RELÓGIO VIRTUAL ESTRITAMENTE CRESCENTE PARA O COMPUTADOR RASPBERRY PI

Edilson Cezar Corrêa

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Claudio Luis de Amorim

Rio de Janeiro
Dezembro de 2014

RELÓGIO VIRTUAL ESTRITAMENTE CRESCENTE PARA O COMPUTADOR RASPBERRY PI

Edilson Cezar Corrêa

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Claudio Luis de Amorim, Ph.D.

Prof. Felipe Maia Galvão França, Ph.D.

Prof. Cristiana Barbosa Bentes, D.Sc.

Prof. Leonardo Bidese de Pinho, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
DEZEMBRO DE 2014

Corrêa, Edilson Cezar

Relógio Virtual Estritamente Crescente para o Computador Raspberry PI/Edilson Cezar Corrêa. – Rio de Janeiro: UFRJ/COPPE, 2014.

XII, 56 p.: il.; 29, 7cm.

Orientador: Claudio Luis de Amorim

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2014.

Referências Bibliográficas: p. 44 – 46.

1. Relógio virtual. 2. Raspberry pi. 3. Contagem de tempo. I. Amorim, Claudio Luis de. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

À minha mãe e ao meu pai (in memoriam) pela dedicação, amor e carinho.

Agradecimentos

À Marinha do Brasil e à Diretoria de Sistemas de Armas da Marinha (DSAM), pela oportunidade concedida para a realização deste curso, que contribuiu para meu crescimento profissional.

Aos professores do Programa de Engenharia de Sistemas e Computação, em especial à Felipe França e Valmir Barbosa.

À secretaria do PESC, em especial à Solange e Guty pela disponibilidade para ajudar.

A Héberte de Moraes pelas dicas simples, porém preciosas para o desenvolvimento deste trabalho.

A Lauro Whatley que, com inteligência e simplicidade, passou informações importantes na elaboração deste trabalho.

A Diego Dutra, incisivo nas idéias e sempre disponível para ajudar, mesmo estando atarefado.

Um agradecimento especial ao Professor Claudio Luis de Amorim, pela oportunidade oferecida e pelo incentivo, passando tranquilidade sempre que me sentia sem idéias.

Um agradecimento especial aos meus filhos Camila, Júlia e Felipe por aceitar minhas ausências.

Um agradecimento especial à minha esposa Sandra pela paciência com minha impaciência e tolerância com minha ansiedade nos momentos difíceis.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

RELÓGIO VIRTUAL ESTRITAMENTE CRESCENTE PARA O COMPUTADOR RASPBERRY PI

Edilson Cezar Corrêa

Dezembro/2014

Orientador: Claudio Luis de Amorim

Programa: Engenharia de Sistemas e Computação

Com as técnicas de gerenciamento de energia em processadores com múltiplos núcleos cada vez mais disseminadas em sistemas computacionais, os métodos comumente utilizados para medir o tempo de execução de uma aplicação, podem não funcionar como esperado em relação aos processadores modernos que possuem a tecnologia de circuito de variação dinâmica de voltagem e frequência e a tecnologia de múltiplos núcleos, que permitem a migração de programas para balanceamento de carga. Comumente, os atuais relógios de sistema não foram projetados para lidar com tais mecanismos o que pode comprometer a execução de aplicações dependentes de medidas de tempo precisas. Esta dissertação avalia o uso do relógio virtual anteriormente proposto, denominado RVEC, que possui a propriedade de contagem de tempo estritamente crescente e precisa que não é afetado pelas tecnologias descritas.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

STRICTLY INCREASING VIRTUAL CLOCK FOR RASPBERRY PI COMPUTER

Edilson Cezar Corrêa

December/2014

Advisor: Claudio Luis de Amorim

Department: Systems Engineering and Computer Science

With the power management techniques in multi-core processors increasingly widespread in computer systems, methods commonly used to measure the execution time of an application may not work as expected for modern processors have the circuit technology dynamic voltage and frequency scaling and the multi-core technology, which allow the migration programs for load balancing. Commonly, the current system clocks are not designed to deal with such mechanisms which can compromise the performance of critical applications with precise time measurements. This dissertation evaluates the use of previously proposed virtual clock, called RVEC, which has the property of strictly increasing timing and is not affected by the described technologies.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Com as tecnologias de gerenciamento de energia e processadores com múltiplos núcleos cada vez mais disseminadas em sistemas computacionais, os métodos comumente utilizados para medir tempo de execução de uma aplicação em alta resolução, podem não funcionar como esperado. Os processadores modernos possuem um circuito de variação dinâmica de voltagem e frequência (*Dynamic Voltage and Frequency Scaling* - DVFS), uma técnica de gerenciamento de energia, que consiste na alteração da duração de um ciclo de processamento. Esta técnica, inicialmente limitada a laptops e dispositivos móveis, está presente nos computadores de mesa e de alta capacidade. Processadores com múltiplos núcleos permitem migração de programas entre os núcleos para balanceamento de carga. Neste caso, não há garantia de sincronização entre os contadores de ciclos de cada núcleo. A operação de ambos os mecanismos expõe o sistema a interrupções, o que contribui para aumentar o desvio na contagem de tempo, reduzindo a precisão do relógio do sistema. Relógios de sistemas, como o High Performance Event Timer (HPET) [1], permitem que aplicações e sistemas de software medirem tempos de execução e implementarem operações de sincronização em um *cluster* de sistemas de computadores usando uma fonte externa como um relógio global. Entretanto, a precisão de tais relógios de sistemas depende de quanto a contagem do tempo foi desviada por interrupções. Além disso, a utilização de um relógio global como o Network Time Protocol (NTP)[2] oferece baixa precisão e está exposto a desvios de tempo.

1.1 Motivação

DUTRA *et al.* [3, 4] propôs um relógio virtual como uma alternativa para o relógio do sistema, com a propriedade de contar o tempo de forma estritamente crescente e precisa para arquitetura x86 em Linux denominado RVEC, sem acarretar ruído adicional ao sistema. Baseado nesta solução, o objetivo deste trabalho é descrever, implementar e avaliar o método do RVEC na plataforma ARM/Raspberry.

Em especial, avaliar o comportamento do RVEC quando submetido à alteração da frequência de operação do processador. O computador Raspberry possui um processador com apenas um núcleo. Entretanto a implementação desenvolvida deverá atender arquiteturas ARM com múltiplos núcleos.

1.2 Contribuição

As principais contribuições desta dissertação são:

1. Validação do relógio virtual RVEC proposto por DUTRA *et al.* [3, 4], como uma solução para relógio de sistema para arquitetura ARM; e

2. Desenvolvimento e avaliação experimental de uma implementação RVEC em um kernel Linux para plataforma ARM/Raspberry com suporte a DVFS.

1.3 Organização da Dissertação

Esta dissertação apresenta no Capítulo 2 os conceitos básicos necessários para o entendimento deste trabalho, entre eles o conceito de processadores com múltiplos núcleos e os contadores de ciclos presentes neste núcleos, além da descrição do problema da contagem de tempo nos sistemas computacionais modernos. O Capítulo 3 descreve o método RVEC e apresenta as premissas para implementação do RVEC no computador Raspberry Pi. No Capítulo 4 são apresentados os resultados para validação da implementação efetuada no computador Raspberry Pi. Os trabalhos relacionados com esta dissertação são apresentados no Capítulo 5. O Capítulo 6 apresenta as conclusões e propõe trabalhos futuros.

Capítulo 2

Conhecimentos Básicos

Neste capítulo estão descritos conceitos que servem como referência para compreensão do problema da precisão do relógio em sistemas que utilizam um número crescente de processadores com múltiplos núcleos e com gerenciamento de energia.

2.1 Processadores com múltiplos núcleos

Em 1965, Gordon Moore afirmou que o número de transistores em um chip seria aproximadamente o dobro a cada ano [5]. Após 10 anos, Moore revisou sua previsão, afirmando que o número de transistores dobraria a cada dois anos. Em 1996, uma nova avaliação feita por um executivo da Intel afirmou que o desempenho do chip de computador dobra a cada 18 meses [6]. Desde o início da década de 1970, quando a Intel fabricou o primeiro microprocessador, o 4-bit 4004, o aumento do desempenho era, e ainda é, relacionado ao aumento da frequência de trabalho do processador. Maior frequência significa um computador mais rápido. Quando o aumento da frequência do processador começou a alcançar o limite de superaquecimento, outros aspectos do desempenho geral de um sistema começaram a ser considerados: o consumo de energia, a dissipação de temperatura, frequência e número de núcleos. O avanço da tecnologia de circuito e a limitação do desempenho levaram os fabricantes a investirem em processadores com múltiplos núcleos [7]. A razão para esta tendência é que tornou-se cada vez mais difícil a melhora do desempenho apenas com o aumento frequência de um processador com apenas um núcleo. Este aumento da frequência demanda maior área no chip de silício para permitir que o processador execute instruções de forma mais rápida, o que acarreta o aumento da quantidade de energia consumida e do calor gerado. Esta mesma área, ocupada com dois núcleos, produz um processador que tem o potencial de fazer duas vezes a quantidade de trabalho. Assim, a maneira mais eficaz de melhorar o desempenho geral é aumentar o número de processos que o processador pode suportar. A solução de múltiplos núcleos aumenta a largura de banda, enquanto diminui o consumo de energia. A

Tabela 2.1 mostra uma comparação entre um processador com apenas um núcleo e outro com oito núcleos usado pelo centro de pesquisa Packaging Research Center na Georgia Tech University. Com a mesma fonte de tensão e com oito núcleos rodando com uma frequência menor, nota-se um aumento de quase dez vezes na largura de banda, enquanto o consumo total de energia é reduzido por um fator de quatro [8].

Tabela 2.1: Processador com um núcleo versus Processador com múltiplos núcleos

	Processador com um núcleo (45nm)	Processador com múltiplos núcleos (45nm)
Vdd	1.0V	1.0V
Frequência de operação	7,8 GHz	4 GHz
Largura de banda	125 GByte/s	1 TeraByte/s
Potência	429,78 W	107,38 W

A utilização de múltiplos núcleos passa a ser, teoricamente, mais um problema de software do que de hardware. Entretanto, se dois núcleos forem colocados em um único chip sem qualquer modificação, o chip iria, em teoria, consumir duas vezes mais potência e gerar uma grande quantidade de calor. Daí a necessidade de se projetar o uso da frequência mais baixa para reduzir o consumo de energia nos núcleos com utilização reduzida. As Figuras 2.1 e 2.2 ilustram os componentes básicos de um processador genérico com múltiplos núcleos [9]. As duas arquiteturas possíveis de múltiplos núcleos diferem na localização da comunicação entre os dois núcleos do processador.

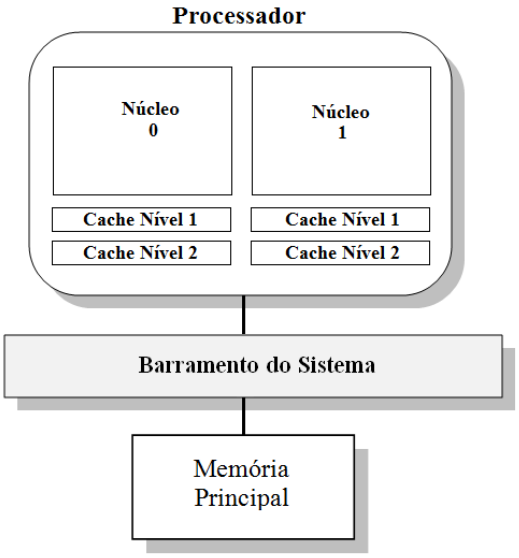

Figura 2.1: Processador multinúcleo com cache nível 2 separada

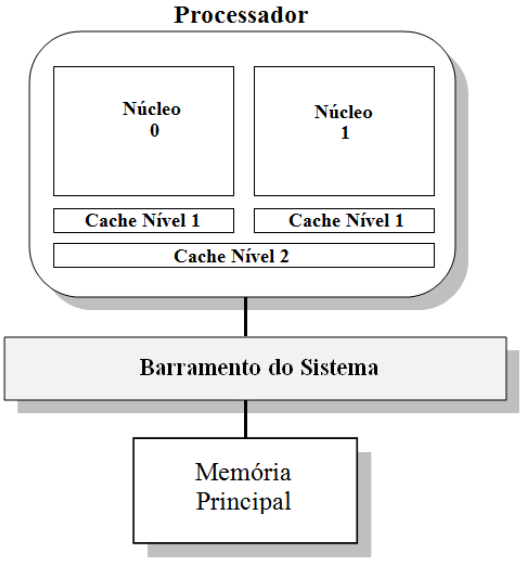

Figura 2.2: Processador multinúcleo com cache nível 2 compartilhada

A Figura 2.1 mostra dois núcleos, cada um com cache nível 2, e na Figura 2.2 os núcleos compartilham o cache de nível 2. Nas duas configurações exemplificadas não há restrição para implementação do relógio virtual estritamente crescente (RVEC).

2.2 Variação Dinâmica de Voltagem e Frequência

Os processadores modernos possuem um circuito de variação dinâmica de voltagem e frequência (*Dynamic Voltage and Frequency Scaling* - DVFS), uma técnica de gerenciamento de energia, que consiste na alteração da duração de um ciclo de processamento, ao modificar a frequência de operação do núcleo durante o seu fun-

cionamento [10]. A redução da frequência induz uma redução correspondente na tensão de alimentação, diminuindo o consumo de energia. Além da redução do consumo, há a diminuição do calor gerado pelo processador. A equação que relaciona o consumo com a frequência e a tensão é mostrada a seguir:

$$P = CfV^2$$

onde P é o consumo de energia, C a capacitância, f a frequência do processador e V a tensão. A redução da tensão e/ou da frequência tem uma desvantagem que é uma redução do desempenho. Outra é a redução da precisão da contagem do tempo em sistemas cujas aplicações são dependentes da medição precisa do tempo. Em sistemas em que o DVFS está disponível, o Linux escalona a frequência através do seu subsistema CPUFreq. A política de escalonamento da frequência pode ser configurada de forma distinta para cada núcleo. Esta flexibilidade pode fazer com que os múltiplos núcleos de um processador trabalhem com diferentes frequências.

2.3 Network Time Protocol (NTP)

O Network Time Protocol (NTP) é um protocolo de Internet usado para sincronizar o relógio de um dispositivo com uma fonte de referência em uma rede, inventado nos anos 80 por Dave L. Mills com o objetivo de alcançar a maior precisão possível na sincronização do tempo entre computadores através da rede. A versão três está definida na RFC 1305 [2]. A arquitetura do NTP consiste de computadores em uma topologia hierárquica dividida em níveis denominados *stratum* (Figura 2.3). O *stratum* de nível mais alto é o zero e o de nível mais baixo é o dezesseis. Os números dos níveis acima de zero representam a distância ao relógio de referência no *stratum* zero, que pode ser um relógio receptor do Sistema de Posicionamento Global (GPS) ou um relógio atômico. Os computadores do *stratum* zero fornecem o tempo para os computadores do *stratum* um, que por sua vez fornecem tempo aos computadores do próprio nível e também aos computadores do *stratum* dois e assim sucessivamente conforme indicam as setas na (Figura 2.3).

