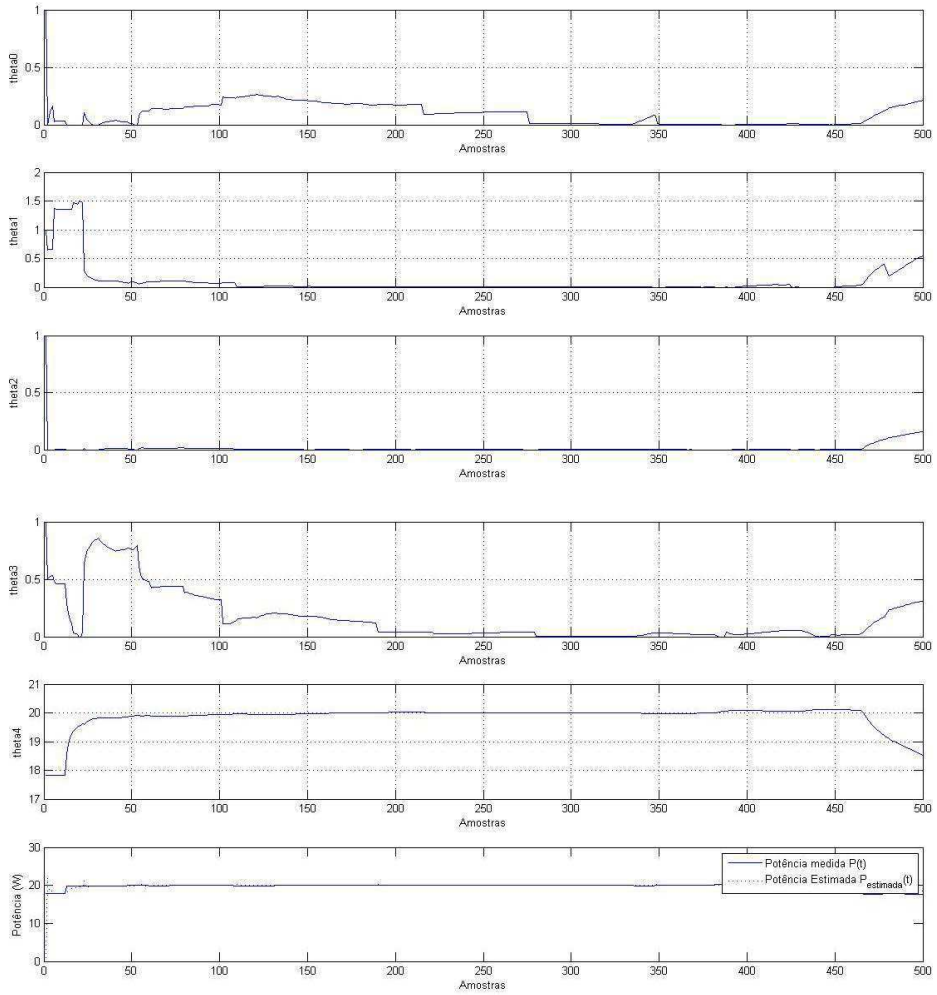

Figura 4: Gráfico dos Parâmetros estimados para utilização de um núcleo lógico do processador

Na Fig. 5 são apresentadas as variáveis φ. Que correspondem às entradas do modelo. É possível verificar pela Fig. 5 que a variável φ_2 possui valores altos, como foi definido no script *Ebizzy* a ocupação de um único núcleo lógico, é possível afirmar que a CPU2 manteve-se ocupada ao longo dos 45 segundos em que o script esteve rodando. Nas últimas 50 amostras verifica-se a queda na potência estimada e medida, devido ao término da execução da aplicação *Ebizzy*.

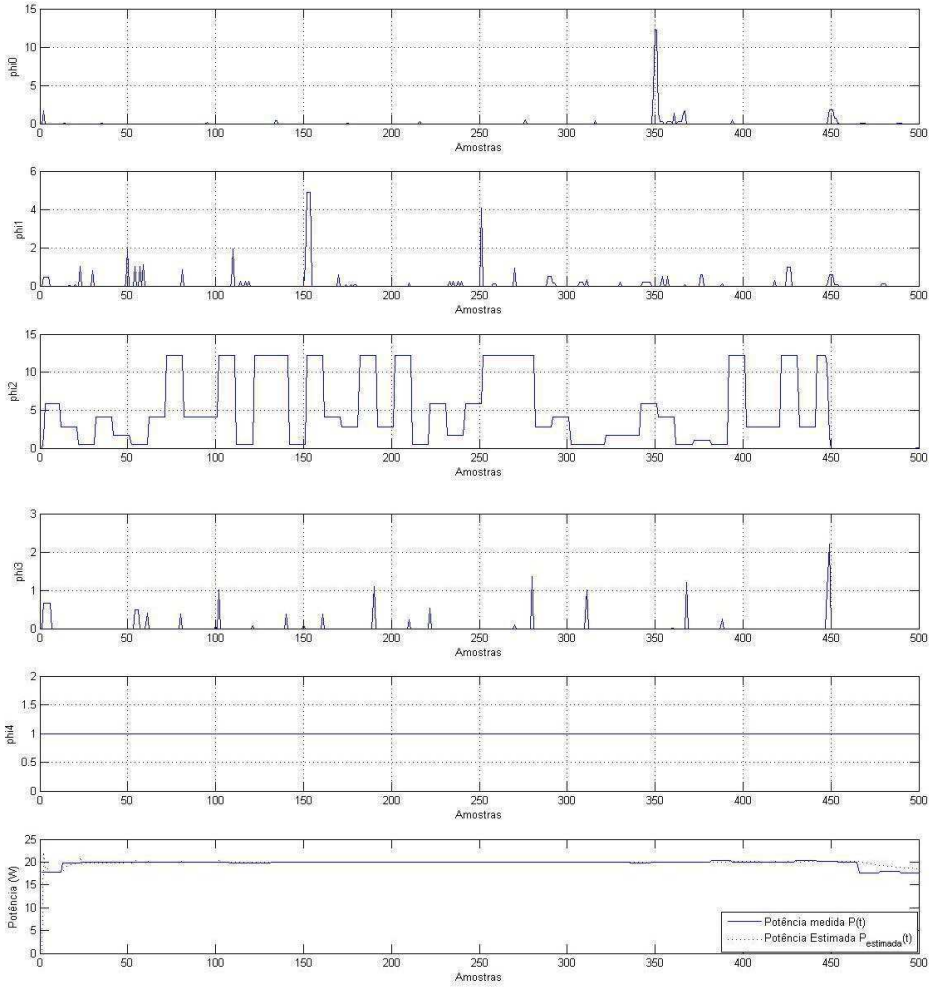

Figura 5: Variáveis de entrada do modelo para utilização de um núcleo lógico do processador

Através da observação da Fig. 6 é possível perceber que as frequências de cada núcleo variam entre seus valores máximos e mínimos de forma aleatória. Esse foi um dos cuidados tomados para fornecer entradas sempre diferentes e assim obter um modelo com maior qualidade.

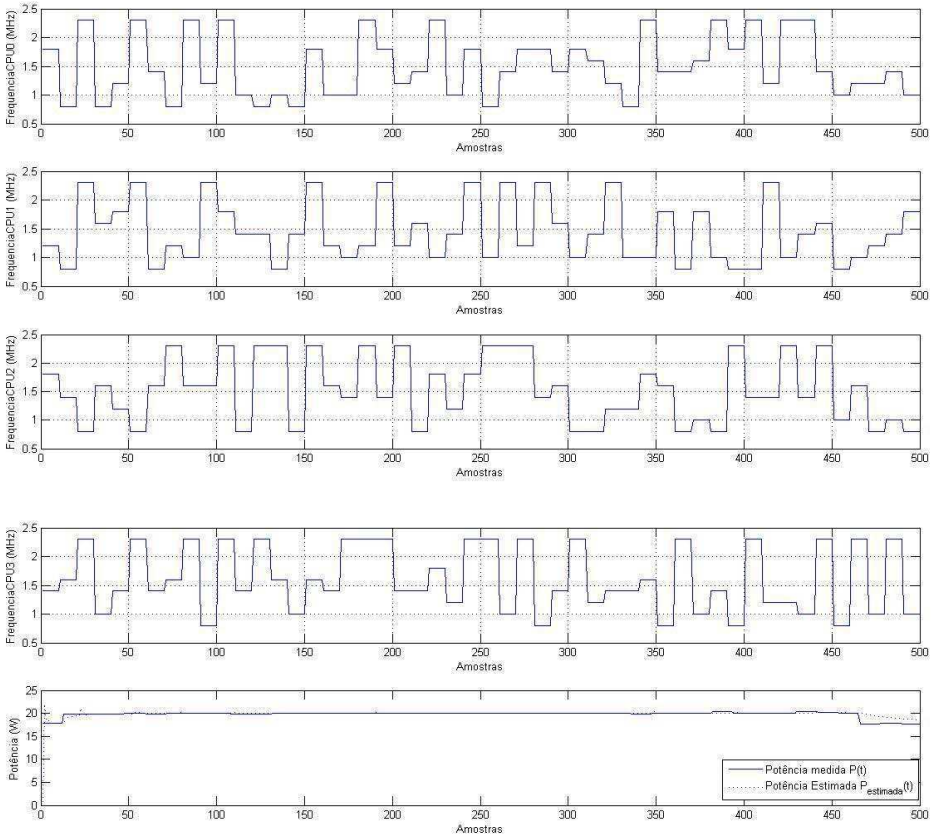

Figura 6: Gráfico com os estados de energia para cada um dos núcleos do processador, potência média e potência estimada, quando na utilização de um núcleo lógico do processador

Na Fig. 7 a amostra de número 100 encontra-se destacada por uma seta, os valores dos parâmetros estimados pelo método dos mínimos quadrados nesse instante são:

$\boldsymbol{\theta_0}$	$\boldsymbol{\theta_1}$	$\boldsymbol{\theta_2}$	$\boldsymbol{\theta_3}$	$\boldsymbol{\theta_4}$	$\boldsymbol{y}$	$\boldsymbol{\hat{y}}$
0,173128	0,064511	0,010657	0,322610	19,938612	20,036680	19,997759

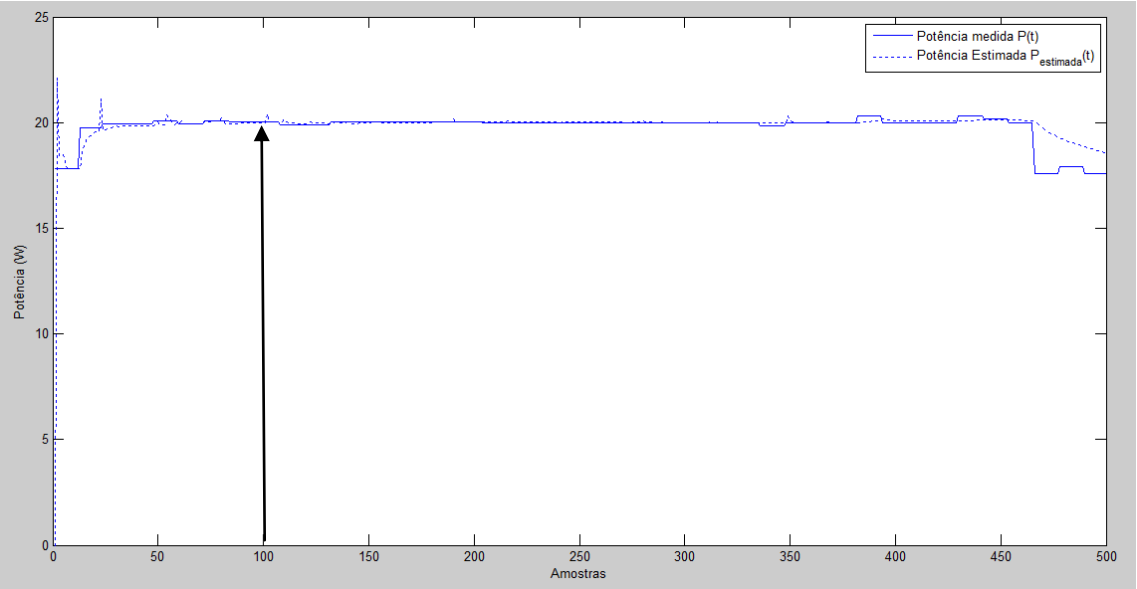

Figura 7: Potência medida e potência estimada quando um núcleo lógico 100% utilizado

5.2 ESTIMAÇÃO DO MODELO COM DOIS NÚCLEOS LÓGICOS 100% UTILIZADOS

Para a realização desse teste, definiu-se o período de amostragem de 100 ms. E na carga de trabalho *Ebizzy* definiu-se que duas threads estariam ativas, assim ocupando dois núcleos com 100% de utilização cada. Os demais núcleos também apresentam atividade, devido às atividades do próprio sistema computacional.

