UNIVERSIDADE FEDERAL DE SANTA CATARINA

HEURÍSTICA DE POWER-CAP EM SISTEMAS MULTICORE DE TEMPO-REAL COM USO DE MONITORAMENTO DE PERFORMANCE

Leonardo Passig Horstmann

Florianópolis - SC

2018 / 2

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

HEURÍSTICA DE POWER-CAP EM SISTEMAS MULTICORE DE TEMPO-REAL COM USO DE MONITORAMENTO DE PERFORMANCE

Leonardo Passig Horstmann

Trabalho De Conclusão De Curso apresentado na Universidade Federal de Santa Catarina (UFSC) como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Florianópolis – SC

2018 / 2

Leonardo Passig Horstmann

HEURÍSTICA DE POWER-CAP EM SISTEMAS MULTICORE DE TEMPO-REAL COM USO DE MONITORAMENTO DE PERFORMANCE

Trabalho De Conclusão De Curso apresentado na Universidade Federal de Santa Catarina (UFSC) como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Orientador: Prof. Dr. Antônio Augusto Fröhlich

INE / UFSC

Coorientador: Prof. Dr. Giovani Gracioli

ECE / UW

Banca Examinadora

Dr. Tiago Rogério Mück

ARM

Prof. Dr. Lucas Francisco Wanner

IDC / UniCamp

SUMÁRIO

LISTA DE FIGURAS 6
LISTA DE TABELAS 9
LISTA DE REDUÇÕES 10
RESUMO 11
INTRODUÇÃO 12
OBJETIVOS 14
OBJETIVOS GERAIS 14
OBJETIVOS ESPECÍFICOS 14
ORGANIZAÇÃO DO TEXTO 14
FUNDAMENTAÇÃO TEÓRICA 16
EPOS 16
INTEL IA-32 16
COMPUTAÇÃO MULTI-CORE 17
PMU 18
PMU NA ARQUITETURA SANDY BRIDGE 19
SISTEMAS DE TEMPO-REAL 19
TIPOS DE SISTEMAS DE TEMPO-REAL 20
ESCALONADORES DE TEMPO REAL 21
RM - RATE MONOTONIC 21
EDF – EARLIEST DEADLINE FIRST 22
POWER-CAP 22
POWER-CAPPING EM HARDWARE E INTERFACE RAPL 23
DVS E MODULAÇÃO DE CLOCK 24
MODULAÇÃO DE CLOCK POR DUTY CYCLE 24
NÃO INTRUSIVIDADE 26
DATA MINING 27
WEKA 27
MATERIAIS UTILIZADOS 28
IMPLEMENTAÇÃO 29
LEITURA DE TEMPERATURA 29
MODULAÇÃO DE CLOCK 30
MEDIÇÃO DE CONSUMO DE ENERGIA DO PROCESSADOR 30
SISTEMA NÃO INTRUSIVO DE CAPTURAS 33
ESTRUTURA DE ARMAZENAMENTO 34
NÃO INTRUSIVIDADE 36
VERIFICAÇÃO DA NÃO INTRUSIVIDADE 37

ESCALONADORES PARA MONITORAMENTO 38
CONFIGURAÇÃO DOS EVENTOS PMU 39
ENVIO PARA O BANCO 40
METADADOS DE ENVIO 40
SISTEMA EXTERNO 41
SISTEMA INTERNO 41
CAPTURA E ANÁLISE DOS DADOS 43
DEFINIÇÃO DOS TASK-SETS E DAS OPERAÇÕES EXECUTADAS 43
CAPTURA DE DADOS 46
ANÁLISE 47
ESTIMATIVA DE CONSUMO 48
TDP 50
PERFIL TÉRMICO E TTV 50
PREPARAÇÃO 53
SELEÇÃO DE DADOS 53
LIMPEZA DE DADOS 53
INTEGRAÇÃO DE DADOS 54
TRANSFORMAÇÃO DOS DADOS 55
LEVANTAMENTO DE CORRELAÇÕES 56
DEFINIÇÃO DO POWER-CAP 59
USO DE CLASSIFICADORES E ÁRVORES DE DECISÃO 60
PREPARAÇÃO E DEFINIÇÃO DE CLASSES 60
GERAÇÃO DE ÁRVORES 61
DESENVOLVIMENTO DA HEURÍSTICA 69
AÇÕES DE CONTROLE 69
SUSPENSÃO CONTROLADA DE TAREFAS 69
LÓGICA DE REDUÇÃO DE FREQUÊNCIA DE OPERAÇÃO (DVS) 70
LÓGICA DE DESVIO DE ESCALONAMENTO 71
IMPLEMENTAÇÃO BASEADA NA ÁRVORE DE DECISÃO 71
RESULTADOS 79
TASK-SETS UTILIZADOS 79
MÉTODO DE AFERIÇÃO ENERGÉTICA 80
DESEMPENHO COM HEURÍSTICA DESABILITADA 80
DESEMPENHO COM HEURÌSTICA HABILITADA E COMPARATIVO 85
CONCLUSÃO 98
TRABALHOS FUTUROS 98
BIBLIOGRAFIA 100
ANEXOS 105