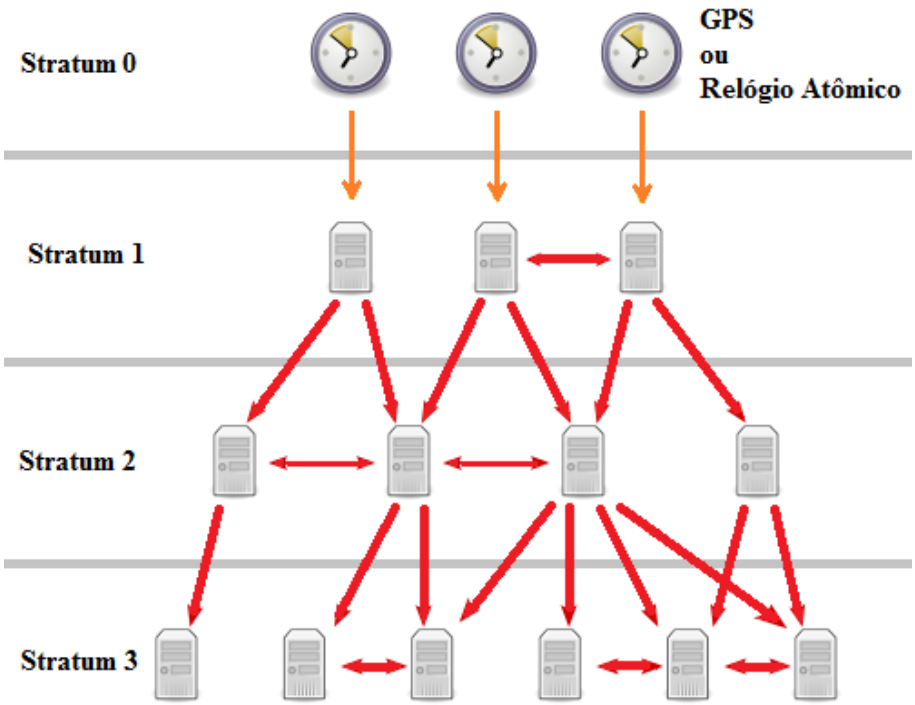

Figura 2.3: Arquitetura do NTP

2.4 Contadores de Ciclos

São registradores que contam e armazenam número de ciclos do sistema. O valor do registrador é incrementado em uma unidade a cada ciclo na frequência de trabalho do processador. Estes contadores normalmente são estáveis e de alta precisão. Um processador, que contenha apenas um núcleo com a frequência de trabalho fixa, possibilita o uso do contador como referência para temporização de uma aplicação. Entretanto, nos processadores modernos, onde há o gerenciamento de energia (DVFS), os incrementos do contador são afetados pela variação da frequência de trabalho. Além disto, com múltiplos núcleos, não há garantia de que os registradores contadores de cada núcleo estejam sincronizados entre si. Estes dois fatores inviabilizam a contagem correta do tempo. Como exemplo, o contador de ciclos *Time Stamp Counter* (TSC) de 64 bits, presente nos processadores x86, conta o número de ciclos desde a inicialização do sistema [11]. Seu acesso é feito por meio da instrução RDTSC [12]. Outro exemplo é o contador CCNT de 32 bits presente no processador do Raspberry que é detalhado no Capítulo 3.

2.5 CLOCK_REALTIME e CLOCK_MONOTONIC

O padrão POSIX 1003.1b [13] (conjunto de normas especificadas pela IEEE) introduziu tipos de temporizadores de software, de alta resolução, para aplicações no modo usuário e de tempo real. Estes temporizadores, também referenciados como relógio do sistema, são chamados de *POSIX timers*. Dentre os relógios que são usados como referência estão o CLOCK_REALTIME e o CLOCK_MONOTONIC.

CLOCK_REALTIME mede o tempo real do sistema. Pode sofrer alterações na temporização caso o relógio do sistema seja alterado, por exemplo, se o administrador do sistema alterar manualmente o relógio ou se houver ajustes incrementais realizados pelo Network Time Protocol (NTP) [14]. CLOCK_MONOTONIC não pode ser alterado manualmente pelo administrador e não é afetado por saltos descontínuos, mas pode ser afetado pelos ajustes incrementais realizados pelo NTP [14]. Para medir o tempo decorrido, CLOCK_MONOTONIC é recomendado. Este relógio não reflete, necessariamente, a hora do dia, mas, ao contrário do CLOCK_REALTIME, mede o tempo de forma linear e crescente. Ambos podem ser acessados por meio de uma requisição de serviço do sistema operacional através da função POSIX *clock_gettime* (chamada de sistema) para obter o tempo corrente do relógio selecionado [14]. A função *clock_gettime* fornece acesso com uma resolução de nanossegundos. A função é implementada diretamente no kernel.

A Figura 2.4 ilustra o uso da função *clock_gettime*.

```
// A estrutura timespec especifica o tempo em segundos
// e nanossegundos

struct timespec {
        time_t   tv_sec;        /* seconds */
        long     tv_nsec;       /* nanoseconds */
};

//Uso da função clock_gettime:

#include <time.h>

int clock_gettime(clockid_t clk_id, struct timespec *tp);

// Onde clk_id é substituído pelo relógio de interesse,
// CLOCK_REALTIME ou CLOCK_MONOTONIC, por exemplo.
// tp é ponteiro para timespec onde clock_gettime() armazena
// o tempo.
```

Figura 2.4: Uso da função *clock_gettime*

2.6 A Arquitetura ARM

A primeira versão da arquitetura (ARMv1) foi introduzida em 1985. Desde então, o processador ARM é um componente chave de muitos sistemas embarcados de 32 bits. A empresa ARM baseia seu sucesso em um projeto original simples e em constante renovação. Enquanto empresas como Intel e AMD projetam e fabricam seus próprios processadores, ARM projeta a arquitetura e a licencia para outras empresas, que fabricam e integram os processadores em seus dispositivos. Empresas como a Apple, NVIDIA e Broadcom fabricam seus próprios processadores, porém a arquitetura do núcleo é licenciada pela ARM.

O núcleo ARM usa uma arquitetura RISC (Reduced Instruction Set Computer). A filosofia RISC consiste na execução de instruções simples, de reduzida complexidade, porém poderosas, em um ciclo do processador [15]. A filosofia RISC visa reduzir a complexidade das instruções executadas pelo hardware. Como resultado, o compilador é mais exigido, o que contrasta com a arquitetura CISC (Complex Instruction Set Computer) presente, por exemplo na família de processadores x86. A Figura 2.5 ilustra esta diferença.

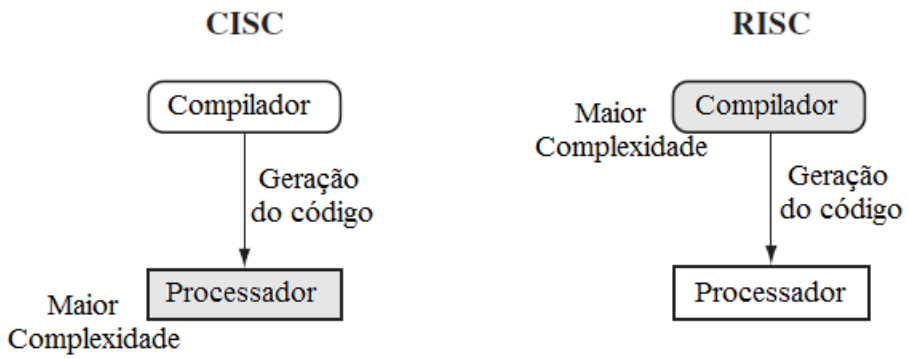

Figura 2.5: RISC-maior complexidade no compilador, CISC-maior complexidade no hardware

O conjunto de instruções ARM é pequeno quando comparado com x86, mas oferece mais registradores de uso geral. O comprimento destas instruções tem largura fixa (16 bits ou 32 bits). ARM utiliza um modelo de load-store para acesso à memória. Isso significa que os dados devem ser movidos da memória para os registros antes de qualquer operação e somente instruções load/store podem acessar a memória (LDR e STR) [16]. Existem dezesseis registradores de 32 bits de uso geral, numerados de R0 a R15 para uso geral, sendo que os três últimos tem significado especial na arquitetura: R13 é o ponteiro de pilha (SP), R14 é o link de registro (RL) e o R15 contador de programa (PC). ARM possui o conceito de co-processadores para dar suporte a instruções adicionais e a configurações em nível de sistema. São dezesseis co-processadores numerados de CP0 a CP15. Para o propósito desta dis-

sertação, o interesse reside no co-processador CP15, onde está localizado o contador de ciclos que servirá de referência para o RVEC. O acesso a este co-processadore é feito somente através de instruções especiais MRC (leitura) e MCR (escrita). Na família de processadores ARM11 o acesso aos co-processadores é feito através de uma interface. Com isso o acesso ao contador no co-processador possui uma pequena sobrecarga adicional. Foi o último núcleo com suporte a co-processadores externos [17]. Nas famílias posteriores o co-processador está integrado ao núcleo.

2.7 O computador Raspberry Pi

Raspberry Pi é um computador do tamanho de um cartão de crédito, de baixo custo e com baixo consumo de energia, criado com o objetivo de ser usado como uma ferramenta de ensino para alunos de computação (Figura 2.6). A idéia surgiu em 2006, no Laboratório de Computação da Universidade de Cambridge, Reino Unido. Em 2008 os processadores projetados para dispositivos móveis tornaram-se mais acessíveis e Eben Upton, então arquiteto de circuito integrado na Broadcom, empresa fabricante de dispositivos de hardware e circuitos integrados, e colegas fundaram a Raspberry Pi Foundation, tornando o projeto uma realidade [18].

Figura 2.6: Computador Raspberry Pi

Existem atualmente dois modelos: modelo A e modelo B. O modelo B, utilizado neste trabalho, possui um controlador Ethernet, duas portas USB e memória de 512 MB, enquanto que o modelo A possui apenas uma porta USB, nenhuma porta de Ethernet e memória de 256 MB. Os modelos consistem de um SOC (system-on-chip) BCM2835 da Broadcom, voltado principalmente para aplicações multimídia, e que possui o processador ARM1176JZ-F da ARM. ARM1176JZ-F é baseado na arquitetura RISC ARMv6, 32 bits. O processador opera na frequência padrão de 700 MHz, que pode ser reconfigurada para até 1GHz. Possui uma unidade de processamento gráfico (GPU) integrada denominada VideoCore IV. O SOC contém cache de nível L2 de 128 KB, usado pela GPU [19]. Outra característica é que, por medida de

economia, o Raspberry Pi não possui relógio de tempo real (*Real Time Clock* RTC) externo. Para atualizar o relógio do sistema é necessário acessar o NTP.

O Raspberry Pi foi projetado para executar o sistema operacional Linux. O código-fonte do sistema operacional está disponível para diversas distribuições. Os procedimentos para instalação estão descritos no Apêndice A.

2.7.1 Contadores do Raspberry pi

O SOC BCM2835 usado no Raspberry Pi possui dois contadores de 32 bits denominados CLO e CHI que podem ser usados para agendar interrupções [20]. A frequência de operação destes contadores é de 1 MHz. Outros contadores estão presentes no co-processador CP15 do ARM1176JZ-F [21]. O propósito do co-processador CP15 é controlar e informar as funções implementadas no processador ARM1176JZF-S. Dentre as funções deste co-processador está a monitoração de performance do sistema (PMU). O ARM11 possui três contadores de performance:

1. O contador de ciclos CCNT;
2. O registrador contador 0 (CR0) conta um dos 21 tipos de eventos suportados pelo processador ARM 1176; e
3. O registrador contador 1 (CR1) também conta um dos 21 tipos de eventos.

O contador CCNT, por padrão, é desabilitado para contagem. Quando habilitado opera sem interrupções e na frequência de operação padrão de 700 MHz. Por estas razões este contador foi escolhido como referência para o RVEC. Os contadores CR0 e CR1 podem contar eventos simultaneamente. Os contadores são controlados e configurados pelo registrador de controle de monitoração de performance (PMCR). Dentre os eventos que podem ser monitorados estão a habilitação dos contadores e zerar contadores. Os contadores de performance informam, por exemplo, *branch mispredicted* e *cache miss*. O acesso a estes contadores é feito através de instruções especiais do ARM11 de leitura (MRC) e escrita (MCR). São instruções privilegiadas que não podem ser utilizadas no nível do usuário a menos que sejam habilitadas no modo *kernel* através de um módulo.

2.7.2 Controle da Frequência

O processador do Raspberry Pi possui tecnologia DVFS, o que possibilita escalonar dinamicamente a frequência do processador em tempo de execução. O escalonamento da frequência foi implementado a partir da versão 2.6 do kernel do Linux através da infraestrutura CPUFreq [22]. CPUFreq consiste de dois elementos:

- Governador - decide qual frequência deverá ser usada; e
- *Driver* - executa a ação baseada na decisão do governador.

Governadores são esquemas de energia pré-configurados para o processador. Um governador monitora continuamente os requisitos de desempenho do sistema e, quando surge a necessidade de alterar a frequência, ele verifica a política atual do processador para os limites de frequência e envia a requisição ao *driver* para mudar a frequência. A política adotada pode ser, por exemplo, a definição da frequência mínima e máxima permitida para cada processador. Outro exemplo são as frequências de operação disponíveis para o processador [23]. A ação do *driver* depende da decisão dos governador. Quando o governador requisita a mudança de frequência ao *driver*, este verifica a disponibilidade de frequências de acordo com a política antes de efetuar a alteração.

Governadores fornecidos pela CPUFreq

No Linux, existem cinco tipos de governadores disponibilizados pela CPUFreq para controlar o escalonamento e configuração da frequência do processador: *performance*, *powersave*, *userspace*, *ondemand* e *conservative*. Apenas um pode ser ativado por vez, com as seguintes características:

1. *Performance* - Em uma política onde os limites de frequências mínima e máxima são definidos, o governador *performance* fixa a frequência de operação do processador no limite máximo.

2. *Powersave* - Em uma política onde os limites de frequências mínima e máxima são definidos, o governador *powersave* fixa a frequência de operação do processador no limite mínimo.

3. *Userspace* - O governador *userspace* permite que o usuário, ou qualquer programa em execução no modo usuário com o identificador *root*, defina uma frequência de operação para o processador, permitida pela política.

4. *Ondemand* - O governador *ondemand* define a frequência de operação de acordo com a carga de trabalho do processador. Quando a carga do sistema for alta, o processador passará a operar na frequência máxima definida pela política. Quando o sistema estiver ocioso o processador passará a operar na frequência mínima definida pela política. O processador deve ter capacidade de chavear entre as frequências de forma rápida. Por exemplo, no caso do Raspberry Pi, a latência máxima de mudança entre as duas frequências é de aproximadamente 355 μs (cálculo efetuado pelo *driver* de frequência).

5. *Conservative* - Como o governador *ondemand*, o governador *conservative* também ajusta a frequência de acordo com o uso. No entanto, enquanto o governador *ondemand* o faz de forma mais agressiva, ou seja, chaveando entre as frequências máxima e mínima, o governador *conservative* altera a frequência de forma gradual. O ajuste da frequência é feito de forma proporcional à carga de trabalho do sistema, em frequências intermediárias entre a mínima e a máxima.

2.8 O problema da contagem de tempo

Um processador possui um contador de ciclos que pode ser incrementado na frequência de operação do processador. Esse contador pode ser utilizado como referência para medir o tempo total de execução de um programa, calculando a diferença dos valores ao iniciar e ao terminar a execução deste programa. O contador típico da arquitetura x86 é o TSC, que pode ser acessado pela instrução RDTSC. Outro exemplo é o contador de ciclos CCNT do processador do Raspberry Pi. O processador pode ser compartilhado por múltiplos programas, com cada um dos programas sendo selecionado por vez para utilizar uma fração de tempo do processador até terminar sua execução [24]. Neste caso, o tempo de execução de um programa será a soma dos intervalos de tempo no qual o processador estava alocado para o programa.

Quando um processador é compartilhado, o relógio do sistema também é usado para aferir os intervalos de tempo em execução de cada um dos programas individualmente. Um intervalo de tempo em execução é obtido pela diferença dos valores do relógio do sistema imediatamente antes de um programa receber do sistema operacional o controle da execução da processador e imediatamente depois que ele retornar o controle de execução do processador ao sistema operacional.