Na Fig. 8 estão apresentados os gráficos que mostram os parâmetros da variável θ, a potência fornecida pela bateria ao sistema, medida no GDE, e a potência estimada pelo método dos *mínimos quadrados recursivos*.

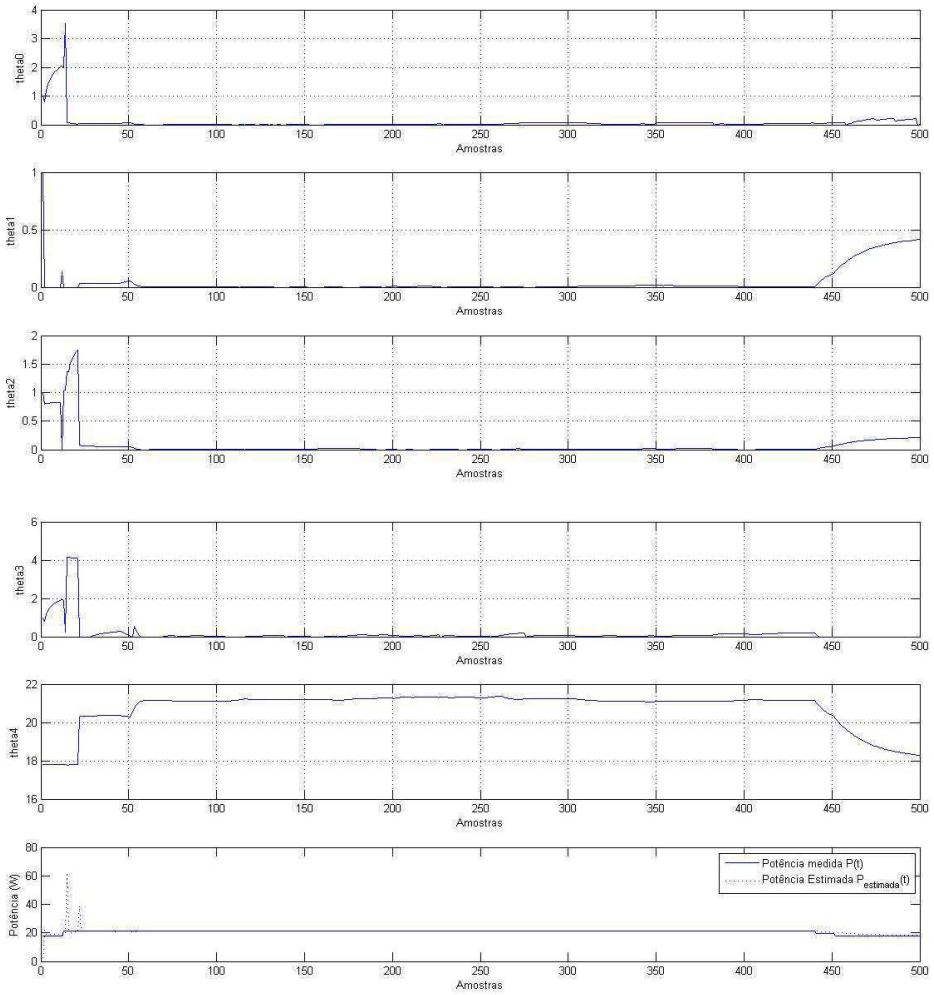

Figura 8: Gráfico dos Parâmetros estimados para utilização de dois núcleos lógicos do processador

Na Fig. 9 são apresentadas as variáveis φ. Que correspondem as entradas do modelo. É possível verificar pela Fig. 8 que a variável φ_1 e φ_2 apresentam valores altos, como foi definido no script *Ebizzy* a ocupação de um dois núcleos lógicos, é possível afirma que a CPU1 e CPU2 mantiveram-se ocupada ao longo dos 45 segundos em que o script esteve rodando. Nas últimas 50 amostras verifica-se a queda na potência estimada e medida, devido a saída da carga gerada pelo script *Ebizzy*.

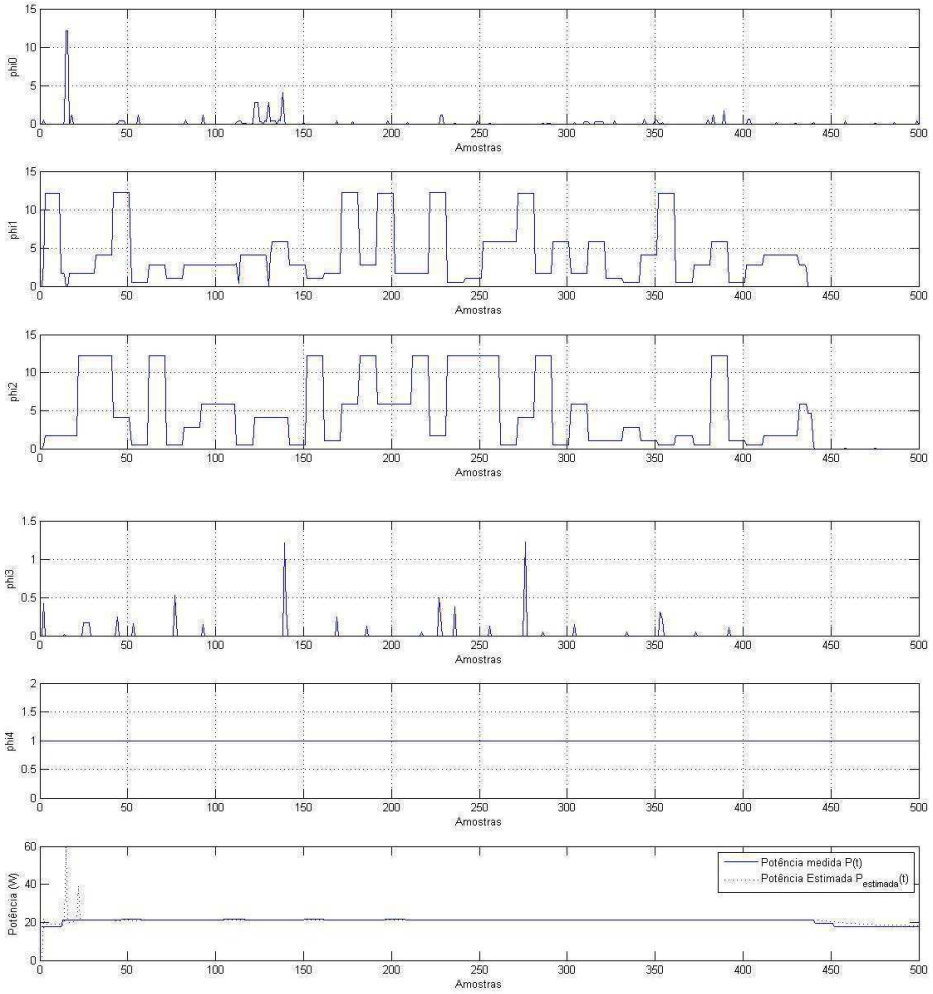

Figura 9: Variáveis de entrada do modelo para utilização de dois núcleos lógicos do processador

Através da observação da Fig. 10 é possível perceber que as frequências de cada núcleo variam entre seus valores máximos e mínimos de forma aleatória. Esse foi um dos cuidados tomados para fornecer entradas sempre diferentes e assim obter um modelo com maior qualidade.

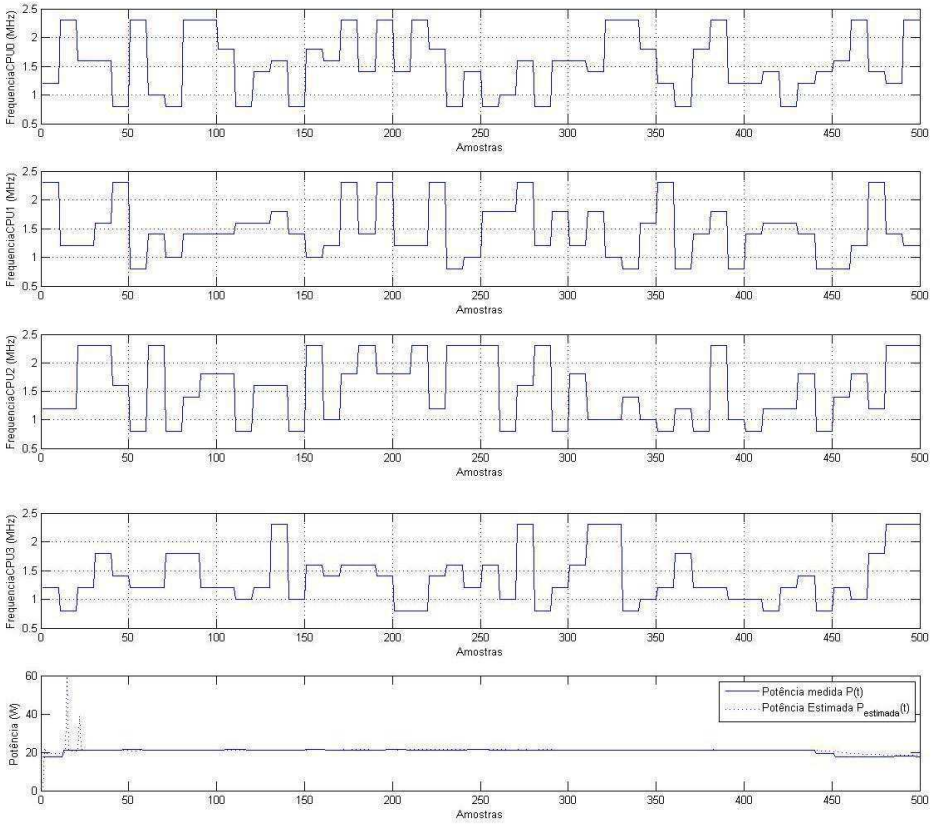

Figura 10: Gráfico com os estados de energia para cada um dos núcleos do processador, potência média e potência estimada, quando na utilização dois núcleos lógicos do processador

Na Fig. 11 a amostra de número 100 encontra-se destacada por uma seta, os valores dos parâmetros estimados pelo método dos mínimos quadrados nesse instante são:

$\boldsymbol{\theta_0}$	$\boldsymbol{\theta_1}$	$\boldsymbol{\theta_2}$	$\boldsymbol{\theta_3}$	$\boldsymbol{\theta_4}$	$\boldsymbol{y}$	$\boldsymbol{\hat{y}}$
0,004166	0,004419	0,005742	0,025439	21,091932	21,157617	21,137003

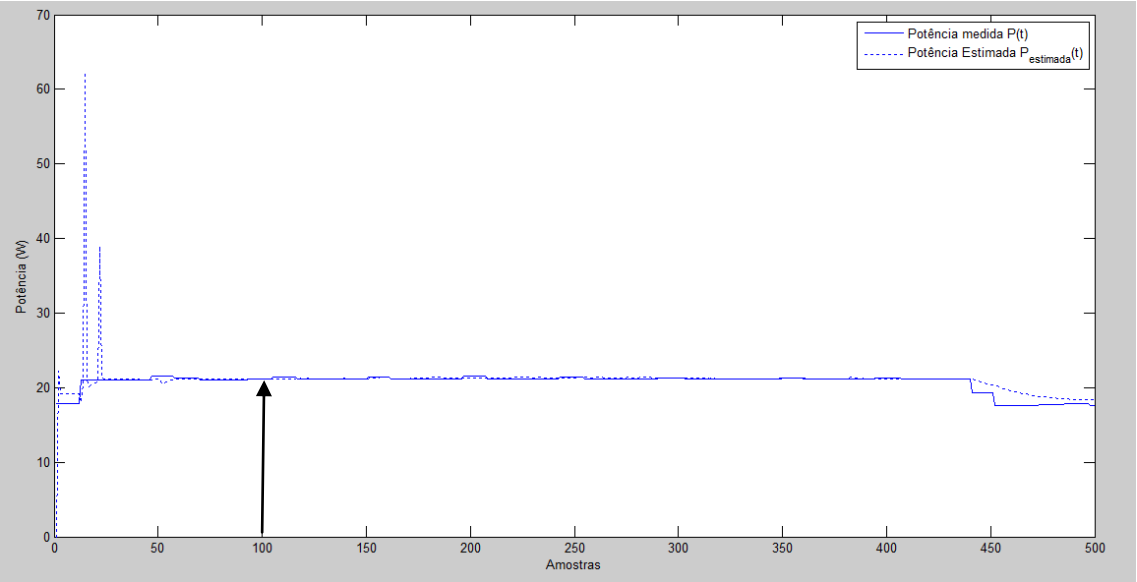

Figura 11: Potência medida e potência estimada quando dois núcleos lógicos 100% utilizados

5.3 ESTIMAÇÃO DO MODELO COM TRÊS NÚCLEOS LÓGICOS 100% UTILIZADOS

Para realização desse teste, definiu-se período de amostragem de 100 ms. E na carga de trabalho *Ebizzy* definiu-se que três núcleos deveriam estar atuando com 100% de sua capacidade. Os demais núcleos também apresentam atividade, devido às necessidades do próprio sistema operacional.

