UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Anderson Morais Barbosa de Araújo

(Matrícula: 21015216)

GERENCIAMENTO DINÂMICO DE ENERGIA

Campina Grande

Maio/ 2011

ANDERSON MORAIS BARBOSA DE ARAÚJO

(Matrícula: 21015216)

GERENCIAMENTO DINÂMICO DE ENERGIA

Trabalho apresentado ao Centro de Engenharia Elétrica e Informática, Departamento de Engenharia Elétrica da Universidade Federal de Campina Grande, como requisito para conclusão de curso de Engenharia Elétrica sob a orientação do M.Sc. Saulo Oliveira Dornellas Luiz.

Campina Grande

Maio/ 2011

ANDERSON MORAIS BARBOSA DE ARAÚJO

(Matrícula: 21015216)

GERENCIAMENTO DINÂMICO DE ENERGIA

Trabalho apresentado ao Centro de Engenharia Elétrica e Informática, Departamento de Engenharia Elétrica da Universidade Federal de Campina Grande, como requisito para conclusão de curso de Engenharia Elétrica sob a orientação do M.Sc. Saulo Oliveira Dornellas Luiz.

Aprovado em: ____/ ____/ _____

EXAMINADORES

_____

M.Sc. Saulo Oliveira Dornellas Luiz (Orientador)
UFCG - Departamento de Engenharia Elétrica

_____

M. Sc. João Batista Morais dos Santos (Examinador Convidado)
UFCG - Departamento de Engenharia Elétrica

AGRADECIMENTO

A Deus por ter-me fortalecido frente às dificuldades.

Aos meus pais fervorosos incentivadores que acreditam em mim.

Às minhas irmãs, tão importantes para mim, por suas presenças nos momentos de alegria e de tristeza, mesmo que estejamos fisicamente distantes.

À minha noiva e aos amigos todos, irmãos que a vida me deu a opção de escolher.

Ao meu professor orientador, como a todos os mestres que contribuíram para minha formação.

"As dificuldades são o aço estrutural que entra na construção do caráter."

Carlos Drummond de Andrade

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE GRÁFICOS

SUMÁRIO

RESUMO

Face aos avanços tecnológicos que a humanidade tem experimentado nas últimas décadas, no que concerne a equipamentos destinados à tecnologia da informação, estudos vêm sendo realizados visando diminuir o consumo de energia destes, através da adoção de políticas de otimização do consumo de energia. A necessidade de redução do consumo de energia impulsionou estudos que ocasionaram o surgimento de inúmeras políticas de gerenciamento de energia. Este trabalho evidencia o Gerenciamento Dinâmico de Energia (GDE), as influências decorrentes das cargas de trabalho sobre a utilização dos processadores e algumas das muitas técnicas dinâmicas de gerenciamento de energia. Assim, apresenta-se inicialmente a técnica conhecida como Escalonamento Dinâmico de Tensão (EDT), seguida do Escalonamento Dinâmico de Tensão e Freqüência (EDTF) e a Movimentação de *Threads.* Este trabalho trata, ainda, de recursos de *hardware* atualmente empregados em processadores reais como é o caso da tecnologia SpeedSteep e Turbo Boost da Intel, além da Cool'n'Quiet da AMD. Neste trabalho são apresentadas, também, políticas de gerenciamento de energia, as quais regulam o consumo de energia via *software,* bem como a influência das cargas de trabalho no procedimento de avaliação das políticas de energia. Realizaram-se, ainda, experimentos visando aferir a utilização e o consumo de energia do processador Celeron D310, e adicionalmente outros experimentos visando promover a alteração dos estados de energia do processador Pentium M 750 submetido a uma carga de trabalho que motivou a propositura de uma nova política de gerenciamento de energia.

ABSTRACT

Facing the technological advances that humanity has experienced in recent decades, which regard to equipments for information technology, studies are being conducted to reduce the energy consumption of these, through the adoption of policies to optimize energy consumption. The need to reduce energy consumption has stimulated studies that led to the emergence of numerous policies of power management. This work shows the Dynamic Power Management (DPM), the influences arising from the workload on the use of processors and some of the many techniques of dynamic power management. Thus there is initially a technique known as Dynamic Voltage Scaling (DVS), followed by Dynamic Voltage and Frequence Scaling (DVFS) and the Thread Motion. It is also discussed the hardware resources currently used in real processors such as the technologies SpeedSteep and Turbo Boost from Intel, and AMD Cool'n'Quiet. This work also presented a power management policy, which regulates the energy consumption via software, as well as the influence of workloads in the evaluation procedure of the power management policy. Furthermore, experiments to assess the use and power consumption of a Celeron D310 processor, and other experiments to change the energy states of the Pentium M 750 subjected to a workload were presented. These experiments motivated the proposal of a new power management policy.

1. INTRODUÇÃO

O avanço tecnológico que a humanidade tem experimentado nas últimas décadas, no que concerne a dispositivos de comunicação móvel e ao avanço de computadores portáteis para fins diversos ou específicos como *notebooks, netbooks, palms,* telefones celulares, MP4 *Players*, IPods, dentre outros tantos que poderiam ser citados, é realmente extraordinário. Entretanto, os avanços não são os mesmos em todos os campos e, em alguns destes, há limites os quais têm mantido os especialistas bastante ocupados, buscando alternativas que solucionem ou minimizem tais limitações.

Algumas destas limitações em relação aos aparelhos citados são as dimensões, o peso e a autonomia de uso devido à capacidade de armazenamento de energia e ao consumo de potência. As duas primeiras, todavia, vem sendo reduzidas ano após ano e proporcionando ainda a incorporação de novas funcionalidades nas versões mais recentes destes aparelhos portáteis. A autonomia, porém, ainda é um problema, visto que a capacidade de armazenamento de energia das baterias tem aumentado, mesmo que em ritmo bastante modesto. No entanto, as novas funções adicionadas causam o aumento no consumo da energia armazenada, de maneira que os avanços na autonomia da bateria se tornam insuficientes.

Paralelo a estas soluções de melhoria das baterias, estudos são focados em diminuir o consumo de energia destes dispositivos através da adoção de políticas de otimização do processamento das informações e de estratégias de desligamento do computador, ou ainda, pela colocação do dispositivo em estado de hibernação, que é um estado intermediário entre os de maior e menor consumo energético.

A fim de obter a redução desejada da demanda energética dos sistemas computacionais, desenvolveram-se muitas técnicas, porém “uma das técnicas mais bem sucedidas empregadas por projetistas no nível do sistema de gerenciamento de energia é dinâmica” Benini et al. (1998) página 1. Portanto, devido à sua própria ação e dinamicidade, esta técnica é denominada de Gerenciamento Dinâmico de Energia.

A necessidade de redução do consumo de energia que impulsiona os estudos com os fins citados anteriormente ocasionou o surgimento de inúmeras políticas de gerenciamento de energia. Este trabalho evidenciará o Gerenciamento Dinâmico de Energia (GDE ou DPM, *Dynamic Power Management*) e as influências decorrentes das cargas de trabalho. Entende-se por cargas de trabalho todas as atividades e tarefas que são executadas pelo processador dos dispositivos, tais como um vídeo, um programa de escritório, uma planilha eletrônica ou programa científico, entre outras.

1.1 Revisão Bibliográfica

Diferentes estudos sobre as várias técnicas de gerenciamento de energia mostram que elas podem ser tratadas em dois momentos distintos. Aquelas adotadas durante o projeto dos sistemas e aquelas utilizadas no dia-a-dia quando o sistema computacional está operando e realizando tarefas do usuário. Sendo, portanto, classificadas respectivamente como Estáticas e Dinâmicas, dentre tais estudos envolvendo técnicas dinâmicas, pode-se citar Luiz et al.(2008), Luiz et al. (2009), Isci et al. (2006), Wang et al. (2009).

> "*As técnicas de redução de consumo de energia são classificadas em: (i) estáticas, usadas na fase de projeto, de que são exemplos a síntese e a compilação para baixo consumo; e (ii) dinâmicas, denominadas Gerenciamento Dinâmico de Energia (GDE) e possibilitam a redução do consumo de energia nesses sistemas em tempo de execução,…*"
>
> Luiz et al.(2008) página 1.

O presente trabalho apresenta algumas das muitas técnicas dinâmicas de gerenciamento de energia disponíveis na vasta bibliografia sobre o assunto onde é possível observar o comportamento dos sistemas em tempo real quando submetidos a tais práticas. Assim, será primeiro apresentada a técnica conhecida como EDT – Escalonamento Dinâmico de Tensão (em inglês DVS – *Dynamic Voltage Scaling*) discutida em Novelli et al. (2005) e Luiz et al. (2009). Em seguida, será apresentada a EDTF – Escalonamento Dinâmico de Tensão e Freqüência (ou DVFS – *Dynamic Voltage and Frequence Scaling*) apresentada por Isci et al. (2006), Rangan et al.

(2009), Wang et al. (2009) que permite a variação tanto da freqüência quanto da tensão de um processador. E de Movimentação de *Threads* em Ragan et al. (2009) e outros.

1.1.1 Escalonamento Dinâmico de Tensão

A política de escalonamento dinâmico de tensão tem seu valor por assegurar redução do consumo de energia, e ao mesmo tempo, do ponto de vista didático facilita a compreensão do processo de escalonamento de tensão e/ou freqüência. "*Contudo, a redução da freqüência operacional do processador, isoladamente, não leva a uma redução de consumo, pois implica no aumento do tempo de execução das tarefas*", Novelli et al. (2005) página 3, permitindo verificar que na prática, escalonar apenas freqüência para um nível inferior ao máximo pode não ter como resultado a economia de energia pretendida.

Um dado processador realiza uma tarefa em um tempo t_o utilizando como valor de tensão $\mathbf{v_{max}}$, Figura 1(a), em seguida este mesmo processador realiza a mesma tarefa utilizando outro valor de tensão de alimentação, $\mathbf{v_o}$, Figura 1(b). Considerando a Equação (1), Araújo et al. (2010) pagina 2 e Intel White Paper (11/2008), abaixo, e analisando que a dependência de consumo dinâmico de energia tem relação quadrática com a tensão de alimentação fica evidenciado que a redução do nível da tensão de alimentação é uma técnica realmente relevante para redução do consumo de energia e potência consumidas pelo processador.

$$P = f \cdot C \cdot V^2 \tag{1}$$

Onde: P – Potência consumida pelo processador;

f – Freqüência de operação do processador;

C – Capacitância de chaveamento do processador;

V – Tensão de alimentação do processador.

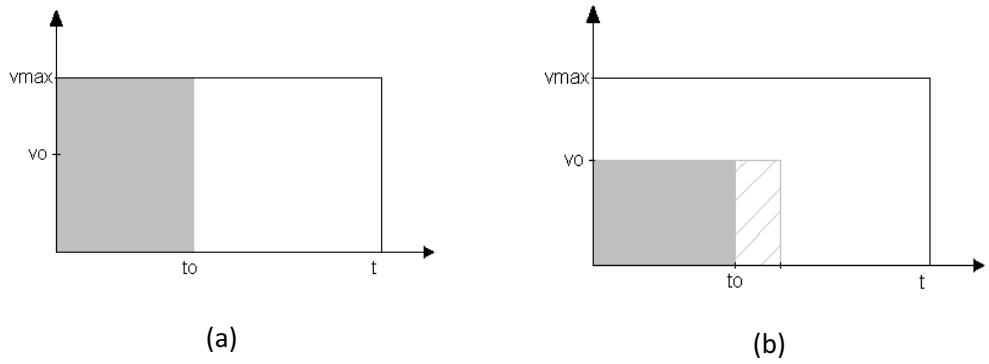

Figura 1: a) Execução de uma tarefa em um processador sem Escalonamento Dinâmico de Tensão.
b) Execução de uma tarefa em um processador com Escalonamento Dinâmico de Tensão

Para realizar a técnica do Escalonamento Dinâmico de Tensão, deve-se considerar anteriormente que é possível reduzir a tensão de operação do processador, de modo que esta redução aconteça para níveis de tensão discretos e finitos, a fim de assegurar sua operacionalidade.

Neste enfoque, para fixar mais apropriadamente, consideram-se os seguintes níveis de tensão distintos:

1- Tensão Máxima – $\mathbf{v_{max}}$;
2- Tensão Ideal – $\mathbf{v_o}$;
3- Tensão nula – 0.

Sobre os níveis de tensão entende-se que:

- Tensão Máxima é a tensão nominal de operação, aquela na qual o processador teria sido projetado para operar caso não houvesse nenhuma política de gerenciamento de energia.
- Tensão Ideal é aquela cujo consumo de energia é calculada pelo sistema computacional como sendo o mais apropriado com base em uma lei preestabelecida.
- Tensão nula representa o estado em que o processador está totalmente inativo, ou seja, uma tensão mínima e significativamente menor que $\mathbf{v_o}$.

> *"... Deve ser notado que, para circuitos integrados reais, admitindo que para uma dada voltagem é sempre utilizada a freqüência máxima correspondente a esse nível, a redução na tensão de alimentação exige uma conseqüente redução na freqüência de operação."*
>
> Novelli et al. (2005) página 3.

Desta forma, este modelo de gerenciamento dinâmico não é útil na prática, mas esclarece o processo de escalonamento e o mesmo princípio pode ser empregado em Escalonamento Dinâmico de Tensão e Freqüência com sucesso, como será explicitado a seguir.

1.1.2 Escalonamento Dinâmico de Tensão e Freqüência

O escalonamento dinâmico de tensão e freqüência é o escalonamento conjunto entre níveis de tensão e níveis de freqüência ditados por uma política global de gerenciamento de energia conforme Ragan et al. (2009) e Isci et al. 2006. Disto percebe-se que o Escalonamento Dinâmico de Tensão é, portanto, um caso particular do Escalonamento Dinâmico de Tensão e Freqüência onde a variável utilizada para controlar o escalonamento é a tensão de alimentação, e a freqüência varia correspondendo à variação de tensão conforme Tabela 1.

Bircher et al. (2008) versa sobre o gerenciamento de energia ativo e inativo, donde o "gerenciamento de energia ativo visa selecionar um ponto de operação ideal, com base na demanda de desempenho do programa", enquanto o "gerenciamento de energia inativo reduz o consumo de energia durante as fases do programa ocioso", estabelecendo assim dois perfis de carga de trabalho.

Desta forma, pode-se afirmar que na primeira, a energia consumida durante o processamento de uma carga de trabalho requer níveis mais elevados de energia para a conclusão da tarefa, ao passo que a segunda trata da energia necessária para manter o processador minimamente alimentado durante os intervalos de tempo nos quais o processador esteja ocioso. Deixando claro que o consumo de energia é fortemente dependente da carga de trabalho à qual o processador está submetido.

Bircher et al. (2008) descreveram estados de operação para o processador ativo padronizado pelo *Advanced Configuration and Power Interface*, ACPI (2010),

citando estados de operação para o processador ativo. Esses estados são denominados de estados-P (P0, P1,..., PN) e para cada estado é associado um par freqüência e tensão de alimentação, conforme exemplificado na Tabela 1 para um dado processador. Descreveram também os estados-C para o gerenciamento de energia inativo (C0, C1,..., CN), exemplificados na Tabela 2. Estes estados indicam a latência da resposta do processador necessária para o mesmo sair do estado de sono para o estado ativo. O P-estado P0 é o estado de maior processamento de informação, enquanto o PN é o de menor processamento. O estado C0, por sua vez, indica que o processador está ativo, ao passo que os estados C1 a CN indicam estados de redução progressiva do consumo de energia aumentando a latência da resposta.

	Frequência (MHz)	VDD (Volts)
P0	Fmax 100%	Vmax 100%
P1	Fmax 85%	Vmax 96%
P2	Fmax 75%	Vmax 90%
P3	Fmax 65%	Vmax 85%
P4	Fmax 50%	Vmax 80%

Tabela 1: Definição de estados-P para um dado processador. Bircher et al. (2008) pág. 2

	Latência da Resposta (us)
C0	0
C1	10
C2	100
C3	1000
C4	10000

Tabela 2: Definição de estados-C para um dado processador. Bircher et al. (2008) pág. 3

A técnica de gerenciamento denominada Escalonamento Dinâmico de Tensão e Freqüência pode ser utilizada em processadores com um único núcleo ou em processadores mais atuais e sofisticados conhecidos como multi-núcleo, que são

processadores com mais de um núcleo. Cada núcleo de processamento pode operar com uma política de Escalonamento Dinâmico de Tensão e Freqüência individual ou com uma política global de gerenciamento dinâmico para todo o processador.