Quando um processador é compartilhado, o sistema computacional geralmente utiliza um contador de tempo de execução e um contador de tempo em execução separadamente para cada um dos programas, processos ou tarefas em execução, e usa o relógio do sistema como fonte dessas informações.

Entretanto, a execução de um programa pode ser interrompida frequentemente e de forma assíncrona por diferentes eventos do sistema computacional que não são precisamente contabilizados no tempo de execução dos programas tais como interrupções com tempos de duração variados do subsistema de entrada/saída, e eventuais perdas de interrupções.

A imprecisão acumulada na contabilidade desses eventos de sistema torna defasada, para mais ou para menos, a contagem do tempo em execução de um programa medida pelo relógio do sistema, tornando imprecisa a medição dos tempos de

execução dos programas.

Uma alternativa comumente utilizada para corrigir a defasagem gerada pelo relógio do sistema é regularmente sincronizar o seu valor corrente com o de outro contador de tempo que seja livre desses eventos do sistema que afetam a precisão da temporização dos programas.

O problema de temporização precisa se torna ainda mais complexo nos sistemas computacionais modernos, que contam com processadores com múltiplos núcleos de processamento. Nestes sistemas cada núcleo tem um circuito contador de ciclos cuja frequência de operação funciona de forma assíncrona em relação aos demais núcleos. A frequência de operação pode ser alterada dinamicamente através do escalonamento de frequências executado pelo sistema operacional.

Nesses sistemas é necessário o auxílio de circuitos externos aos núcleos de processamento, por exemplo, utilizando contadores tais como o High Precision Event Timers (HPET) para as medições temporais.

Estes circuitos externos emitem interrupções a uma frequência fixa, que são atualizadas em um contador de tempo na memória principal, livre dos eventos do sistema que defasam a contagem de tempo.

Ocorre que um contador externo de tempo utilizado como relógio pelo sistema também ficará com o valor de tempo decorrido defasado, em relação ao valor inicial do relógio, devido à latência de acesso à memória principal e a perdas de interrupções não contabilizadas na temporização do sistema. Desta forma, é necessária a ressincronização deste contador de tempo externo a intervalos periódicos tomando como referência uma base de tempo confiável como a fornecida externamente por um relógio atômico que é acessado utilizando um protocolo de comunicação como o NTP.

A ressincronização através do NTP produz uma solução de temporização de baixa precisão, que varia de milissegundos até segundos, dependendo do tráfego da rede de comunicação como à utilizada para acessar o relógio atômico [2].

Capítulo 3

Relógio Virtual Estritamente Crescente - RVEC

Neste capítulo está descrito o método RVEC e apresenta as premissas para implementação do RVEC no computador Raspberry Pi. São descritos o contador de ciclos presente no co-processador do Raspberry Pi e a implementação no Linux.

3.1 Descrição do RVEC

RVEC é um relógio virtual que garante a temporização de forma estritamente crescente e precisa de aplicações executadas em um sistema computacional composto de um sistema operacional e processadores com um ou mais núcleos. A Figura 3.1 ilustra um exemplo de uma arquitetura típica de um sistema computacional [25], com um processador contendo dois núcleos onde o RVEC pode ser utilizado. O método utiliza o processador e a memória principal para criar e manter as informações necessárias para funcionamento do relógio virtual. A comunicação entre o processador e a memória principal é feita através do barramento do sistema.

Cada núcleo de um processador possui um registrador contador de ciclos, cuja frequência de operação funciona de forma assíncrona e independente das frequências dos demais núcleos. Para o contar o tempo de forma estritamente crescente e precisa, o RVEC utiliza o contador de ciclos como referência, que opera sem interrupções. O incremento deste contador é feito através de um circuito de hardware, comumente um oscilador com alta precisão e estabilidade. A taxa em que os pulsos são gerados é a frequência expressa em Hertz (Hz).

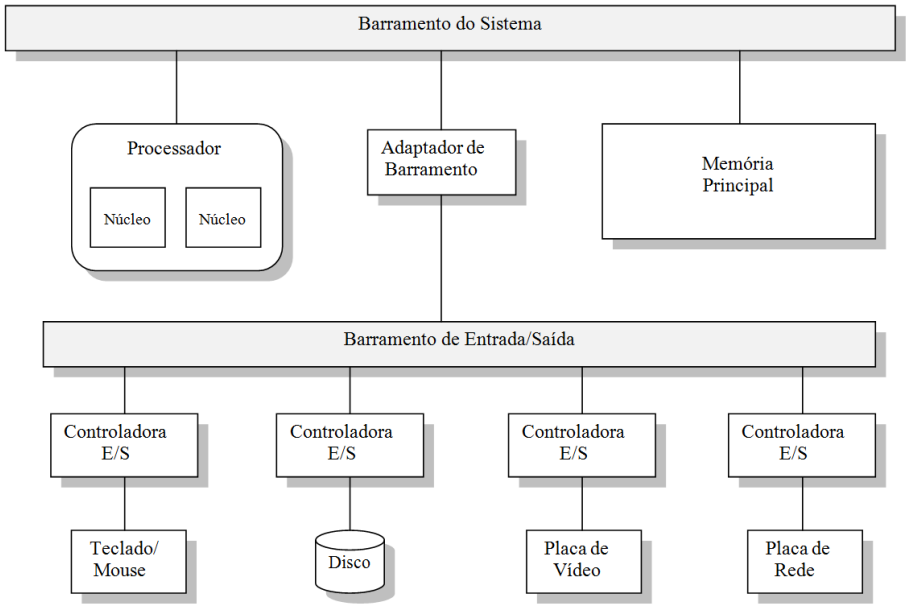

Figura 3.1: Arquitetura típica de um sistema computacional

O inverso da frequência é o ciclo ou intervalo dos pulsos expresso em segundos (Figura 3.2).

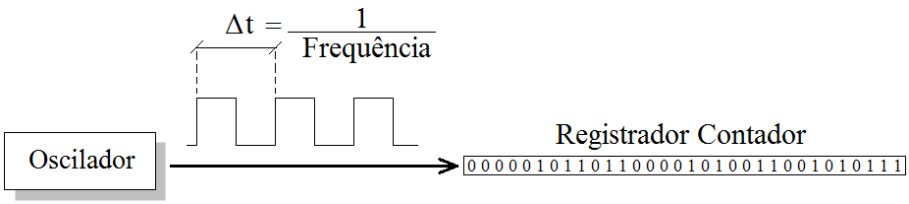

Figura 3.2: Contador de ciclos

Entretanto, a utilização pelo RVEC dos contadores de ciclos dos núcleos como referência implica no tratamento de duas potenciais fontes de desvio na contagem precisa do tempo:

- Mudança na frequência do contador de ciclos de um núcleo; e
- Migração da execução de um programa entre dois núcleos de processamento.

Os processadores modernos que possuem DVFS podem modificar a frequência de operação do núcleo durante o seu funcionamento. A redução da frequência induz uma redução correspondente na tensão de alimentação, diminuindo o consumo de energia. Esta técnica de gerenciamento de energia impacta na correta contagem do tempo.

A migração de programas ou processos entre núcleos é um mecanismo útil para balancear a carga em um sistema computacional. O balanceamento de carga em um sistema pode ser feito por meio de transferência de um processo e de sua estrutura

de um núcleo muito carregado para outro. Entretanto, não há garantia de que os contadores presentes nos núcleos estejam sincronizados.

Para os dois casos acima descritos o RVEC se protege e garante a temporização correta e de alta precisão dos programas.

A Figura 3.3 ilustra um processador com dois núcleos que pode ser utilizado para implementar o RVEC. Neste exemplo, o processador possui contador de ciclos e suporte a DVFS em cada núcleo. O contador pode ser, por exemplo, o TSC (arquiteturas x86) ou o CCNT utilizado no processador da família ARM11. Estes contadores são incrementados a cada ciclo. O DVFS pode alterar a duração de um ciclo ao modificar a frequência de operação de um núcleo. O valor do contador é acessado por meio de uma instrução específica do processador utilizado e armazenado em um registrador ou na memória.

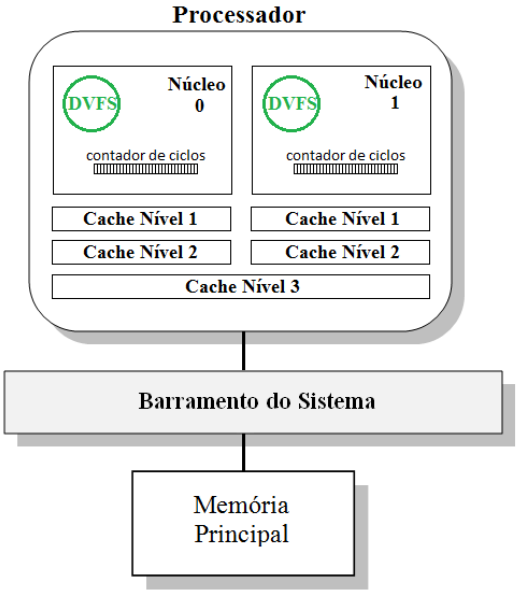

Figura 3.3: Processador com dois núcleos com o suporte DVFS

3.1.1 Procedimento de Alteração de Frequência

A Figura 3.4 mostra o procedimento de alteração de frequência, constituído por etapas que são executadas por diferentes subsistemas. Nas caixas sombreadas estão as alterações introduzidas pelo *driver* RVEC. O pedido de alteração é feito pelo sistema operacional ou por um programa de usuário. O pedido recebido pelo sistema operacional é repassado ao *driver* RVEC que passa a gerenciar o procedimento de alteração de frequência do núcleo. Neste etapa, o RVEC atualiza sua estrutura de controle (ver 3.1.3) contabilizando nesta etapa o tempo decorrido de execução do núcleo que sofrerá alteração em sua frequência de operação. O *driver* RVEC enviará o pedido de alteração de frequência para o *driver* DVFS, que fará a alteração da

frequência. Após a alteração o controle retorna ao *driver* RVEC o qual atualizará as estruturas de dados e passará o controle ao sistema operacional.

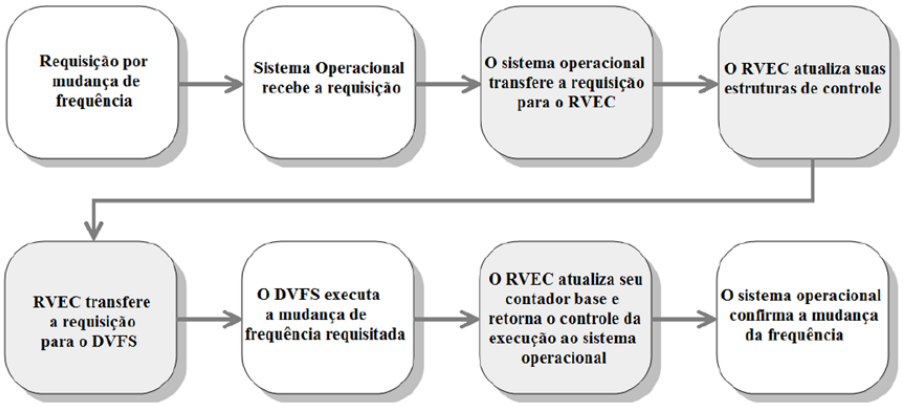

Figura 3.4: Etapas da alteração da frequência

3.1.2 Migração de programa entre núcleos

A Figura 3.5 mostra as etapas que são executadas no procedimento de migração de um programa por diferentes subsistemas. Nas caixas sombreadas estão as alterações introduzidas pelo *driver* RVEC. O pedido de migração é feito pelo sistema operacional ou por um programa de usuário. O sistema operacional recebe o pedido e o encaminha para o subsistema de escalonamento, responsável por executar a migração do programa entre núcleos de processamento. O escalonador escolhe o novo núcleo de processamento para onde o processo será migrado e informa ao RVEC que será realizada uma migração entre núcleos. O *driver* RVEC executa o algoritmo de migração e retorna o controle da execução ao escalonador. Este, por sua vez, realiza a migração do programa entre os núcleos informando o término ao *driver* RVEC. O RVEC, ao terminar o algoritmo de migração, atualiza suas estruturas de controle e retorna o controle para o escalonador. O escalonador então informa o sistema operacional que a migração foi concluída.

3.1.3 O Método RVEC

RVEC é construído a partir de um conjunto de instruções e estruturas de controle e de dados para cada instância RVEC armazenadas na memória principal formando uma camada de *software* entre a aplicação e os valores retornados pelo contador de ciclos. Uma estrutura de dados é composta de dois valores: o valor da última leitura do contador, feita pela lógica de controle, e o valor da passagem consolidada de tempo até o instante da última operação de atualização. Uma representação desta estrutura está ilustrada na Figura 3.6.

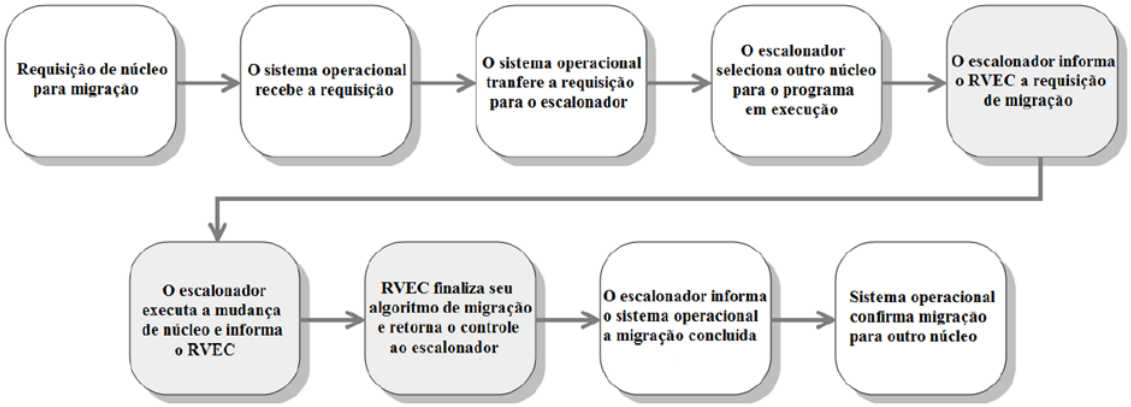

Figura 3.5: Etapas da migração de programa

```
contador_base(ciclos);
tempo_consolidado(segundos);
```

Figura 3.6: Diagrama da estrutura de Dados do RVEC

Nesta estrutura, o contador_base armazena o último valor do número de ciclos (ou *ticks*) lido no contador e o tempo_consolidado mantém correto o tempo decorrido de execução de um programa tanto na presença de mudanças de frequência de operação do núcleo de processamento, como na migração da execução do programa entre núcleos. A estrutura é atualizada na inicialização do núcleo e sempre que o núcleo tem a sua frequência de funcionamento alterada. Após a mudança da frequência o RVEC corrige a contagem do tempo através da execução dos procedimentos mostrados na Figura 3.7.

```
aux_contador = obter_contador();
tempo_consolidado_final = tempo_consolidado_inicial + (aux_contador - contador_base) / frequência_processador;
contador_base = aux_contador;
```

Figura 3.7: Diagrama dos procedimentos de atualização do RVEC

Na iminência da mudança da frequência, o valor lido no contador é armazenado no contador_base e o tempo consolidado final é atualizado com base no tempo consolidado anterior, na nova frequência e nos valores do contador lidos durante a passagem do tempo. Por exemplo, a Figura 3.8 ilustra a contagem do tempo usando o RVEC com a mudança da frequência do núcleo.