A1. TABELA DE EVENTOS INTEL SANDY BRIDGE NO EPOS 105
A2. CÓDIGO DO SISTEMA DE MONITORAMENTO DE PERFORMANCE 119
A3. CÓDIGO DA HEURÍSTICA DESENVOLVIDA 161
A4. ARTIGO PADRÃO SBC 179

LISTA DE FIGURAS

Figura 2.1 - Exemplo de *Layout* de um MSR……………………………………………………18

Figura 2.2 - *Layout* do registrador MSR_PKG_POWER_LIMIT……………..……….......23

Figura 2.3 - *Layout* do MSR IA32_CLOCK_MODULATION…………………….……......25

Figura 2.4 - Descrição dos *Duty Cycles*………………………………...…….……….25

Figura 3.1 - Divisão de componentes utilizada pela Interface RAPL…..…………………..30

Figura 3.2 - *Layout* do registrador MSR_RAPL_POWER_UNIT……………...….……....31

Figura 3.3 - *Layout* do MSR para leitura dos dados energéticos do Package…………....32

Figura 3.4 - *Layout* dos MSRs de leitura dos componentes PP0 e PP1…………………..32

Figura 3.5 - Estrutura de um Moment…………………………………………………………34

Figura 3.6 - Estrutura de Armazenamento....……………………………………………....35

Figura 4.1 - Diagrama de montagem do circuito de medição - Fluke………………………48

Figura 4.2 a) - Perfil Térmico do Processador Intel® Core™ i7-2600………...………...51

Figura 4.2 b) - Variação de Psi-CA para vários $T_{AMBIENT}$………………...…………………...52

Figura 4.3 - Aplicação do algoritmo *CorrelationAttributeEval* sobre dados capturados.....56

Figura 4.4 a) - Exemplo de evento ignorado por semelhança ao UnHalted Cycles……...58

Figura 4.4 b) - Exemplo de evento ignorado por semelhança ao Instructions Retired…..59

Figura 4.5 a) - Árvore de decisão simples - Descrição Textual…...………………………..62

Figura 4.5 b) - Árvore de decisão simples - Descrição Gráfica…..……………...………...62

Figura 4.6 a) - Árvore de decisão processo 2 - Representação Textual………..………...63

Figura 4.6 b) - Árvore de decisão processo 2 - Representação Gráfica………..………...64

Figura 4.7 - Árvore de decisão utilizada - Descrição Textual Completa………..………...65

Figura 4.8 - Árvore de decisão utilizada - Descrição Textual Sintetizada…………………67

Figura 5.1 a) - Pseudocódigo da heurística - parte dispatch………………………….......72

Figura 5.1 b) - Diagrama de fluxo de execução - Parte 1…………………………………...75

Figura 5.1 c) - Diagrama de fluxo de execução - Parte 2…………………………………...76

Figura 5.1 d) - Pseudocódigo da heurística - parte idle………………………………......78

Figura 6.1 - Histograma de consumo - Operações recursivas……………………………...81

Figura 6.2 - Frequências de consumo - Operações recursivas……………………………..81

Figura 6.3 – Sumário de consumo - Operações recursivas…………………………………82

Figura 6.4 - Histograma de consumo - Operações iterativas…………………..…………82

Figura 6.5 - Frequências de consumo - Operações iterativas……………………………...83

Figura 6.6 – Sumário de consumo - Operações iterativas.…………………………………83

Figura 6.7 - Histograma de consumo - Operações em memória..…………………………84

Figura 6.8 - Frequências de consumo - Operações em memória..…………………………84

Figura 6.9 - Sumário de consumo - Operações em memória..………………...………...85

Figura 6.10 - Histograma de consumo - operações recursivas com heurística…………...86

Figura 6.11 - Frequências de consumo - operações recursivas com heurística..………..86