Na Fig. 12 estão apresentados os gráficos que mostram os parâmetros da variável θ, a potência fornecida pela bateria ao sistema, calculada no GDE, e a potência estimada pelos método dos *mínimos quadrados recursivos*.

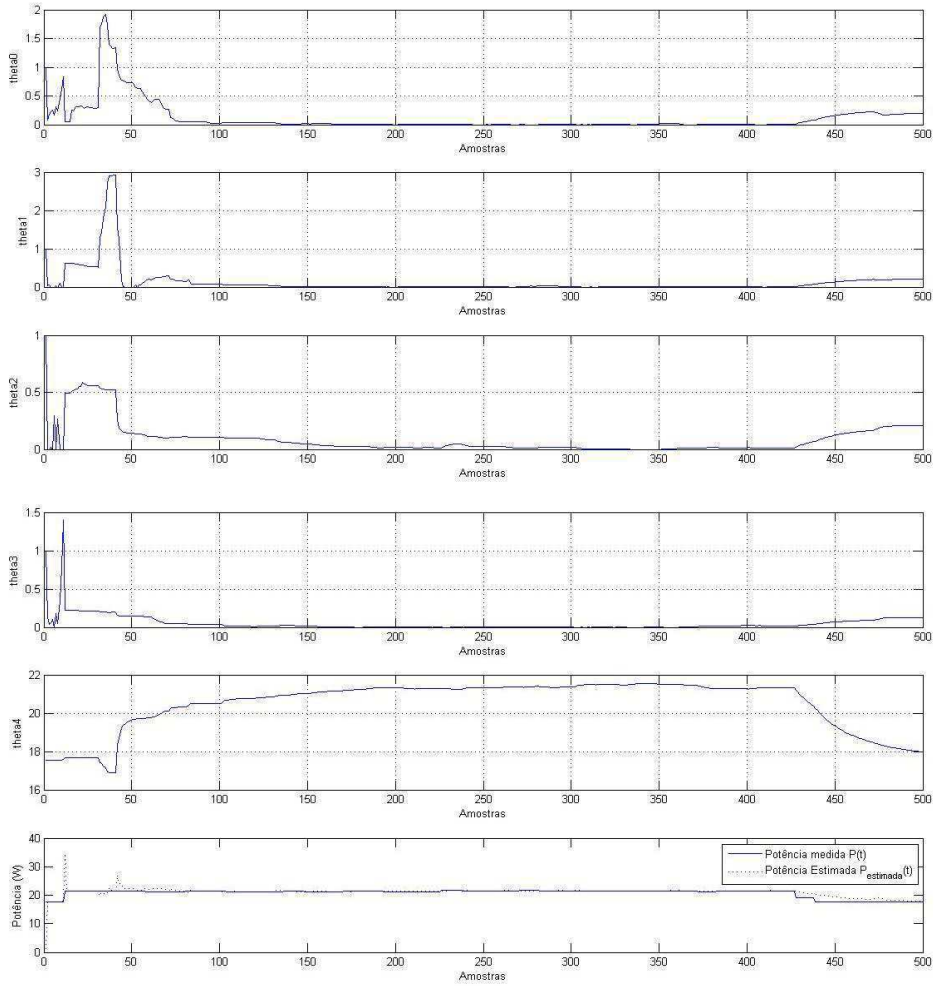

Figura 12: Gráfico dos Parâmetros estimados para utilização de três núcleos lógicos do processador

Na fig. 13 são apresentadas as variáveis φ. Que correspondem as entradas do modelo. É possível verificar pela Fig. 13 que a variável φ_1, φ_2 *e* $\varphi_4$apresentam valores altos, como foi definido no script *Ebizzy* a ocupação de um três núcleos lógicos, é possível afirma que a CPU1, CPU3 e CPU4 mantiveram-se ocupadas ao longo dos 45 segundos em que o script esteve rodando. Entretanto, se observa maior utilização do núcleo, no que não foi definido para operar com o total de sua capacidade. Nas últimas 50 amostras verifica-se a queda na potência estimada e medida, devido a saída da carga gerada pelo script *Ebizzy*.

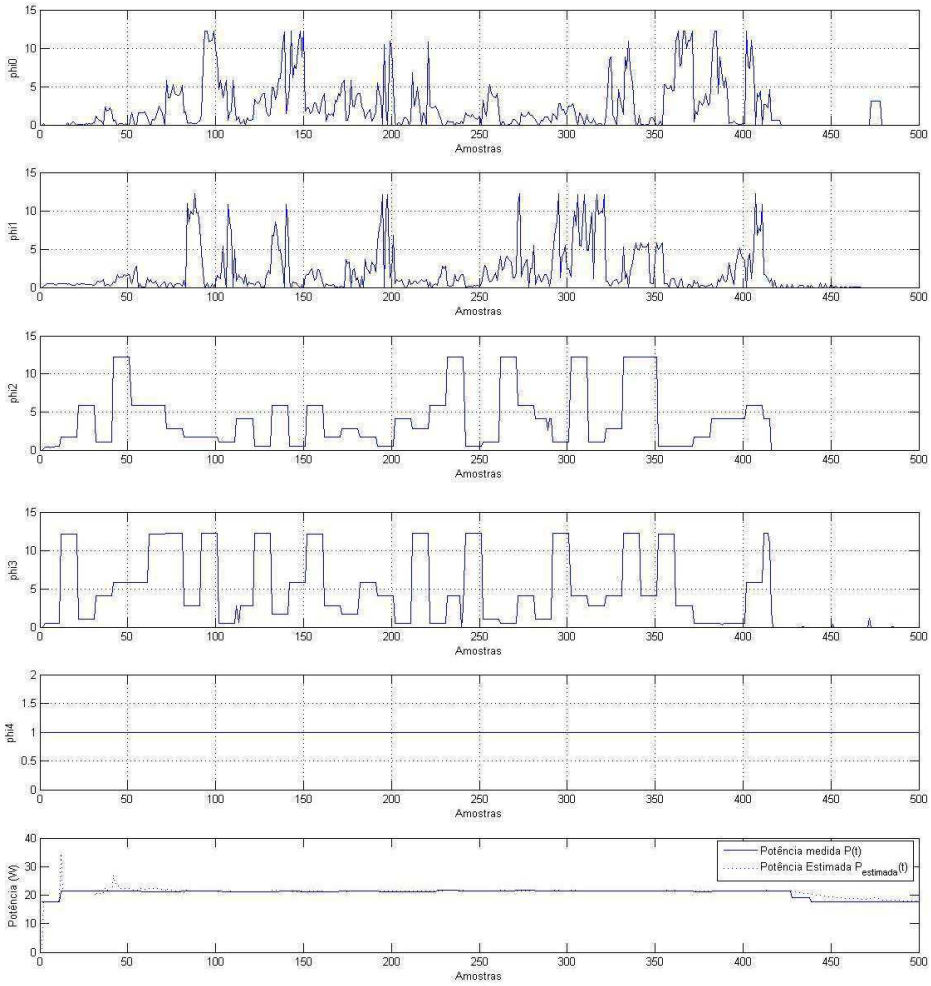

Figura 13: Variáveis de entrada do modelo para utilização de três núcleos lógicos do processador

Através da observação da Fig. 14 é possível perceber que as frequências de cada núcleo variam entre seus valores máximos e mínimos de forma aleatória. Esse foi um dos cuidados tomados para fornecer entradas sempre diferentes e assim obter um modelo com maior qualidade.

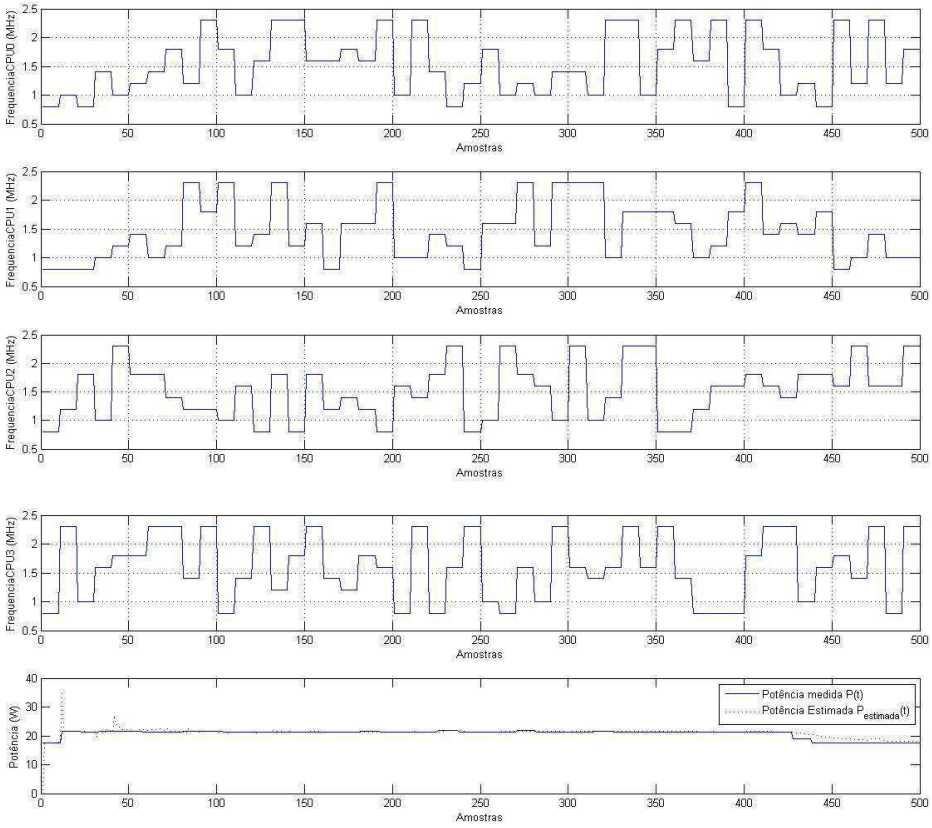

Figura 14: Gráfico com os estados de energia para cada um dos núcleos do processador, potência média e potência estimada, quando na utilização de três núcleos lógicos do processador

Na Fig. 15 a amostra de número 100 encontra-se destacada por uma seta, os valores dos parâmetros estimados pelo método dos mínimos quadrados nesse instante são:

$\boldsymbol{\theta_0}$	$\boldsymbol{\theta_1}$	$\boldsymbol{\theta_2}$	$\boldsymbol{\theta_3}$	$\boldsymbol{\theta_4}$	$\boldsymbol{y}$	$\boldsymbol{\hat{y}}$
0,027235	0,066403	0,105532	0,037476	20,507980	21,458209	21,456991