No exemplo, o cálculo do tempo consolidado no instante em que o contador é igual a 4, retorna o seguinte resultado:

$$\text{tempo\_consolidado} = 0 \text{ ns} + (4 - 0)/800 \text{ MHz} = 5 \text{ ns}$$

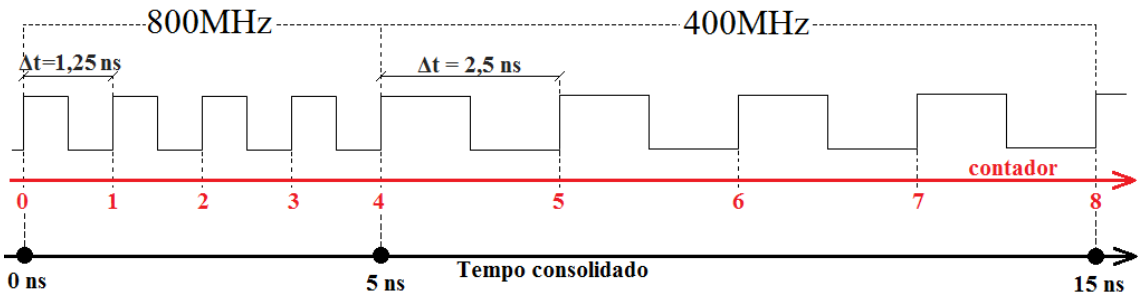

Figura 3.8: Cálculo da contagem do tempo com a mudança da frequência do núcleo

Antes do núcleo mudar a frequência de 800 para 400 MHz, o contador_base lê o contador e armazena o valor 4. Quando o contador chega à 8, o cálculo do tempo dará o valor de:

$$\text{tempo\_consolidado} = 5 \text{ ns} + (8 - 4)/400 \text{ MHz} = 15 \text{ ns}.$$

No próximo exemplo, a Figura 3.9 ilustra a contagem do tempo usando o RVEC com a a migração de processo. Note que os contadores dos núcleos 0 e 1 possuem valores diferentes, pois contam de forma assíncrona.

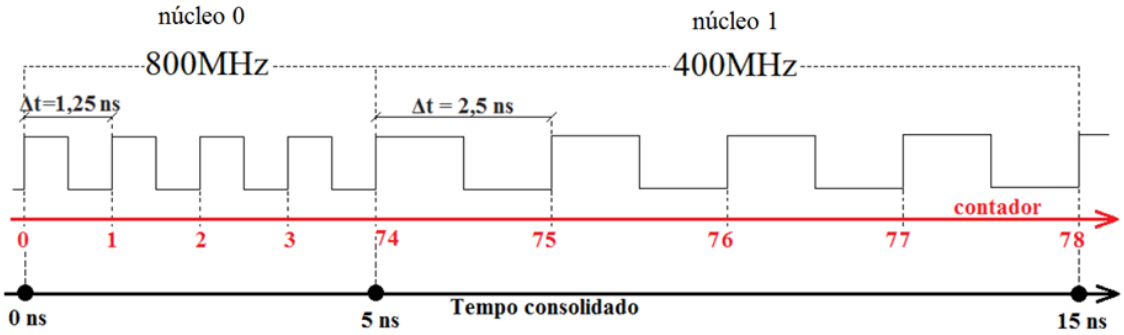

Figura 3.9: Cálculo da contagem do tempo com a migração de processo

De forma similar ao exemplo anterior, o cálculo do tempo consolidado retorna o seguinte resultado:

$$\text{tempo\_consolidado} = 5 \text{ ns} + (78 - 74)/400 \text{ MHz} = 15 \text{ ns}$$

O RVEC cria uma abstração de contagem de tempo para um processo específico, desde o instante da sua criação e para outras instâncias do mesmo processo, com o RVEC de cada processo sujeito a pontos de atualização que são os instantes em que uma unidade de frequência do núcleo é alterada ou um processo migre para outro núcleo. A estrutura de controle do RVEC mantém o tempo consolidado desde a inicialização do núcleo, que é calculado usando os valores do contador lidos durante a passagem do tempo, o valor da frequência atual do núcleo e os valores da estrutura de dados do RVEC associados ao núcleo (Figura 3.10).

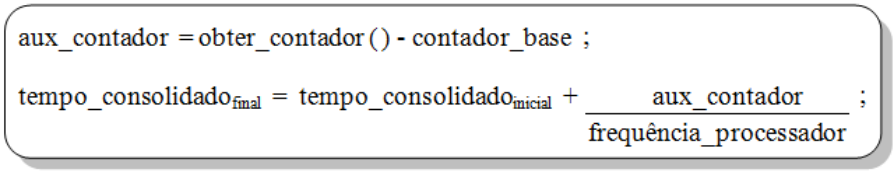

Figura 3.10: Procedimento de leitura do RVEC de um núcleo

3.2 RVEC no computador Raspberry Pi

3.2.1 Premissas

A implementação do RVEC na plataforma ARM-Raspberry Pi permite testar o método quando submetido à mudança de frequência, uma vez que o processador utilizado possui a tecnologia DVFS. Entretanto, pelo fato do processador ARM utilizado no computador Raspberry Pi possuir apenas um núcleo, não foi possível testar a migração de processos. Note que a limitação quanto à migração é restrita a esta plataforma. O código do RVEC implementado nesta arquitetura pode ser utilizado em uma plataforma ARM com processadores contendo múltiplos núcleos. Desta forma, para fundamentar o estudo e a implementação do RVEC no Raspberry Pi, é necessário definir as seguintes premissas:

1. O registrador contador não deve reiniciar durante a contagem do tempo (*overflow*); e

2. O registrador contador deve possuir a propriedade de contagem estritamente crescente e precisa que assegure que duas leituras consecutivas do contador, T1 e T2, retorne medidas de tempo T2 >T1 para qualquer intervalo de tempo entre as duas leituras.

3.2.2 O registrador contador de ciclos CCNT

O RVEC utiliza o contador de ciclos como referência para contagem de tempo. Na arquitetura x86 a referência é o TSC. O acesso a este registro é feito através da instrução RDTSC. De forma similar, o processador do Raspberry Pi possui um contador de ciclos de 32 bits denominado CCNT [21]. O acesso a este contador é feito pela transferência do dado do registrador c15, presente no co-processador CP15, ao registrador Rd (registrador de destino) do processador. Embora presentes no mesmo chip, a comunicação entre o co-processador e processador é feita através da interface presente no núcleo (Figura 3.11).

O co-processador pode ser acessado através de um grupo especial de instruções que estendem o conjunto padrão de instruções da arquitetura ARM. Dentre as instruções especiais estão MRC e MCR, que são utilizadas apenas por núcleos com

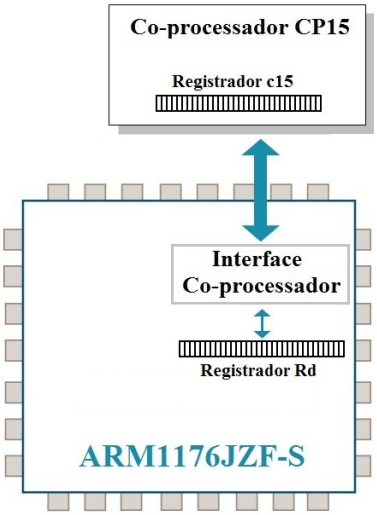

Figura 3.11: Processador ARM-1176 e o Co-processador CP15

co-processador. O registrador contador c15 do co-processador CP15 é lido e escrito com as instruções especiais MRC e MCR, respectivamente. A Tabela 3.1 lista as instruções MRC e MCR do co-processador suportados pelo processador.

Tabela 3.1: Instruções do Co-processador

Instrução	Descrição
MRC	Transfere dados do co-processador para os registradores do núcleo
MCR	Transfere dados dos registradores do núcleo para o co-processador

A sintaxe e o significado dos campos estão ilustrados nas Figuras 3.12 e 3.13.

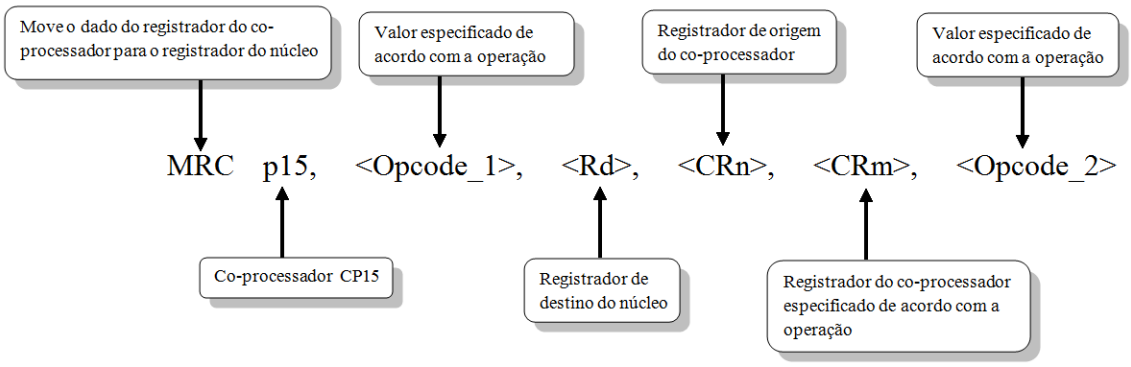

Figura 3.12: Sintaxe da instrução MRC

O contador de ciclos está sempre acessível no modo kernel. Para ler este registro, utilizando a instrução MCR, os campos devem ter as seguintes definições:

- Opcode_1 definido como 0;

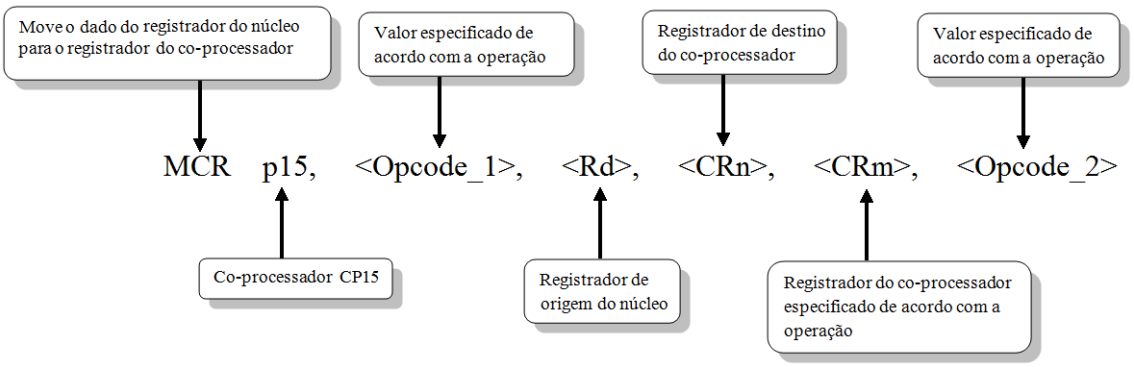

Figura 3.13: Sintaxe da instrução MCR

- CRn definido como c15;
- CRm definido como c12; e
- Opcode_2 definido como 1.

A instrução para mover o conteúdo do registrador contador de ciclos c15 para o registrador Rd do núcleo do processador será:

MRC p15, 0, Rd, c15, c12, 1

MRC e MCR são instruções privilegiadas e normalmente são executadas pelo sistema operacional. A execução destas instruções no modo usuário retorna a informação de uma instrução ilegal. No entanto, o sistema operacional pode permitir que programas no modo usuário possam ler o contador de ciclos através da instrução MRC. Para isto, é necessário executar um módulo no kernel com a instrução:

MCR p15, 0, #1, c15, c9, 0

A instrução é uma operação de escrita no bit 0 do registrador c15 do co-processador que, além de contador, possui também a função de validação de controle de acesso do modo usuário. Quando o bit 0 é definido para 1, é concedido o acesso ao contador de ciclos no modo usuário. Finalmente, por padrão, o contador está desabilitado, sendo necessária a seguinte instrução de escrita no co-processador para início da contagem:

MCR p15, 0, #1, c15, c12, 0

Após todos os procedimentos para acessar o contador CCNT para servir de referência para o RVEC, o fato de ser um contador de 32 bits limita seu uso para poucos segundos, o que pode violar a premissa de que o contador não pode sofrer *overflow* durante a contagem de tempo. Por exemplo, na frequência de 800 MHz, o tempo total entre *overflows* é igual a:

2^{32} ciclos * (1 segundo/800.000.000 ciclos) = 5,3687 segundos

O RVEC não poderá utilizar um contador de 32 bits como referência caso uma aplicação necessite contar além de 5,3687 segundos, na frequência de 800 Mhz. Para superar esta limitação, utiliza-se um algoritmo homologado no Linux (include/linux/cnt32_to_63.h) para estender um contador de 32 para 63 bits, o que permite uma contagem sem *overflows* de 370,66 anos para a mesma frequência. Na implementação deste algoritmo, os bits 0 a 31 são fornecidos pelo CCNT enquanto os bits 32 a 62 são armazenados na memória. O bit 63 é usado para sincronizar com meio período de contagem do CCNT. Quando o último bit de ambos os contadores (CCNT e o contador armazenado na memória) diferirem, então o contador da memória será incrementado quando o CCNT reiniciar a contagem. As restrições para que o algoritmo funcione de forma adequada são:

1. O algoritmo deve ser chamado pelo menos uma vez para cada meio período de contagem do CCNT; e

2. A função correspondente ao algoritmo não pode ser interrompida (preempção) por um tempo maior do que metade da contagem do CCNT menos o período mais longo entre duas chamadas do algoritmo.

Neste ponto, ao obter o acesso ao CCNT através das instruções especiais e, além disto, estender o limite de contagem, uma aplicação pode fazer uso deste contador utilizando, por exemplo, um código similar ao ilustrado na Figura 3.14, desde que não haja alteração na frequência.

```
static inline unsigned int get_cyclecount (void)
{
   unsigned int ccnt;
   // Ler registrador CCNT
   asm volatile ("mrc p15, 0, %0, c15, c12, 1":"=r"(ccnt));
   return ccnt;
}
unsigned int t = get_cyclecount();
float tempo = 0;

// chama funcao();

// obter o tempo em microsegundos para uma frequência de 800MHz
tempo = ((float)t)/800;
```

Figura 3.14: Contagem de tempo utilizando o CCNT

Entretanto, a frequência do processador do Raspberry Pi pode variar visando economia de energia, o que resultaria em uma contagem de tempo errada. O RVEC, por sua vez, usa o CCNT como referência e efetua a contagem de tempo de forma correta, mesmo que ocorra a mudança da frequência do processador.

3.3 Implementação no Linux

A implementação do RVEC foi feita sobre a versão 3.2.27 do *kernel* do Linux. Consiste na integração das estruturas de controle do RVEC nos subsistemas de gerenciamento do núcleo e de gerenciamento de tarefas do *kernel* e da implementação no *driver* de escalonamento de frequência (CPUFreq).

Embora o Raspberry Pi possua um processador com apenas um núcleo, a implementação do código suporta a configuração para múltiplos núcleos e está integrada no procedimento de inicialização do Linux. Além disso, foi feita a integração de modo que o sistema ofereça suporte para tarefas nos modos *kernel* e usuário, o que permite a construção de novas instâncias RVEC, quando necessário.