Figura 6.12 - Sumário de consumo - operações recursivas com heurística……..….……87

Figura 6.13 - Histograma de consumo - operações em memória com heurística………...88

Figura 6.14 - Frequências de consumo - operações em memória com heurística……….89

Figura 6.15 - Sumário de consumo - operações em memória com heurística……………89

Figura 6.16 - Histograma de consumo - operações em memória com heurística………...90

Figura 6.17 - Frequências de consumo - operações em memória com heurística……….90

Figura 6.18 - Sumário de consumo - operações em memória com heurística...………….91

Figura 6.19 - Histograma de consumo - operações iterativas com heurística…....……….92

Figura 6.20 - Frequências de consumo - operações iterativas com heurística……………93

Figura 6.21 - Sumário de consumo - operações iterativas com heurística…...…………….93

Figura 6.22 - Histograma de consumo - operações recursivas com heurística…………...94

Figura 6.23 - Frequências de consumo - operações recursivas com heurística…………94

Figura 6.24 - Sumário de consumo - operações recursivas com heurística...…………….95

Figura 6.25 - Consumos Médios Sob Conjunto de Tarefas 2……………………………….97

LISTA DE TABELAS

Tabela 3.1 - Conjunto de Tarefas Utilizado para os testes de não intrusividade……...….37

Tabela 4.1 - Conjunto de tarefas 1..…………………………………………………..…....43

Tabela 4.2 - Conjunto de tarefas 2……………………………………………………………..44

Tabela 4.3 - Conjunto de tarefas 3……………………………………………………………..45

Tabela 4.4 - Uso de cada CPU nos Conjuntos de Tarefas 1, 2 e 3………………………...45

Tabela 4.5 - Tabela de principais correlações..…………………………………….……....57

Tabela 4.6 - Mapeamento de eventos…..………………………………………………….65

LISTA DE REDUÇÕES

MSR	Model Specific Register
PMU	Performance Monitoring Unit
PMC	Performance Monitoring Counter
EPOS	Embedded Parallel Operating System
LISHA	Laboratório de Integração de Software e Hardware

RESUMO

Com a crescente no uso de sistemas embarcados e dispositivos móveis alimentados a bateria e o aumento do poder computacional e das necessidades dos mesmos, têm-se feito cada vez mais necessário o estudo de técnicas para melhor utilização dos recursos energéticos de modo a maximizar o tempo de funcionamento de um sistema. Uma das alternativas para uma melhor exploração da fonte energética é a aplicação de Power-Cap, limitando o consumo energético de maneira a não extrapolar o nível desejado. O uso de Power-Cap em sistemas de Tempo-Real, no entanto, deve levar em conta a existência de tarefas críticas que não devem perder seus tempos limite para execução (deadlines). Para se obter dados que possam atestar a necessidade de tomada de ação para que o consumo não extrapole o limite estabelecido, uma das alternativas é a aplicação de monitoramento de performance, outra área que vem ganhando grande importância nos dias de hoje. Tendo em vista este cenário, o presente Trabalho de Conclusão de Curso implementou um sistema, não intrusivo, de monitoramento de performance e estudou eventos de Software e Hardware coletados, desenvolvendo uma heurística de Power-Cap para um sistema Multi-Core de Tempo-Real.

Palavras chave: Multicore, Energy-Aware, DVS, PMU, Monitoramento, Tempo-Real, Power-Cap.

1 INTRODUÇÃO

Nos dias de hoje, os dispositivos móveis e sistemas embarcados, que, em sua maioria, operam tendo como fonte de alimentação uma bateria, representam a maior parte dos sistemas computacionais desenvolvidos. Isto traz à tona, ainda mais, a importância da aplicação de Power-Cap (Tampa de Energia ou, em outras palavras, um limite máximo para consumo energético).

Existem basicamente duas maneiras de se aplicar Power-Capping, a primeira delas busca uma abordagem a nível de Software e a segunda delas a nível de Hardware. Segundo Zhang e Hoffmann (2016), as abordagens de Software apresentam flexibilidade, permitindo a coordenação de múltiplos recursos de hardware, porém apresentam lentidão para alcançar o objetivo, requerendo muito tempo para convergir para o Power-Cap. Abordagens de Hardware, por sua vez, tendem a convergir muito rapidamente, mas controlam apenas voltagem e frequência de operação, limitando assim a performance num todo.

Dentre as opções para Power-Capping a nível de Software, destacam-se o controle da frequência de clock (modulação de clock) através de DVS e o desenvolvimento de estratégias de escalonamento focadas no consumo energético (power-aware).