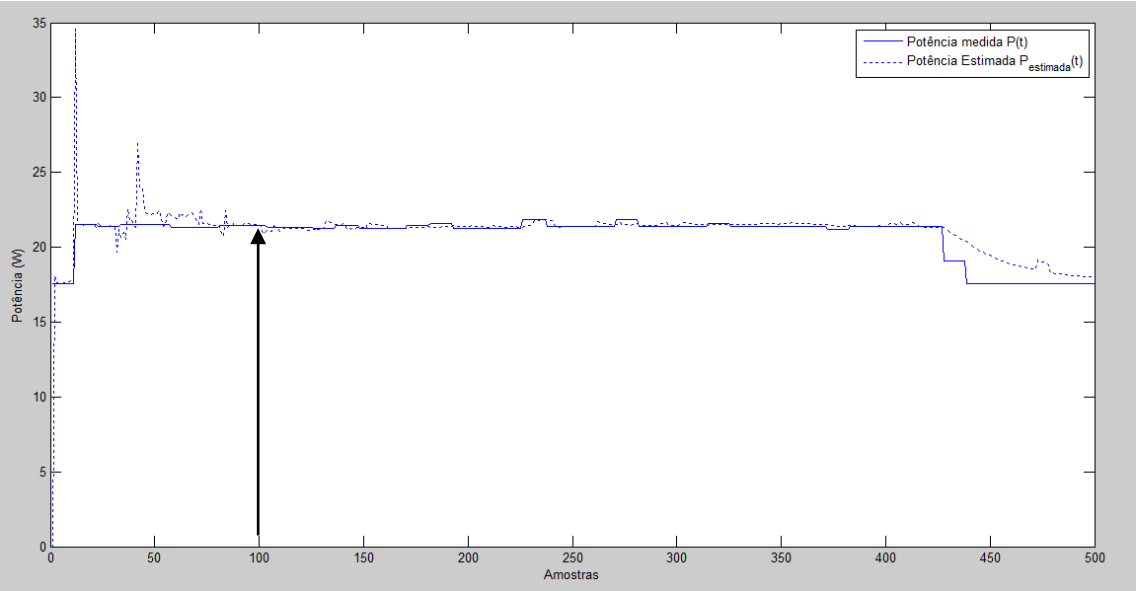

Figura 15: Potência medida e potência estimada quando três núcleos lógicos 100% utilizados

5.4 ESTIMAÇÃO DO MODELO COM QUATRO NÚCLEOS LÓGICOS 100% UTILIZADOS

Para realização desse teste, definiu-se período de amostragem de 100 ms. E na carga de trabalho *Ebizzy* definiu-se os quatro núcleos deveriam estar atuando com 100% de sua capacidade.

Na Fig. 16 estão apresentados os gráficos que mostram os parâmetros da variável θ, a potência fornecida pela bateria ao sistema, calculada no GDE, e a potência estimada pelos método dos *mínimos quadrados recursivos*.

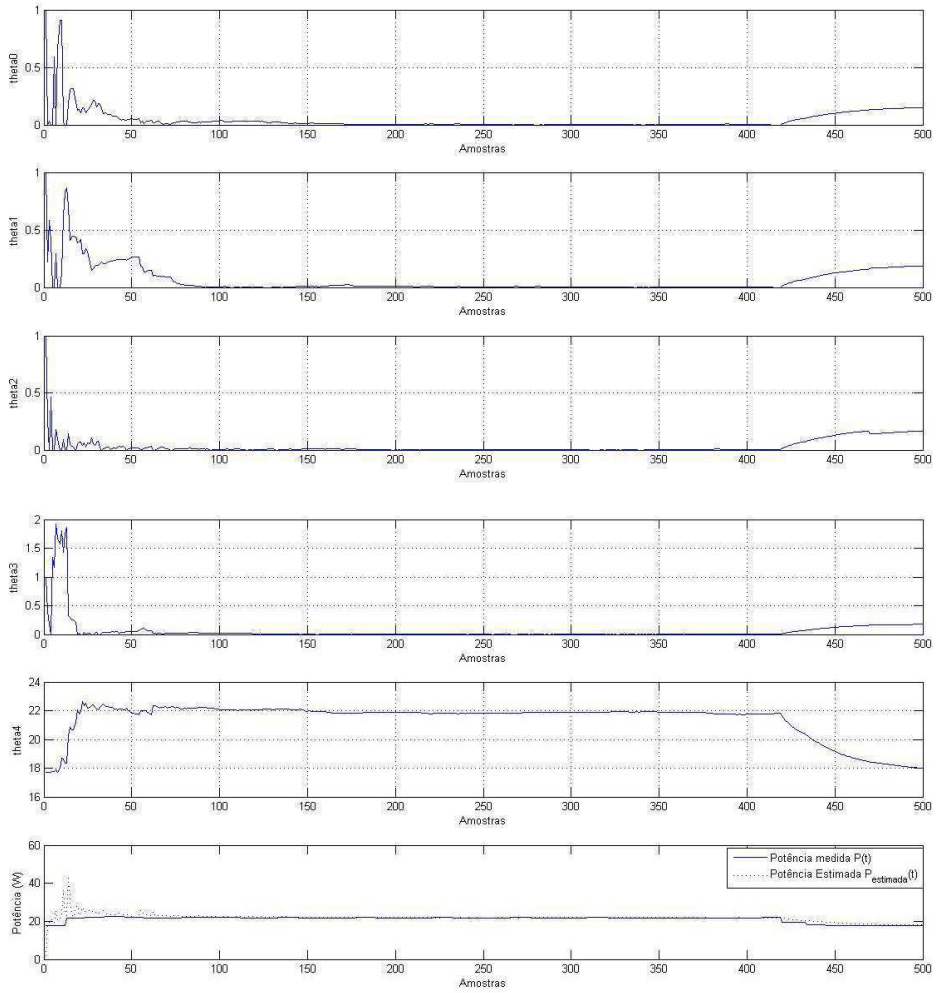

Figura 16: Gráfico dos Parâmetros estimados para utilização dos quatro núcleos lógicos do processador

Na Fig. 14 são apresentadas as variáveis φ. Que correspondem as entradas do modelo. É possível verificar pela Fig. 17 que as variáveis $\varphi_1, \varphi_2, \varphi_3 \, e \, \varphi_4$ apresentam valores altos, pois foi definido no script *Ebizzy* a ocupação de todos os núcleos lógicos. É possível que todas as CPU's mantiveram-se ocupadas ao longo dos 45 segundos em que o script esteve rodando. Nas últimas 50 amostras verifica-se a queda na potência estimada e medida, devido à saída da carga gerada pelo script *Ebizzy*.

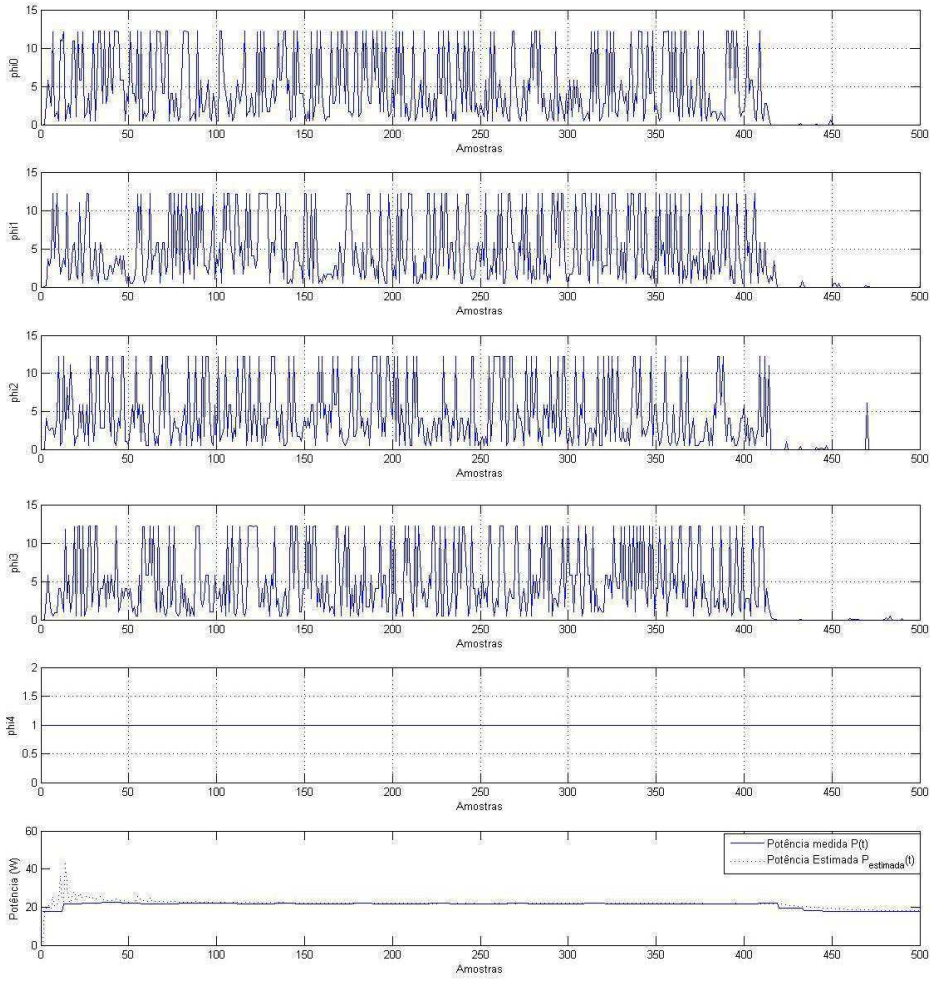

Figura 17: Variáveis de entrada do modelo para utilização dos quatro núcleos lógicos do processador

Através da observação da Fig. 18 é possível perceber que as frequências de cada núcleo variam entre seus valores máximos e mínimos de forma aleatória. Esse foi um dos cuidados tomados para fornecer entradas sempre diferentes e assim obter um modelo com maior qualidade.

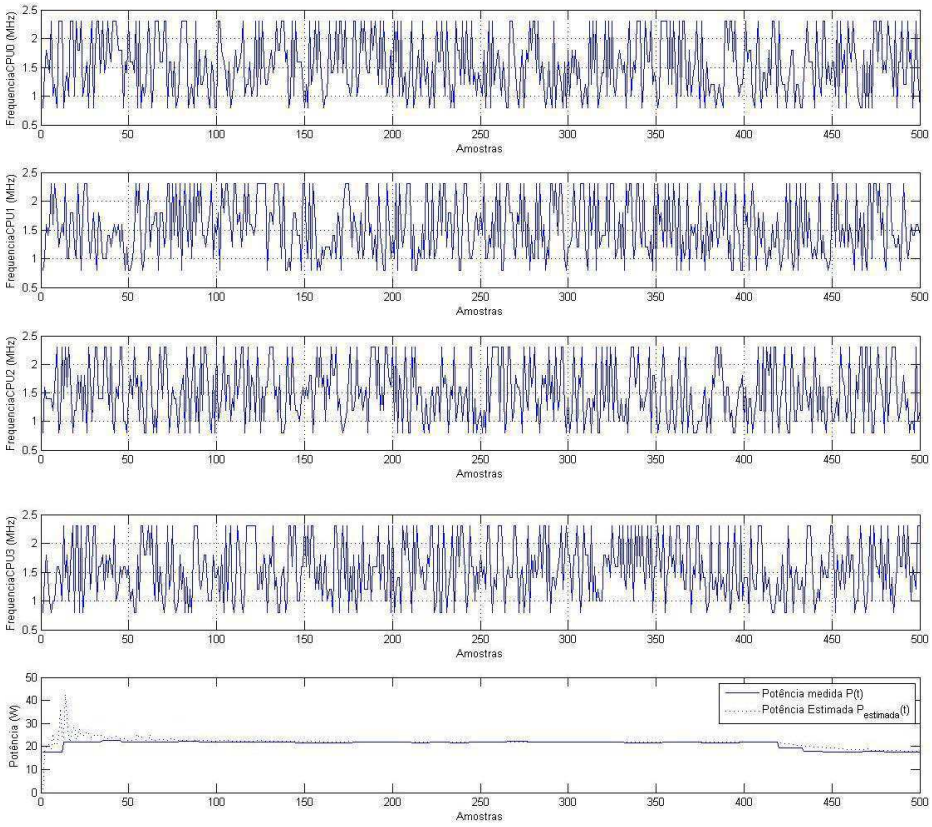

Figura 18: Gráfico com os estados de energia para cada um dos núcleos do processador, potência média e potência estimada, quando na utilização dos quatro núcleos lógicos do processador

Para a amostra 100, destacada na Fig. 19 pela seta, os parâmetros estimados pelo modelo são:

$\boldsymbol{\theta_0}$	$\boldsymbol{\theta_1}$	$\boldsymbol{\theta_2}$	$\boldsymbol{\theta_3}$	$\boldsymbol{\theta_4}$	$\boldsymbol{y}$	$\boldsymbol{\hat{y}}$
0,034691	0,005118	0,004360	0,024182	22,074043	21,920202	22,201550

Inicialmente o modelo tem dificuldade de acompanhar os valores de potência medida, devido à entrada da carga no sistema, após 10 s, ou 100 amostras, percebe-se que a potência medida e a potência estimada são praticamente iguais.