Entretanto, em processadores multi-núcleo, a técnica de Escalonamento Dinâmico de Tensão e Freqüência torna-se mais complexa para ser utilizada porque, nestes casos, deve-se considerar se a política será aplicada a cada núcleo individualmente ou a todos os núcleos. E evidentemente que cada caso terá suas peculiaridades, além de vantagens e desvantagens.

1.1.3 Movimentação de *Threads*

Alguns trabalhos apresentam a utilização da técnica de Escalonamento Dinâmico de Tensão e Freqüência associada à Movimentação de *Threads* (MT) devido à limitação do Escalonamento Dinâmico de Tensão e Freqüência sozinho operar na ordem de nanossegundos. “O *objetivo da Movimentação de Threads é permitir os benefícios do DVFS em escalas de tempo em nanossegundos, explorando as variações decorrentes de eventos microarquiteturais*” Ragan et al. (2009) pág. 4.

A Movimentação de *Threads* proporciona a movimentação das aplicações dentre os vários núcleos de processamento para permitir com isto a execução mais rápida da tarefa que está sendo executada por um único núcleo. Ao distribuir mais uniformemente as tarefas a serem executadas, os núcleos poderão processar estas informações utilizando um nível de tensão e freqüência mais baixas, implicando em economia de energia. Registra-se então, duas razões possíveis e principais: a primeira por utilizar um estado-P maior, ou seja, valores de tensão e freqüência menores, e a segunda, ao dividir a tarefa, o tempo total de realização será reduzido.

Além das razões citadas anteriormente, outra possibilidade é a de utilizar um estado-P menor para que a tarefa seja concluída utilizando valores mais elevados de tensão e freqüência, caso se deseje maior desempenho em detrimento da economia de energia. As possibilidades de consumo e desempenho são muitas quando se utiliza a combinação de tensão e freqüência.

Na Fig. 2 (a) é ilustrado um processador hipotético que contém quatro núcleos de processamento onde L1 representa a unidade responsável por ditar a política de gerenciamento dinâmico de energia, bem como distribuir as tarefas de processamento entre os quatro núcleos. A unidade L1 pode ser implementada em *hardware* ou *software*. Assim, o núcleo N1 poderia estar processando a aplicação A, enquanto o núcleo N3 poderia estar processando a aplicação B. Após algum tempo de processamento, a aplicação A poderá ser transferida para o núcleo N3, ao passo que a aplicação B passaria a ser processada pelo núcleo N1.

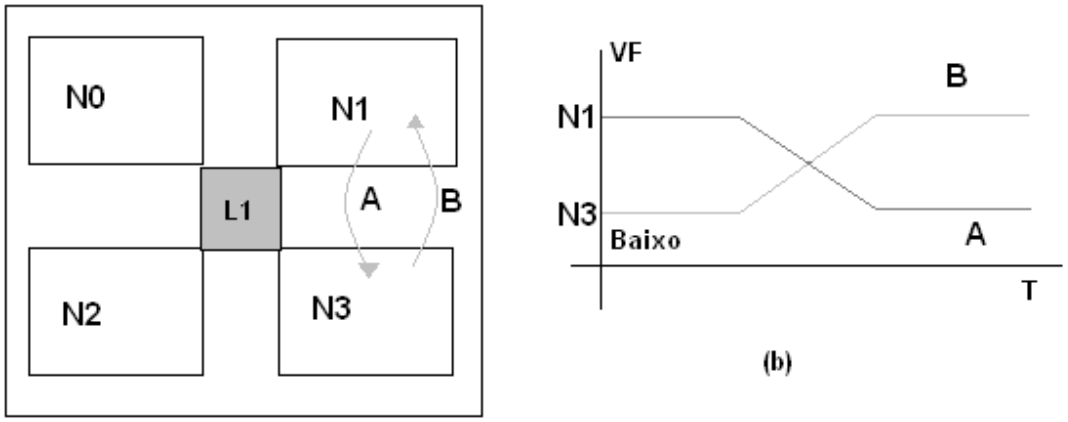

Figura 2: (a) Representação de um processador com 4 núcleos de processamento.
(b) Transições de níveis de tensão e freqüência das aplicações A e/ou B.

Da Fig. 2 (b) podemos inferir que a aplicação A está inicialmente em um estado de tensão e freqüência altas, enquanto a aplicação B está em estado de baixa energia. Após um intervalo de tempo estas aplicações passam por uma transição fazendo com que comecem a ser processadas nos níveis de tensão e freqüência comutados, devido a uma decisão de L1 para acelerar o processamento de B, ao passo que a aplicação A fica em um estado de economia de energia.

Araújo et al. (2010), contudo, expõe que se deve optar, quando se está utilizando esta técnica, ou pelo desempenho ou pela economia de energia, pois ao se utilizar freqüências mais elevadas no processamento da aplicação ocorrerá um maior consumo de energia. Entretanto ao operar com freqüências mais baixas evidencia-se uma redução de desempenho de processamento.

Após explanarmos acerca destas políticas, suas principais virtudes e algumas desvantagens que possuem, verifica-se que em todas existe uma dependência

estreita entre o desempenho, a economia de energia e aquilo que se está processando, ou seja, a carga de trabalho que está sendo processada.

1.2 Objetivos Gerais

- Realizar atividades experimentais utilizando um *software* GDE, Anexo A, desenvolvido em linguagem de programação C com a função de monitorar a utilização do processador. Este *software* será executado no sistema operacional Linux com distribuição Ubuntu 10.10 com versão do núcleo 2.6.35-22-generic (i686).
- Calcular a energia utilizada pelo processador durante a execução de cargas de trabalho.
- Comparar o comportamento da utilização do processador e outras informações mediante a execução de várias cargas de trabalho conhecidas, disponíveis em *Benchmarks*, e da realização de testes em uma carga de trabalho qualquer.

1.3 Objetivos específicos

Os objetivos específicos deste trabalho são:

- Conhecer a política implementada em GDE para gerenciamento de energia;
- Executar GDE em espaço de usuário do Linux para obtenção de valores de utilização do processador, tempo ocioso, energia consumida, temperatura do processador, freqüência de operação do processador;
- Variar as freqüências dos núcleos do processador para comparação do comportamento de consumo de energia de cada um, bem como o tempo de utilização de cada núcleo;
- Executar a EBIZZY e Carga Alta, Anexo F;

- Chamar Carga Alta Periodicamente, Anexo D;
- Plotar, utilizando o MatLab, os gráficos obtidos com os dados de GDE.

2. GERENCIAMENTO DINÂMICO DE ENERGIA PARA PROCESSADORES

Nesta seção serão apresentados alguns dos recursos reais de *hardwares* disponíveis no mercado de processadores, bem como algumas de suas principais vantagens e desvantagens na utilização destes, além de discorrer sobre a política que estes processadores implementam para realizar a conservação de energia e ampliação de desempenho.

2.1 Recursos de Hardware para Gerenciamento de Energia

Existem recursos de gerenciamento de energia instalados nos diversos computadores buscando torná-los mais eficientes e econômicos, dentre estes podem ser citados: *Advanced Power Management (*APM), ou seja, gerenciamento avançado de energia, e Interface de Configuração e Gerenciamento de Energia Avançada (ACPI).

Entre os recursos apresentados no parágrafo anterior, pode-se registrar que o APM possibilita que sistemas computacionais gerenciem suas características de utilização e consumo de energia. Operando em nível de BIOS busca reduzir o consumo de energia do sistema quando este não estiver em uso, reduzindo a freqüência do processador, desligando o monitor e o disco rígido por exemplo.

A partir de um paralelo entre o APM já apresentado e o ACPI, constata-se que este último é mais sofisticado. Isto deve-se ao fato de que o ACPI é uma camada de gestão energética que executa em nível de sistema operacional e tem os recursos da APM e muitos outros como: desligamento da máquina por teclas especiais, controle de brilho e contraste, do monitor de temperatura, periféricos, dentre outros. Desta maneira o ACPI varia de um sistema para outro com relação ao suporte dos recursos especiais que cada um possui e armazena as informações em tabelas.

Contudo, estes recursos de gerenciamento de energia são genéricos e apesar de controlar o consumo via *hardware*, são ferramentas de *software.* Entretanto, tais recursos ainda permitem a cada fabricante de processadores implementar em seus

produtos, ferramentas e técnicas de gerenciamento de energia em *hardware*, como é o caso do Intel SpeedStep, Intel White Paper (11/2008), do Intel Turbo Boost, Intel White Paper (03/2004), e Cool'n'Quiet, AMD *Aplication Note* (05/2005), e a PowerNow!, AMD *Installation Guide for AMD Athlon* (06/2004), da AMD, que serão destacadas nos subitens a seguir.

2.1.1 Intel SpeedStep

Esta é uma tecnologia da Intel que visa gerenciar funções do processador a fim de obter maior economia de energia e controlar seu aquecimento. Atualmente os processadores da Intel que utilizam esta tecnologia são: Centrino, Atom, Core 2 e Core 2 duo, além dos mais novos Core i3, Core i5 e Core i7.

O SpeedStep reduz a tensão e a freqüência do *clock* do processador quando este está sendo pouco utilizado o que implica simultaneamente numa redução do consumo energético, proporcionando assim, o resfriamento dos componentes. Decorre destas vantagens o aumento de vida útil do processador.

Apesar das vantagens apresentadas pelo SpeedStep, esta tecnologia possui limitações. Pode-se citar que há a possibilidade de que o processador não atinja as freqüências mais elevadas de processamento e o comprometimento à visualização de vídeos ou outras cargas de trabalho quando o SpeedStep está ativo.

2.1.2 Intel Turbo Boost

Outra tecnologia da Intel, é a Turbo Boost, que é implementada em alguns dos seus processadores, o Core i3, Core i5 e Core i7, e permite o aumento dinâmico da freqüência do *clock* da CPU quando necessário. Desta forma, quando o sistema operacional solicita o estado de alto desempenho do processador, o Turbo Boost é ativado.

Os vários níveis de desempenho do processador com Intel Turbo Boost são ativados quando ocorre uma requisição do sistema operacional. Desta forma o

desempenho é controlado via ACPI que é suportado pelos principais sistemas operacionais disponíveis atualmente no mercado.

A Turbo Boost tem em sua tecnologia mecanismos que funcionam da seguinte maneira: enquanto o processador não atingir os seus limites térmico e elétrico e a carga de trabalho do usuário necessitar de desempenho adicional, a freqüência de *clock* do processador irá aumentar dinamicamente em incrementos com intervalos curtos e regulares, até que o limite de energia térmica seja atingido ou que a velocidade máxima para o número de núcleos ativos seja alcançada. Por outro lado, quando qualquer um dos limites for atingido, a frequência do processador automaticamente sofrerá reduções também curtas e regulares, em intervalos que podem ser de 100 ou 133MHz de acordo com a arquitetura do processador.

É muito raro ver todas as vantagens do Turbo Boost na prática, pois ele tem potencial para acelerar as tarefas que estão sendo executadas por um único núcleo, mas os outros núcleos são mantidos inativos, ou seja, mantém um único núcleo ativo por vez. Esta tecnologia está avançando para que seja possível obter o máximo de todos os núcleos.

2.1.3 AMD Cool'n'Quiet

Outra tecnologia presente no mercado de processadores é a AMD Cool'n'Quiet que, apesar de se assemelhar à tecnologia Intel SpeedStep, e à AMD PowerNow!, difere porém, porque estas foram desenvolvidas para serem utilizadas em dispositivos portáteis os quais utilizam baterias que aumentam sua autonomia, e a Cool'n'Quiet se volta para processadores utilizados em *desktop*.

A tecnologia Cool'n'Quiet que foi lançada pela AMD na linha de processadores Athlon 64, e funciona como um limitador de freqüência e gerenciador de energia. Desta forma promove a redução da frequência do *clock* e da tensão do processador quando este estiver ocioso. Com isto, reduz o consumo de energia e a dissipação de calor além de favorecer que o sistema de resfriamento trabalhe mais lento e silenciosamente.

A Cool'n'Quiet que inicialmente foi lançada para o Athlon 64 é também utilizada como tecnologia de gerenciamento de energia nos processadores Athlon II, Sempron e Phenom, enquanto a PowerNow! é utilizada nos processadores Mobile Athlon 64, Mobile Sempron, Turion 64 e X2, dentre outros da AMD pelas vantagens de que ela apresenta.

2.2 Políticas de Gerenciamento

As políticas de gerenciamento são especificações disponíveis para regular o consumo de energia por meio de *software* num computador, mais especificamente o processador. No sistema operacional Linux, a maioria dos *drivers* de gerenciamento de energia, denominados de cpufreq, realizam por meio de algoritmos escalonadores de freqüência a configuração da velocidade de processamento. A fim de oferecer escalonamento dinâmico de freqüência, a interface cpufreq deve ser capaz de enviar um comando ao processador para atingir uma "freqüência alvo". Assim estes *drivers* específicos são adaptados para oferecer um "alvo" ao invés de utilizar as configurações previamente estabelecidas pela política de gerenciamento, ou seja, o estabelecimento da freqüência alvo torna-se dinâmica.

Hoje, existem políticas de gerenciamento de energia disponíveis para serem utilizadas pela cpufreq: *Powersave* e a *Performance*, que configuram uma freqüência estaticamente para ser a menor ou a maior, respectivamente. Além de outras políticas de escalonamento dinâmico de freqüências que possibilitam a alteração inclusive do desempenho do processador baseando-se no histórico de utilização, na demanda de desempenho, ou ainda para obter uma condição de conservação energética. Estas políticas são: *Userspace*, *Conservative* e *Ondemand*. Para decidir que política cpufreq deve ser utilizada, dispõem-se de "gerenciadores de cpufreq" que decidirão com base em testes extensivos qual é o melhor gerenciador.

O fluxo decisório de qual política o processador deve utilizar é mostrado na Figura 3. O fluxo pode iniciar em duas situações distintas, na primeira a CPU pode ser configurada para variar as freqüências dentro de um limite específico. Enquanto na outra, a CPU pode operar fixa em uma freqüência pré-estabelecida. Em seguida, a política é regulada pelo gerenciador e configura a freqüência respeitando um limite

mínimo e outro máximo. E por fim, ocorre a decisão de utilizar os limites de freqüência de uma política configurada pelo usuário ou pelo gerenciador, neste caso pode ser estática ou dinâmica.

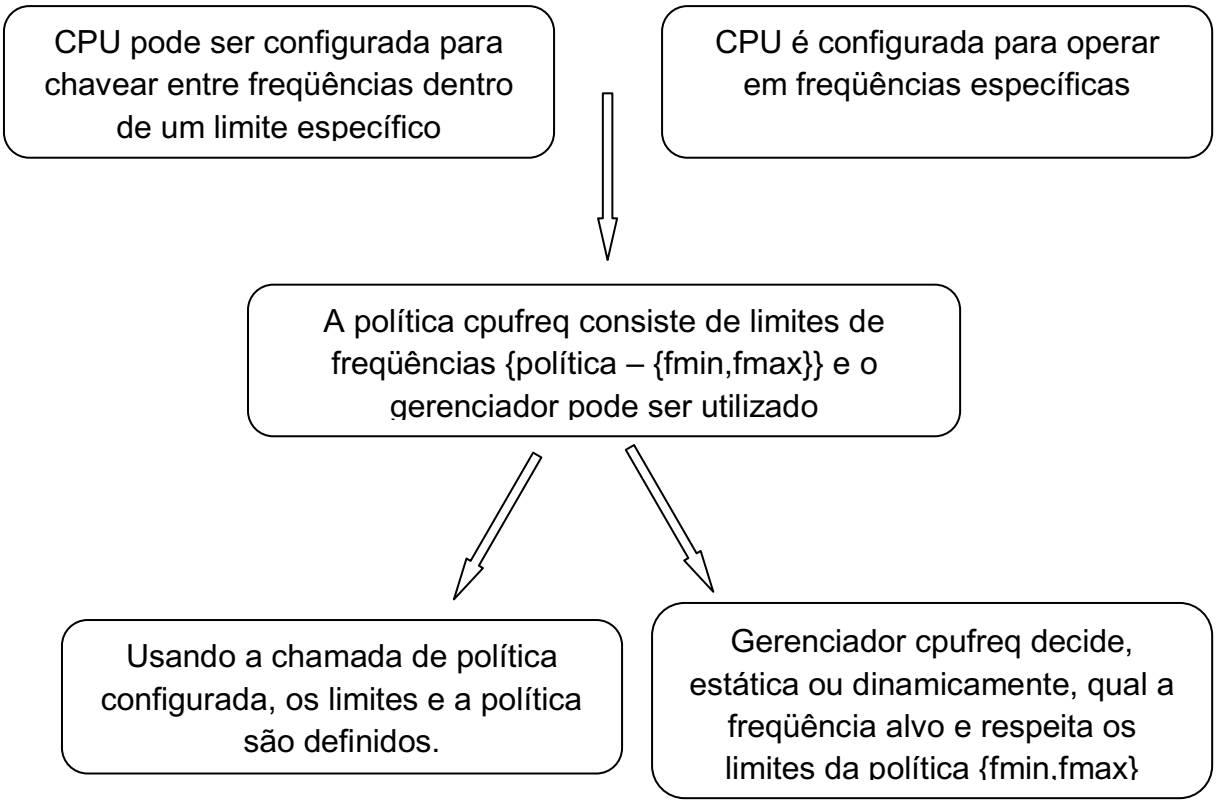

Figura 3: Fluxo decisório da política de gerenciamento do cpufreq.