Seguindo a ideia de ZHANG, LANG, PAKIN e FU (2014), o desenvolvimento de escalonadores de tarefas paralelas focados em consumo energético tem sido reconhecido como uma demanda para a computação de alto desempenho (high performance computing - HPC).

Ainda neste cenário, o uso de técnicas de DVS tende a aumentar a duração e a vida útil de suas baterias, uma vez que, conforme frizado por Islam e Lin (2017), dentre os vários componentes num dispositivo computacional, o processador é um dos maiores consumidores de energia, sendo sua performance diretamente relacionada com sua dissipação energética, e consome aproximadamente de 18% a 30% de toda energia consumida pelo dispositivo.

Por outro lado, o monitoramento de performance tem tomado um importante papel no que se refere, principalmente, a predição de eventos, permitindo operar o mecanismo de DVS no melhor momento. Mück, Sarma e Dutt (2015), ao descreverem o modelo "*Run-DMC: Runtime Dynamic Heterogeneous Multicore Performance and Power Estimation for Energy Efficiency*", previam uma etapa de coleta de dados (por eles chamada de "*sensing*"), na qual eram realizadas leituras de contadores de performance de Hardware (HPCs - Hardware Performance Counter).

No modelo proposto, bem como no presente trabalho, as coletas de dados eram realizadas na ordem de trocas de contexto, de modo a ser possível manter as métricas de performance separadas por threads.

Tendo em mente o desenvolvimento já feito, o foco do presente trabalho é se utilizar de um sistema operacional embarcado de baixa interferência (EPOS) para coletar dados de execução provenientes da Unidade de Monitoramento de Performance da Intel (Performance Monitoring Unit - PMU), tornando possível o correlacionamento dos diferentes contadores de performance e a busca de novas variáveis de interesse para o desenvolvimento de Heurísticas de Power-Cap para sistemas Multi-Core de Tempo-Real.

1.1. OBJETIVOS

Tendo em vista o foco determinado para o trabalho, foram estabelecidos os seguintes objetivos gerais e específicos

1.1.1. OBJETIVOS GERAIS

Utilizando o sistema operacional EPOS, desenvolvido pelo LISHA, funcionando em uma arquitetura Intel IA-32 em um servidor fornecido pelo laboratório, em modo Bare Metal, coletar dados provindo de contadores de performance durante a execução do sistema, a fim de aplicar algoritmos de Data Mining para obtenção de correlações entre os dados que descrevam situações durante a execução e, a partir da análise feita, desenvolver uma heurística de Power-Cap para sistemas Multi-Core de Tempo-Real.

1.1.2. OBJETIVOS ESPECÍFICOS

- Desenvolver, em conjunto com o discente José Luis Conradi Hoffmann, de matrícula 15100745, um sistema não intrusivo de captura de dados de monitoramento de performance em tempo de execução e um sistema de envio para o banco de dados inteligentes de IoT Lisha (iot.lisha.ufsc.br);
- Utilizar táticas de Data Mining para validar os dados capturados e encontrar uma correlação entre os mesmos;
- Desenvolver uma heurística de Power-Cap utilizando as variáveis correlacionadas.

1.2. ORGANIZAÇÃO DO TEXTO

As seções a seguir separam o conteúdo do presente trabalho em:

- Fundamentação Teórica: onde são apresentados os principais conceitos necessários para o entendimento do trabalho;
- Implementação: contém os detalhes do desenvolvimento do sistema de monitoramento de performance não intrusivo;
- Captura e Análise de Dados: descreve o processo de coleta, separação e análise dos dados, além das técnicas utilizadas;
- Desenvolvimento da Heurística: descreve os processos que levaram da análise realizada ao código da heurística;
- Resultados: Relata os aspectos referentes ao desempenho da heurística;
- Conclusão do Trabalho;
- Referências Bibliográficas.

2 FUNDAMENTAÇÃO TEÓRICA

A presente seção tem como objetivo descrever os conceitos e conhecimentos utilizados para o desenvolvimento deste trabalho, tais como o sistema EPOS, a arquitetura IA-32, o conceito de sistemas não intrusivos, alguns aspectos importantes da computação de Tempo-Real, os conceitos de Power-Cap e DVS e uma introdução a PMU.

2.1. EPOS

EPOS [1], ou Sistema Operacional Paralelo Embarcado (*Embedded Parallel Operating System*), é um sistema operacional desenvolvido no Laboratório de Integração Software/Hardware - LISHA [2].