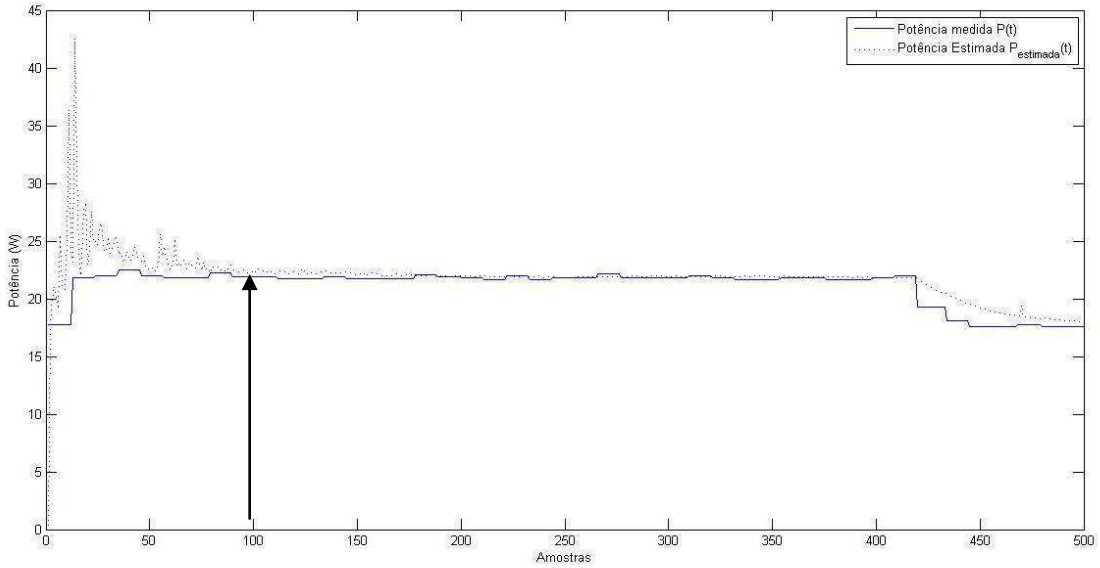

Figura 19: Potência medida e potência estimada quando quatro núcleos lógicos 100% utilizados

6 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho, foi realizada a estimação do modelo de potência do sistema computacional considerando-se a potência que o processador acrescenta ao sistema. Para tanto, foram realizadas investigações sobre gerenciamento dinâmico de energia de sistemas computacionais, e estimação de modelos por meio da técnica de Mínimos Quadrados Recursivos. Por fim, a técnica proposta foi implementada em um *software* de gerenciamento de energia, e foram avaliados os resultados experimentais

Outros dispositivos podem ser utilizados na estimação do modelo de potência, como brilho do LCD, uso de memória, utilização de dispositivos de redes, etc. O modelo apresentado pode ser expandido de forma a receber variáveis relacionadas aos dispositivos citados e fornecer uma estimativa de potência.

APÊNDICE A: CÓDIGO EM LINGUAGEM C DO *SOFTWARE* LIVRE GDE

```
/*
 * gdebat.c  Gerenciamento Dinamico de Energia
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 *
 * Saulo O. D. Luiz
 * Embedded / UFCG
 * saulo@ee.ufcg.edu.br
 * Janeiro, 2011
 *
 * Implementação da Política Proposta em TCC:
 * Bruna M. J. Cruz
 * Embedded / UFCG
 * bruna.cruz@ee.ufcg.edu.br
 * Agosto, 2012 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <fcntl.h>

#include "gdelib.h"
#include "modelodosistema.h"

#define NMODULADOR 2
```

```
#define USEC_PER_SEC 1000000l

#define timev(var) ((long)(var.tv_sec*USEC_PER_SEC + var.tv_usec))

struct timeval t0, t1, t2;
long periodoModulador;                  /* Período para o modulador */
long periodo = 1000000l;        /* Periodo para a politica de 1 segundo */
long tPoliticaTotal, tDormindo, tErro, t1Anterior, delta, tPoliticaAtivaTotal = 0;
float tPoliticaMedio, tPoliticaAtivaMedio, percentualtPoliticaAtivaMedio, tau;

float distribuicao[NFREQ] = { 0.125, 0.125, 0.125, 0.125, 0.125, 0.125, 0.125, 0.125 };
float distribuicao2[2] = { 0.5, 0.5 };

int fps[NNUCLEOS];                      /* descritor do arquivo do estado de energia */

unsigned char buflog[1000];     /* buffer para registrar o estado do sistema */
int fpslog;                     /* arquivo para registrar o estado do sistema */
FILE *fpsStat = NULL;                   /* descritor do arquivo de estatísticas */
int statuslog;                          /* returna o status de chamadas do sistema */
int tamLinhaBuflog;

//Estrutura periodic_info, e funções make_periodic e wait_period para tornar a política de energia periódica através de
timerfd
struct periodic_info
{
        int timer_fd;
        unsigned long long wakeups_missed;
};

struct periodic_info info;
static int make_periodic(unsigned int period, struct periodic_info *info)
{
        int ret;
        unsigned int ns;
        unsigned int sec;
        int fd;
        struct itimerspec itval;

        /* Create the timer */
        fd = timerfd_create (CLOCK_MONOTONIC, 0);
        info->wakeups_missed = 0;
        info->timer_fd = fd;
        if (fd == -1)
                return fd;

        /* Make the timer periodic */
        sec = period/1000000;
        ns = (period - (sec * 1000000)) * 1000;
        itval.it_interval.tv_sec = sec;
        itval.it_interval.tv_nsec = ns;
        itval.it_value.tv_sec = sec;
```

```
		itval.it_value.tv_nsec = ns;
		ret = timerfd_settime (fd, 0, &itval, NULL);
		return ret;
	}
	static int wait_period(struct periodic_info *info)
	{
		unsigned long long missed;
		int ret;

		/* Wait for the next timer event. If we have missed any the
		   number is written to "missed" */
		ret = read (info->timer_fd, &missed, sizeof (missed));
		if (ret == -1)
		{
			perror ("read timer");
			return ret;
		}

		/* "missed" should always be >= 1, but just to be sure, check it is not 0 anyway */
		if (missed > 0)
			info->wakeups_missed += (missed - 1);

		return ret;
	}

	int
	main (int argc, char *argv[])
	{

	  if (argc != 6)
	    {
	      printf ("Uso %s limiteInterior referencia limiteSuperior periodoAmostragem(us) tempo\n", argv[0]);
	      return 1;
	    }

	  sscanf (argv[1], "%lf", &Li_controleL[0]);
	  Li_controleL[1] = Li_controleL[0];
	  sscanf (argv[2], "%lf", &uc_controleL[0]);
	  uc_controleL[1] = uc_controleL[0];

	  sscanf (argv[3], "%lf", &Ls_controleL[0]);
	  Ls_controleL[1] = Ls_controleL[0];

	    for (j = 0; j < NNUCLEOS; j++)
			{
			  sinal_i_filtrado[j] = 1 - uc[nucleo]; //O sinal de inativa percental já é inicializado com 1 -
referênciaDeAtivoPercentual
			}

	  sscanf (argv[4], "%ld", &periodo);
	  sscanf (argv[5], "%d", &n);
```

```
/* Abrindo o device do arquivo do estado de energia */
fps[0] =
  open ("/sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed", O_RDWR);
if (fps[0] < 0)
  {
    perror
            ("Abertura de /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed falhou.");
    exit (1);
  }

fps[1] =
  open ("/sys/devices/system/cpu/cpu1/cpufreq/scaling_setspeed", O_RDWR);
if (fps[1] < 0)
  {
    perror
            ("Abertura de /sys/devices/system/cpu/cpu1/cpufreq/scaling_setspeed falhou.");
    exit (1);
  }
fps[2] =
  open ("/sys/devices/system/cpu/cpu2/cpufreq/scaling_setspeed", O_RDWR);
if (fps[2] < 0)
  {
    perror
            ("Abertura de /sys/devices/system/cpu/cpu2/cpufreq/scaling_setspeed falhou.");
    exit (1);
  }
fps[3] =
  open ("/sys/devices/system/cpu/cpu3/cpufreq/scaling_setspeed", O_RDWR);
if (fps[3] < 0)
  {
    perror
            ("Abertura de /sys/devices/system/cpu/cpu3/cpufreq/scaling_setspeed falhou.");
    exit (1);
  }

/* Abrindo arquivo para registrar o estado do sistema */
fpslog = open ("./log.txt", O_RDWR);
if (fpslog < 0)
  {
    perror ("Abertura de log.txt falhou.");
    exit (1);
  }

     /* Abrindo arquivo de estatisticas do processador */
     fpsStat = fopen("/proc/stat", "r");
     if (!fpsStat) {
            perror("Abertura de /proc/stat falhou.");
            exit(1);
     }
```

```
  /*Configurando a semente do gerador de numeros aleatorios com a hora do dia */
  srand ((unsigned int) time (NULL));

  /*Inicializando o vetor de penalidade de desempenho esperado para cada estado do processador */
  for (i = 0; i < NFREQ; i++)
    {
      l_S[i] = frequencia_S[0] / frequencia_S[i];
//      printf("l_S[%d] = %f\n", i, l_S[i]);
    }
 inicializaMatrizIdentidade (&matrizIdentidade[0][0], NU);

  i = 0;                                //Contador para as condições inicias
  indiceModulador = 0;                          //Contador para o modulador

                make_periodic(periodo, &info);
                gettimeofday(&t0, NULL);

                /******Politica de energia*******/
  while (1)
    {

 #ifdef TESTE2
      i++;
      if (i > n)                        //i funciona aqui tanto como contador para as considções iniciais,
                //quanto como um contador para um período n de teste da política
                {
                fclose(fpsStat);
                break;
                }
 #else
      if (i <= t0RLS)                   //Sendo i contador para as considções iniciais, só é necessário incrementar i durante as
condições iniciais
                i++;
 #endif

  //Implentação de: phi = [u1(t_atual-1), u2(t_atual-1),u3(t_atual-1), u4(t_atual-1)...];

                uRLS[0] = u1;
                uRLS[1] = u2;
                uRLS[2] = u3;
                uRLS[3] = u4;
                uRLS[4] = 1.0;

                  preencher_phi (phi,&uRLS[0], NU);
                  /*Estimativa da próxima saída de acordo com o modelo obtido de mínimos quadrados recursivo */
                  ySimuladoRLS_menos1= ySimuladoRLS;
                  produtoMatrizes (&ySimuladoRLS, phi, theta_t, 1, NU, 1); //Phi transposto * theta

      /*Sensor bateria */
      bateria (&correnteBateria, &capacidadeRestanteBateria, &tensaoBateria);
```

```
            /*Sensor de temperatura do Processador */
                    //temperaturaDoProcessador = temperaturaProcessador ();

            /*Sensor atividade do processador */
            estatisticasProcessador (fpsStat, tempoCPU, tempoCPUinativa);
            for (j = 0; j < NNUCLEOS+1; j++)
                    {
                     if (tempoCPU[j] - tempoCPU_anterior[j] != 0){
                             sinal_i[j] = (float) (tempoCPUinativa[j] - tempoCPUinativa_anterior[j]) /
                                     (float) (tempoCPU[j] - tempoCPU_anterior[j]);
                     }
                     //sinal_i_filtrado[j] = alfa * sinal_i[j] + (1-alfa) * sinal_i_filtrado[j];
           #ifdef DEBUG
            if( j== 4){
                    printf("(tempoCPUinativa[j] - tempoCPUinativa_anterior[j]) = %d", (tempoCPUinativa[j] -
tempoCPUinativa_anterior[j]));
                    printf("(tempoCPU[j] - tempoCPU_anterior[j]) = %d", (tempoCPU[j] - tempoCPU_anterior[j]));
                    }

          #endif
                     //inativaPercentual[j] = sinal_i_filtrado[j];
                     inativaPercentual[j] = sinal_i[j];

                     ativaPercentual[j] = 1.0 - inativaPercentual[j];
                     tempoCPU_anterior[j] = tempoCPU[j];
                     tempoCPUinativa_anterior[j] = tempoCPUinativa[j];

                    }// fim for do nucleo

                    /* Frequencia do Processador */
                    frequencia_processador(&frequenciaCPU0, &frequenciaCPU1, &frequenciaCPU2, &frequenciaCPU3);
                    float_fCPU0 = (float) frequenciaCPU0 * 0.000001;
                    float_fCPU1 = (float) frequenciaCPU1 * 0.000001;
                    float_fCPU2 = (float) frequenciaCPU2 * 0.000001;
                    float_fCPU3 = (float) frequenciaCPU3 * 0.000001;

                    u1 = ativaPercentual[1]*float_fCPU0*float_fCPU0*float_fCPU0;
                    u2 = ativaPercentual[2]*float_fCPU1*float_fCPU1*float_fCPU1;
                    u3 = ativaPercentual[3]*float_fCPU2*float_fCPU2*float_fCPU2;
                    u4 = ativaPercentual[4]*float_fCPU3*float_fCPU3*float_fCPU3;
                    y=correnteBateria*tensaoBateria*0.000001;

                        /*Início de Identificação com mínimos quadrados recursivos - LJUNG, System Identification. pág. 365 */

                        //Implentação de: L_t = P_t_menos_1 * phi / (lambda + phi' * P_t_menos_1 * phi);
                        produtoMatrizes (produto_P_t_menos_1_phi,
                                         &P_t_menos_1[0][0], phi,
                                         NU, NU, 1);
                        produtoMatrizes (&produto_phi_P_t_menos_1_phi,
                                         phi, produto_P_t_menos_1_phi,
                                         1, NU, 1);
```

```
                q1 =
                        1.0 / (lambda + produto_phi_P_t_menos_1_phi);
                produtoMatrizes (L_t, produto_P_t_menos_1_phi,
                                 &q1, NU, 1, 1);

                //Implentação de: P_t = 1/lambda * (eye(n+m,n+m) - L_t * phi')* P_t_menos_1;
                produtoMatrizes (&produto_L_t_phi[0][0], L_t,
                                 phi, NU, 1, NU);
                produtoEscalarMatriz (&produto_L_t_phi[0][0], -1.0,
                                      &produto_L_t_phi[0][0], NU,
                                      NU);
                somaMatrizes (&sub_Identidade_produto_L_t_phi[0][0],
                              &matrizIdentidade[0][0],
                              &produto_L_t_phi[0][0], NU, NU);
                produtoMatrizes (&P_t[0][0],
                                 &sub_Identidade_produto_L_t_phi[0][0],
                                 &P_t_menos_1[0][0], NU, NU,
                                 NU);
                produtoEscalarMatriz (&P_t[0][0], 1.0 / lambda,
                                      &P_t[0][0], NU, NU);

                        //Implentação de: theta_t = theta_t_menos_1 + L_t * (y(t_atual) - phi' * theta_t_menos_1);
                        produtoMatrizes (&produto_phi_theta_t_menos_1,
                                         phi, theta_t_menos_1, 1,
                                         NU, 1);
                erroRLS =
                        y - produto_phi_theta_t_menos_1;
                produtoEscalarMatriz (produto_L_t_erroRLS,
                                      erroRLS, L_t, NU, 1);
                somaMatrizes (theta_t, theta_t_menos_1,
                              produto_L_t_erroRLS, NU, 1);

                //Implentação de: theta_t_menos_1 = theta_t;
                copiaMatrizes (theta_t_menos_1, theta_t,
                               NU, 1);

                //Implentação de: P_t_menos_1 = P_t;
                copiaMatrizes (&P_t_menos_1[0][0], &P_t[0][0],
                               NU, NU);

                //Implentação de: L_t_menos_1 = L_t;
                copiaMatrizes (L_t_menos_1, L_t, NU, 1);

            //Fim de Identificação com mínimos quadrados recursivos - LJUNG, System Identification. pág. 365

for (nucleo = 0; nucleo < NNUCLEOS; nucleo++){
            comandoDoGE[nucleo]= rando (&distribuicao[0], NFREQ);

             //Atuador
             if(comandoDoGE[nucleo] != comandoDoGE_menos1[nucleo]){
```

```
                        mudarEstadoDeEnergiaDoProcessador (fps[nucleo],
                                                parametro_S[comandoDoGE[nucleo]],
                                                tamanhoStringParametro_S
                                                [comandoDoGE[nucleo]]);
                    }

                    comandoDoGE_menos1[nucleo] = comandoDoGE[nucleo];

                }

                /*REGISTRO */
                    tamLinhaBuflog =
                        sprintf (buflog, "%4.3f %4.3f %4.3f %4.3f %f %f %f %f %f %4.3f %4.3f %4.3f %4.3f %4.3f %4.3f %4.3f
%4.3f %4.3f %f %f\n", float_fCPU0, float_fCPU1, float_fCPU2, float_fCPU3, theta_t[0], theta_t[1], theta_t[2],
theta_t[3],theta_t[4], phi[0], phi[1], phi[2], phi[3],phi[4], ativaPercentual[1], ativaPercentual[2], ativaPercentual[3],
ativaPercentual[4], y, ySimuladoRLS);

                    statuslog = write (fpslog, buflog, tamLinhaBuflog);/* Mudando o estado de operacao */
                    if (statuslog != tamLinhaBuflog)
                        perror ("wrote wrong number of bytes at log");

                //Atuador
                    if(comandoDoGE[nucleo] != comandoDoGE_menos1[nucleo]){
                        mudarEstadoDeEnergiaDoProcessador (fps[nucleo],
                                                parametro_S[comandoDoGE[nucleo]],
                                                tamanhoStringParametro_S
                                                [comandoDoGE[nucleo]]);
                    }

                        comandoDoGE_menos1[nucleo] = comandoDoGE[nucleo];

                    } // fim for nucleo

                    /*Espera o inicio do proximo periodo de amostragem */
                    if(wait_period(&info) == -1) //Primeiro tentamos pelo wait_period. Se nao der certo (retorno -1), tentamos pelo
usleep (menos preciso)
                    {
                        usleep (periodo);
                        indiceModulador = 0;
                    }

                  }         //end while política de energia

                gettimeofday (&t2, NULL);
                tPoliticaTotal = timev (t2) - timev (t0);
                tPoliticaMedio = ((float) tPoliticaTotal) / ((float) n);
                printf ("tPoliticaMedio =   %5.4f us\n", tPoliticaMedio);
                tPoliticaAtivaMedio =  ((float) tPoliticaAtivaTotal) / ((float) n);
                printf ("tPoliticaAtivaMedio = %5.4f us\n", tPoliticaAtivaMedio);
                percentualtPoliticaAtivaMedio = ((float)tPoliticaAtivaTotal) / ((float)tPoliticaTotal) * 100.0;
                printf ("percentual tPoliticaAtivaMedio = %5.4f\n", percentualtPoliticaAtivaMedio);
```

```
  return 0;
}
```