O RVEC foi integrado ao subsistema de gerenciamento do *kernel* através da inserção de *struct tb* na fila de execução do núcleo. Desta forma, esta estrutura é atualizada na inicialização do núcleo e sempre que o núcleo tem a sua frequência de funcionamento alterada (Figura 3.15).

```
struct tb{
   u64 base_counter;
   u64 age_time_ns;}
```

Figura 3.15: Estrutura de Dados do RVEC

O código do programa ilustrado na Figura 3.16 executa a atualização dos campos base_counter e age_time_ns após uma mudança de frequência. A alteração efetuada no *driver* CPUFreq do Linux garante a correção destes campos.

```
void update_rvec (struct tb *ptb, u64 CoreHz){
   aux_ccnt = get_counter();
   ptb->age_time_ns += (aux_ccnt - ptb->base_counter)/CoreHz;
   ptb->base_counter = aux_ccnt;}
```

Figura 3.16: Procedimento de atualização do RVEC

A estrutura RVEC também mantém o tempo consolidado desde a inicialização do núcleo, que é calculado usando os valores do CCNT lidos durante a passagem do tempo, o valor da frequência atual do núcleo e os valores dos campos do RVEC associados ao núcleo (Figura 3.17).

```
return u64 rvec_core_gettime(void){
   aux_ccnt = (get_counter() - ptb->base_counter);
   return ptb->age_time_ns + aux_ccnt/get_CoreHz();}
```

Figura 3.17: Procedimento de leitura de RVEC de um núcleo

A Figura 3.18 ilustra a lógica de controle adicional introduzida no *driver* CPUfreq. As modificações foram feitas no *driver* bcm2835-cpufreq usado na família de

processadores ARM11 do Raspberry Pi. A chamada de função é usada para alterar a frequência do núcleo. A função original bcm2835_cpufreq_set_clock() é estendida em duas funções RVEC, rvec_cpu_freq_change_pre() e rvec_cpu_freq_change_pos(). A função rvec_cpu_freq_change_pre() lida com a configuração de antes da mudança da frequência (pre-change) para o relógio RVEC no subsistema de gerenciamento do núcleo, executando uma implementação de atualização do RVEC. A função rvec_cpu_freq_change_pos() define (após a mudança da frequência) a soma do tempo gasto desde a execução da função rvec_cpu_freq_change_pre() mais o tempo gasto com a frequência atual e armazena o valor atual do CCNT do núcleo no base_counter.

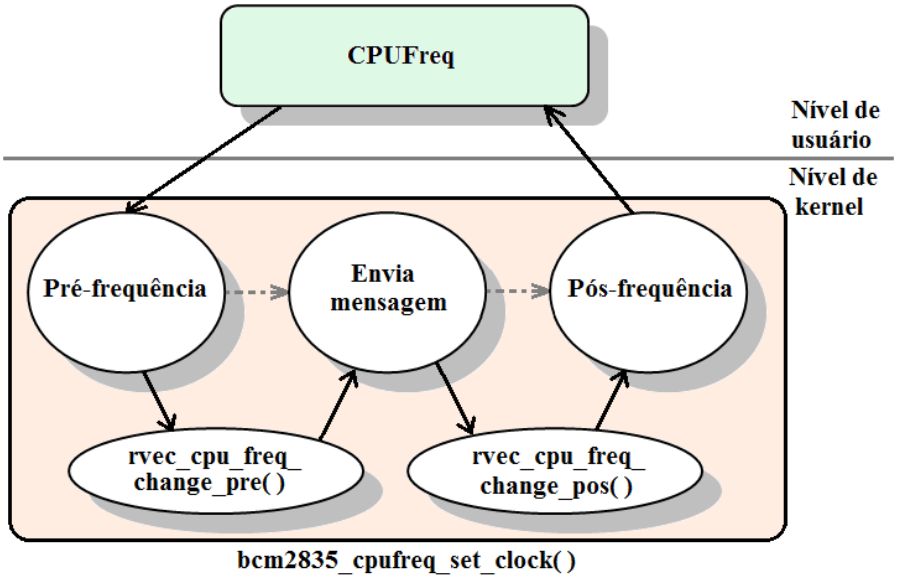

Figura 3.18: Driver bcm2835-cpufreq modificado para o RVEC

A instância struct tb também foi criada para cada tarefa (*thread*) para que implementação do RVEC suporte a execução de múltiplas *threads* no *kernel* do Linux. Após a inicialização de uma nova tarefa para ser executada em um núcleo específico, o valor corrente de RVEC associado a este núcleo será copiado e armazenado no campo base_counter do RVEC que está associado com a nova tarefa. A Figura 3.19 ilustra o RVEC para tarefas e para núcleos. A atualização de base_counter na estrutura do RVEC para tarefas é realizada sempre que a tarefa sofre migração entre os núcleos, garantindo assim ao RVEC a propriedade da contagem de tempo estritamente crescente e preciso. Desta forma, os dados atualizados são passados para a aplicação. O controle da atualização é executado no subsistema de escalonamento do Linux, de tal maneira que RVEC é protegido de potenciais fontes de desvios de tempo.

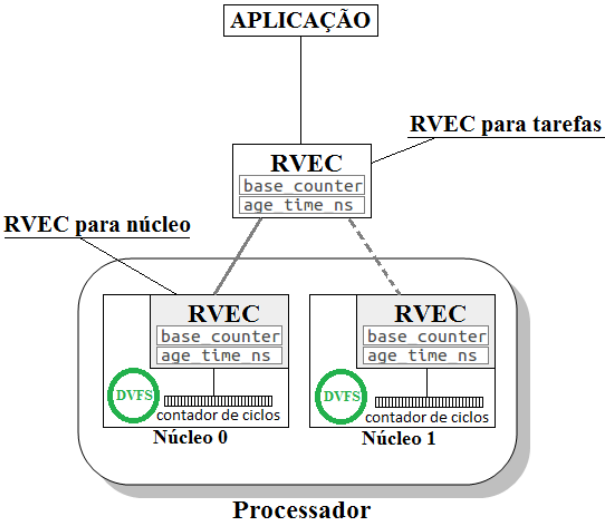

Figura 3.19: RVEC para núcleos e para tarefas

A implementação RVEC permite diferentes tarefas no sistema para verificar os seus RVECs associados através da chamada de sistema clock_gettime(), que vai diretamente para o subsistema de cronometragem do Linux (timekeeping). Neste caso, uma aplicação acessa a essa função com o identificador do relógio desejado CLOCK_RVEC como o parâmetro de entrada.

Capítulo 4

Validação Experimental

Neste capítulo, são apresentados os resultados de uma avaliação experimental do RVEC com dois objetivos:

1. Verificar a propriedade da contagem de tempo estritamente crescente e precisa (*Strictly Increasing and Precise* - SIP) do RVEC com e sem alteração da frequência; e

2. Comparar a sobrecarga e precisão do RVEC com o CCNT, CLOCK_MONOTONIC e CLOCK_REALTIME.

Os experimentos foram feitos em um computador Raspberry Pi, Modelo B (processador ARM1176JZF-S, memória 520 MB) com sistema operacional Linux (Debian weezy, *kernel* 3.2.27+). Os resultados são apresentados com o valor da média e desvio-padrão em microssegundos (μs) para um intervalo de confiança de 99.9%.

4.1 O contador CCNT

O contador CCNT pode acelerar ou desacelerar sua contagem de acordo com a variação da frequência de operação do processador. Além disto, para um tempo maior do que 5,368 segundos (2^{32}/800.000.000 Hz), ocorre o *overflow* deste contador na frequência fixa de 800 MHz. O próximo experimento mostra os valores obtidos do contador CCNT a cada 3.600.000 operações aritméticas. A Figura 4.1 mostra um extrato do gráfico da contagem do tempo do CCNT. Note que após o tempo de 5,368 segundos ocorre o *overflow* do contador.

O mesmo foi feito para o RVEC (Figura 4.2). Este experimento foi estendido para uma hora tanto para o CCNT quanto para o RVEC. Para o RVEC a contagem permaneceu estritamente crescente e precisa durante todo tempo, enquanto que para o CCNT a contagem reinicia a cada intervalo de 5,368 segundos.

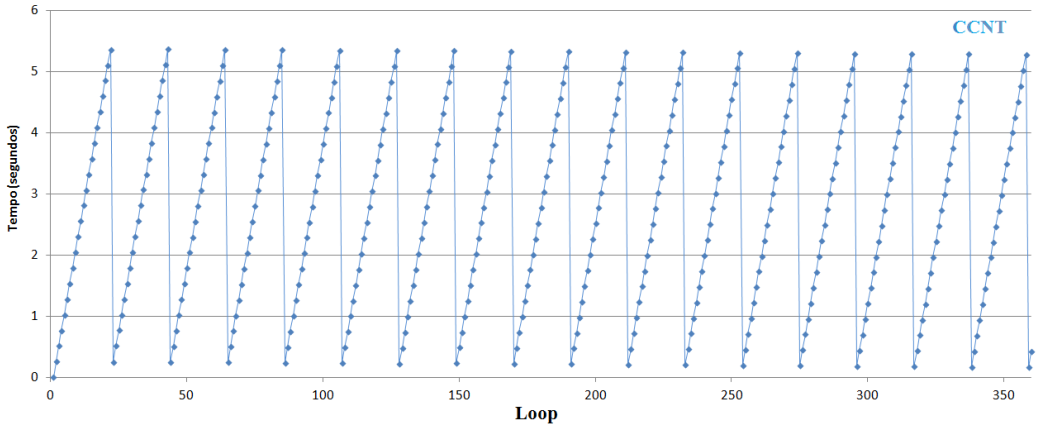

Figura 4.1: Contagem do tempo do registrador CCNT

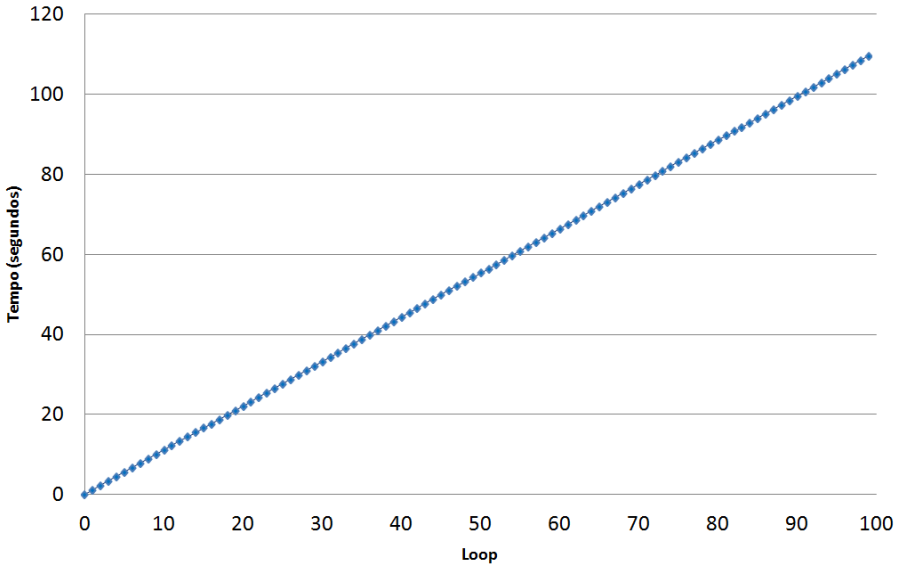

Figura 4.2: Contagem do tempo usando o RVEC

4.2 Propriedade Estritamente Crescente e Precisa (*Strictly Increasing and Precise* - SIP) do RVEC

Para verificar a propriedade SIP do RVEC, primeiro verificou-se se a propriedade SIP se mantém para o CCNT. O programa de teste foi um *microbenchmark* que implementou um laço com um bloco de 12 operações de soma por iteração, executado 400 vezes em cada iteração. O experimento foi executado 100 vezes. Avaliou-se o tempo de execução do *microbenchmark* usando uma instrução de acesso (MRC) para leitura do registrador CCNT de 32 bits. A avaliação da propriedade SIP do CCNT mostrou um tempo médio de execução de 361,91 μs e desvio de 18,33 μs com frequência do processador fixa em 800 MHz. Não ocorreu violação dos tempos de

execução individuais ou dos tempos de execução médios (Figura 4.3).

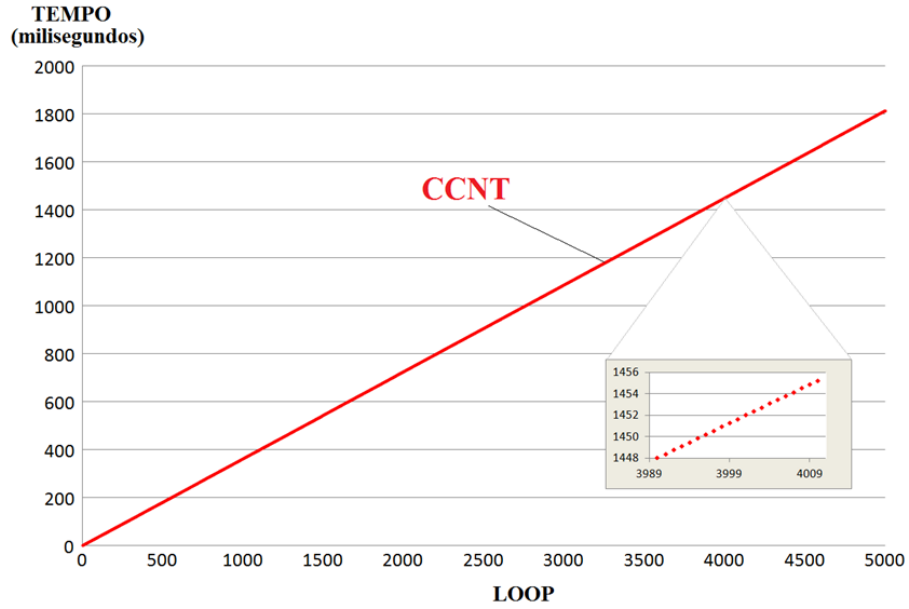

Figura 4.3: Propriedade SIP-CCNT

Tendo em vista a comprovação da propriedade SIP para o CCNT, utilizou-se o mesmo *microbenchmark* para o RVEC, e o experimento foi executado 100 vezes para verificar se o RVEC também obedece a propriedade SIP. A avaliação da propriedade SIP do RVEC mostrou o tempo médio de execução de 372,88 μs e desvio de 20,02 μs com frequência do processador fixa em 800 MHz. Assim como para o CCNT, também não ocorreu violação dos tempos de execução individuais ou dos tempos de execução médios (Figura 4.4).

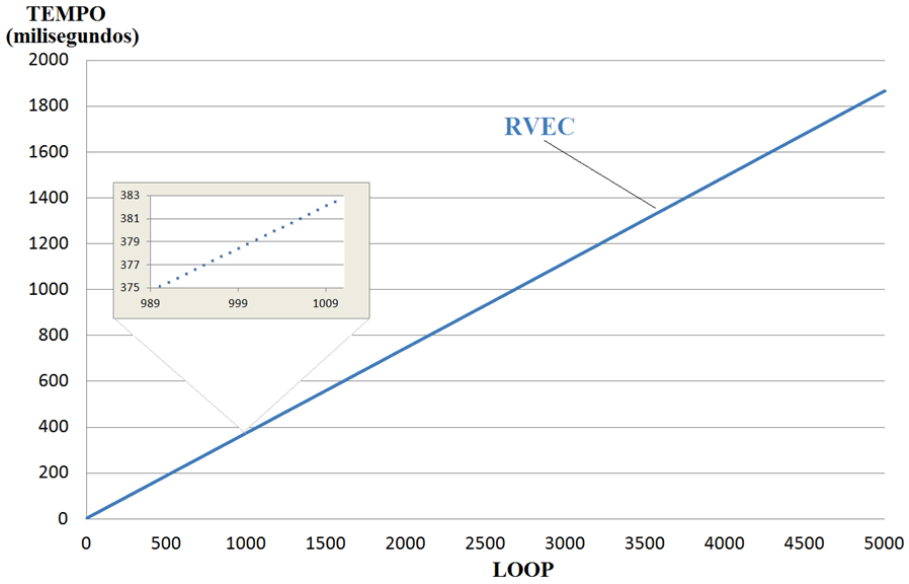

Figura 4.4: Propriedade SIP-RVEC

Testou-se a propriedade SIP com alteração da frequência do processador para

o RVEC (Figura 4.5), com execução de um laço com 5.000 instruções aritméticas, porém com diminuição da frequência de 800 para 400 MHz após 2.500 instruções. A figura mostra o aumento do tempo médio de execução do bloco de instruções para o RVEC, como esperado.