O EPOS conta com o Método de Design de Sistemas Embarcados Dirigido à Aplicação (ADESD), proposto pelo professor Antônio Augusto Fröhlich, para guiar o desenvolvimento de componentes de software e hardware, que podem ser adaptados automaticamente para atender aos requisitos de aplicações específicas [3].

O sistema operacional EPOS pode ser configurado para executar tanto em um processador Intel, quanto em um processador ARM. Neste trabalho estaremos focados em sua versão para Intel IA-32, executando os testes em modo bare-metal, a qual presta suporte aos recursos, aqui necessários.

2.2. INTEL IA-32

A arquitetura Intel IA-32, também conhecida como i386, é a versão 32 bits do conjunto de instruções x86 (x86 instruction-set architecture), implementada pela primeira

vez nos microprocessadores Intel 80386 em 1985, sendo a primeira implementação de x86 a suportar 32-bits.

Nesta arquitetura, eram disponibilizados modo de operação protegido, modelo de memória segmentada, paginação, suporte para estágios paralelos e um barramento com suporte para 4-Gbytes de memória física.

A Intel, por questões de compatibilidade, manteve intacto o conjunto de instruções, sendo assim, os novos processadores continuam tendo suporte para as instruções utilizadas, apesar do ganho de desempenho.

2.3. COMPUTAÇÃO MULTI-CORE

Existem duas abordagens, básicas, de computação Multi-Core, a primeira é o paralelismo físico, também conhecido como chip multi-processador (CPM) onde o paralelismo é explorado em hardware, no caso, se tem mais de um Core físico dentro de uma mesma unidade de processamento (CPU). Neste tipo de paralelização, cada Core é um processador completo, e as diferenças quanto a um processador single Core são vistas, por exemplo, no caso das abordagens quanto a Cache.

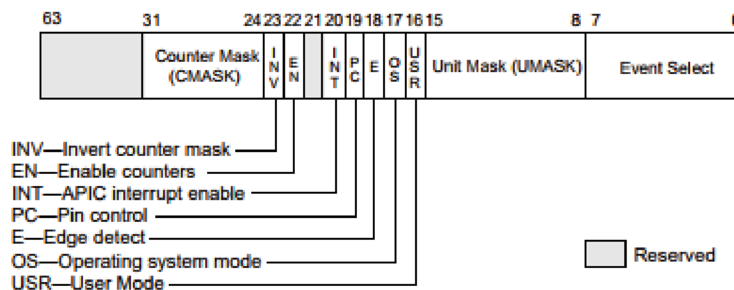
A segunda opção é o paralelismo explorado logicamente, esta abordagem leva o nome de hyperthreading, introduzido pela Intel no processador Pentium 4, que, de uma maneira simples, permite que a CPU manter o estado de duas diferentes threads e fazer switchs entre elas na escala de nano-segundos. Num exemplo, se um dos processos precisa fazer uma leitura de uma palavra da memória [13], uma CPU com multithread pode fazer a troca pela outra thread para ganhar em paralelismo.

Um aspecto importante é que o hyperthreading não traz um paralelismo real, uma vez que apenas uma thread está executando num instante de tempo. Por outro lado, para o sistema operacional, cada thread aparece como uma diferente CPU.

2.4. PMU

O monitoramento de performance foi introduzido pela Intel nos processadores Pentium, com um conjunto de contadores de monitoramento de performance específicos (model-specific performance-monitoring counter – MSRs). Estes contadores permitiam a seleção de parâmetros de performance dos processadores para serem monitorados e medidos [4].

Figura 2.1 – Exemplo de *Layout* de um MSR.



Fonte: Intel® 64 and IA-32 Architectures Software Developer's Manual (2018)

As novas gerações de processadores da Intel suportam uma gama maior de eventos de performance arquiteturais e não-arquiteturais.

- Eventos de performance arquiteturais: esta classe suporta amostragem de eventos por contagem e interrupção (dado um valor, cada vez que o contador alcança este, uma interrupção é disparada). O comportamento

visível desta classe de eventos é consistente entre as diferentes implementações de processadores;

- Eventos de performance não-arquiteturais: também suporta amostragem de eventos por contagem e interrupção, porém, os eventos desta classe variam de um modelo de processador para outro. São específicos da microarquitetura.