APÊNDICE B: BIBLIOTECA DE FUNÇÕES PARA AUXILIAR O GDE

```
/*
 * gdelib.c  Biblioteca de funcoes para auxiliar a politica de
 * gerenciamento de energia
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * Copyright (C) 2002, International Business Machines Corporation
 * All Rights Reserved
 *
 * Saulo O. D. Luiz
 * Embedded / UFCG
 * saulo@dee.ufcg.edu.br
 * Janeiro, 2011
 *
* Implementação da Política Proposta em TCC:
 * Bruna M. J. Cruz
 * Embedded / UFCG
 * bruna.cruz@ee.ufcg.edu.br
 * Agosto, 2012 */
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include "gdelib.h"
```

```
#define NNUCLEOS 4
#define NFREQ 8

int rando(float *distribuicao, int nElementos)
{
	int indice = 0;
	float numeroAleatorio, probabilidadeAcumulada;

    /*Configurando a semente do gerador de numeros aleatorios com a hora do dia*/
    /*srand( (unsigned int)time( NULL ) );*/

	/*Obtendo um numero aleatorio no intervalo [0, 1)*/
	numeroAleatorio = ( (float)rand() / ((float)(RAND_MAX)+(float)(1)) );
	#ifdef DEBUG
	printf("numeroAleatorio = %1.4f\n", numeroAleatorio);
	#endif
	probabilidadeAcumulada = *distribuicao;
	while((numeroAleatorio > probabilidadeAcumulada) && (indice < nElementos - 1)){
		indice++;
		probabilidadeAcumulada += *(distribuicao + indice);
	}
	return indice;
}

void inicializaJanela(int *janela, int nElementosRS, int comprimentoJanela)
{
	int nElementosJanela, b;
	nElementosJanela = nElementosRS * comprimentoJanela;
	for(b = 0; b < nElementosJanela; b++){
		*(janela + b) = 0;
	}
}
void deslocaJanela(int *janela, int nElementosRS, int comprimentoJanela, int bufferRequisicaoAnterior, int estadoDoRS)
{
	int b, inicio;
	inicio = bufferRequisicaoAnterior * comprimentoJanela;
	for(b = comprimentoJanela - 1; b > 0; b--){
	*(janela + inicio + b) = *(janela + inicio + b - 1);
    }
	*(janela + inicio) = estadoDoRS;
}
float interpUniDim(float x, float x1, float x2, float fx1, float fx2){
	return ( (fx2 - fx1) * x + x2*fx1 - x1*fx2 ) / (x2 - x1);
}
int temperaturaProcessador(){
	unsigned char bufTemp[100];	/* buffer para guardar string do arquivo de temperatura*/
	int fpsTemp;			/* descritor do arquivo de temperatura */
	int cnt;			/* para a leitura do arquivo de temperatura */
	unsigned char temporario[100];	/* buffer temporario */
	int temperatura;		/* valor de temperatura*/
```

```
        /* Abrindo arquivo de temperatura do processador */
        fpsTemp = open("/proc/acpi/thermal_zone/TZCR/temperature", O_RDWR);
        if (fpsTemp < 0) {
            perror("Abertura de /proc/acpi/thermal_zone/TZCR/temperature falhou.");
            exit(1);
        }

            /* Lendo arquivo de temperatura do processador */
            cnt = read(fpsTemp, bufTemp, 29);

            /* Identificando o valor de temperatura */
            //temperature:            48 C
            sscanf(bufTemp, "%s %d %s", temporario, &temperatura, temporario);
            #ifdef DEBUG
            printf("\nTemperatura lida: %d C\n", temperatura);
            #endif
            close(fpsTemp);
            return temperatura;
    }

    void brilhoLCD(int *Nniveis, int *nivelAtual){
            unsigned char bufLCD[100];
            int fpsBrilho;
            int cnt;
            unsigned char temporario[100];
            int brilho, temp;
            /* Abrindo o arquivo do brilho do LCD */
            fpsBrilho = open("/proc/acpi/video/GFX0/DD02/brightness", O_RDWR);
            if (fpsBrilho < 0) {
                    perror("Abertura de /proc/acpi/video/GFX0/DD02/brightness falhou. ");
                    exit(1);
            }
            /*Lendo arquivo de brilho do LCD*/
            cnt =  read(fpsBrilho, bufLCD, 50 );

            /*Identificando o valor do brilho do LCD */
            //levels:  0 1 2 3 4 5 6 7
            //current: 1
            sscanf(bufLCD,  "%s  %d  %d  %d  %d  %d  %d  %d  %d\n  %s  %d",temporario,
&temp,&temp,&temp,&temp,&temp,&temp,&temp, Nniveis, temporario, nivelAtual );

            #ifdef DEBUG
            printf("\nBrilho do LCD lido: %d C\n", *nivelAtual);
            #endif
            close(fpsBrilho);
    }
    void frequencia_processador(int *frequenciaCPU0, int *frequenciaCPU1, int *frequenciaCPU2, int *frequenciaCPU3){
            unsigned char bufFreq0[100];
            unsigned char bufFreq1[100];
            unsigned char bufFreq2[100];
            unsigned char bufFreq3[100];
```

```
int fpsFreq0;
int fpsFreq1;
int fpsFreq2;
int fpsFreq3;
int cnt;
int brilho, temp;
/* Abrindo o arquivo de frequencia CPU0 */
fpsFreq0 = open("/sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq", O_RDONLY);
if (fpsFreq0 < 0) {
        perror("Abertura de /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq falhou. ");
        exit(1);
}
/* Abrindo o arquivo de frequencia CPU1 */
fpsFreq1 = open("/sys/devices/system/cpu/cpu1/cpufreq/scaling_cur_freq", O_RDONLY);
if (fpsFreq1 < 0) {
        perror("Abertura de /sys/devices/system/cpu/cpu1/cpufreq/scaling_cur_freq falhou. ");
        exit(1);
}
/* Abrindo o arquivo de frequencia CPU2 */
fpsFreq2 = open("/sys/devices/system/cpu/cpu2/cpufreq/scaling_cur_freq", O_RDONLY);
if (fpsFreq2 < 0) {
        perror("Abertura de /sys/devices/system/cpu/cpu2/cpufreq/scaling_cur_freq falhou. ");
        exit(1);
}
/* Abrindo o arquivo de frequencia CPU3 */
fpsFreq3 = open("/sys/devices/system/cpu/cpu3/cpufreq/scaling_cur_freq", O_RDONLY);
if (fpsFreq3 < 0) {
        perror("Abertura de /sys/devices/system/cpu/cpu3/cpufreq/scaling_cur_freq falhou. ");
        exit(1);
}
/*Lendo arquivo de Frequencia da CPU */
cnt =  read(fpsFreq0, bufFreq0, 50 );
cnt =  read(fpsFreq1, bufFreq1, 50 );
cnt =  read(fpsFreq2, bufFreq2, 50 );
cnt =  read(fpsFreq3, bufFreq3, 50 );

/*Identificando o valor da Frequencia da CPU */
//800000
sscanf(bufFreq0, "%d",frequenciaCPU0);
sscanf(bufFreq1, "%d",frequenciaCPU1);
sscanf(bufFreq2, "%d",frequenciaCPU2);
sscanf(bufFreq3, "%d",frequenciaCPU3);
#ifdef DEBUG
printf("\nFrequencia CPU0: %d C\n", *frequenciaCPU0);
printf("\nFrequencia CPU1: %d C\n", *frequenciaCPU1);
printf("\nFrequencia CPU2: %d C\n", *frequenciaCPU2);
printf("\nFrequencia CPU3: %d C\n", *frequenciaCPU3);
#endif

close(fpsFreq0);
close(fpsFreq1);
```

```
        close(fpsFreq2);
        close(fpsFreq3);


}
void bateria(int *corrente, int *capacidadeRestante, int *tensao){

        unsigned char bufbat[1000];     /* buffer para registrar o estado da bateria*/
        int fpsbat;                     /* descritor do arquivo da bateria */
        int cnt;                        /* para a leitura do arquivo da bateria */
        unsigned char temporario[100];  /* buffer temporario */

    /* Abrindo arquivo de bateria */
    fpsbat = open("/proc/acpi/battery/BAT0/state", O_RDWR);
    if (fpsbat < 0) {
        perror("/proc/acpi/battery/BAT0/state");
        exit(1);
    }

        /* Lendo o arquivo de bateria */
        cnt = read(fpsbat, bufbat, 205);

        /* Identificando os valores de Corrente, Capacidade restante e Tensao*/
        //present:              yes
        //capacity state:       ok
        //charging state:       charged
        //present rate:         0 mA
        //remaining capacity:   2714 mAh
        //present voltage:      12438 mV
        sscanf(bufbat, "%s %s\n%s %s %s\n%s %s %s\n%s %s %d %s\n%s %s %d %s\n%s %s %d %s", temporario,
temporario, temporario, temporario, temporario, temporario, temporario, temporario, temporario, temporario, corrente, temporario,
temporario, temporario, capacidadeRestante, temporario, temporario, temporario, tensao, temporario);

        #ifdef DEBUG
        printf("\nCorrente: %d mA", *corrente);
        printf("\nCapacidade restante: %d mAh", *capacidadeRestante);
        printf("\nTensao: %d mV\n", *tensao);
        #endif
        close(fpsbat);
}

void estatisticasProcessador(FILE *fpsStat, int *soma, int *idle){//, float *idlePercentual){

   char bufStat[500];    /* buffer para guardar string do arquivo de estatísticas*/
   int cnt;          /* para a leitura do arquivo de estatísticas */
   unsigned char temporario[500];    /* buffer temporario */
   int estatisticasCPU[5][10];       /* Estatisticas do processador 0=todo, 1 = cpu0, 2 = cpu1*/ //NÃO ENTENDO ISSO
   int somaEstatisticasCPU[5] = {0, 0, 0, 0, 0};    /* Soma das estatisticas do processador */ //NÃO ENTENDO ISSO
   int i,j;

   rewind(fpsStat);
```

```
    fflush(fpsStat);
    /* Lendo arquivo de estatisticas do processador */
    cnt = fread(&bufStat, sizeof(char), 400, fpsStat);
    //if( !fgets(bufStat, 400, fpsStat)) perror ("Leitura de log.txt falhou.");

    #ifdef DEBUG
    printf("\n%s", bufStat);
    #endif

    /* Identificando os valores */

  /*
  cpu  5452 23 2180 473379 4409 0 11 0 0 0
  cpu0 2625 3 854 114302 3144 0 7 0 0 0
  cpu1 935 12 478 120019 360 0 3 0 0 0
  cpu2 1650 0 645 118684 590 0 0 0 0 0
  cpu3 241 7 202 120373 314 0 0 0 0 0
  */

     sscanf(bufStat, "%s %d %d %d %d %d %d %d %d %d\n%s %d %d %d %d %d %d %d %d %d\n%s %d %d
%d %d %d %d %d %d %d %d\n%s %d %d %d %d %d %d %d %d %d\n%s %d %d %d %d %d %d %d %d %d\n",
temporario, &estatisticasCPU[0][0], &estatisticasCPU[0][1], &estatisticasCPU[0][2], &estatisticasCPU[0][3],
&estatisticasCPU[0][4], &estatisticasCPU[0][5], &estatisticasCPU[0][6], &estatisticasCPU[0][7], &estatisticasCPU[0][8],
&estatisticasCPU[0][9], temporario, &estatisticasCPU[1][0], &estatisticasCPU[1][1], &estatisticasCPU[1][2],
&estatisticasCPU[1][3], &estatisticasCPU[1][4], &estatisticasCPU[1][5], &estatisticasCPU[1][6], &estatisticasCPU[1][7],
&estatisticasCPU[1][8], &estatisticasCPU[1][9], temporario, &estatisticasCPU[2][0], &estatisticasCPU[2][1],
&estatisticasCPU[2][2], &estatisticasCPU[2][3], &estatisticasCPU[2][4], &estatisticasCPU[2][5], &estatisticasCPU[2][6],
&estatisticasCPU[2][7], &estatisticasCPU[2][8], &estatisticasCPU[2][9],temporario, &estatisticasCPU[3][0],
&estatisticasCPU[3][1], &estatisticasCPU[3][2], &estatisticasCPU[3][3], &estatisticasCPU[3][4], &estatisticasCPU[3][5],
&estatisticasCPU[3][6], &estatisticasCPU[3][7], &estatisticasCPU[3][8], &estatisticasCPU[3][9], temporario,
&estatisticasCPU[4][0], &estatisticasCPU[4][1], &estatisticasCPU[4][2], &estatisticasCPU[4][3], &estatisticasCPU[4][4],
&estatisticasCPU[4][5], &estatisticasCPU[4][6], &estatisticasCPU[4][7], &estatisticasCPU[4][8], &estatisticasCPU[4][9]);

    for(i = 0; i<5;i++){
       for(j = 0; j<10;j++){
       somaEstatisticasCPU[i] += estatisticasCPU[i][j];
       }
       *(soma+i) = somaEstatisticasCPU[i];
       *(idle+i) = estatisticasCPU[i][3];
    }

    //*idlePercentual = ((float) estatisticasCPU[3]) / ((float) somaEstatisticasCPU);

    #ifdef DEBUG
    printf("\nsomaEstatisticasCPU: %d %d", somaEstatisticasCPU[4], i);
    printf("\nidle: %d", estatisticasCPU[4][3]);
    //printf("\nidle/somaEstatisticasCPU: %.3f\n", ((float) estatisticasCPU[3]) / ((float) somaEstatisticasCPU));
    #endif


  }
```

```
void mudarEstadoDeEnergiaDoProcessador(int fpsProcessador, char *nomeEstado, int tamanhoNomeEstado){

        unsigned char buf[1000]; /* buffer para escrever o estado de energia*/
        int status;   /* retorna o status de chamadas do sistema */

        #ifdef DEBUG
        //printf("Frequencia = %d, Processador = d\n", parametro_S[estado_S], fpsProcessador);
        #endif
        /* mudando o estado de operacao */
    strcpy(buf, nomeEstado);
    status = write(fpsProcessador, buf, tamanhoNomeEstado); /* Mudando o estado de operacao */
    if (status != tamanhoNomeEstado)
        perror("wrote wrong number of bytes at scaling_setspeed");
}

int lerFrequenciaProcessador(int processador){
        unsigned char buf[100];         /* buffer para guardar string do arquivo de frequencia*/
        int fps;                /* descritor do arquivo de frequencia */
        int cnt;                        /* para a leitura do arquivo de frequencia */
        unsigned char temporario[100] = "/sys/devices/system/cpu/cpu";   /* buffer temporario */
        unsigned char temporario2[100] = "/cpufreq/scaling_cur_freq";    /* buffer temporario */
        int frequencia;                 /* valor de frequencia*/
        char caractereProcessador[2][2] = {"0","1"};


        strcat(temporario, caractereProcessador[processador]);
        strcat(temporario, temporario2);
    // Abrindo arquivo de temperatura do frequencia
    fps = open(temporario, O_RDWR);
    if (fps < 0) {
                printf("%s\n", temporario);
        perror("Abertura de /sys/devices/system/cpu/cpuX/cpufreq/scaling_cur_freq falhou.");
        exit(1);
    }

    // Abrindo arquivo de temperatura do frequencia
        switch(processador){
                case 0:
                fps = open("/sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq", O_RDWR);
                        break;
                case 1:
                fps = open("/sys/devices/system/cpu/cpu1/cpufreq/scaling_cur_freq", O_RDWR);
                        break;
                case 2:
                fps = open("/sys/devices/system/cpu/cpu2/cpufreq/scaling_cur_freq", O_RDWR);
                        break;
                case 3:
                fps = open("/sys/devices/system/cpu/cpu3/cpufreq/scaling_cur_freq", O_RDWR);
                        break;
```

```
		}

	if (fps < 0) {
		perror("Abertura de /sys/devices/system/cpu/cpuX/cpufreq/scaling_cur_freq falhou.");
		exit(1);
	}

		/* Lendo arquivo de frequencia do processador */
		cnt = read(fps, buf, 29);

		/* Identificando o valor de frequencia */
		sscanf(buf, "%d", &frequencia);

		#ifdef DEBUG
		printf("Frequencia lida: %d\n", frequencia);
		#endif

		close(fps);

		return frequencia;
}

/* Função que desloca os elementos de um vetor para a direita e coloca um novoElemento na posição da esquerda
 * Considere x >>> y como mover x para a posição de y
 * novoElemento >>> *p >>> *(p+1) >>> *(p+2) >>> *(p+3) >>> ... >>> *(p+num_elementos-1)
 * Obs.: primeiro ocorre *(p+num_elementos-2) >>> *(p+num_elementos-1), e depois os elementos menos significativos
 */
void deslocarVetorParaADireita(double novoElemento, double *p, int num_elementos){
		int i;
		for(i = num_elementos-1; i>0; i--){
			*(p+i) = *(p+i-1);
		}
		*p = novoElemento;
		return;
}

//phi(n) = [u(t-1) ... un(t-1)]
void preencher_phi(double *pphi, double *pu, int nu){
		int i;

		for (i=0; i<nu; i++){
			*(pphi+i) = *(pu+i);
		}
}

/*P[N][M] = A[N][K] * B[K][M] */
void produtoMatrizes(double *pP, double *pA, double *pB, int n, int k, int m){
		int i, j, l;
		double soma;
		for(i = 0; i<n; i++){
			for(j=0; j<m; j++){
```

```
                                soma = 0.0;
                                for(l=0; l<k; l++){
                                        soma += *(pA+i*k+l) * *(pB+l*m+j);
                                }
                                *(pP+i*m+j) = soma;
                        }
                }
                return;
}

/*S[N][M] = A[N][M] + B[N][M] */
void somaMatrizes(double *pP, double *pA, double *pB, int n, int m){
                int i, j;
                for(i = 0; i<n; i++){
                        for(j=0; j<m; j++){
                                *(pP+i*m+j) = *(pA+i*m+j) + *(pB+i*m+j);
                        }
                }
                return;
}

/* D[N][M] = O[N][M] */
/* copia origem para destino */
void copiaMatrizes(double *pD, double *pO, int n, int m){
                int i, j;
                for(i = 0; i<n; i++){
                        for(j=0; j<m; j++){
                                *(pD+i*m+j) = *(pO+i*m+j);
                        }
                }
                return;
}

void inicializaMatrizIdentidade(double *pI, int n){
                int i, j;
                for(i = 0; i<n; i++){
                        for(j=0; j<n; j++){
                                if(i==j){
                                        *(pI+i*n+j) = 1.0;
                                }else{
                                        *(pI+i*n+j) = 0.0;
                                }
                        }
                }
                return;
}

/*P[N][M] = x * A[N][M] */
void produtoEscalarMatriz(double *pP, double x, double *pA, int n, int m){
                int i, j;
                for(i = 0; i<n; i++){
```

```
                        for(j=0; j<m; j++){
                                *(pP+i*m+j) = x * *(pA+i*m+j);
                        }
                }
                return;
}

double
detrm (double a[ORDEM][ORDEM], int k)
{
 double s = 1, det = 0, b[ORDEM][ORDEM];
 int i, j, m, n, c;
 if (k == 1)
   {
    return (a[0][0]);
   }
 else
   {
    det = 0;
    for (c = 0; c < k; c++)
            {
             m = 0;
             n = 0;
             for (i = 0; i < k; i++)
               {
                 for (j = 0; j < k; j++)
                        {
                         b[i][j] = 0;
                         if (i != 0 && j != c)
                           {
                             b[m][n] = a[i][j];
                             if (n < (k - 2))
                                n++;
                             else
                                {
                                 n = 0;
                                 m++;
                                }
                           }
                        }
               }
             det = det + s * (a[0][c] * detrm (b, k - 1));
             s = -1 * s;
            }
   }
 return det;
}

void
cofact (double *inv, double num[ORDEM][ORDEM], int f)
{
```

```
 double b[ORDEM][ORDEM], fac[ORDEM][ORDEM];
 int p, q, m, n, i, j;
 for (q = 0; q < f; q++)
  {
   for (p = 0; p < f; p++)
             {
              m = 0;
              n = 0;
              for (i = 0; i < f; i++)
               {
                for (j = 0; j < f; j++)
                          {
                           b[i][j] = 0;
                           if (i != q && j != p)
                             {
                              b[m][n] = num[i][j];
                              if (n < (f - 2))
                                        n++;
                              else
                                        {
                                         n = 0;
                                         m++;
                                        }
                             }
                          }
               }
              fac[q][p] = pow (-1, q + p) * detrm (b, f - 1);
             }
  }
 trans (inv, num, fac, f);
}

void
trans (double *inv, double num[ORDEM][ORDEM], double fac[ORDEM][ORDEM], int r)
{
 int i, j;
 double b[ORDEM][ORDEM], d;
 for (i = 0; i < r; i++)
  {
   for (j = 0; j < r; j++)
             {
              b[i][j] = fac[j][i];
             }
  }
 d = detrm (num, r);

//  printf("\nThe inverse of the matrix :\n\n");
 for (i = 0; i < r; i++)
  {
   for (j = 0; j < r; j++)
             {
```

```
                    *(inv+i*r+j) = b[i][j]/d;
                }
            }
        }
```