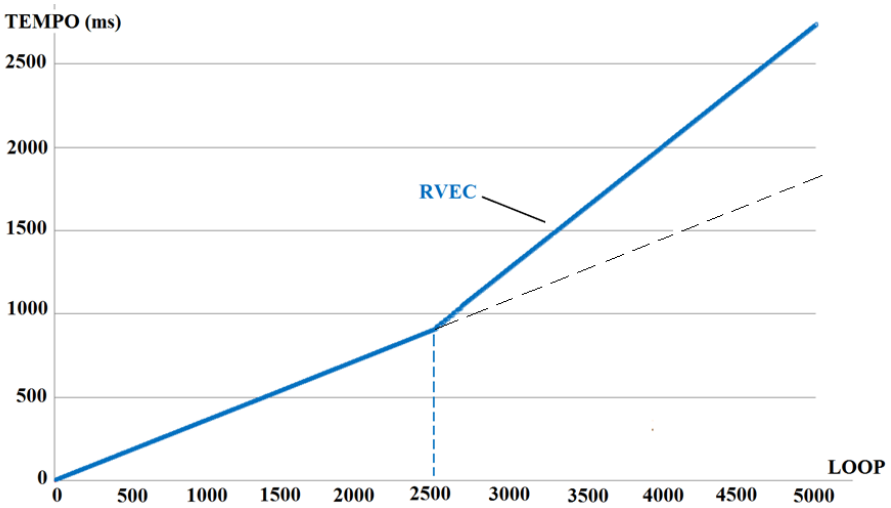

Figura 4.5: Propriedade SIP com alteração da frequência de 800 para 400 MHz

4.2.1 Sobrecarga

A sobrecarga do RVEC foi medida tomando-se como referência o CCNT, que oferece a melhor precisão disponível para um relógio do sistema. Utilizando este artifício, mediu-se a também sobrecarga do próprio CCNT, do MONOTONIC e do REALTIME. A frequência do processador foi fixada em 800 MHz. As configurações utilizadas estão ilustradas na Figura 4.6.

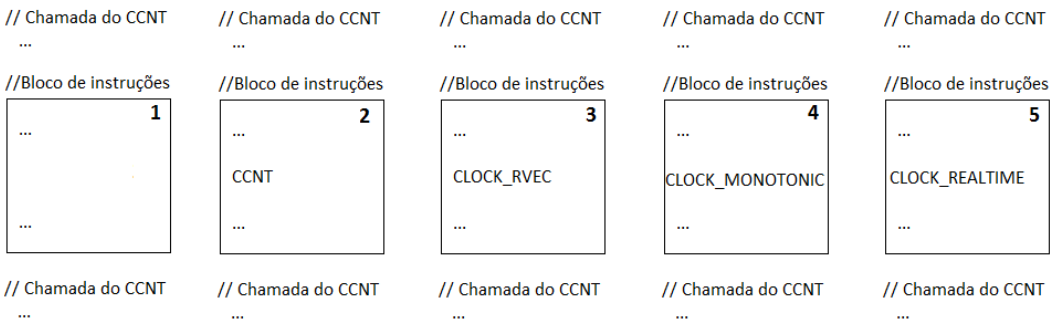

Figura 4.6: Configurações para cálculo da sobrecarga para cada relógio

Na configuração 1 da Figura 4.6 mediu-se o tempo base como referência. Nas demais configurações foi incluído um relógio no meio do ciclo do bloco de instruções.

Sobrecarga para um bloco de 2400 instruções

O experimento consiste na medição do tempo de execução após 1200 instruções (no meio do ciclo) de um laço com 2400 instruções aritméticas. O laço foi executado 1.000 vezes e, em cada iteração, foram avaliados CCNT, RVEC, MONOTONIC e REALTIME usando a chamada de sistema o clock_gettime() para estes três últimos relógios. O experimento foi executado 100 vezes. O tempo médio de execução do programa de testes, sem ler os relógios do sistema, foi de 170,354 μs com um desvio padrão de 6,507 μs. A chamada do CCNT no meio do ciclo aumenta o tempo médio de execução em 1,333 μs em relação à execução do programa de teste sem a leitura dos relógios. O tempo médio de execução para as demais configurações estão mostradas na Tabela 4.1 para a frequência de 800 MHz. O experimento foi repetido para as frequências de 600 MHz (Tabela 4.2) e 400 MHz (Tabela 4.3).

Tabela 4.1: Tempo de Execução vs. Opção de Relógio (800 MHz) - Bloco de 2400 instruções

Opções de Relógio	Média (μs)	Desvio Padrão (μs)	Sobrecarga (μs)	Sobrecarga (%)
CCNT	171,687	6,729	1,333	0,78
REALTIME	173,013	6,891	2,659	1,56
MONOTONIC	173,107	6,918	2,753	1,62
RVEC	174,502	7,419	4,148	2,43

Tabela 4.2: Tempo de Execução vs. Opção de Relógio (600 MHz) - Bloco de 2400 instruções

Opções de Relógio	Média (μs)	Desvio Padrão (μs)	Sobrecarga (μs)	Sobrecarga (%)
CCNT	229,571	13,315	1,781	0,78
REALTIME	231,262	13,349	3,472	1,52
MONOTONIC	231,382	13,343	3,592	1,57
RVEC	233,113	13,831	5,323	2,33

Tabela 4.3: Tempo de Execução vs. Opção de Relógio (400 MHz) - Bloco de 2400 instruções

Opções de Relógio	Média (μs)	Desvio Padrão (μs)	Sobrecarga (μs)	Sobrecarga (%)
CCNT	344,381	19,831	2,702	0,79
REALTIME	346,804	19,720	5,125	1,50
MONOTONIC	346,867	19,187	5,188	1,52
RVEC	349,680	20,448	8,002	2,34

Na Figura 4.7 estão todos os resultados do tempo de execução de cada relógio para as frequências de 800, 600 e 400 MHz comparados com o tempo base para um bloco de 2400 instruções.

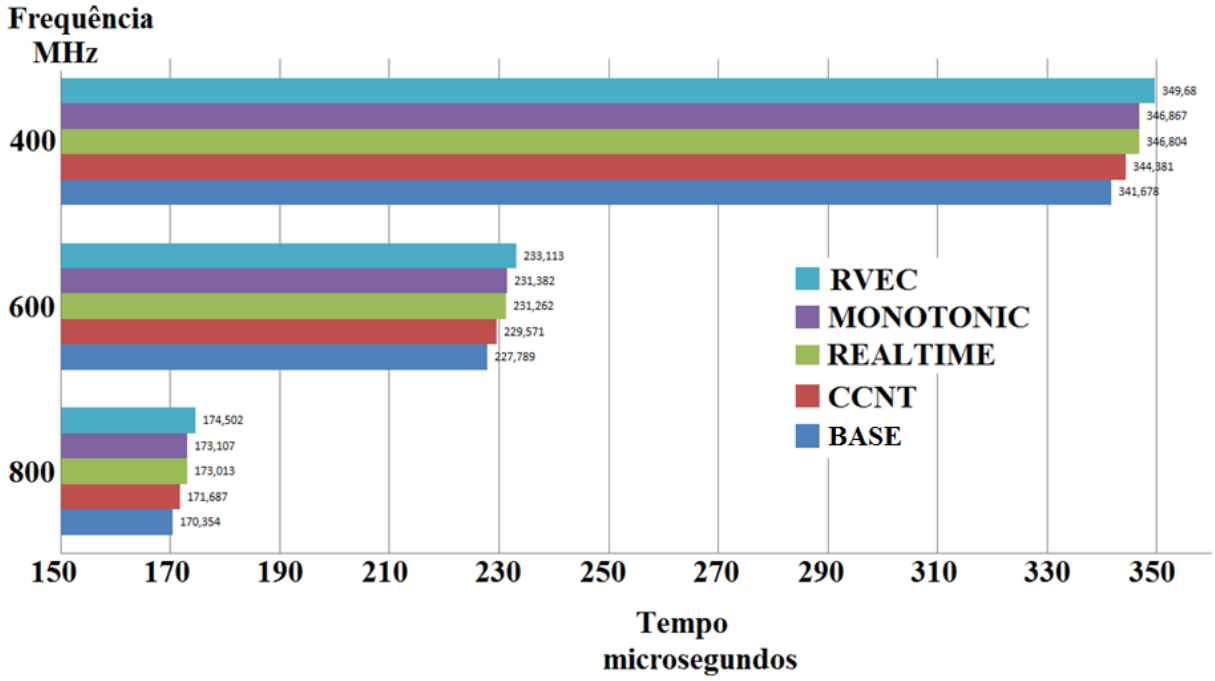

Figura 4.7: Tempo de execução dos relógios em um bloco de 2400 instruções

Sobrecarga para um bloco de 1200 instruções

O mesmo experimento de teste de sobrecarga foi repetido, porém em um bloco de 1200 instruções aritméticas:

Tabela 4.4: Tempo de Execução vs. Opção de Relógio (800 MHz) - Bloco de 1200 instruções

Opções de Relógio	Média (μs)	Desvio Padrão (μs)	Sobrecarga (μs)	Sobrecarga (%)
CCNT	86,023	4,930	0,67	0,78
REALTIME	87,285	5,390	1,932	2,26
MONOTONIC	87,438	5,309	2,085	2,44
RVEC	88,789	5,614	3,436	4,02

Tabela 4.5: Tempo de Execução vs. Opção de Relógio (600 MHz) - Bloco de 1200 instruções

Opções de Relógio	Média (μs)	Desvio Padrão (μs)	Sobrecarga (μs)	Sobrecarga (%)
CCNT	114,99	7,787	0,926	0,81
REALTIME	116,548	8,157	2,484	2.17
MONOTONIC	116,700	7,989	2,636	2,31
RVEC	118,330	8,075	4,266	3,74

Tabela 4.6: Tempo de Execução vs. Opção de Relógio (400 MHz) - Bloco de 1200 instruções

Opções de Relógio	Média (μs)	Desvio Padrão (μs)	Sobrecarga (μs)	Sobrecarga (%)
CCNT	172,428	11,96	0,743	0,43
REALTIME	174,438	10,62	2,753	1,60
MONOTONIC	174,746	13,17	3,061	1,78
RVEC	177,462	14,67	5,777	3,36

Na Figura 4.8 estão todos os resultados do tempo de execução de cada relógio para as frequências de 800, 600 e 400 MHz comparados com o tempo base para um bloco de 1200 instruções.

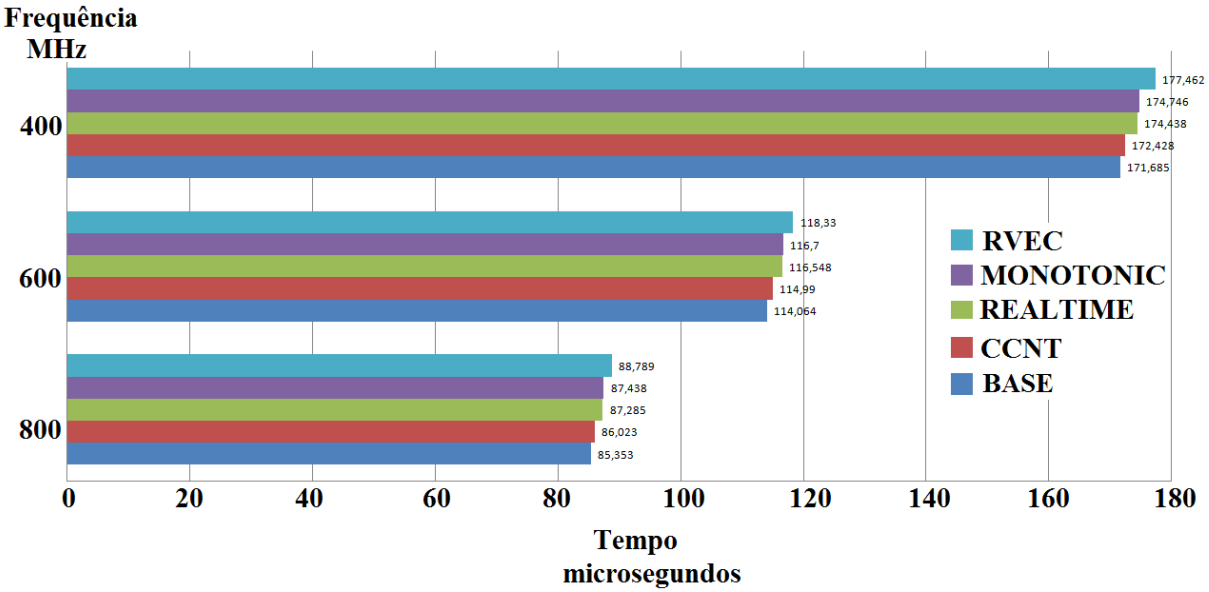

Figura 4.8: Tempo de execução dos relógios em um bloco de 1200 instruções

Em todas as frequências, para um bloco de 1200 instruções, o RVEC apresentou maior sobrecarga. O relógio de menor sobrecarga em todas as frequências foi o próprio CCNT.

Sobrecarga no acesso ao contador

A Tabela 4.7 mostra as sobrecargas medidas para cada acesso ao contador CCNT para as frequências de 800, 600 e 400 MHz. Os valores são os mesmos para os blocos de 1200 e 2400 instruções.

Tabela 4.7: Sobrecarga no acesso ao contador CCNT

Frequência (MHz)	Tempo de acesso (ns)
800	33,75
600	45,00
400	67,00

4.2.2 Precisão

Neste experimento, comparou-se a precisão do tempo dos relógios do sistema RVEC, MONOTONIC e REALTIME contra a precisão do CCNT.

Precisão para um bloco de 2400 instruções

O experimento foi repetido 100 vezes e mediu-se a duração média de um laço com 2400 instruções aritméticas, onde o laço foi executado 1.000 vezes. Os resultados são apresentados nas Tabelas 4.8, 4.9 e 4.10 para as frequências de 800, 600 e 400MHz, respectivamente.

Tabela 4.8: Precisão de Tempo vs. Opção de Relógio (800 MHz) - Bloco de 2400 instruções

Opções de Relógio	Média (μs)	Desvio Padrão (μs)
CCNT	170,439	6,554
REALTIME	171,459	7,140
MONOTONIC	172,047	7,086
RVEC	172,820	7,248

Tabela 4.9: Precisão de Tempo vs. Opção de Relógio (600 MHz) - Bloco de 2400 instruções

Opções de Relógio	Média (μs)	Desvio Padrão (μs)
CCNT	227,761	12,441
REALTIME	229,205	13,689
MONOTONIC	229,994	14,019
RVEC	230,960	13,908

Tabela 4.10: Precisão de Tempo vs. Opção de Relógio (400 MHz) - Bloco de 2400 instruções

Opções de Relógio	Média (μs)	Desvio Padrão (μs)
CCNT	341,642	19,060
REALTIME	343,705	20,004
MONOTONIC	344,722	18,882
RVEC	346,403	20,295

Precisão para um bloco de 1200 instruções

Neste experimento, comparou-se a precisão do tempo dos relógios do sistema RVEC, MONOTONIC e REALTIME contra a precisão do CCNT. O experimento foi repetido 100 vezes e mediu-se a duração média de um laço com 1200 instruções aritméticas, onde o laço foi executado 1.000 vezes. Os resultados são apresentados nas Tabelas 4.11, 4.12 e 4.13 para as frequências de 800, 600 e 400MHz, respectivamente.