2.2.1 Performance

O gerenciador cpufreq *Performance* configura a CPU estaticamente, de modo a assegurar maior desempenho estabelecendo a maior freqüência dentro dos limites de uma escala de freqüências máxima e mínima.

2.2.2 *Powersave*

O gerenciador cpufreq *Powersave* configura a CPU estaticamente para proporcionar uma economia de energia mais significativa, definindo menor freqüência dentro dos limites da escala.

2.2.3 *Userspace*

O gerenciador cpufreq *Userspace* (espaço de usuário) permite ao usuário, ou a um programa em execução dentro do espaço de usuário configurar a CPU para trabalhar em uma freqüência específica, tornando um arquivo sysfs "scaling_setspeed", disponível no diretório do dispositivo da CPU.

2.2.4 *Ondemand*

O gerenciador cpufreq *Ondemand* (sob demanda) configura a CPU, de acordo com o uso atual. Para isso, a CPU deve ter a capacidade de mudar a freqüência muito rapidamente. Há certo número de parâmetros acessíveis no arquivo sysfs: taxa de amostragem, latência da transição, exibição da taxa de amostragem mínima, exibição da taxa de amostragem máxima, dentre outras. No momento em que a utilização da CPU passa de um limite máximo, é escolhida a freqüência máxima. Se a utilização da CPU fica menor que o limite máximo, a freqüência da CPU é reduzida suavemente.

2.2.5 *Conservative*

O gerenciador cpufreq Consevartive (conservação energética), é muito parecido com o gerenciador *Ondemand*, pois configura a CPU, dependendo do uso corrente. Difere de comportamento quanto à forma de aumentar e diminuir a velocidade da CPU, pois ao invés de saltar para a freqüência máxima no momento em que a utilização da CPU passa de um limite máximo, este gerenciador varia suavemente. Este comportamento é mais adequado a um ambiente alimentado por bateria.

O gerenciador *Conservative* está preparado da mesma maneira que o gerenciador *Ondemand* através do arquivo sysfs com a adição de freq_step que permite definir o passo de variação da freqüência entre zero e cem por cento, mas é definido por padrão em 5% e down_threshold tem a mesma que o up_threshold encontrado para o gerenciador *Ondemand*, porém em sentido oposto, ou seja, quando definide-se o valor padrão de “20" significa que o uso da CPU deve ser inferior a 20% entre as amostras para que a freqüência diminua.

2.3 Avaliação das Políticas Energéticas

Como todo produto lançado no mercado, faz-se necessário acompanhar seu desempenho e qualidade, o que não é diferente com as tecnologias. Neste caso a avaliação das políticas de gerenciamento de energia para processadores se dá por meio de comparação entre os resultados obtidos quando são utilizadas uma ou várias cargas de trabalho sob condições que possam ser controladas e repetidas. Para tanto, é comum utilizar-se cargas de trabalho desenvolvidas para fins de teste deste tipo.

2.3.1 Cargas de Trabalho

Algumas destas cargas de trabalho estão disponíveis em *Benchmark Suite* para serem executadas no Linux, a fim de que sejam definidos padrões de utilização do processador e da demanda energética. São exemplos destas: *Splash-2 Suite*, e a carga EBIZZY, que juntamente com o programa GDE, possibilitam a coleta de dados necessários à construção dos gráficos representativos da utilização do processador e da energia consumida por este no processo de avaliação das políticas, estabelecendo, portanto, uma avaliação confiável.

2.3.1.1 *Splash-2 Suite*

Neste trabalho, utilizou-se um conjunto de programas pertencentes à suíte Splash-2, os quais “representam uma variedade de cálculos em engenharia, ciência e computação gráfica” Woo et al. (1995) página 3, com o objetivo de realizar testes de utilização do processador e calcular o consumo da energia. Splash-2 que é constituído por oito algoritmos distintos, onde cada um deles simula um comportamento do evento. Dentre estes, pode-se destacar:

- Barnes – simula a interação de um sistema de corpos em três dimensões sobre uma série de intervalos de tempo;

- FMM – como Barnes, o algoritmo da FMM simula um sistema de corpos ao longo de etapas de tempo, entretanto este simula interações em apenas duas dimensões.

- Radiosidade – calcula a distribuição do equilíbrio de luz em uma cena, utilizando um método interativo. A cena é modelada inicialmente como um polígono com grande número de entradas.

- *Raytrace* - este algoritmo processa uma cena tridimensional utilizando traçado de raios. Um raio é traçado através de cada pixel no plano da imagem, e reflete de forma imprevisível os objetos que ele atinge.

2.3.1.2 EBIZZY

Este programa foi projetado para replicar uma aplicação comum de comércio eletrônico. Muitos aplicativos de busca, assim como a carga de trabalho EBIZZY têm como padrão básico: começar um pedido para encontrar um determinado registro, o endereçamento da memória que o contém, copiar em outro endereço de memória, e depois realizar uma busca binária do que é desejado.

Pode-se destacar como sendo características interessantes da carga de trabalho EBIZZY: o grande conjunto de trabalhos realizados simultaneamente, a alocação de cópias de dados, e a inprevisibilidade do acesso aos padrões de dados. A alocação de cópias de dados faz com que esta carga de trabalho tenha diversos acessos a mémoria, representando do ponto de vista da utilização do processador, períodos de maior tempo ocioso devido ao acesso à memória ser mais lento que tempo de acesso a *cache*.

3 PLATAFORMA EXPERIMENTAL

Até o momento, foi enfatizada uma abordagem teórica voltada de forma mais específica para embasar este trabalho, a fim de proporcionar ao leitor informações necessárias à realização dos estudos desenvolvidos. Contudo, é importante detalhar os recursos que foram utilizados para realizar a coleta de dados. Tal detalhamento abordará as características do processador disponível nos experimentos, os recursos de *software* do sistema operacional utilizado como ambiente de desenvolvimento do programa GDE empregado para obter informações de utilização e consumo de energia, além das políticas de gerenciamento de energia utilizadas para alterar os estados de energia do processador.

Vale salientar que para a realização dos experimentos constantes neste trabalho, foram usadas duas plataformas. A primeira, um computador de mesa, com processador Celeron D310 com 2,13GHz de freqüência de processamento, dispondo de 640MBytes de memória RAM DDR acessível a 533MHz e disco rígido de 40Gbytes ATA-100Mbps com rotação de 7200rpm para os testes de medição da utilização do processador empregando diversas cargas de trabalho. Enquanto a segunda plataforma, um computador portátil com processador Pentium M 750 com possibilidade de variar a freqüência de operação entre 1,867GHz, 1,600GHz, 1,333GHz, 1,067GHz e 0,800GHz para experimentos envolvendo alteração dos estados de energia.

O sistema operacional Linux com distribuição Ubuntu 9.04 e versão do núcleo 2.6.31-22-generic disponível para experimentação no computador de mesa, dispõe de uma política de gerenciamento de energia *Userspace* o que permite ao usuário variar a freqüência de processamento de acordo com o que se deseja verificar nos testes. Registra-se também, uma aplicação que permite ao usuário observar o histórico de uso da CPU do processador, dentre outros recursos, Figura 4.

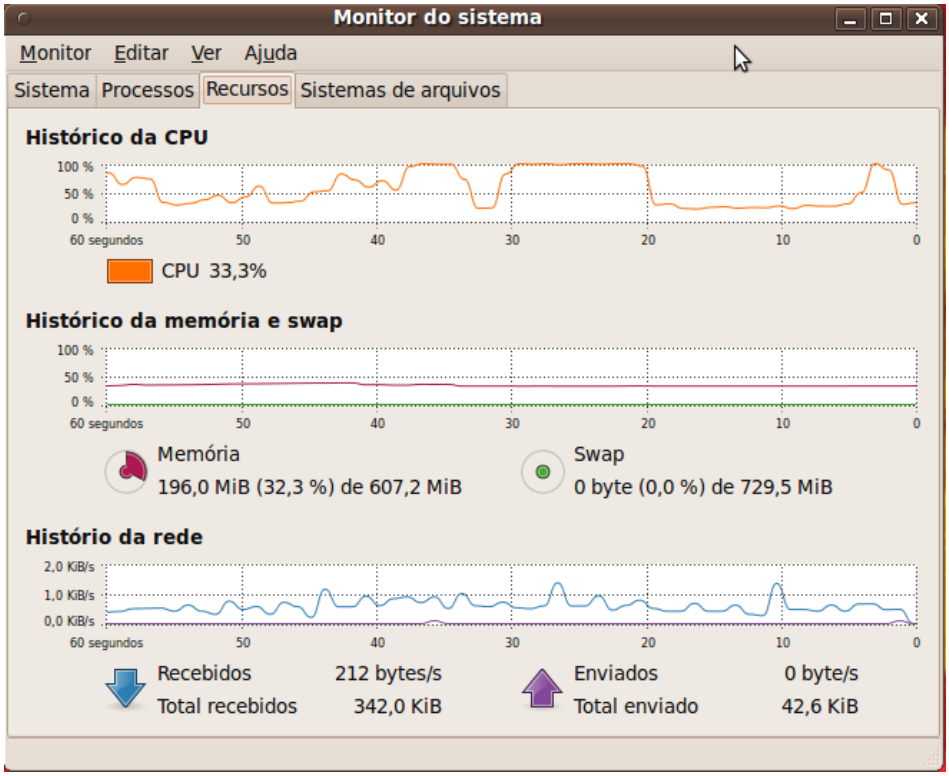

Figura 4: Monitor do Sistema do Ubuntu 9.04 permite ao usuário observar o histórico de utilização da CPU, memória, e rede.

Por outro lado, o sistema operacional instalado no computador portátil utilizado foi o Linux com distribuição Ubuntu 10.10 e versão do núcleo 2.6.35-22-generic (i686), dispondo dos mesmos recursos que a versão anteriormente citada.

3.1 Processadores

Buscando compreender a influência dos processadores utilizados em cada conjunto de experimentos e a importância que estes têm nos resultados, será realizada uma pequena apresentação destes elementos, tão relevantes no contexto deste trabalho.

Portanto, será iniciada tal apresentação com uma abordagem ao processador Celeron D310 e em seguida ao Pentium M 750 ambos da Intel, enfatizando algumas de suas vantagens, desvantagens e características.

3.1.1 Processador Celeron D310 da Intel

Neste contexto, onde se busca apresentar a plataforma experimental, faz-se necessário o conhecimento do processador Celeron D310 da Intel (*Intel® Celeron® D Processor 310 (2011)*) que apesar do seu baixo custo, executa muitos aplicativos satisfatoriamente embora apresente algumas limitações quanto ao seu desempenho ao executar aplicativos pesados (jogos, aplicações em 3D,...).

Assim, mesmo com as limitações citadas anteriormente, a família de processadores Celeron cresceu. Em junho de 2004, foi lançado o Prescott-256 Celeron D, com um *cache* L2 de 256kbytes, baseado no núcleo Prescott do Pentium 4 que dispõe de um barramento de 533MHz.

Tais características viabilizaram que os experimentos realizados com este processador focassem em medição da utilização deste, pois ele foi projetado para obter máximo desempenho, impossibilitando ao usuário a variação de sua freqüência de operação. Assim é definida na especificação do processador D310 a freqüência de processamento como sendo de 2,13GHz, contudo, a letra D não se refere à existência de dois núcleos.

3.1.2 Processador Pentium M 750

O Intel Pentium M 750 (*Intel® Pentium® M Processor 750 (2011)*), que busca otimização para um consumo eficiente de potência, e emprega a tecnologia SpeedStep discutida na seção 2.1.1 deste trabalho. Esta tecnologia proporciona um aumento na durabilidade da bateria dos computadores portáteis nos quais este processador está instalado.

O Pentium M 750 pertence à segunda geração dos processadores Pentium M, a qual chamou-se de “Dothan” que teve avanços em relação a primeira geração por ter sido desenvolvido com transistores de 90nm. Devido a isto foi possível dobrar a *cache* L2 para 2MBytes sem com isto alterar significativamente o tamanho do *chip* que consome de 21 a 27Watt dependendo da freqüência de operação de processamento. Pode-se registrar, que este processador possui ainda, uma comunicação de barramento de 533MHz.

A opção de variar freqüências de operação deste Pentium M 750 tornou-se bastante conveniente para a realização dos ensaios que visavam alteração do estado de energia do processador, bem como propor e testar uma política de gerenciamento de energia (*Userspace*) a partir da freqüência de operação e comparar com aquela instalada automaticamente no sistema operacional (*Ondemand*).

3.2 Medição da Utilização do processador

Durante o processo em que se realizou a coleta de dados e a medição da utilização do processador, obtiveram-se as informações contidas no /proc/stat do núcleo do Linux, através de chamadas do programa GDE para em seguida calcular a utilização, pois desta forma a estatística obtida do /proc/stat fornece o tempo ocioso do processador juntamente com outras informações.

Para esclarecer como se dá as relações entre espaço de usuário, núcleo do Linux e o processador, é necessário estabelecer que espaço de usuário é o ambiente ao qual o operador tem acesso para trabalhar e realizar suas tarefas ou executar programas de seu interesse. Núcleo do Linux é o centro de armazenamento das informações e de controle das tarefas requisitadas pelo usuário ordenando as tarefas a fim de não ocorrer conflitos entre as ordens, assegurando, assim, que o processador execute os serviços.

Na Figura 5, é possível observar que o espaço de usuário contém todos os outros blocos, ou seja, tudo que será feito é devido a requisições de serviços oriundas do usuário ou sistema operacional onde estão as aplicações deste. Para compreender as relações entre as partes de interesse dos experimentos é pertinente entender como os blocos se relacionam e quais suas funções.

O bloco RS realiza a requisição de serviços que são alocadas em uma fila de serviços, FS, no interior do núcleo do Linux. A fila escalona as requisições de modo a serem atendidas, evidentemente isto é uma idealização, a fim de simplificar a compreensão. Por sua vez a fila comunica-se com o provedor de serviços (PS), ou seja, o processador, que atende as requisições advindas da fila de serviços. Este

por sua vez, poderá concluir a tarefa ou, caso a requisição não tenha sido totalmente processada esta retorna a fila. O processador, enquanto realiza as tarefas, se comunica com o bloco /proc/stat que é um arquivo que armazena o estado de contadores os quais são função de uma variedade de estatísticas acerca do processador desde a última reinicialização do computador.