APÊNDICE C: MODELO DO SISTEMA

```
/*
 * modelodosistema.h  Modelo do sistema
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * Copyright (C) 2002, International Business Machines Corporation
 * All Rights Reserved
 *
 * Saulo O. D. Luiz
 * Embedded / UFCG
 * saulo@dee.ufcg.edu.br
 * Novembro, 2007
 *
*
* Implementação da Política Proposta em TCC:
 * Bruna M. J. Cruz
 * Embedded / UFCG
 * bruna.cruz@ee.ufcg.edu.br
 * Agosto, 2012 */
 */

#ifndef __MODELODOSISTEMA_H__
#define __MODELODOSISTEMA_H__

#define NNUCLEOS 4
#define NFREQ 8

float T = 1.0; /*periodo (1s)*/
int ti = 0; /*instante inicial*/
int i, j, n = 100, iFreq;//100000; /*numero de iteracoes da simulacao*/


/*Frequencias do processador*/
/*Plataforma não suportada*/
```

```
char parametro_S[8][10] = {"2301000", "2300000", "1800000", "1600000", "1400000", "1200000", "1000000", "800000"};
int tamanhoStringParametro_S[8] = {7, 7, 7, 7, 7, 7, 7, 6};
//float frequencia_S_normalizada[8] = {1.0, 1.300075019, 1.624179944, 2.16625};
float frequencia_S[8] = {1733000, 1333000, 1067000, 800000};
float l_S[8];
int nucleo;


/*Identificação da carga de trabalho através de Mínimos Quadrados Recursivo*/
//y(n) = b(1)*u1(t-1) + ... + bnu*unu(t-1)
#define NA 1 //ordem do polinômio A
#define NB 1 //ordem do polinômio B
#define NU 5 //quantidade de entradas
int t0RLS = 50;//tempo limite para estabelecer condições iniciais
double lambda = 0.95;
double R_barra_t0[NU][NU], L_t0[NU], produto_P_t0_theta_t0[NNUCLEOS][NA+NB];
double P_t[NU][NU] = {
        {1000, 0, 0, 0, 0},
    {0, 1000, 0, 0, 0},
    {0, 0, 1000, 0, 0},
    {0, 0, 0, 1000 ,0},
        {0, 0, 0, 0, 1000}};
double theta_t_menos_1[NU]={1, 1, 1, 1, 1};

double produto_P_t0_phi[NNUCLEOS][NA+NB], produto_phi_P_t0_phi[NNUCLEOS], q1; //Condições iniciais
double phi[NU], produto_phi_y[NNUCLEOS][NA+NB], theta_t[NU], L_t[NU], P_t[NU][NU], b1, b2, b3, A; //Variáveis no instante t
double produto_phi_phi[NNUCLEOS][NA+NB][NA+NB];
double L_t_menos_1[NU], P_t_menos_1[NU][NU] = {
        {1000, 0, 0, 0, 0},
        {0, 1000, 0, 0, 0},
        {0, 0, 1000, 0, 0},
        {0, 0, 0, 1000 ,0},
        {0, 0, 0, 0, 1000}}; //Variáveis no instante t-1
//double theta_t_menos_1[NU], L_t_menos_1[NU], P_t_menos_1[NU][NU]; //Variáveis no instante t-1
double produto_P_t_menos_1_phi[NU], produto_phi_P_t_menos_1_phi;  //Variáveis no instante t-1
double produto_phi_theta_t_menos_1, erroRLS, produto_L_t_erroRLS[NU];
double uRLS[NU]; //variáveis para colocar em phi
double ySimuladoRLS, ySimuladoRLS_menos1;//Estimativa da próxima saída de acordo com o modelo obtido de mínimos quadrados recursivo
double matrizIdentidade[NU][NU], produto_L_t_phi[NU][NU], sub_Identidade_produto_L_t_phi[NU][NU];

/*Controlador RST - Controle de utilizacao da CPU*/
float pt1 = 3016.4, r0 = 1.0, r1 = 0.9997, s1 = 3014.7;
double u1, u2, u3, u4, y, uMenos1[NNUCLEOS] = {1}, uc[NNUCLEOS], ucOriginal[NNUCLEOS], Ls[NNUCLEOS] = {2.0}, Li[NNUCLEOS] = {0.0};
int comandoDoGE[NNUCLEOS]; /*comando do gerenciador de energia*/
int comandoDoGE_menos1[NNUCLEOS] = {0}; /*comando do gerenciador de energia*/

/* Estado da bateria */
```

```
int correnteBateria;
int capacidadeRestanteBateria;
int tensaoBateria;

/*Frequencia do Processador */
int frequenciaCPU0;
int frequenciaCPU1;
int frequenciaCPU2;
int frequenciaCPU3;
float float_fCPU0, float_fCPU1, float_fCPU2, float_fCPU3;

/* Estado do processador */
int temperaturaDoProcessador;
int tempoCPU[NNUCLEOS+1]={0,0,0,0,0}, tempoCPUinativa[NNUCLEOS+1]={0,0,0,0,0};     /* tempoCPU = tempo que a CPU gasta realizando as várias tarefas*/
        /* tempoCPUinativa = tempo que a CPU gasta inativa*/
int tempoCPU_anterior[NNUCLEOS+1]={0,0,0,0,0}, tempoCPUinativa_anterior[NNUCLEOS+1]={0,0,0,0,0};
float  inativaPercentual[NNUCLEOS+1]={0.0,0.0,0.0,0.0,0.0,0.0};            /* inativaPercentual = (float)tempoCPUinativa / (float)tempoCPU */
float  ativaPercentual[NNUCLEOS+1]={0.0,0.0,0.0,0.0,0.0,0.0};           /* ativaPercentual = 1 - (float)tempoCPUinativa / (float)tempoCPU */
float atividade;
float sinal_i[NNUCLEOS+1]={0,0,0,0,0}, sinal_i_filtrado[NNUCLEOS+1]={0,0,0,0,0}, alfa = 0.0920;

float numeroAleatorio;

#endif /*__MODELODOSISTEMA_H__*/
```