Tabela 4.11: Precisão de Tempo vs. Opção de Relógio (800 MHz) - Bloco de 1200 instruções

Opções de Relógio	Média (μs)	Desvio Padrão (μs)
CCNT	85,330	4,999
REALTIME	86,427	5,582
MONOTONIC	86,745	5,594
RVEC	87,758	5,667

Tabela 4.12: Precisão de Tempo vs. Opção de Relógio (600 MHz) - Bloco de 1200 instruções

Opções de Relógio	Média (μs)	Desvio Padrão (μs)
CCNT	114,072	7,697
REALTIME	115,36	7,812
MONOTONIC	115,765	7,642
RVEC	117,123	7,762

Tabela 4.13: Precisão de Tempo vs. Opção de Relógio (400 MHz) - Bloco de 1200 instruções

Opções de Relógio	Média (μs)	Desvio Padrão (μs)
CCNT	171,629	12,616
REALTIME	173,026	12,940
MONOTONIC	173,517	12,552
RVEC	175,616	13,424

4.3 RVEC e CLOCK_MONOTONIC

Um experimento simples, sem utilização de blocos de instruções foi feito para avaliar a propriedade SIP no RVEC e no CLOCK_MONITONIC. Consiste de um programa com uma chamada sequencial do relógio para leitura do valor do tempo dentro de um laço. Para cada relógio comparou-se a leitura dos valores de tempos subsequentes. Cada vez que o valor do tempo posterior fosse menor ou igual ao valor do tempo anterior, o programa era interrompido, o que significa violação da propriedade SIP. Em todos os experimentos feitos para o RVEC, o programa não foi interrompido nenhuma vez, portanto não houve violação da propriedade SIP. O mesmo experimento efetuado com o CLOCK_MONOTONIC frequentemente, e de forma aleatória, apresentava valores de tempos consecutivos iguais, o que ocasionava a saída do programa, violando a propriedade SIP.

Capítulo 5

Trabalhos Relacionados

- "Attaining Strictly Increasing and Precise Time Count in Energy-efficient Computer Systems"[3] é a proposta original do RVEC como uma alternativa para o relógio do sistema, que tem a propriedade de contar o tempo de forma estritamente crescente e precisa (Strictly Increasing and Precise - SIP). Neste trabalho os autores mostraram que os nós em um *cluster* de computadores usando RVEC podem permanecer sincronizados globalmente após sincronização inicial de seus RVECs através do uso de um algoritmo cliente-servidor de sincronização remoto. Como resultado desta sincronização, todo nó terá construído um Relógio Global de Alta Precisão (High Precision Global Clock - HPGC) local, o qual não somente será sincronizado com o HPGC do servidor de sincronização como também será sincronizado globalmente com todos os outros HPGCs no *cluster*. Com isso o HPGCs ficam livres da necessidade de ressincronização. Neste trabalho utilizou-se contadores TSC presentes na arquitetura x86/64, enquanto nesta dissertação foram utilizados contadores CCNT presentes na arquitetura ARM/Raspberry Pi.

- Em "Clock synchronization in highend computing environments: a strategy for minimizing clock variance at runtime"[26], os autores desenvolveram um esquema de sincronização do relógio cliente-servidor baseado em software para MPI que melhorou a variância tempo médio de um conjunto de nós de 20,0 ms sob padrão NTP para 2,29 μs. O esquema é baseado em uma fase de sincronização inicial entre o nó de referência e todos os outros nós de um sistema de computador paralelo em conjunto com várias etapas de ressincronização realizadas durante chamadas coletivas de MPI. Em comparação ao trabalho original do RVEC, a solução HPGC só requer uma fase de sincronização inicial entre o RVEC do nó de referência e os RVECs dos outros nós graças a propriedade SIP do RVEC.

- "Robust Synchronization of Absolute and Difference Clocks Over

Networks"[27] (RADClock) é um sistema construído em relógios do sistema (por exemplo, HPET) ou contadores de tempo (por exemplo, TSC), que fornece informações sobre o tempo global, bem como o tempo global absoluto para a sincronização de nós da rede. No entanto, RADClock depende de NTP para ressincronização periódica. Além disso, o uso TSC limita a sua aplicabilidade em processadores com múltiplos núcleos.

- "High-Precision Relative Clock Synchronization Using Time Stamps Counters"[11] trata do desenvolvimento de um relógio global usando o TSC como relógio de base juntamente com um algoritmo de sincronização remoto que é semelhante ao do NTP. Devido ao uso direto de TSC, tal relógio global não pode funcionar corretamente com processadores que possuem DVFS ou vários núcleos.

- Em "Hardware supported synchronization primitives for clusters"[28], os autores propuseram um hardware auxiliar de sincronização de rede utilizando um gerador de pulsos remoto. Em um *cluster* de computadores, esta sincronização de rede assegura que todos os nós do *cluster* recebem simultaneamente o pulso de clock remoto e o usa para atualizar o relógio do nó local sem envolver o sistema operacional. Apesar de garantir a contagem estritamente crescente e precisa de relógios locais, a solução proposta depende de hardware dedicado.

Capítulo 6

Conclusão e Trabalhos Futuros

O presente trabalho consiste na descrição do método do RVEC como um relógio virtual feito inteiramente em software para o sistema operacional Linux e sua implementação e avaliação na plataforma ARM/Raspberry Pi. RVEC é uma alternativa para o relógio do sistema, proposto originalmente por DUTRA *et al.* [3, 4], com a propriedade de contagem de tempo estritamente crescente e precisa para arquitetura x86. A construção do RVEC para arquitetura x86 tem como base o contador de ciclos TSC. Similarmente, para o Raspberry foi utilizado o contador denominado CCNT presente no processador como base para contagem do tempo. A avaliação experimental mostrou que o RVEC manteve contagem estritamente crescente e precisa por longos períodos. A adaptação do código foi feita sem otimização. Desta forma, o RVEC apresentou sobrecarga adicional em relação aos relógios representativos do Linux, CLOCK_MONOTONIC e CLOCK_REALTIME.

Entretanto, o experimento descrito na Seção 4.3, onde é feita a comparação do RVEC e o CLOCK_MONOTONIC, mostra que o RVEC possui propriedade SIP, o que não ocorre com o CLOCK_MONOTONIC.

Os resultados dos testes realizados no RVEC submetido à mudança de frequência, mostraram a contagem crescente e precisa além de manter a contagem correta e coerente de forma independente da frequência utilizada. Os resultados obtidos demonstram que o RVEC pode ser uma alternativa eficaz para o relógio de sistema, especialmente para sistemas de computadores onde há variação de frequência visando economia de energia. Esta técnica de economia, inicialmente limitada a computadores portáteis e dispositivos móveis, vem sendo aplicada também em computadores de alta performance [29].

6.1 Trabalhos Futuros

Os trabalhos futuros relacionados ao RVEC para arquitetura ARM são:

- Investigar a maior sobrecarga do RVEC em relação à sobrecarga apresentada pelos relógios CLOCK_MONOTONIC e CLOCK_REALTIME. A implementação do código do RVEC na arquitetura ARM foi feita sem otimização. Para otimizar o código é necessário o uso de ferramentas de monitoração e performance [30]. Como visto no Capítulo 2, o Raspberry possui contadores de desempenho que contam eventos microarquiteturais em nível de hardware. Estes eventos podem indicar a presença de gargalos. As informações obtidas da contagem destes eventos auxiliam na depuração e no ajuste do código para um melhor desempenho.

- Implementar o RVEC e avaliar o impacto do escalonamento de frequência em arquiteturas ARM com múltiplos núcleos. O Raspberry possui um processador com apenas um núcleo, o que inviabilizaram testes frente à migração de processos, embora o código tenha sido implementado e preparado para tais testes. Como exemplos de processadores com mais de um núcleo podem ser citados os processadores ARM Cortex-A9 MPCore e ARM Cortex-A15 MPCore.

- Implementar o RVEC e avaliar o desempenho em dispositivos móveis baseados na arquitetura ARM. A popularização dos celulares, *smartphones* e *tablets* vem aumentando a demanda por desenvolvimento de soluções para estas plataformas. Os dispositivos móveis modernos possuem processadores com temporizadores de hardware extremamente precisos, como por exemplo o ARM Cortex A9, presente no *smartphone* Samsung Galaxy SII, que possui um registrador contador temporizador global de 64 bits. A leitura deste contador consome apenas um ciclo do processador [31].

Referências Bibliográficas

[1] INTEL. "IA-PC HPET (High Precision Event Timers) Specification". Disponível em `http://www.intel.com/content/dam/www/public/us/en/documents/technical-specifications/software-developers-hpet-spec-1-0a.pdf`, 2004.

[2] MILLS, D. "Network Time Protocol (Version 3) Specification, Implementation". 1992.

[3] DUTRA, D., WHATELY, L., AMORIM, C. "Attaining Strictly Increasing and Precise Time Count in Energy-efficient Computer Systems", *Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 65–72, out. 2013.

[4] DUTRA, D., WHATELY, L., AMORIM, C. *A Highly Effective System Clock for Energy-efficient Computer Systems*. Tech. Rep. ES-745/13, COPPE,UFRJ, Rio de Janeiro, 2013.

[5] MOORE, G. "Cramming More Components Onto Integrated Circuits", *Proceedings of the IEEE*, v. 86, n. 1, pp. 82–85, Jan 1998. ISSN: 0018-9219. doi: 10.1109/JPROC.1998.658762.

[6] MOLLICK, E. "Establishing Moore's Law", *Annals of the History of Computing, IEEE*, v. 28, n. 3, pp. 62–75, July 2006. ISSN: 1058-6180. doi: 10.1109/MAHC.2006.45.

[7] PENG, L., PEIR, J.-K., PRAKASH, T., et al. "Memory Performance and Scalability of Intel's and AMD's Dual-Core Processors: A Case Study". In: *Performance, Computing, and Communications Conference, 2007. IPCCC 2007. IEEE Internationa*, pp. 55–64, April 2007. doi: 10.1109/PCCC.2007.358879.

[8] MUTHANA, P., SWAMINATHAN, M., TUMMALA, R., et al. "Packaging of multi-core microprocessors: tradeoffs and potential solutions", pp. 1895–1903 Vol. 2, May 2005. ISSN: 0569-5503. doi: 10.1109/ECTC.2005.1442057.

[9] KECKLER, S. *Multicore processors and systems*. New York London, Springer, 2009. ISBN: 978-1-4419-0263-4.

[10] SULEIMAN, D., IBRAHIM, M., HAMARASH, I. "Dynamic voltage frequency scaling (DVFS) for microprocessors power and energy reduction". In: *4th International Conference on Electrical and Electronics Engineering*, 2005.

[11] TIAN, G.-S., TIAN, Y.-C., FIDGE, C. "High-Precision Relative Clock Synchronization Using Time Stamp Counters". In: *Engineering of Complex Computer Systems, 2008. ICECCS 2008. 13th IEEE International Conference on*, pp. 69–78, March 2008. doi: 10.1109/ICECCS.2008.39.

[12] INTEL CORPORATION. *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. N. 248966-018. March 2009.

[13] "Standard for Information TechnologyPortable Operating System Interface (POSIX(R)) Base Specifications, Issue 7", *IEEE Std 1003.1, 2013 Edition (incorporates IEEE Std 1003.1-2008, and IEEE Std 1003.1-2008/Cor 1-2013)*, pp. 1–3906, April 2013. doi: 10.1109/IEEESTD.2013.6506091.

[14] CLIFFORD, N., BROUWER, A. *Linux Programmer's Manual*. Disponível em: `http://man7.org/linux/man-pages/man2/clock_gettime.2.html`.

[15] SLOSS, A., SYMES, D., WRIGHT, C. *ARM System Developer's Guide: Designing and Optimizing System Software (The Morgan Kaufmann Series in Computer Architecture and Design)*. Morgan Kaufmann, 2004. ISBN: 9780080490496.

[16] FURBER, S. *ARM System-on-Chip Architecture (2nd Edition)*. Addison-Wesley Professional, 2000. ISBN: 0201675196.

[17] LANGBRIDGE, J. A. *Professional Embedded ARM Development*. Wrox, 2014.

[18] UPTON, E., HALFACREE, G. *Raspberry Pi user guide*. Wiley, 2012.

[19] SCHMIDT, M., CARTER, J. *Raspberry Pi*. The Pragmatic Bookshelf, 2012.

[20] BROADCOM. *BCM2835-ARM-Peripherals*.

[21] ARM. *ARM1176JZ-S Technical Reference Manual*, .

[22] HOPPER, J. "The CPUfreq subsystem". Disponível em `http://www.ibm.com/developerworks/library/l-cpufreq-1/l-cpufreq-1-pdf.pdf`.

[23] BRODOWSKI, D., GOLDE, N. "CPU frequency and voltage scaling code in the Linux kernel". Disponível em `https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt`.

[24] LOVE, R. *Linux Kernel Development*. 3 ed. New York, Pearson Education, Inc., 2010.

[25] BRYANT, R. E., O'HALLARON, D. R. *Computer systems : a programmer's perspective*. Boston, Prentice Hall, 2011. ISBN: 978-0136108047.

[26] JONES, T., KOENIG, G. A. "Clock synchronization in high-end computing environments: a strategy for minimizing clock variance at runtime", *Concurrency and Computation: Practice and Experience*, v. 25, n. 6, pp. 881–897, 2013. ISSN: 1532-0634. doi: 10.1002/cpe.2868. Disponível em: <`http://dx.doi.org/10.1002/cpe.2868`>.

[27] VEITCH, D., RIDOUX, J., KORADA, S. "Robust Synchronization of Absolute and Difference Clocks Over Networks", *Networking, IEEE/ACM Transactions on*, v. 17, n. 2, pp. 417–430, April 2009. ISSN: 1063-6692. doi: 10.1109/TNET.2008.926505.

[28] DE SOUZA, A. F., DE SOUZA S. F., DE AMORIM, C. L., et al. "Hardware supported synchronization primitives for clusters". In: *PDPTA'08*, pp. 520–526, 2008.

[29] MINARTZ, T., LUDWIG, T., KNOBLOCH, M., et al. "Managing hardware power saving modes for high performance computing". In: *Green Computing Conference and Workshops (IGCC), 2011 International*, pp. 1–8, July 2011. doi: 10.1109/IGCC.2011.6008581.

[30] PARADIS, C., WEAVER, V. *Enabling Raspberry Pi Performance Counter Support on Linux perf_event*. Tech. Rep. UMaine ECE Tech Report 2014-2, University of Maine, Maine, 2014.

[31] ARM. *Cortex-A9 MPCore Technical Reference Manual*, .

[32] "R-Pi Configuration File". Disponível em `http://elinux.org/R-Pi_ConfigurationFile`. Último acesso em outubro de 2014.

Apêndice A

Preparação do Ambiente de Testes

O desenvolvimento do software foi feito a partir da alteração do *kernel* no Linux. Para isto o ambiente para testes consta de:

Hardware:

- Computador Raspberry Pi;
- Um cabo micro USB com fonte de 5V e 0,7A para alimentar a placa;
- Um cartão SD para gravar o sistema operacional;
- Um monitor com entrada HDMI; e
- Um teclado e um mouse USB.

Software: Sistema operacional Raspbian "wheezy"(3.2.27), versão otimizada do Debian. O Raspberry utilizado é um computador com apenas um processador com um núcleo e 512 Kb de memória.