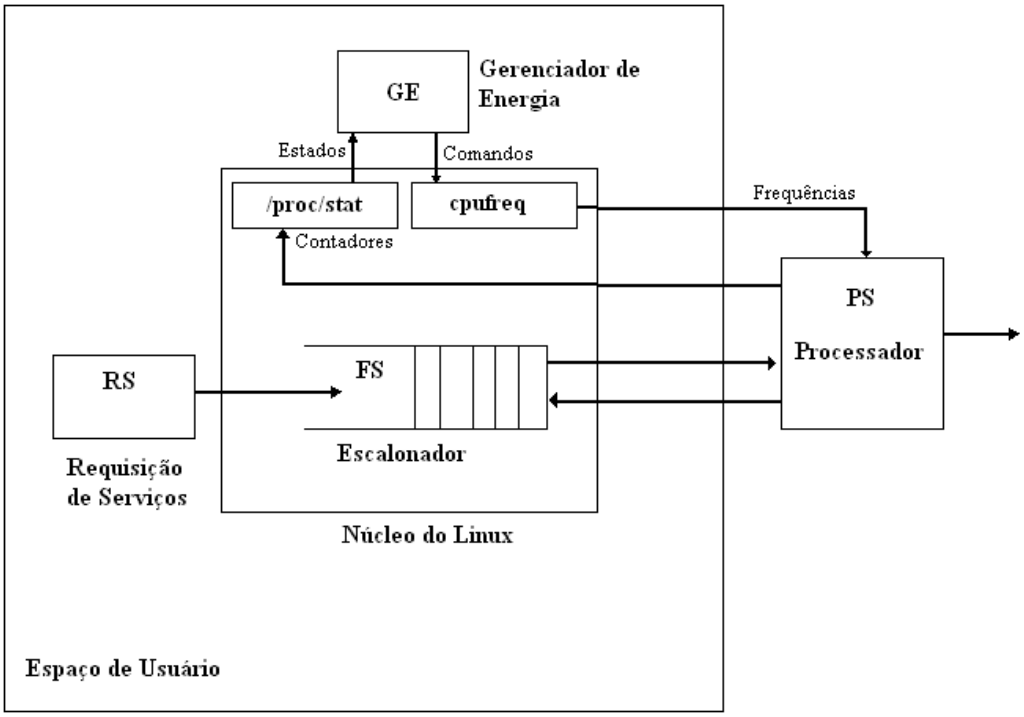

Figura 5: Relação entre espaço de usuário, núcleo do Linux e o processador.

Continuando com os esclarecimentos da Figura 5, o estado do /proc/stat é lido pelo gerenciador de energia, que com estas informações calcula a nova freqüência de operação do processador e também envia o comando ao bloco cpufreq que, por sua vez, altera a freqüência de operação do processador.

No /proc/stat as estatísticas são armazenadas em linhas sucessivas, onde a primeira registra o somatório das estatísticas de todas as CPU's, caso o processador seja multi-núcleo. As linhas seguintes discriminam as estatísticas de cada CPU identificando-as de "cpu0" a "cpuN-1", como exemplificado na Figura 6.

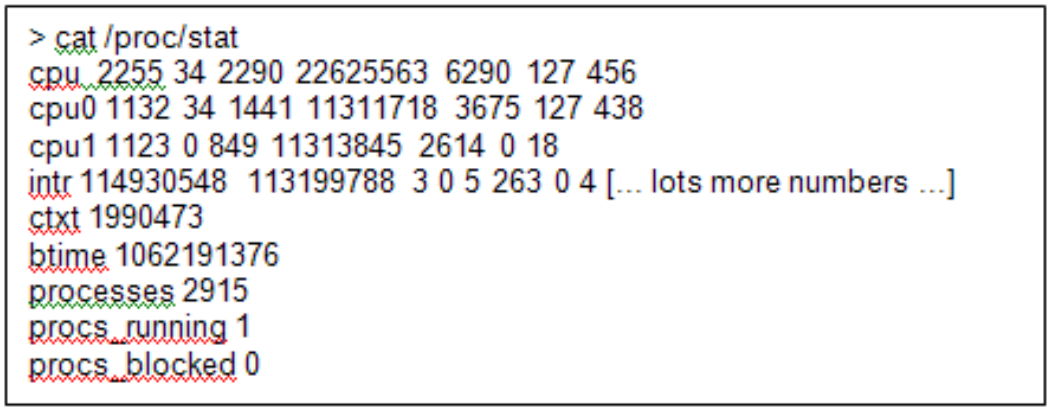

Figura 6: Ilustra as estatísticas armazenadas no /proc/stat do núcleo do Linux

Além disto, saber o significado dos valores em cada coluna nas linhas “cpu” a “cpuN-1” é importante, pois, a partir destas estatísticas serão realizadas as medições da utilização do processador e também os testes referentes ao estado de energia. Considere que a seqüência da esquerda para a direita na Figura 6 da linha “cpu” é a mesma que segue:

- <usuário>: tempo de processos executando em modo usuário;
- <nice>: tempo de processos de usuário em modo de baixa prioridade;
- <sistema>: tempo de processos em execução do próprio sistema;
- <ocioso>: tempo de CPU ocioso;
- <espera I/O>: tempo de sistema em espera por entradas ou saídas;
- <irq>: tempo de interrupções de serviço de *hardware*;
- <softirq>: tempo de interrupções por *software.*

Desta maneira, o /proc/stat armazena os dados em forma de matriz, onde as primeiras linhas mostram o resultado de cada núcleo do processador em aproximadamente oito colunas, portanto as chamadas de GDE são feitas em intervalos de tempo de um segundo para montar uma matriz onde o número de colunas é fixo e o número de linhas é definido quando o usuário estabelece quanto tempo o processo de coleta permanecerá sendo executado pelo GDE que por sua vez é executado em espaço de usuário.

Com base nos dados obtidos do núcleo do Linux, o GDE realizará o cálculo do tempo de CPU inativa e depois do tempo de CPU ativa, ou seja, quanto tempo a unidade de processamento realmente ficou inativa e quanto tempo esta, de fato, ficou em atividade. A equação (2) é utilizada para realizar o cálculo de tempo de CPU inativa, enquanto a equação (3) destina-se a estimativa do tempo de CPU ativa.

$$CPU_{Inat}(t) = \frac{i(t) - i(t-1)}{s(t) - s(t-1)} \tag{2}$$

$$CPU_{Ativa}(t) = 1 - CPU_{Inat}(t) \tag{3}$$

Onde: $CPU_{Inat}(t)$ – Razão entre Tempo de CPU inativa e Tempo total;

$CPU_{Ativa}(t)$ – Razão entre Tempo de CPU ativa e Tempo total;

$i(t)$ – Contador do tempo de CPU ocioso <ocioso>;

$s(t)$ – Tempo total de utilização da CPU: soma dos contadores <usuário>, <nice>, <sistema>, ... , <softirq>.