2.4.1. PMU NA ARQUITETURA SANDY BRIDGE

A versão arquitetural da PMU utilizada durante o desenvolvimento e os testes (vide tópico 2.10) é a Sandy Bridge (suportada, também, pelo EPOS). Nesta versão, são suportados 3 canais fixos (*Instructions Retired, CPU Clock Unhalted e CPU Clock Unhalted Ref-TSC* [4]) 8 configuráveis por core (4 por thread) , possuindo os eventos típicos da versão arquitetural 3, somados aos eventos não-arquiteturais disponíveis apenas para a versão Sandy Bridge.

Além das configurações básicas, a PMU em sua versão Sandy Bridge conta com os recursos PEBS, PDIR, PMI e contagem Any-Thread. Estes recursos não serão explorados no presente trabalho.

A descrição da versão arquitetural 3 da PMU pode ser encontrada na seção 18.2.3 do manual de desenvolvedores da Intel [4] e a descrição dos eventos não-arquiteturais da microarquitetura Sandy Bridge é acessível na seção 19.6 do mesmo manual.

2.5. SISTEMAS DE TEMPO-REAL

Sistemas de Tempo-Real são sistemas utilizados em sistemas de controle críticos, que dependem de resposta em tempo hábil e dos resultados computados para funcionar

corretamente. Estes sistemas diferem dos demais pelo fato de reagirem a eventos do mundo físico dentro de um certo período de tempo [9].

Em um sistema de Tempo-Real, as tarefas (tasks) possuem algumas propriedades que ajudam a sua compreensão e execução [10]:

- Tempo de Liberação (Release Time): Tempo requerido para que qualquer thread esteja pronta;
- Deadline: Tempo no qual cada tarefa deve ser completada depois da sua liberação;
- WCET (Worst Case Execution Time): Maior tempo necessário para concluir uma tarefa com sucesso sob parâmetros críticos e injustos do sistema;
- Tempo de execução (Run Time): Tempo necessário, sem interrupções, para completar uma tarefa depois de sua liberação.

2.5.1. TIPOS DE SISTEMAS DE TEMPO-REAL

A perda de uma deadline não é desejável em um sistema de Tempo-Real. Mas existem diferentes tipos de sistemas de Tempo-Real e, em cada um deles as perdas de deadlines tem diferentes consequências:

- Sistema de Tempo-Real Crítico (Hard Real-Time System): Neste sistema, as deadlines (limites temporais) são estáticas e não mudam em nenhuma circunstância. A saída deste sistema é zero se a execução não for completada com sucesso dentro do tempo limite [10] e o atraso não é aceito. Muitas vezes este tipo de sistema lida com a vida humana e atrasos podem acarretar em catástrofes;

- Sistema de Tempo-Real Não-Crítico (Soft Real-Time System): Sistema no qual as deadlines são dinâmicas e podem ser flexíveis, assim as tarefas podem ser completadas com sucesso [10].

2.5.2. ESCALONADORES DE TEMPO REAL

Para sistemas de Tempo-Real simples, a resposta é rápida o suficiente. Contudo, para muitos sistemas de Tempo-Real, mais complexos, é necessário uma coordenação sofisticada, e obter respostas rápidas o bastante torna-se um desafio [9].

Existem dois tipos básicos de escalonadores de Tempo-Real, os escalonadores de Prioridade Estática, nos quais as prioridades das tarefas variam durante a execução do sistema, e os de Prioridade Dinâmica, cujas prioridades são constantes.

2.5.2.1 RM - RATE MONOTONIC

Escalonador de prioridade fixa. Segue a filosofia de que as prioridades mais altas são atribuídas as tarefas de maior frequência. O escalonador sempre escolhe para execução a tarefa de maior prioridade. Conforme cresce o número de tarefas a escalonar, a escalabilidade deste algoritmo tende a 69% [9], isto é, a maior porcentagem de uso sem perda deadline é de 69% a partir de um determinado número de tarefas.

Um teste possível para a escalabilidade deste algoritmo é checar a sentença lógica Un < Wn, onde:

- Un é a utilização e pode ser calculado como o somatório das taxas de uso de cada tarefa. Imaginando que cada tarefa i tenha período Ti e WCET Ci, Un = Somatório (i = 1.. n) { Ci / Ti };

- Wn é a escalabilidade máxima para o número n de tarefas, podendo ser calculado por: $n*(2^{1/n} – 1)$.

2.5.2.2 EDF – EARLIEST DEADLINE FIRST

Escalonador de prioridade dinâmica. Usa a abordagem de que as tarefas com a deadline mais próxima devem receber maior prioridade. Esta abordagem tende a zerar o tempo de idle do processador, podendo alcançar 100% de uso. Contudo, o EDF apresenta uma limitação pela não-possibilidade de se prever a tarefa que falhará durante um overload momentâneo [9].