REFERÊNCIAS BIBLIOGRÁFICAS

[1] WANG, Y.; MA, K.; WANG, X. Temperature-constrained power control for chip multiprocessors with online model estimation. *SIGARCH Comput. Archit. News*, ACM, New York, NY, USA, v. 37, n. 3, p. 314--324, 2009. ISSN 0163-5964.

[2] LUIZ, S. O. D.; PERKUSICH, Angelo; LIMA, Antonio Marcus Nogueira; SILVA, J. J.; ALMEIDA, H. System Identification and Energy-aware Processor Utilization Control. IEEE Transactions on Consumer Electronics, v. 58, p. 32-37, 2012.

[3] MIAN, D.; ZHONG, L. Self-Constructive High-Rate System Energy Modeling for Battery-Powered Mobile Systems. **Proc. MobiSys**, 2011. 335-348.

[4] SHEARER, F. Power Management in Mobile Devices. 1. ed. Burlington, USA: Newnes, 2008. ISBN 978-0-7506-7958-9.

[5] TIAN, Y. C. et al. Control-theoretic dynamic voltage scaling for embedded controllers. **IET Computers & Digital Techniques**, v. 2, p. 377–385, 2008. ISSN 5.

[6] LINUX , 2003. Disponivel em: <http://liw.iki.fi/liw/texts/linux-the-big-picture.html>. Acesso em: 25 Julho 2012

[7] Oliveira, P. J. R. de. Estimação de parâmetros de um motor de indução utilizando um modelo contínuo no tempo. 1998.

[8] LJUNG, L. *System Identification: Theory for the User*. 2. ed. Englewoods Cliffs, New Jersey 07632: Prentice Hall PTR, 1998. Hardcover. ISBN 0138816409.

[9] MINAMI, K., VÉLEZ-REYES, M., ELTEN, D., VERGHESE, G.C. e FILBERT, D., (1991), *Multi-Stage Speed and Parameter Estimation for Induction Machines*, in IEEE Proceeding of IAS'91, Grenoble, França, pp. 596-604.

[10] **Intel**. Disponivel em: <http://www.intel.com.br/content/www/br/pt/processors/core/core-i5-processor-office-use/Corei5Business.html?wapkw=%28core+i5+2410M%29>. Acesso em: 25 Julho 2012.

[11] HOPPER, J. Subsistema CPUfreq. **DeveloperWorks**, 2009. Disponivel em: <http://www.ibm.com/developerworks/br/linux/library/l-cpufreq-1/>. Acesso em: 25 Julho 2012.

[12] Governors.txt. Disponível em: http://www.mjmwired.net/kernel/Documentation/cpu-freq/governors.txt. Acesso em: 05 de Maio de 2012

[13] Luiz, S. O. D. Gerenciamento Dinâmico de Energia em Processadores com Cargas de Trabalho Variantes no Tempo. Tese (Doutorado), UFCG, 2012.

[14] Rodrigues, R. A. B. Métricas e Ferramentas Livres para Análise de Capacidade de Servidores Linux, 2009.

[15] HENSON, V. *ebizzy*. Agosto 2010. Disponível em: <http://sourceforge.net/projects/ebizzy/>. Acesso em 22 de Julho de 2012.