Como pode ser constatado no trecho de código extraído do programa GDE, Figura 7.

```
  /*Sensor atividade do processador */
  estatisticasProcessador (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]);
  }

  inativaPercentual[j] = sinal_i[j];

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

Figura 7: Trecho de código extraído do Anexo A onde é calculada a utilização Percentual da CPU.

3.3 Alteração dos Estados de Energia do processador

Nesta parte do trabalho serão discutidos os recursos empregados para promover alterações de estados de energia e as políticas utilizadas para realizar os experimentos, além de discorrer brevemente sobre as ferramentas empregadas para propor uma política de gerenciamento dinâmico de energia alternativa àquelas existentes a fim de avaliar o desempenho obtido em ambos os casos.

Considerando na Figura 5, o bloco do gerenciador de energia GE, percebe-se que este recebe informações de estados a partir do núcleo do Linux e envia comandos para a interface *cpufreq* que controla as variações dos estados de energia do processador. Então, alterando a política instalada no gerenciador de energia, pode-se alterar o desempenho e o consumo de energia. Inicialmente foi realizada a coleta de dados de entrada e saída utilizando um vídeo como carga de trabalho submetido à gestão energética do próprio computador que estava configurado para utilizar a política de gerenciamento de energia *Ondemand.*

Evidentemente a política de conservação de energia implementada é bastante eficiente, tornando difícil a tarefa de conseguir utilizar alguma política que seja ainda melhor que esta. Contudo foi realizada a tentativa de implementar uma política destinada a uma carga de trabalho específica pode obter bons resultados, visto que a política automática é genérica para aplicações diversas.

A fim de obter os dados com informações necessárias à análise do comportamento de consumo de energia, dispõe-se do programa GDE, cujo código está em Anexo A, executando a carga de trabalho. De posse das informações coletadas, estas foram armazenadas para ser analisadas, desta vez, fazendo uso do *software* MatLab 7.7.0 (R2008b) para construir os gráficos das curvas de utilização do processador, do consumo de energia, dentre outras. Além disto, utilizou-se a ferramenta de identificação de sistemas, Figura 8, disponível em seu *Toolbox* para identificar e validar o modelo da política que será proposta e testada.

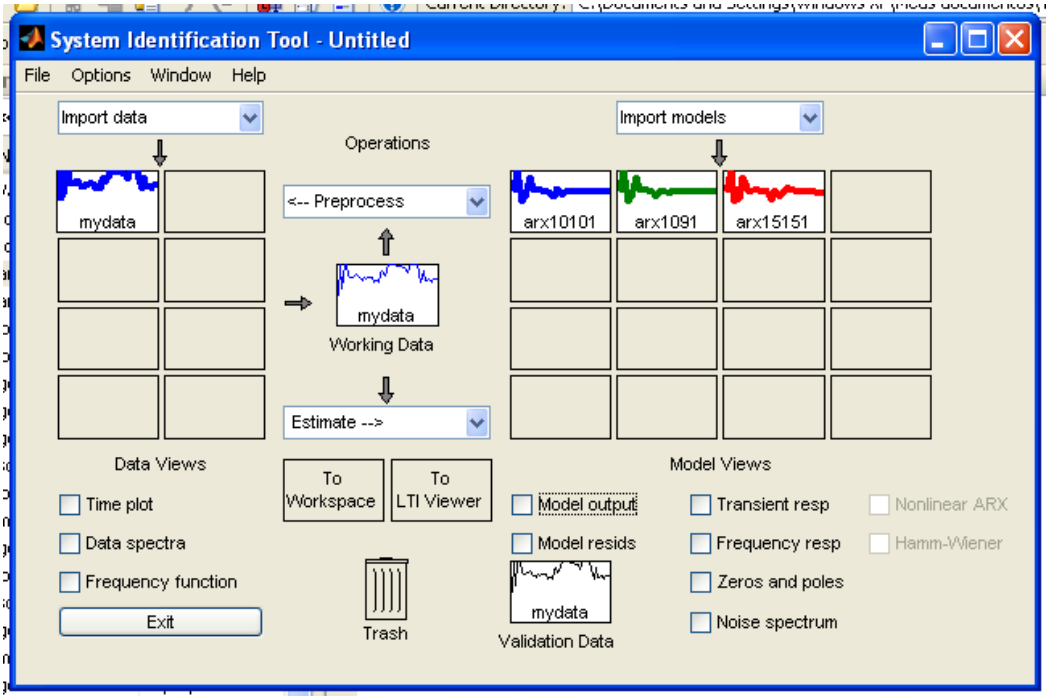

Figura 8: Ferramenta de Identificação de Sistemas do MatLab 7.7.0 (R2008b).

Mas o que se deseja ao propor esta política de gerenciamento de energia? Considerando-se, portanto, uma entrada u(t), igual à razão entre a freqüência anterior e a freqüência atual do processador, empregada como variável controlável. Espera-se que a saída do sistema y(t), ou seja, a utilização do processador, apresente uma tendência de acompanhar o comportamento da entrada. Entretanto, sabe-se que a saída presente depende de seus estados passados e dos valores de entrada. Assim, considerando a entrada u(t) dada na equação (4).

$$u(t) = \frac{f_{anterior}}{f_{atual}} = \frac{f(t-1)}{f(t)} \tag{4}$$

Na saída y(t), os estados do sistema, no instante posterior dependerá dos estados passados e da entrada como na equação (5), onde “a” e “b” são coeficientes. Após substituir u(t) na equação (6), a qual evidencia as dependências da saída dos estados, chega-se a equação (7) onde se lê que a saída do estado seguinte do sistema é função do estado presente, da freqüência atual e da freqüência anterior do processador quando se está variando os estados de energia de processamento.

$$y(t+1) = ay(t) + bu(t) \tag{5}$$

$$y(t+1) = ay(t) + b\frac{f(t-1)}{f(t)} \tag{6}$$

$$y(t+1) = F(y(t), f(t), f(t-1)) \tag{7}$$

Porém, o modelo obtido com utilização da ferramenta de identificação do MatLab fornece um modelo discretizado, conforme equação (8), onde A(q) e B(q) são matrizes linha de coeficientes.

$$y(t) = \big(1 - A(q)\big)y(t) + B(q)u(t) \tag{8}$$

Esta por sua vez pode ser colocada na forma da equação (9) a qual será representada até a terceira ordem apenas, porém a ordem é definida no momento da identificação.

$$y(t) = -a_1 y(t-1) - a_2 y(t-2) - a_3 y(t-3) + b_1 u(t-1) + b_2 u(t-2) + b_3 u(t-3) \tag{9}$$

Tendo sido esta a forma inserida no programa GDE como uma nova política, que se completa ao serem consideradas duas coisas. A primeira é que a variável de controle na equação (9) é u(t – 1), e a segunda é que na pratica a variação de freqüências deve respeitar uma restrição de desempenho, L, imposta pelo usuário conforme Figura 9. Considerando-se que a estimativa da saída y(t) no instante t é máxima para a freqüência mínima permitida F_{min}, e mínima para a freqüência máxima permitida F_{max}, deseja-se encontrar a menor freqüência f_0, tal que y(t) é menor ou igual à restrição de desempenho, L.

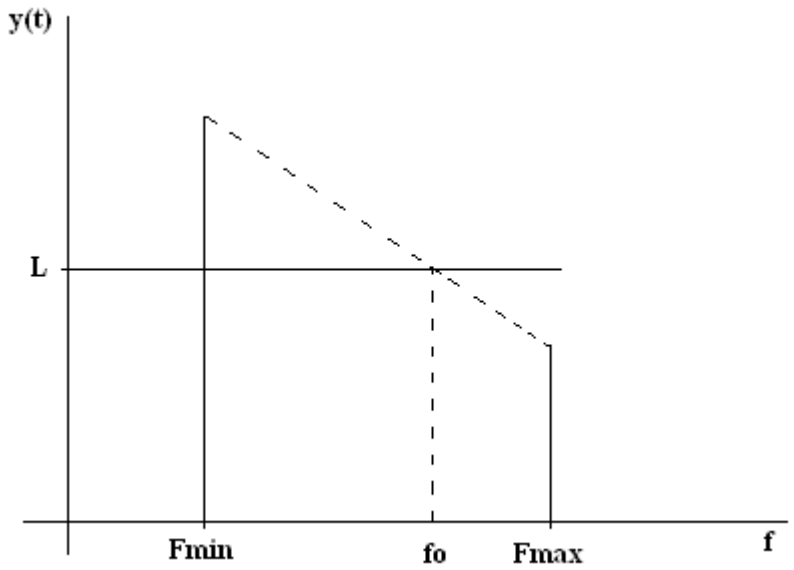

Figura 9: Esboço da relação do desempenho do sistema mediante uma política cuja variável de controle é a freqüência respeitando uma restrição L.

Agora que as plataformas experimentais já foram apresentadas, bem como aquilo que é esperado de cada uma delas, pode-se iniciar a apresentação dos resultados obtidos nos experimentos, tanto aqueles destinados à verificação da utilização do processador quanto aqueles destinados a observar e propor a eficiência das políticas de gerenciamento de energia utilizadas neste trabalho.

4 RESULTADOS EXPERIMENTAIS

Neste capítulo serão apresentados os resultados obtidos nos experimentos realizados durante a construção deste trabalho. Tais experimentos tiveram duas linhas de investigação principais, a saber: verificação e medição da utilização do processador e aferições dos estados de energia do processador frente à variação da freqüência de operação, utilizando estes resultados para propor uma política alternativa de energia para a carga de trabalho utilizada, em seguida comparando os resultados desta política proposta com a política de gerenciamento de energia *Ondemand.*

4.1 Experimentos de Medição de Utilização do Processador

Esta série de experimentos foi realizada na plataforma cujo processador é o Intel Celeron D310, o qual possui a freqüência de operação constante e igual a 2,13GHz, o qual tornou-se, no contexto deste trabalho, um ambiente propício à medição da utilização do processador. A utilização do processador é neste trabalho denominada a relação entre o tempo em que este permaneceu ativo e o tempo total, como na equação (3). Contudo, o parâmetro aferido é o tempo de CPU inativo, equação (2). Estes dados são coletados com o programa GDE, cujo período de amostragem estabeleceu-se em um segundo, e armazenados em arquivo chamado “log.txt” que possibilita ser lido e tratado utilizando uma aplicação de análise desenvolvida em MatLab e, com este, montar gráficos de utilização para as várias cargas de trabalho.

Inicialmente, realizou-se a coleta de dados sem cargas de trabalho, ou melhor, executando apenas as aplicações indispensáveis ao sistema operacional Linux com distribuição Ubuntu 9.04 e versão do núcleo 2.6.31-22-generic, além do próprio programa para inferência da utilização GDE, a fim de confirmar que a influência desta é reduzida. No Gráfico 1, pode-se verificar a atividade da CPU sem emprego de cargas de trabalho, cuja média é de aproximadamente 3%, portanto consideraremos este o referencial para as medições.

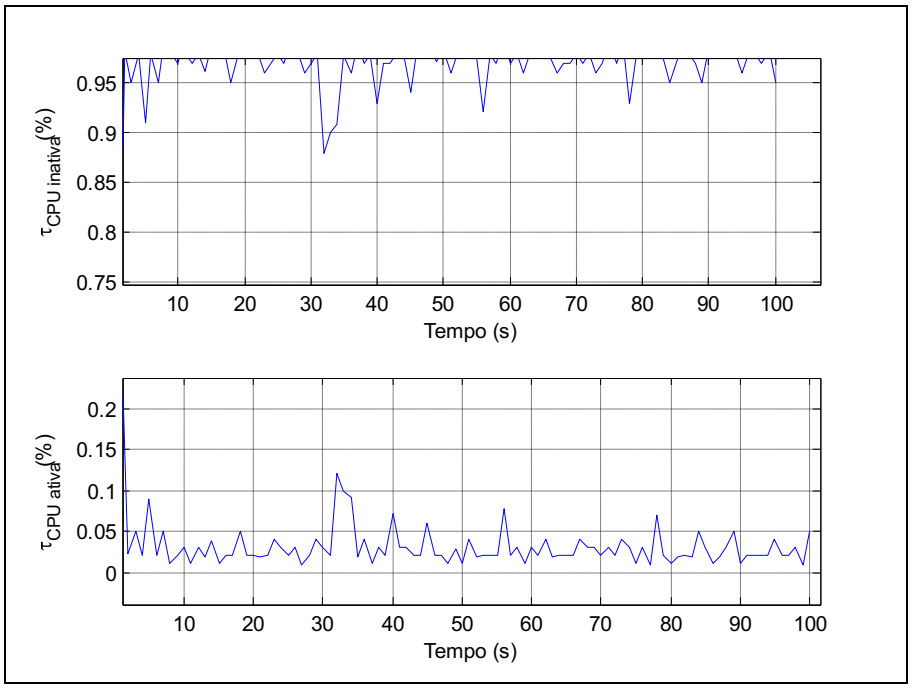

Gráfico 1: Atividade de CPU sem utilização de carga de trabalho.

Aproveitando a apresentação do Gráfico 1, é conveniente, neste momento, estabelecer a seguinte padronização: todos os gráficos desta seção serão apresentados aos pares, sendo que o elemento gráfico localizado acima, refere-se ao tempo de CPU inativa, ao passo que o elemento gráfico situado abaixo, refere-se ao tempo de CPU ativa.

Na seqüência, foi utilizada a carga de trabalho EBIZZY, projetada para simular o comportamento de um tipo de comércio eletrônico de larga escala, ou seja, é uma carga que apresenta grande variação de requisições por unidade de tempo podendo ser, em alguns momentos, baixa e em outros tomar totalmente o tempo do processador, ou seja, espera-se um comportamento bastante irregular de atividades.

Iniciado o processo de aquisição de dados, obteve-se o comportamento apresentado no Gráfico 2, que mostra-se bastante incomum, pois em seus instantes iniciais, de cinco a vinte segundos, tem uma característica de acessos com pequena variação permanecendo próximo a 56% de atividade. Mas à medida que o tempo passa, estabelece-se um padrão com variação mais intensa, contudo o valor médio permanece o mesmo.

No Gráfico 3, também foi utilizada a carga de trabalho EBIZZY, entretanto só foi realizada a aquisição de dados após a estabilização do sistema de acessos o qual a carga de trabalho simula. Com isto, o comportamento em regime permanente da atividade do processador está estabelecido, onde verifica-se o valor médio de 57%. Mesmo assim, registra-se em vários instantes a utilização máxima do processador.

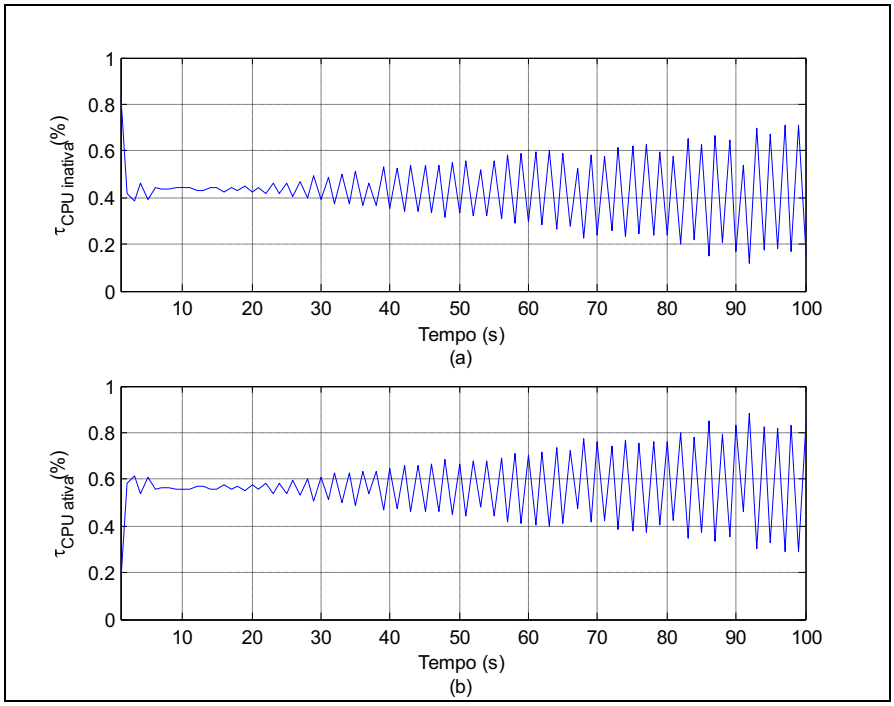

Gráfico 2: Inicialização da Carga de trabalho EBIZZY.

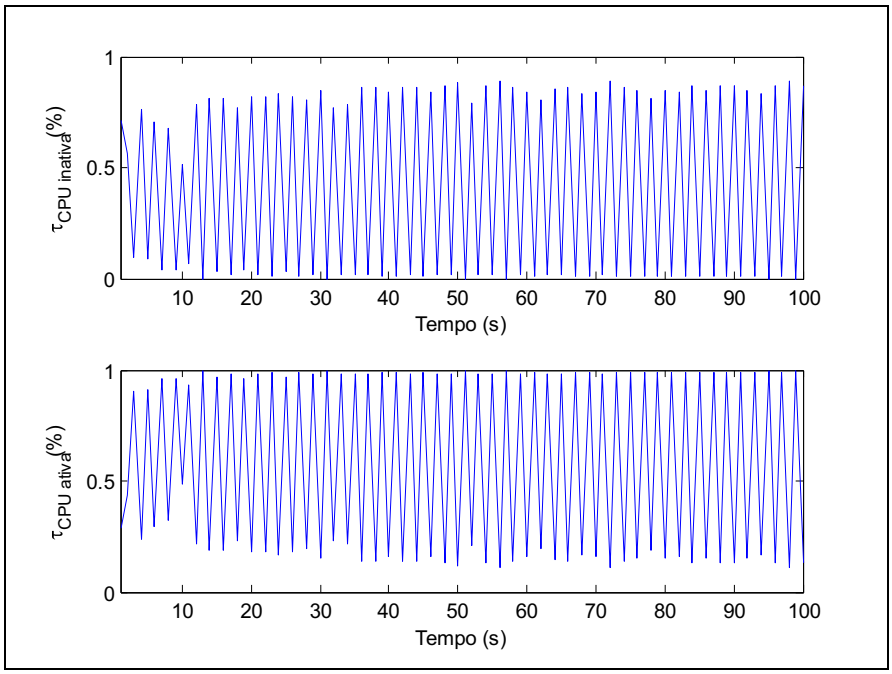

Gráfico 3: Atividade do Processador com carga EBIZZY em regime permanente.

Continuando com o processo de medições da utilização do processador Celoron D310, realizou-se a chamada de uma carga preparada (Anexo D) para exigir maior atividade da CPU do computador de mesa, além de apresentar um padrão de repetição da atividade a partir de um período estabelecido pelo usuário. Assim, caso o operador utilize valores demasiadamente grandes, poderá não ser possível identificar a periodicidade, pois o GDE coleta as amostras durante cem segundos, utilizado como padrão para este trabalho.

Neste experimento, obteve-se uma média de tempo de CPU ativa de 79%, que é uma boa utilização, mas para cargas com padrão de utilização alta ocorre um maior consumo de energia devido ao reduzido tempo em que o processador esteve inativo.

Contudo, a periodicidade da carga não fica evidenciada no Gráfico 4, certamente devido ao ensaísta ter utilizado um período de repetição consideravelmente maior que o período utilizado no programa de gerenciamento dinâmico de energia, GDE, o que não invalida o experimento por ter possibilitado o verificação do comportamento do sistema sob demanda elevada de trabalho.

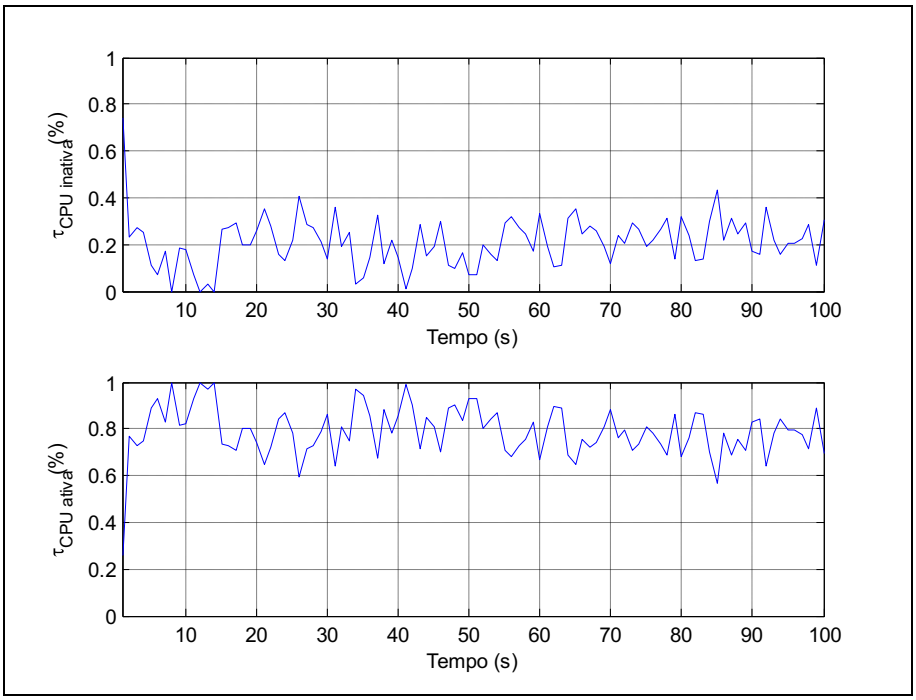

Gráfico 4: Utilização do processador Celeron D310 submetido a carga de trabalho Chamar Carga Alta Periodicamente.

Nesta seção foram então apresentados diversos resultados de utilização do processador, considerando que estes foram os mais relevantes por permitirem ver simulações de situações reais de utilização do processador, tais como: sem carga de trabalho, uma carga de trabalho que simula um sistema de comércio eletrônico, tendo esta, dois momentos distintos, o momento da inicialização e em regime permanente de acessos e saídas de clientes, além de uma carga com forte demanda de utilização do processador. Esta última carga causa no processador um consumo mais elevado de energia devido ao reduzido tempo de CPU inativa que permite a entrada em modo de baixo consumo de energia.

Na seção que segue serão realizados experimentos com alteração do estado de energia através da variação da freqüência de operação do processador e a propositura de uma política para uma carga de trabalho e comparar-se-á com a política de gerenciamento de energia *Ondemand* que se mostra bastante eficiente na conservação de energia.

4.2 Experimentos de Alteração dos Estados de Energia do Processador

Para realizar as atividades desta parte do presente trabalho, empregou-se o processador Intel Pentium M 750 por permitir variar a freqüência de operação e conseqüentemente seu estado de energia. Utilizando novamente a aplicação em linguagem C, GDE, para nova coleta de informações e alterar as freqüências de trabalho para um vídeo como carga de trabalho. Tal coleta permitiu realizar estudos mais detalhados que simples observações da utilização do processador. Dentre estes, a identificação de um modelo de política de gerenciamento de energia, seguida da validação e subseqüente implementação desta política para comparar com uma política de gerenciamento mais sofisticada como a *Ondemand*.

O procedimento inicial de coleta de dados utilizando o GDE é o mesmo do empregado na Seção 4.1, ou seja, executa a aplicação GDE, com período de amostragem de um segundo, enquanto é executada paralelamente a carga de trabalho, neste caso, um vídeo. Durante a execução do GDE, Anexo A, abre-se o arquivo “log.txt” onde serão armazenados os valores lidos pelo programa. Este procedimento repetiu-se até que se tivesse um número suficiente de arquivos

“logN.txt”, assim renomeado para que os novos dados não se sobrepusessem aos obtidos das vezes anteriores.

Com este arquivo “logN.txt” foi possível construir um conjunto de gráficos com a média das várias informações pertinentes ao computador portátil com o qual se fizeram os experimentos utilizando o processador Intel Pentium M 750. Obtendo, portanto, o Gráfico 5 no qual são apresentadas as seguintes curvas em relação ao tempo em segundos:

- f(GHz) – freqüência de operação em gigahertz;
- $\tau_{CPU\ at}$(%) – Tempo de utilização ativa do processador;
- T_{CPU}(C) – Temperatura do processador em °C;
- I_B(mA) – Corrente da bateria em mA;
- P_B(mW) – Potência da bateria em mW;
- Q_B(mAh) – Capacidade restante da bateria em mAh.

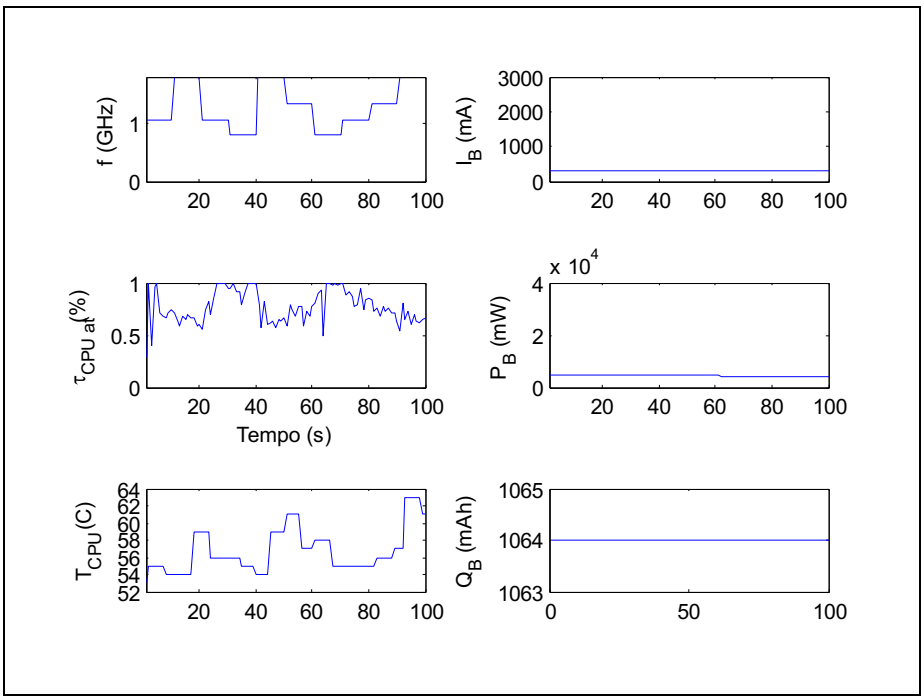

Gráfico 5: Análise do consumo de energia para um computador portátil executando um vídeo.

Analisando o Gráfico 5, percebe-se algumas situações bem características, por exemplo, nos instantes iniciais a temperatura do processador está em torno de 54 °C, com freqüência de 1,067GHz. Nesta situação é exigida uma maior utilização do processador, ou seja, quando o processador estiver em modo de economia de energia tem uma redução de desempenho, por outro lado, nos instantes em que a freqüência é maior, modo de maior desempenho e maior consumo de energia, a utilização do processador diminui e a temperatura aumenta indicando maior dissipação de calor.

As curvas de corrente, potência e capacidade restante da bateria mantêm-se praticamente constantes devido ao fato de que no momento em que os ensaios foram realizados o computador portátil estava sendo alimentado pela rede de energia.

Considere, agora, que a entrada do sistema é a variável u como na equação (4), tendo condição inicial de freqüência máxima igual a 1,867GHz. Como saída, y, tempo de utilização ativa do processador, conforme equação (7), permitindo montar o Gráfico 6.

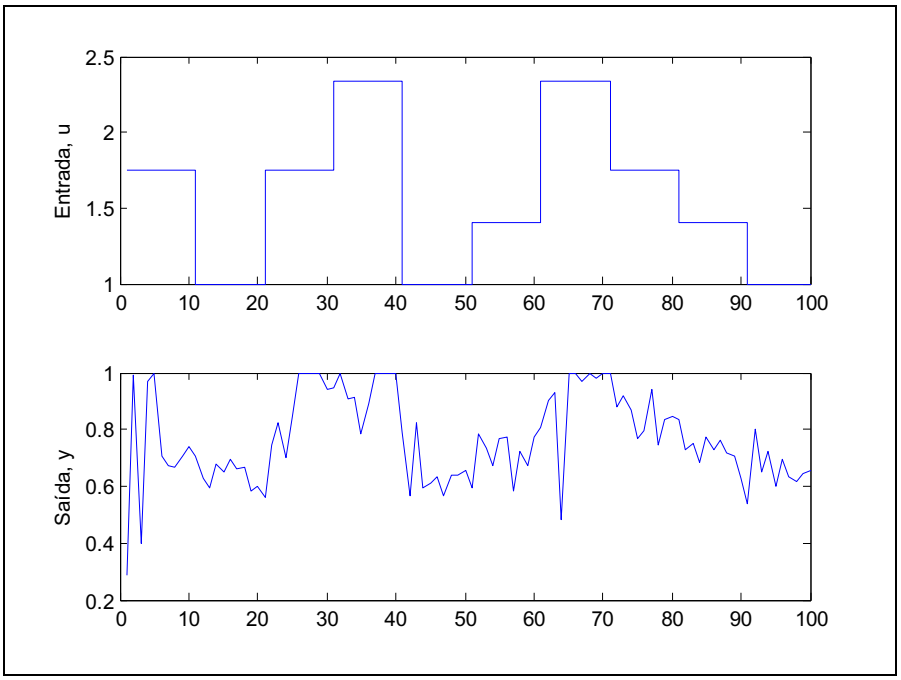