Um teste simples de escalabilidade é:

- Somatório de (Ci / Ti) ≤ 1, tomando Ci como o WCET de uma tarefa i e Ti como o período desta tarefa.

2.6. POWER-CAP

Power-Cap ou tampa de energia consiste em determinar um limite máximo de consumo de energia para a execução de um sistema [21]. As opções para aplicação de Power-Cap (power-capping) usam abordagens de Software e Hardware.

As abordagens de Software apresentam flexibilidade, permitindo a coordenação de múltiplos recursos de hardware, porém apresentam lentidão para alcançar o objetivo, requerendo muito tempo para convergir para o Power-Cap. Abordagens de Hardware, por sua vez, tendem a convergir muito rapidamente, mas controlam apenas voltagem e frequência de operação, limitando assim a performance num todo [22].

2.6.1. POWER-CAPPING EM HARDWARE E INTERFACE RAPL

Conforme já citado, um dos caminhos para aplicação Power-Cap é em nível de Hardware. Esta configuração pode ser realizada, na arquitetura Intel IA32, através da interface RAPL.

Segundo o “Manual de Desenvolvedores de Software das Arquiteturas IA32 e Intel® 64” [4], fornecido pela Intel, em seu tópico 14.9.3 “Package RAPL Domain” (página 3189), a interface RAPL fornece o registrador MSR_PKG_POWER_LIMIT, que permite, ao software, definir uma limitação de consumo para o domínio do “pacote” (“package” é a região de Hardware onde estão agrupados componentes como o processador, a cache e a placa de vídeo integrada).

Ainda seguindo a definição presente no manual, este limite é dado na forma de consumo médio em Watts sobre uma janela de tempo definida no mesmo registrador. O recurso permite a configuração de dois limites a serem aplicados em duas janelas de tempo diferentes, conforme apresentado na Figura 2.2.

Figura 2.2 – *Layout* do registrador *MSR_PKG_POWER_LIMIT*,

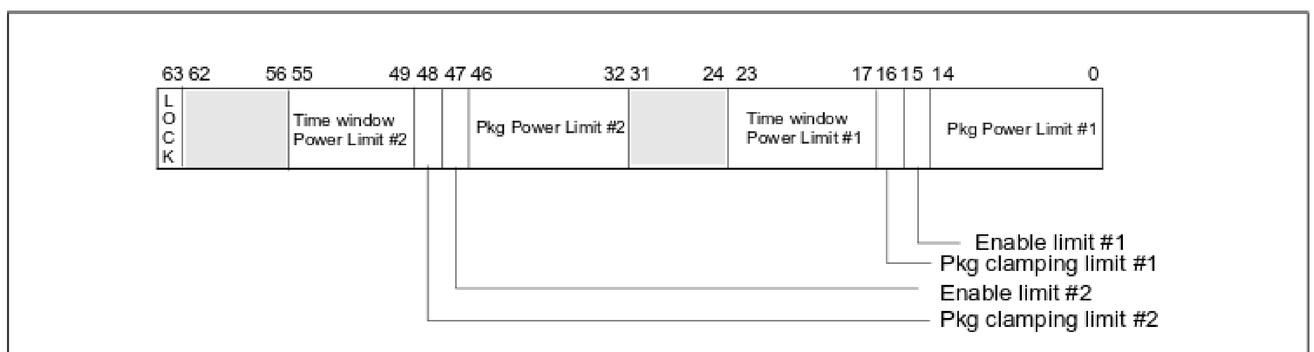


Figure 14-36. MSR_PKG_POWER_LIMIT Register

Fonte: Intel® 64 and IA-32 Architectures Software Developer’s Manual (2018)

Em alguns processadores, a interface permite configuração de limites específicos de memória RAM e para o processador, sendo possível, ainda, ler status de consumo específico para cada componente e do pacote como um todo, levando em consideração as unidades de medida utilizadas para o modelo.

2.7. DVS E MODULAÇÃO DE CLOCK

Dynamic Voltage and Frequency Scaling (DVS ou DVFS), ou Escalonamento Dinâmico de Voltagem e Frequência, é aceito como uma técnica para reduzir potência e consumo de energia de microprocessadores [11].

2.7.1. MODULAÇÃO DE CLOCK POR DUTY CYCLE

A Intel define modulação de clock como um segundo método de controle térmico disponível para os processadores. Esta modulação é efetuada desligando e ligando o clock rapidamente em duty cycle (ciclos "ocupados") e pode resultar em reduções de até 50% na dissipação de calor (aplicando duty cycle de 30-50%) [20, Thermal Guide. 6.2.2.2 - Clock Modulation].