Procedimentos para instalação do sistema operacional no cartão SD

1. Baixar a imagem do linux de `http://downloads.raspberrypi.org/images/raspbian/2012-12-16-wheezy-raspbian/2012-12-16-wheezy-raspbian.zip`;
2. Descompactar o arquivo e extrair o arquivo imagem. O arquivo .img só pode ser gravado no cartão SD por um software especial para este fim (por exemplo, o Win32DiskImager);
3. Baixar o Win32DiskImager (específico para windows) de `http://sourceforge.net/projects/win32diskimager/files/latest/download` e instalar. Este é um programa para salvar e restaurar imagens de unidades removíveis. Ele pode ser usado para escrever as imagens de inicialização em um cartão SD ou dispositivo USB, tornandos-o inicializáveis; e

4. Para salvar o sistema operacional, inserir o cartão SD no PC e efetuar a gravação (Figura A.1).

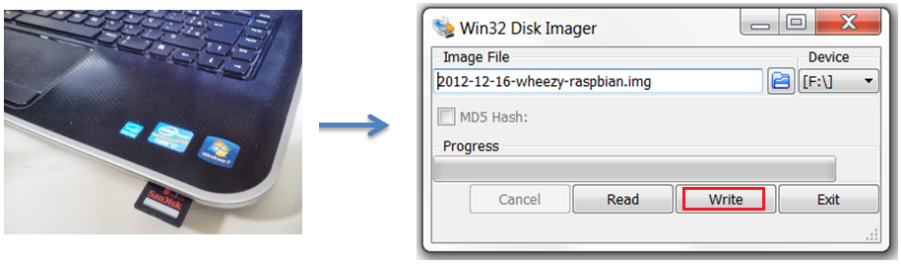

Figura A.1: Gravação do sistema operacional no cartão SD

Processo de boot do Raspberry

O cartão SD preparado com o sistema operacional possui duas partições: na primeira, formatada em FAT32, estão os arquivos bootcode.bin, start.elf, fixup.dat, config.txt, kernel.img e cmdline.txt e na segunda está o sistema de arquivos do root.

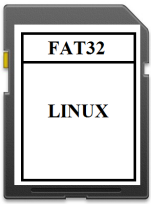

Figura A.2: As partições do cartão SD

Ao iniciar o Raspberry, a GPU assume o controle do boot antes do processador principal (a CPU ARM1176JZ-F) e o primeiro código a ser executado está armazenado em ROM, dentro do SoC. Este código procura na primeira partição (formatada em FAT32) do cartão SD o arquivo bootcode.bin, carrega na cache L2. Ao executar, inicializa a SDRAM e carrega os arquivos binários start.elf e fixup.dat para serem executados na GPU com a responsabilidade de configurar o hardware de acordo com o arquivo config.txt. Dentre as várias configurações possíveis, pode-se definir mapeamento de memória e especificar parâmetros para carregar o *kernel* Linux. Uma especificação completa encontra-se em `http://elinux.org/RPi_config.txt`. Após ler o arquivo de configuração, ele irá carregar o arquivo kernel.img que é a imagem do *kernel* Linux. Antes de executar a imagem do *kernel*, ele é capaz de passar as linhas de comandos para o *kernel*, que pode ser definida no arquivo cmdline.txt, ou em uma variável no arquivo config.txt. No primeiro boot do sistema surge a tela da Figura A.3 para configuração do sistema. É necessário selecionar a opção *expand-rootf*, caso a capacidade de armazenamento do cartão de boot utilizado seja maior do que 2GB. Habilitar *ssh* também é importante para comunicação com o Raspberry via rede.

Figura A.3: Tela para configuração do sistema

Apêndice B

Compilação Cruzada e Modificação do Kernel

Compilar um novo *kernel* no Raspberry leva horas. A solução foi usar um compilador em outra plataforma mais potente para gerar os binários (compilação cruzada). Para executar esta compilação, utilizamos um computador com o processador Intel i7 2.1GHz, 8GB RAM e sistema operacional Ubuntu 12.04. Para isto é necessário instalar um conjunto de ferramentas de compilação, denominado toolchain, no computador mais potente e efetuar a compilação cruzada.

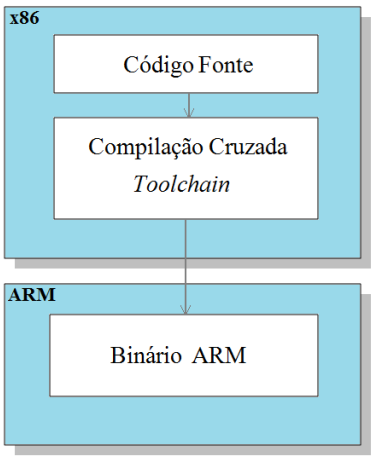

Figura B.1: Compilação Cruzada - Toolchain

O bloco de cima na Figura B.1 representa o computador de desenvolvimento (arquitetura x86) e o bloco de baixo a arquitetura que representa o raspberry. Utilizando o toolchain, a compilação cruzada vai gerar um binário para o raspberry.

Procedimentos para geração do novo kernel no Raspberry

Os procedimentos para geração do *kernel* são executados a cada nova alteração do código. Na primeira geração do *kernel* todos os itens a seguir deverão ser executados.

Para gerações subsequentes, basta seguir a partir do item 5.

1. O comando abaixo, instala dependências, git e toolchain no Ubuntu:

   sudo apt-get install git-core gcc-4.6-arm-linux-gnueabi

2. Criação de links simbólicos (*symlink*) para o compilador:

   sudo ln -s /usr/bin/arm-linux-gnueabi-gcc-4.6
   /usr/bin/arm-linux-gnueabi-gcc

3. Criar o diretório para os fontes e ferramentas e criar clones com o git:

   mkdir rpi
   cd rpi
   git clone `https://github.com/rpi/tools.git`
   git clone `https://github.com/rpi/linux.git`
   cd linux

4. Alterando a versão do linux para rpi-3.2.27

   git checkout rpi-3.2.27

5. Definir as variáveis de ambiente:

   export CCPREFIX=/home/edilson/rpi/tools/arm-bcm2708/arm-bcm2708-
   linux-gnueabi/bin/arm-bcm2708-linux-gnueabi-
   export KERNEL_SRC=/home/edilson/rpi/linux

6. Obter /proc/config.gz do Raspberry e extrair na máquina com o Ubuntu:

   scp pi@10.10.10.211:/proc/config.gz ./ (transfere para Ubuntu)
   zcat config.gz >.config (descompacta .config)
   mv .config $KERNEL_SRC (move .config para KERNEL_SRC)

7. No Ubuntu - preparar o kernel com old config:

   make ARCH=arm CROSS_COMPILE=${CCPREFIX} oldconfig

8. No Ubuntu: habilitar o paralelismo com o flag -j e usar os 7 núcleos para aumentar a velocidade do processo de compilação do kernel

make ARCH=arm CROSS_COMPILE=${CCPREFIX} -j8

9. Após a compilação, criar diretórios para os módulos:

   mkdir ../modules

10. Compilar e instalar os modulos no diretório temporário:

    make modules_install ARCH=arm CROSS_COMPILE=${CCPREFIX} INSTALL_MOD_PATH=../modules/

11. Usar o executável imagetool-uncompressed.py do repositório tools para preparar o kernel para o raspberry.

    cd ../tools/mkimage/ ./imagetool-uncompressed.py ../../linux/arch/arm/boot/Image

12. Isto cria kernel.img no diretório atual. Plugar o cartão SD com a imagem onde será instalado o novo kernel. Apagar kernel.img do cartão e substituir pelo novo kernel.img na partição de boot. No Ubuntu:

    sudo rm /media/id-particao-boot/kernel.img (id-particao-boot é o identificador montado)
    sudo mv kernel.img /id-particao-boot/

13. Remover os diretórios /lib/modules e lib/firmware, substitutindo "id-particao-rootfs" com o identificador da partição do sistema de arquivos do root montado no Ubuntu.

    sudo rm -rf /media/id-particao-rootfs/lib/modules/ sudo rm -rf /media/id-particao-rootfs/lib/firmware/

14. Copiar os novos módulos e firmware:

    cd ../../modules/ sudo cp -a lib/modules/ /media/id-particao-rootfs/lib/ sudo cp -a lib/firmware/ /media/id-particao-rootfs/lib/ sync

15. Ejetar o cartão e reiniciar o raspberry com o novo *kernel*.

A aplicação do *patch* do RVEC, originalmente feito para x86, versão 3.2 do Linux, foi aplicada na versão 3.2.27 (Raspberry). A Tabela B.1 mostra a correlação dos arquivos entre as duas arquiteturas.

Tabela B.1: Correlação dos arquivos x86 x ARM

x86	ARM
arch/x86/Kconfig	arch/arm/Kconfig
arch/x86/kernel/smpboot.c	xxxxxxxxxxxxxxxx
arch/x86/vdso/vclock_gettime.c	xxxxxxxxxxxxxxxx
drivers/cpufreq/acpi-cpufreq.c	drivers/cpufreq/acpi-cpufreq.c
include/linux/sched.h	include/linux/sched.h
include/linux/time.h	include/linux/time.h
init/main.c	init/main.c
kernel/fork.c	kernel/fork.c
kernel/kthread.c	kernel/kthread.c
kernel/posix-timers.c	kernel/posix-timers.c
kernel/sched.c	kernel/sched.c
kernel/time/timekeeping.c	kernel/time/timekeeping.c

Apêndice C

Controle de Frequência no Raspberry

Por padrão, o Raspberry vem configurado com a frequência fixa em 700 MHz e o governador configurado para *ondemand*. Neste caso, o governador não terá influência na frequência de operação do processador, com a política determinando a frequência fixa. Entretanto, é possível definir as frequências mínima e máxima de operação editando o arquivo /boot/config.txt [32]. No exemplo a seguir, as frequências mínima e máxima são definidas em 400 e 800 MHz, respectivamente:

```
sudo nano /boot/config.txt
```

```
. . .
#uncomment to overclock the arm. 700 MHz is the default.
arm_freq=800
arm_freq_min=400
. . .
```

Os governadores disponíveis no Raspberry podem ser verificados através dos seguintes comandos como usuário *root*:

```
cd /sys/devices/system/cpu
```

```
cat cpu0/cpufreq/scaling_available_governors
```

Para alterar para alterar para o governador *userspace*:

```
echo userspace >cpu0/cpufreq/scaling_governor
```

O comando a seguir mostra qual o governador está ativo:

```
cat cpu0/cpufreq/scaling_governor
```

Para controlar as frequências de operação do Raspberry e definir quais governa-

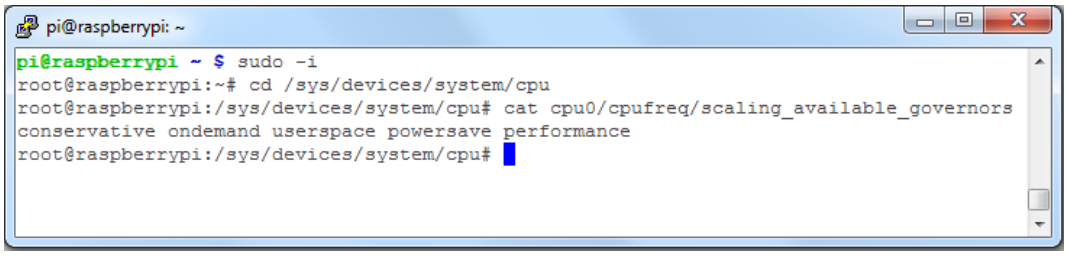

Figura C.1: Governadores disponíveis no Raspberry

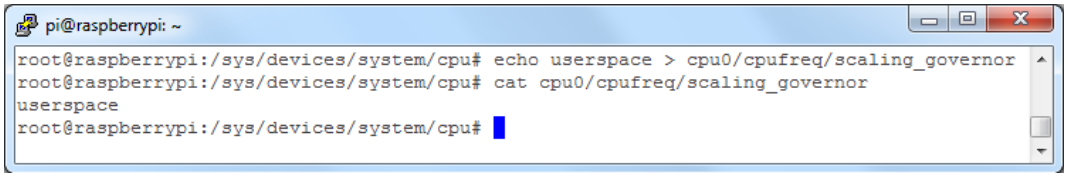

Figura C.2: Alterando o governador para *userspace*

dores utilizar para execução dos testes de modo mais fácil, é necessário instalar um utilitário denominado cpufrequtils. Com este utilitário é possível ajustar as opções de gerenciamento de energia a partir da linha de comando. Está disponível nos repositórios da maioria das distribuições Linux tradicionais de software. Para instalar no Raspberry:

```
sudo apt get cpufrequtils
```

O utilitário possui dois programas principais: cpufreq-info e cpufreq-set. O comando cpufreq-info informa as configurações de frequência do processador:

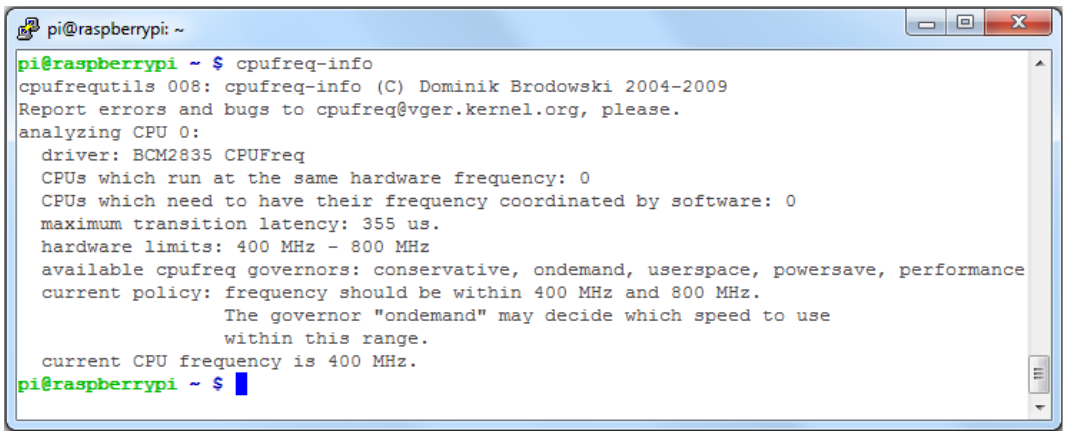

Figura C.3: Configuração de frequência do processador

O comando cpufreq-set permite definir as seguintes configurações:

- -d frequência mínima;
- -u frequência máxima;

- -f define uma frequência específica (apenas no modo governador *userspace*);
- -g define um governador específico; e
- -c define um processador específico.

Por exemplo, para alterar o governador para *ondemand*, basta executar o seguinte comando:

```
sudo cpufreq-set -g ondemand
```

Neste caso, com as frequências mínima e máxima definidas em 400 e 800 MHz, respectivamente, é possível observar o valor corrente da frequência de operação do processador através de outro *shell* utilizando o seguinte comando:

```
watch -n 1 'cpufreq-info — grep "current CPU frequency is"'
```

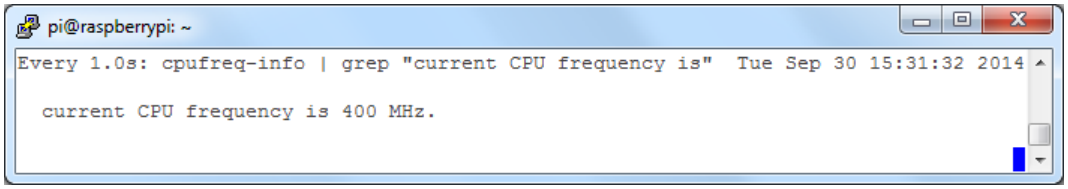

Figura C.4: Observar o valor corrente da frequência

Com o governador definido como *ondemand*, a frequência de operação do processador será de 400 MHz quando ocioso e de 800 MHz quando houver carga de trabalho. Para fixar a frequência de operação é necessário definir antes o governador para *userspace*. Os comandos necessários, por exemplo, para fixar a frequência de operação em 800 MHz, são:

```
sudo cpufreq-set -g userspace
```

```
sudo cpufreq-set -f 800MHz
```

O controle da frequência de operação através dos comandos descritos (em linha de comando ou usando os comandos do utilitário cpufrequtils) permitiu avaliar o comportamento dos relógios na execução dos testes para diversas configurações.