Gráfico 6: Utilização do processador, y, em função da relação das freqüências anterior e atual, u.

Após fazer uma relação entre a utilização do processador e as freqüências anterior e atual, decidiu-se por utilizar uma ferramenta matemática para realizar a identificação de um polinômio e com este, propor uma política de gerenciamento de energia para aquela carga de trabalho a fim de proporcionar maior economia de energia.

A ferramenta utilizada para realizar a identificação foi o sistema de identificação do MatLab, Figura 8, por ser um sistema robusto e capaz de realizar, também a validação de maneira simples e eficiente. Esta ferramenta permite que diversos testes sejam feitos com métodos diferentes, contudo, optou-se pelo método dos mínimos quadrados.

No entanto, para identificar o modelo da política no domínio do tempo, utilizou-se um polinômio de terceira ordem, cuja modelo geral é mostrado na equação (8) com o qual se obtêm os valores dos coeficientes “a” e “b” para diferentes ordens de polinômios, Gráfico 7. Ao lado deste gráfico há um quadro indicando a semelhança percentual entre a curva de cada ordem e a curva original em preto. Verifica-se que, aumentando a ordem, melhora-se a semelhança percentual. No entanto, a ordem 3 já apresenta uma semelhança percentual suficiente para o problema deste trabalho. De posse destes coeficientes e realizando algumas manipulações matemáticas, chegou-se ao modelo da equação (9). Assim, o modelo resultante do procedimento de identificação é mostrado na equação (10).

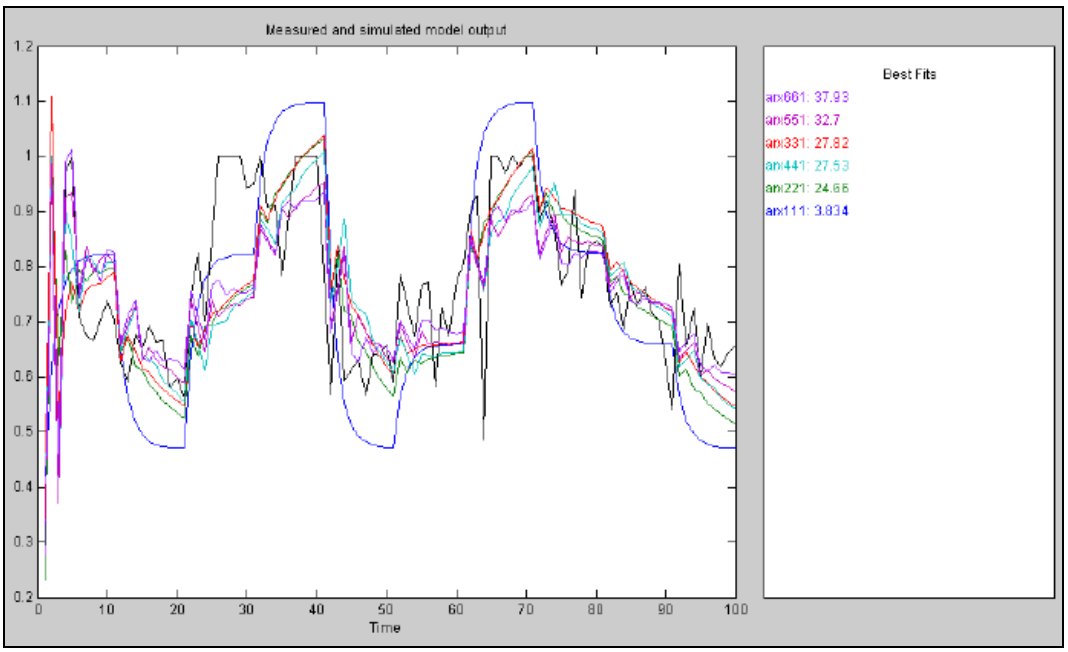

Gráfico 7: Identificação da Política e Utilização Real da Utilização.

$$y(t) = 0{,}2962y(t-1) + 0{,}2803y(t-2) + 0{,}2111y(t-3) + 0{,}218u(t-1) - 0{,}1203u(t-2) + 0{,}004112u(t-3) \quad (10)$$

De posse deste modelo, fez-se necessário validar a política proposta. Desta maneira, cumpriu-se o procedimento de validação, carregando no sistema de identificação do MatLab um novo conjunto de dados obtidos em uma das amostragens realizadas, desde que este fosse diferente daquele empregado durante a identificação, o que possibilitou a construção do Gráfico 8, onde é mostrada a curva de utilização real do processador e a curva construída a partir do modelo proposto.

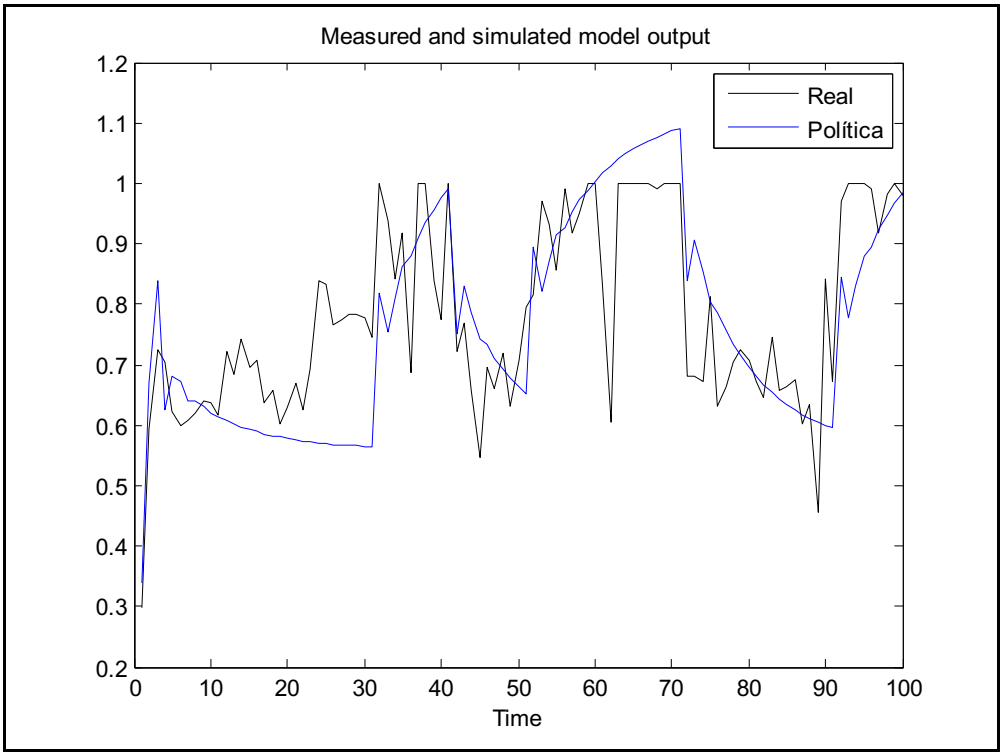

Gráfico 8: Validação da Política e Utilização Real da Utilização

Validado o modelo da política de gerenciamento dinâmico de energia, este foi implementado em linguagem de programação C no GDE, para realizar os testes finais desta série de experimentações que consistiu em executar a carga de trabalho estudada, submetendo-a a política aqui proposta e posteriormente, comparar os resultados com a política de gerenciamento *Ondemand*.

Em busca de obter maior credibilidade à comparação entre as políticas de gerenciamento dinâmico de energia, utilizou-se a montagem onde é apresentado o diagrama elétrico, Figura 10. Nesta, evidencia-se a utilização de um módulo de aquisição de dados NI USB-6210 com canais ligados a um divisor de tensão e um resistor shunt conectados aos terminais da fonte do computador portátil. Os dados coletados pelo módulo são enviados a um computador de mesa com o programa LabVIEW instalado, e cuja montagem pode ser vista na Figura 11, que por sua vez apresenta os resultados de corrente e tensão em tempo real, ao mesmo tempo que armazena as informações de corrente e tensão.

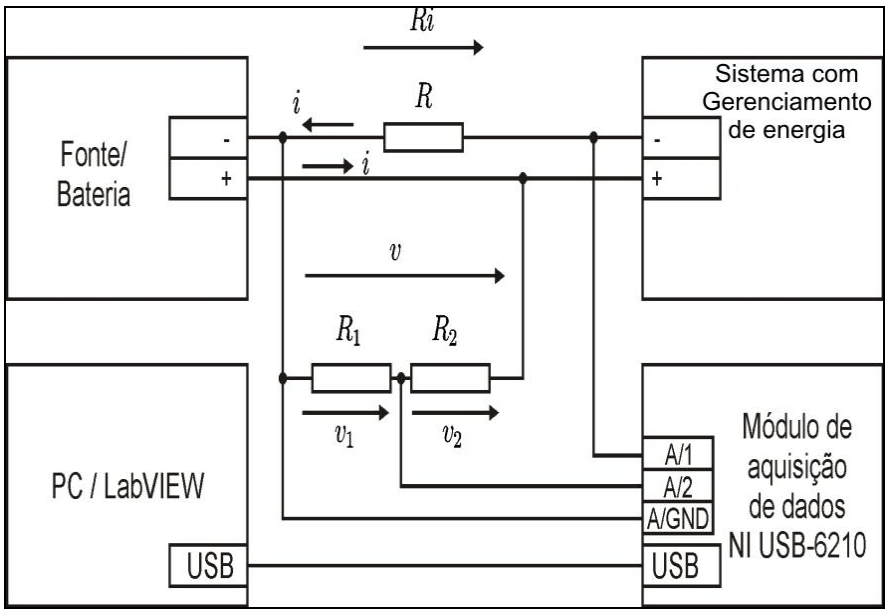

Figura 10: Diagrama elétrico para medição de tensão e corrente do sistema com gerenciamento de energia através do módulo de aquisição de dados NI USB-6210.

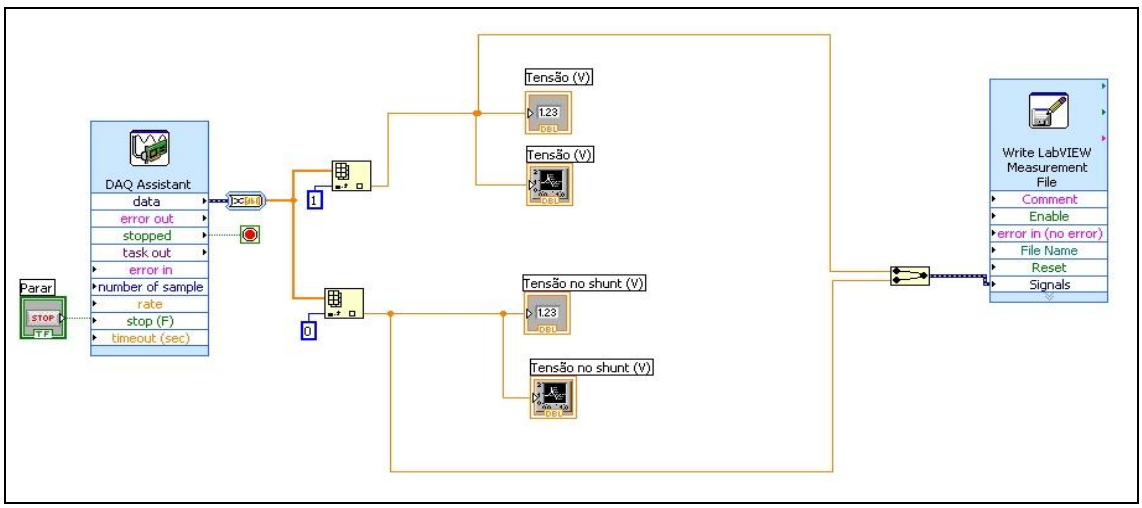

Figura 11: Diagrama de blocos no programa LabVIEW.

Com este esquema montado, iniciaram-se os experimentos utilizando a política proposta executando o GDE juntamente com a estrutura descrita na Figura 10. Os testes tiveram dois focos: alterar o estado de energia através da variação de freqüência e alteração dos estados de energia considerando uma restrição de desempenho mínimo.

Observa-se no Gráfico 9, que quando se processa os serviços com uma freqüência menor, será demandada uma maior utilização do processador, ao passo que quando a freqüência cresce a utilização decai. A potência, ao contrário, cresce com o aumento da freqüência. Além disto, no mesmo gráfico, constata-se que ocorre uma maior penalidade de desempenho quando a freqüência for 0,800GHz, indo a zero quando a máxima freqüência for atingida.

Comparando então, com a *Ondemand,* pode-se destacar que no tocante a análise do foco que envolve a variação de freqüência, objetiva-se menor utilização do processador média, sendo assim a política proposta mostrou-se satisfatória, pois na média mostrou-se 3% menor, aproximadamente.

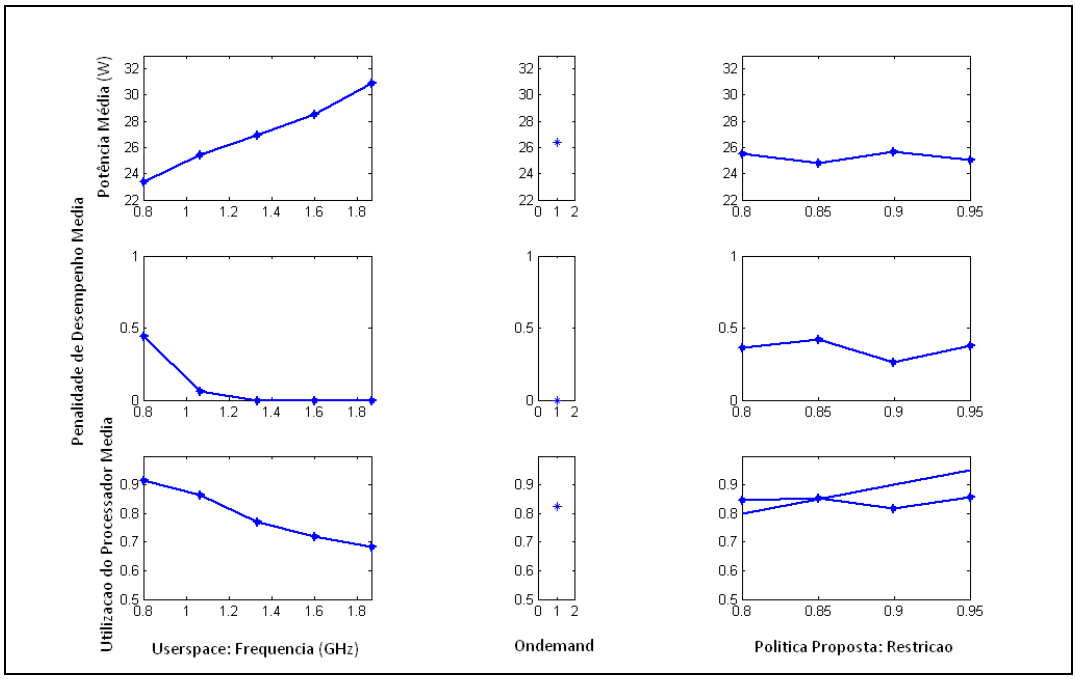

Gráfico 9: Comparação da política proposta e a *Ondemand.*

Prosseguindo com a análise dos testes, do ponto de vista da restrição de desempenho, verifica-se que variando a restrição de 0,8 a 0,95 obteve-se utilização do processador média de 84%, potência média 25,3W e penalidade de 0,36.

Entretanto, a política de gerenciamento de energia *Ondemand*, utilizada como referência, apresenta 82%, 26,3W e 0 para utilização de processador, potência e penalidade de desempenho média respectivamente.

Os resultados acima apresentados foram construídos, após o término de todos os experimentos propostos para este trabalho, portanto, o registro dos dados favoreceu a construção de gráficos, figuras e análise de informações que a partir dos referenciais teóricos conduziu às conclusões que serão descritas no capítulo seguinte.

5 CONCLUSÕES E TRABALHOS FUTUROS

O gerenciamento dinâmico de energia tem sido bastante estudado sob as mais diversas óticas, e por esta razão vem apresentando uma evolução constante em todas suas vertentes. Sua importância cresce dia-a-dia concomitantemente com o surgimento dos mais variados dispositivos eletrônicos portáteis e da crescente necessidade da redução do consumo de energia e ampliação da autonomia destes.

Diante deste desafio para a ciência e para a engenharia, no presente trabalho, propôs-se a estudar o comportamento dos processadores frente a um conjunto de cargas de trabalho através de medições de variáveis de tempo necessárias ao estabelecimento de sua utilização.

Assim, os experimentos apresentados ao longo deste texto foram realizados utilizando duas plataformas de experimentação distintas. A primeira foi utilizada para medir a utilização do processador a partir dos tempos de CPU inativa e de CPU ativa. A segunda plataforma foi utilizada com a finalidade de propor uma política de gerenciamento de energia para uma carga de trabalho específica.

Na seção 4.1, os resultados mostraram que a utilização do processador quando não havia cargas de trabalho em execução, ou seja, quando estava rodando apenas as funções imprescindíveis do sistema operacional, era de aproximadamente 3% na média. Esta constatação permitiu que tal contribuição pudesse ser desprezada nos demais experimentos.

Em seguida, os resultados do tempo de CPU ativa para a carga EBIZZY, a qual simulou um sistema de comércio eletrônico, teve média de 57%, com muitos picos onde a utilização foi máxima. Este resultado não era esperado visto que normalmente a carga EBIZZY faz com que o processador tenha utilização total por longos intervalos de tempo.

A carga de trabalho que ocupou o processador por mais tempo foi Chamar Carga Alta Periodicamente, Anexo D. Onde o tempo de CPU ativo ficou em torno de 80%, e por esta razão também foi aquela que mais consumiu energia durante o processamento. Porém, não se registrou nas amostras o comportamento periódico

desta carga. O que ocorreu devido a grande diferença de tempo estabelecido para a periodicidade e o de coleta de dados utilizado no programa GDE.

Durante a realização dos experimentos da seção 4.2, onde se buscava alterações dos estados de energia do processador, realizaram-se coletas de dados de utilização da CPU, temperatura do processador, corrente na bateria entre outras, verificou-se que a política de gerenciamento poderia ser alterada a fim de reduzir o consumo de energia.

Para tanto, realizou-se com o auxílio da ferramenta de identificação do MatLab a identificação de um modelo para implementar uma nova política para uma carga de trabalho específica. Assim, propôs-se uma política com um modelo polinomial de terceira ordem.

Foi utilizando o programa LabVIEW para coleta em tempo real da corrente e da tensão da bateria, juntamente com o programa GDE para coleta do tempo de CPU ativa e freqüência empregada. Estes dados permitiram a comparação entre a nova política de gerenciamento dinâmico de energia proposta e a política *Ondemand*.

Os resultados mostraram que a política proposta foi mais econômica do ponto de vista de energia quando comparado à política *Ondemand* sob custo de uma maior penalidade de desempenho média.

Sabe-se, contudo, que os avanços não pararam e que cada vez mais é necessário estudar, experimentar e propor políticas ainda mais sofisticadas que proporcionem economia de energia ainda maior para atender aos paradigmas da sociedade em uma época onde os dispositivos de tecnológicos se fazem mais e mais presentes na vida das pessoas.

Sendo assim, colocam-se como propostas para estudos futuros:

- Realizar estudos utilizando a política proposta neste trabalho para processadores multi-núcleos.
- Aferir a utilização de múltiplas cargas de trabalho simultâneas para estimar o comportamento de processadores.
- Realizar comparações entre a política de gerenciamento dinâmico de energia proposta neste trabalho e as políticas *Conservative*, *Powersave,* e outras políticas de usuário.

- Realizar estudos visando melhorar a eficiência da política aqui proposta a fim de possibilitar a utilização desta em outras cargas de trabalho.

REFERÊNCIAS BIBLIOGRÁFICAS

ARAÚJO, A. S. Cavalheiro, G. G. H. *Eficiência Energética Através do Escalonamento Dinâmico de Threads e Frequência de Processamento em Ambiente Anahy Sobre Arquiteturas Multi-Core*. Publicado no XIX CIC (XIX Congresso de Iniciação Científica), XII ENPOS (XII Encontro de Pós-Graduação), II Mostra Científica, Pelotas, RS, Brasil. 2010.

BENINI, L. Bogliolo, A. Paleologo, G. A. Micheli, G. De. *Policy Optimization for Dynamic Power Management.* Publicado na DAC '98 durante a 35th annual Design Automation Conference, New York, NY, USA. 1998. ISBN:0-89791-964-5.

BERGAMASCHI, R. Nair, I. Dittmann, G. Patel, H. Janssen, G. Dhanwada, N. Buyuktosunoglu, A. Acar, E. Acar, E. Kucar, D. Bose, P. *Darringer, J. Han, G. Performance Modeling for Early Analysis of Multi-Core Systems.* Publicado na CODES+ISSS '07 durante a 5th IEEE/ACM International Conference on Hardware/software codesign and system synthesis. New York, NY, USA. 2007. ISBN: 978-1-59593-824-4.

BIRCHER, W. L., John, L. K. *Analysis of Dynamic Power Management on Multi-Core.* Publicado na 22nd annual international conference on Supercomputing. New York, NY, EUA, 2008. ISBN: 978-1-60558-158-3.

CAI, Y. Reddy, S. M. Pomeranz, I. Al-Hashimi, B. M. *Battery-aware Dynamic Voltage Scaling in Multiprocessor Embedded System.* Publicado no ISCAS '05 durante o IEEE Internacional Symposiun on Circuits and Systems. Kobe, Japão. 2005. ISBN: 0-7803-8834-8.

CHOI, K. Lee, W. Soma, R. Pedram, M. *Dynamic Voltage and Frequency Scaling Under a Precise Energy Model Considering Variable and Fixed Components of the System Power Dissipation.* Publicado no ICCAD '04 durante a IEEE/ACM International Conference on Computer-aided Design. Washington, DC, USA. 2004. ISBN: 0-7803-8702-3.

DONALD, J., Martonosi, M. *Techniques for Multicore Thermal Management: Classification and New Exploration.* Publicado no ISCA '06 durante o 33rd Annual International Symposium on Computer Architecture. Washington, DC, USA, 2006. ISBN:0-7695-2608-X.

ISCI, C. Buyuktosunoglu, A. Cher, Chen-Yong, Bose, P. Martonosi, M. *An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget.* Publicado no MICRO 39 durante a 39th Annual IEEE/ACM International Symposium on Microarchitecture. Washington, DC, USA, 2006. ISBN: 0-7695-2732-9.

LIMA, A. M. N. Luiz, S. O. D. Perkusich, A. *Modeling, Estimation and Control for Power Management in Embedded Systems.* Publicado na DINCON'09, Brazilian Conference on Dynamics, Control and Applications. Bauru, SP, Brazil, 2009.

LIMA, A. M. N. Luiz, S. O. D. Perkusich, A. Gorgônio, K. Técnica de gerenciamento dinâmico de energia orientada à autonomia da bateria para sistemas embarcados. Publicado no XVII Congresso Brasileiro de Automática (CBA 2008), Juiz de Fora, MG, Brasil. 2008.

LUNGU, A. Bose, P. Sorin, D. J. German, S. Janssen, G. *Multicore Power Management: Ensuring Robustness via Early-Stage Formal Verification.* Publicado no ISLPED '10 durante o 16th ACM/IEEE International Symposium on Low Power Electronics and Design, New York, NY, USA. 2010. ISBN: 978-1-4503-0146-6.

NOVELLI, B. A. Leite, J. C. B. Urriza, J. M. Orozco, J. D. Regulagem Dinâmica de Voltagem em Sistemas de Tempo Real. Publicado no XXXII Seminário Integrado de Software e Hardware, São Leopoldo, Brasil, 2005.

RANGAN, K. K. Wei, Gu-Yeon. Brooks, D. *Thread Motion: Fine-Grained Power Management for Multi-Core Systems.* Publicado no ISCA '09 durante o 36th Annual International Symposium on Computer Architecture. Washington, DC, USA, 2009. ISBN: 978-1-60558-526-0.

WANG, Y. Kai Ma, Wang, X. *Temperature-Constrained Power Control for Chip Multiprocessors with Online Model Estimation.* Publicado no ISCA '09 durante o 36th Annual International Symposium on Computer Architecture. Washington, DC, USA, 2009. ISBN: 978-1-60558-526-0.

WOO, S. C. Ohara, M. Torrie, E. Singh, J. P. Gupta, A. The *SPLASH-2 Programs: Characterization and Methodological Considerations*. Publicado no ISCA '95 durante o 22nd Annual International Symposium on Computer Architecture. New York, NY, USA. 1995. ISBN: 0-89791-698-0.

(ACPI) Advanced Configuration and Power Interface Revisão 4.0a de 2010, disponível em http://www.acpi.info/DOWNLOADS/ACPIspec40a.pdf acessado em 21/03/2011.

Intel White Paper, *Intel Turbo Boost Technology in Intel Core*, Novembro de 2008

Intel White Paper, *Enhaced Intel SpeedStepTechnology for the Intel Pentium M Processor*, Março de 2004

Advanced Micro Devices (AMD), *Processor Utilization with Microsoft Windows Media Center Edition on Systems Enabled with Cool'n'Quiet and AMD PowerNow! Technologies*, *Aplication Note*. Maio de 2005.

Advanced Micro Devices (AMD), *Cool 'n' Quiet™ Technology Installation Guide for AMD Athlon™ 64 Processor Based Systems*. Junho de 2004

Intel® Celeron® D Processor 310, disponível em http://ark.intel.com/Product.aspx?id=27104 acessado em 21/03/2011.

Intel® Pentium® M Processor 750, disponível em http://ark.intel.com/Product.aspx?id=27593 acessado em 21/03/2011.

ANEXO A

```
/*
 * 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
 *
 * Versão original:
 * Saulo O. D. Luiz
 * Embedded / UFCG
 * saulo@ee.ufcg.edu.br
 * Janeiro, 2011
 *
 * Implementação da política proposta em TCC:
 * Anderson Morais
 * UFCG
 * ambamorais@gmail.com
 * Maio, 2011
 */

#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 USEC_PER_SEC 10000001
#define timev(var) ((long)(var.tv_sec*USEC_PER_SEC + var.tv_usec))

struct timeval t0, t1, t2;
long periodo = 10000001;    /* Periodo para a politica de 1 segundo */
long tPoliticaTotal, tDormindo, tErro, t1Anterior, delta,
tPoliticaAtivaTotal = 0;
float tPoliticaMedio, tPoliticaAtivaMedio, percentualtPoliticaAtivaMedio;
float distribuicao[5] = { 0.2, 0.2, 0.2, 0.2, 0.2};
int fps[2];         /* 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 */
int statuslog;          /* returna o status de chamadas do sistema */
int tamLinhaBuflog;
```

```
//Estrutura periodic_info, e funÃ§Ãµes make_periodic e wait_period para
tornar a polÃtica de energia periÃ³dica atravÃ©s 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 void 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;
    }
    /* "missed" should always be >= 1, but just to be sure, check it is not
0 anyway */
    if (missed > 0)
        info->wakeups_missed += (missed - 1);
}
int
main (int argc, char *argv[])
{
  if (argc != 3)
    {
      printf ("Uso %s periodoAmostragem(us) tempo\n", argv[0]);
      return 1;
    }
  sscanf (argv[1], "%d", &periodo);
  sscanf (argv[2], "%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);
    }
  /* 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);
    }
  /*Configurando a semente do gerador de numeros aleatorios com a hora do
dia */
  srand ((unsigned int) time (NULL));
  i = 0;

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

    /******Politica de energia*******/
  while (1)
    {
      /*Sensor bateria */
      //bateria (&correnteBateria, &capacidadeRestanteBateria,
&tensaoBateria);
      /*Sensor atividade do processador */
      estatisticasProcessador (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]);
      }
      inativaPercentual[j] = sinal_i[j];
      ativaPercentual[j] = 1.0 - inativaPercentual[j];
      tempoCPU_anterior[j] = tempoCPU[j];
      tempoCPUinativa_anterior[j] = tempoCPUinativa[j];
    }
      for (nucleo = 0; nucleo < NNUCLEOS; nucleo++)
    {
      /*Sensor de atividade */
      y[nucleo] = ativaPercentual[1 + nucleo];

// Minha Politica de Energia
deslocarVetorParaADireita(y[nucleo], &y_calculado[1], ORDEM);
deslocarVetorParaADireita(0.0, &u_calculado[1], ORDEM);

for(indiceFreq = NFREQ -1; indiceFreq > 0 ; indiceFreq--){
  u_calculado[1] = frequencia_S[comandoDoGE_menos1[nucleo]] /
frequencia_S[indiceFreq];

  //Calcula estimativa de y
  y_calculado[0] = 0.0;
  for ( i=1; i<=ORDEM; i++ ){
```

```
        y_calculado[0] += -a[i]*y_calculado[i] + b[i]*u_calculado[i];
  }
  //Saturacao
  y_calculado[0] = (y_calculado[0] < 0.0) ? 0.0 : y_calculado[0];
  y_calculado[0] = (y_calculado[0] > 1.0) ? 1.0 : y_calculado[0];

  //Verificar se estimativa de y menor ou igual a restricao
  if(y_calculado[0] <= restricao){
    break;
  }
}
if (indiceFreq < 0){
    indiceFreq = 0;
} else if (indiceFreq >= NFREQ){
    indiceFreq = NFREQ - 1;
} else {
    comandoDoGE[nucleo] = indiceFreq;
}
// Fim Minha Politica de Energia

      /*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, "%s %d %d %d %f %f\n",
         parametro_S[comandoDoGE[0]],
         0, correnteBateria,
         capacidadeRestanteBateria, y_calculado[0],
         ativaPercentual[1]);
       statuslog = write (fpslog, buflog, tamLinhaBuflog);   /* Mudando o
estado de operacao */
      if (statuslog != tamLinhaBuflog)
    perror ("wrote wrong number of bytes at log");
       wait_period(&info);
       i++;
      if (i >= n)       //i funciona aqui tanto como contador para as
considÃ§Ãµes iniciais,
    //quanto como um contador para um perÃodo n de teste da polÃtica
    break;
    }               //end while polÃtica 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;
}
```

ANEXO B

```
/*
 * 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
 */

#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 1
#define NFREQ 5

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;
}
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 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/BAT1/state", O_RDWR);
        if (fpsbat < 0) {
                perror("/proc/acpi/battery/BAT1/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(int *soma, int *idle){//, float
*idlePercentual){
    unsigned char bufStat[500]; /* buffer para guardar string do arquivo de
estatÃsticas*/
    int fpsStat;            /* descritor do arquivo de estatisticas */
    int cnt;           /* para a leitura do arquivo de estatisticas */
    unsigned char temporario[500];  /* buffer temporario */
    int estatisticasCPU[NNUCLEOS+1][10];        /* Estatisticas do
processador 0=todo, 1 = cpu0, 2 = cpu1*/
    int somaEstatisticasCPU[NNUCLEOS+1] = {0, 0};   /* Soma das
estatisticas do processador */
    int i,j;

        /* Abrindo arquivo de temperatura do processador */
        fpsStat = open("/proc/stat", O_RDWR);
        if (fpsStat < 0) {
                perror("Abertura de /proc/stat falhou.");
                exit(1);
        }
    /* Lendo arquivo de temperatura do processador */
    cnt = read(fpsStat, bufStat, 400);

    /* Identificando os valores */
    sscanf(bufStat, "%s %d %d %d %d %d %d %d %d %d %d\n%s %d %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]);
    for(i = 0; i<NNUCLEOS+1;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", somaEstatisticasCPU);
    printf("\nidle: %d", estatisticasCPU[3]);
    //printf("\nidle/somaEstatisticasCPU: %.3f\n", ((float)
estatisticasCPU[3]) / ((float) somaEstatisticasCPU));
    #endif
    close(fpsStat);
}
void mudarEstadoDeEnergiaDoProcessador(int fpsProcessador, char
*nomeEstado, int tamanhoNomeEstado){
    unsigned char buf[1000]; /* buffer para escrever o estado de energia*/
    int status;   /* returna 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"};

        // 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;
    }
        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;
}
/* Funcao que desloca os elementos de um vetor para a direita e coloca um
novoElemento na posicao da esquerda
 * Considere x >>> y como mover x para a posicao 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;
}
```

ANEXO C

```
/*
 * 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
 *
 */
#ifndef __MODELODOSISTEMA_H__
#define __MODELODOSISTEMA_H__

#define NNUCLEOS 1
#define NFREQ 5
#define ORDEM 3    //Ordem de minha Politica

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

/*Frequencias do processador*/
char parametro_S[NFREQ][8] = {"1867000", "1600000", "1333000", "1067000",
"800000"};
int tamanhoStringParametro_S[NFREQ] = {7, 7, 7, 7, 6};
float frequencia_S_normalizada[NFREQ] = {1.0, 1.166875, 1.40060015,
1.749765698, 2.33375};
float frequencia_S[NFREQ] = {1867000, 1600000, 1333000, 1067000, 800000};
float l_S[NFREQ];
int nucleo;
int indiceFreq;

/*Controlador*/
double u[NNUCLEOS], y[NNUCLEOS], uMenos1[NNUCLEOS] = {1}, uc[NNUCLEOS],
Ls[NNUCLEOS], Li[NNUCLEOS], restricao;
int comandoDoGE[NNUCLEOS]; /*comando do gerenciador de energia*/
int comandoDoGE_menos1[NNUCLEOS] = {0}; /*comando do gerenciador de
energia*/
float a[ORDEM+1] = { 1.0, -0.2962, -0.2803, -0.2111 }, b[ORDEM+1] = { 0,
0.218, -0.1203, 0.004112 };  // Coeficientes de Minha Politica
```

```
float y_calculado[ORDEM+1], u_calculado[ORDEM+1];         // Y eh saida de
Minnha Politica e U eh a entrada

/* Estado da bateria */
int correnteBateria;
int capacidadeRestanteBateria;
int tensaoBateria;

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

#endif /*__MODELODOSISTEMA_H__*/
```

ANEXO D

```
//./chamarCargaAltaPeriodicamente 15000000
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

#define N_VALORES_RAND 10
#define USEC_PER_SEC 10000001
#define timev(var) ((long)(var.tv_sec*USEC_PER_SEC + var.tv_usec))

long periodo = USEC_PER_SEC / 10;   /* Periodo para a politica de 0.1
segundo */
struct timeval t1, t2, t3;
long i, j, delta, offset;
long u;
long valoresu[N_VALORES_RAND];
float distribuicao[N_VALORES_RAND];
char comando[50];
FILE *fpipe;

int rando (float *distribuicao, int nElementos);
int
main (int argc, char *argv[])
{
  if (argc != 2) {
    printf ("Uso %s n\n", argv[0]);
    return 1;
  }
  sscanf (argv[1], "%ld", &u);
  for (i = 1; i <= N_VALORES_RAND; i++) {
    valoresu[i - 1] = u / N_VALORES_RAND * i;
    distribuicao[i - 1] = 1.0 / N_VALORES_RAND;
    printf ("%ld %.3f\n", valoresu[i - 1], distribuicao[i - 1]);
  }
  /*Configurando a semente do gerador de numeros aleatorios com a hora do
dia */
  srand ((unsigned int) time (NULL));
  while (1) {
    gettimeofday (&t1, NULL);
    printf ("%ld\n", timev (t1) - timev (t3));
    t3 = t1;
    u = valoresu[rando (distribuicao, 100)];
    sprintf (comando, "./cargaAlta %ld &", u);
    if (!(fpipe = (FILE *) popen (comando, "r"))) {
      perror ("Problema quando tentou abrir pipe\n");
      exit (1);
    }
    pclose (fpipe);
    if (!(fpipe = (FILE *) popen (comando, "r"))) {
      perror ("Problema quando tentou abrir pipe\n");
      exit (1);
    }
    pclose (fpipe);
    gettimeofday (&t2, NULL);
    delta = timev (t2) - timev (t1);
    if (delta <= periodo) {
      usleep (periodo - delta);
```

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

  /*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;
}
```

ANEXO E

```
%Codigo realizado em MatLab para realizar comparacao entre conjuntos %de
dados amostrados
%Comparacao.m

close all;
clear all;
clc;

%1 aquisicao0800.lvm
%2 aquisicao1070.lvm
%3 aquisicao1333.lvm
%4 aquisicao1600.lvm
%5 aquisicao1867.lvm
%6 aquisicaoOndemand.lvm

nLinhas = 3;
nColunas = 3;
     R = 0.009;
    R1 = 9912.2;
    R2 = 9931.5;

%%%%%%%%%%%%%%%%%%%%%%%Potencia%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%Outras%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

S = [0.800 1.067 1.333 1.600 1.867];

instanteInicialPolitica = [1134 1126 1113 1195 1158 1097];
nPassos = 6;

potenciaMediaPasso = zeros(nPassos,1);
tensaoMediaPasso = zeros(nPassos,1);
correnteMediaPasso = zeros(nPassos,1);

Tm = 1;
fa = 100; %frequencia de amostragem

for indicePasso = 1:1:nPassos
    dadosOriginal= load(sprintf('aquisicaoGDE%d.lvm',indicePasso));
    dados =
dadosOriginal(instanteInicialPolitica(indicePasso):instanteInicialPolitica(
indicePasso)+round(100*Tm*fa),:);

    fa = 100; %frequencia de amostragem
    tempo = 1/fa*(1:size(dados(:,1),1));

    v = dados(:,1);
    vR = dados(:,2);

    tensao = (R1+R2)/R2 * (v - vR);
    corrente = vR/R - (v - vR)/R2;
    potencia = tensao.*corrente;

    potenciaMediaPasso(indicePasso) = mean(potencia);
    tensaoMediaPasso(indicePasso) = mean(tensao);
    correnteMediaPasso(indicePasso) = mean(corrente);
```

```
end

figure(1)

subplot(nLinhas,nColunas,1);
plot(S, potenciaMediaPasso(1:5),'.:');
axis([S(1) S(5) 22 33]);
% xlabel('freq');
%ylabel('Potência Média (W)');

subplot(nLinhas,nColunas,2);
plot(1, potenciaMediaPasso(6),'-');
axis([0 2 22 33]);
% xlabel('con');
%ylabel('Potência Média (W)');
pbaspect([1 4 1])

%%%%%%%%%%%%%%%%%%%%%%%%%%Log%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%Outras%%%%%%%%%%%%%%%%%%%%%%

%Entrada de controle
nPassos = 6;
potenciaPassos = zeros(nPassos,1);
penalidadeDesempenhoPassos = zeros(nPassos,2);
ativaPercentualRAWPassos = zeros(nPassos,2);

for indicePasso = 1:1:nPassos

    % vetor de valores de potencia
    dados=load(sprintf('log%d.txt',indicePasso));
    n = size(dados,1);
    %n=2000;
    intervalo = 1:1:n;

    Tm = 1;
    t=Tm * (1:1:n)';

    comandoDoGE = dados(intervalo,1);
    yCalculado = dados(intervalo,5);
    ativaPercentual = dados(intervalo,6);

    for j=1
        penalidadeDesempenho = zeros(n,1);
        for i=1:n
            if (ativaPercentual(i,j) >=1)
                penalidadeDesempenho(i) = 1;
            end
        end

        penalidadeDesempenhoPassos(indicePasso,j) =
mean(penalidadeDesempenho(1:n));
        ativaPercentualRAWPassos(indicePasso,j) =
mean(ativaPercentual(1:n,j));
    end
end

figure(1)

subplot(nLinhas,nColunas,nColunas+1);
```

```
plot(S, penalidadeDesempenhoPassos(1:5,1),'.:');
axis([S(1) S(5) 0 1]);
% xlabel('Userspace')
%xlabel('a')
%ylabel('L_1')
% [C,I] = min(penalidadeDesempenho_uc(:,1));
% fprintf('\n%s\n', ['Menor L1: ' num2str(C) ' em ' num2str(uc(I))])

subplot(nLinhas,nColunas,nColunas+2);
plot(1, penalidadeDesempenhoPassos(6,1),'-');
axis([0 2 0 1]);
%title('Userspace')
% xlabel('con')
%ylabel('L_1')
% [C,I] = min(penalidadeDesempenho_uc(:,1));
% fprintf('\n%s\n', ['Menor L1: ' num2str(C) ' em ' num2str(uc(I))])
pbaspect([1 4 1])

subplot(nLinhas,nColunas,2*nColunas+1);
plot(S, ativaPercentualRAWPassos(1:5,1),'.:');
axis([S(1) S(5) 0.5 1]);
%title('Userspace')
% xlabel('Userspace')
%ylabel('a_1')
%[C,I] = min(abs(0.75-ativaPercentualRAW_uc(:,1)));
%fprintf('\n%s\n', ['Menor erro a1: ' num2str(C) ' em ' num2str(uc(I))])

subplot(nLinhas,nColunas,2*nColunas+2);
plot(1, ativaPercentualRAWPassos(6,1),'-');
axis([0 2 0.5 1]);
%title('Userspace')
% xlabel('con')
%ylabel('a_1')
%[C,I] = min(abs(0.75-ativaPercentualRAW_uc(:,1)));
%fprintf('\n%s\n', ['Menor erro a1: ' num2str(C) ' em ' num2str(uc(I))])
pbaspect([1 4 1])

%%%%%%%%%%%%%%%%%%%%%%%%%%Potencia%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%Controle%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

uc = 0.80:0.05:0.95;
Nuc = size(uc,2);

%8 aquisicao200ms.lvm aqui tem 6 passos

instanteInicialPolitica = [1160 1519 1147 1865];
%Espacamento = 0.11*550*100+1000 = 7050

nPassos = size(uc,2);

potenciaMediaPasso = zeros(nPassos,1);
tensaoMediaPasso = zeros(nPassos,1);
correnteMediaPasso = zeros(nPassos,1);

Tm = 1;
fa = 100; %frequencia de amostragem

for indicePasso = 1:1:nPassos
    dadosOriginal= load(sprintf('aquisicaoGDE%4.2f.lvm', uc(indicePasso)));
```

```
    dados =
dadosOriginal(instanteInicialPolitica(indicePasso):instanteInicialPolitica(
indicePasso)+round(100*Tm*fa),:);

    tempo = 1/fa*(1:size(dados(:,1),1));

    v = dados(:,1);
    vR = dados(:,2);

    tensao = (R1+R2)/R2 * (v - vR);
    corrente = vR/R - (v - vR)/R2;
    potencia = tensao.*corrente;

    potenciaMediaPasso(indicePasso) = mean(potencia);
    tensaoMediaPasso(indicePasso) = mean(tensao);
    correnteMediaPasso(indicePasso) = mean(corrente);
end

figure(1)

subplot(nLinhas,nColunas,3);
plot(uc, potenciaMediaPasso,'.:');
axis([uc(1) uc(Nuc) 22 33]);
% xlabel('uc');
% ylabel('Potência Média (W)');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Log%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%Controle%%%%%%%%%%%%%%%%%%%%%%%%%%

%Entrada de controle
uc = 0.80:0.05:0.95;
Nuc = size(uc,2);

potencia_uc = zeros(Nuc,1);
penalidadeDesempenho_uc = zeros(Nuc,2);
ativaPercentualRAW_uc = zeros(Nuc,2);

for indice_uc = 1:1:Nuc
    % vetor de valores de potencia
    dados=load(sprintf('log%1.2f.txt', uc(indice_uc)));
    n = size(dados,1);
    %n=2000;
    intervalo = 1:1:n;

    Tm = 1;
    t=Tm * (1:1:n)';

    comandoDoGE = dados(intervalo,1);
    yCalculado = dados(intervalo,5);
    ativaPercentual = dados(intervalo,6);

    for j=1
        penalidadeDesempenho = zeros(n,1);
        for i=1:n
            if (ativaPercentual(i,j) >=1)
                penalidadeDesempenho(i) = 1;
            end
        end
```

```
        penalidadeDesempenho_uc(indice_uc,j) =
mean(penalidadeDesempenho(1:n));
        ativaPercentualRAW_uc(indice_uc,j) = mean(ativaPercentual(1:n,j));
    end
end

figure(1)

subplot(nLinhas,nColunas,nColunas+3);
plot(uc, penalidadeDesempenho_uc(:,1),'.:');
axis([uc(1) uc(Nuc) 0 1]);
%title('Userspace')
%xlabel('a')
% ylabel('L_1')
[C,I] = min(penalidadeDesempenho_uc(:,1));
fprintf('\n%s\n', ['Menor L1: ' num2str(C) ' em ' num2str(uc(I))])

subplot(nLinhas,nColunas,2*nColunas+3);
plot(uc, ativaPercentualRAW_uc(:,1),'.:');
axis([uc(1) uc(Nuc) 0.5 1]);
hold on
plot(uc, uc,'-');
hold off
%axis([0 1100 0 2.1]);
%title('Userspace')
%xlabel('a')
% ylabel('a_1')
%[C,I] = min(abs(0.75-ativaPercentualRAW_uc(:,1)));
%fprintf('\n%s\n', ['Menor erro a1: ' num2str(C) ' em ' num2str(uc(I))])
```

ANEXO F

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

//./cargaAlta 250000000
#define USEC_PER_SEC 10000001
#define timev(var) ((long)(var.tv_sec*USEC_PER_SEC + var.tv_usec))

struct timeval t1, t2;
long i,j, y, r = 1000000, delta, offset;
long u;
double K=20, I = 0, h = (double)USEC_PER_SEC, Ti = (double)2*USEC_PER_SEC;

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

    if (argc != 2){
        printf("Uso %s n\n",argv[0]);
        return 1;
     }
    sscanf(argv[1], "%ld", &u);

    gettimeofday(&t1, NULL);
        for(i=0;i<u;i++){ //1 segundo

        }
    gettimeofday(&t2, NULL);
    printf("%ld\n", timev(t2) - timev(t1));


    return 0;
}
```