Segundo a definição da Intel [4, seção "14.7 THERMAL MONITORING AND PROTECTION", página 3177], "duty cycle" não se refere ao ciclo de trabalho real do sinal de clock, e sim ao período de tempo durante o qual o sinal de clock pode acionar o chip do processador. Usando o mecanismo de parada do clock para controlar com que freqüência o processador é "cronometrado" (recebe os sinais do clock), o consumo de energia do processador pode ser modulado.

Ainda segundo a Intel, os processadores a partir do Pentium 4, Intel Xeon e Pentium M apresentam suporte para modulação de clock controlada por software, permitindo que o sistema operacional implemente uma política de redução de consumo de

energia dos processadores [4, seção “14.7.3 Software Controlled Clock Modulation”, páginas 3179 e 3180].

O controle de modulação de clock é exercido através de escritas no MSR IA32_CLOCK_MODULATION, conforme Figura 2.3 e a descrição abaixo da mesma.

Figura 2.3 – *Layout* do MSR IA32_CLOCK_MODULATION,

Figure 14-25. IA32_CLOCK_MODULATION MSR

Fonte: Intel® 64 and IA-32 Architectures Software Developer’s Manual (2018)

- *On-Demand Clock Modulation Enable*, bit 4 - Habilita o controle de software de modulação de clock por demanda quando em 1 e o desabilita quando 0.
- *On-Demand Clock Modulation Duty Cycle*, bits 1 through 3 - Seleciona o *duty cycle* da modulação de clock por demanda (ver Figura 2.4). Esse campo só é válido quando o bit 4 (descrito acima) esta habilitado.

Figura 2.4 – Descrição dos *Duty Cycles*

Table 14-3. On-Demand Clock Modulation Duty Cycle Field Encoding

Duty Cycle Field Encoding	Duty Cycle
000B	Reserved
001B	12.5% (Default)
010B	25.0%
011B	37.5%
100B	50.0%
101B	63.5%
110B	75%
111B	87.5%

Fonte: Intel® 64 and IA-32 Architectures Software Developer's Manual (2018)

É importante ressaltar que em sistemas multicore, cada core controla seu duty cycle, porém quando se usa a tecnologia de hyperthreading, o duty cycle do core físico será configurado para o valor mais alto entre as threads.

2.8. NÃO INTRUSIVIDADE

A não intrusividade na computação é tratada como a capacidade de um sistema, ou aplicação, de executar minimizando o seu impacto sobre o desempenho do resto do sistema. Segundo Harelick e Stoyen (1999), um sistema de monitoramento não intrusivo, deve respeitar o Princípio de Heisenberg, descrito por Rosenberg (1996) como a garantia de que o processo monitor não afete o comportamento do processo monitorado.

Ainda conforme Harelick e Stoyen (1999), um sistema de monitoramento em Tempo-Real deve cumprir com objetivos severos. A instrumentação não pode fazer com que as tarefas percam suas deadlines e não deve requerer recompilação, além de que, talvez, o monitor pode ter a necessidade de operar sobre os dados antes de imprimí-los.

2.9. DATA MINING

Data Mining, ou mineração de dados, é o nome dado a todo processo de se aplicar uma metodologia baseada na computação, incluindo novas técnicas, para extrair conhecimento de um conjunto de dados [15].

Ainda segundo Kantardzic [15], Data Mining é um processo interativo no qual cada progresso é definido por descobertas, por métodos manuais ou automáticos, sendo que o seu uso é mais útil num cenário de análise exploratória onde não se tem ideias premeditadas do que seria uma saída interessante.

São dois os objetivos primários da aplicação de Data Mining:

- Predição: com intuito de produzir um modelo do sistema descrito pelo conjunto de dados;
- Descrição: com intuito de produzir informação nova e não trivial sobre os dados.

Para alcançar tais objetivos usam-se métodos de mineração de dados para agrupamento (clusterização ou clustering), que consiste em separar registros em grupos, associação, que é utilizado para identificar atributos (ocorrências) que se relacionam com um determinado atributo (evento), e, ainda, classificação, que é utilizado para identificar a qual conjunto de categorias/classes pertence um determinado evento.

2.9.1. WEKA

Weka (Waikato Environment for Knowledge Analysis - Ambiente para Análise de Conhecimento da Waikato) é uma coleção de algoritmos de aprendizado de máquina (machine learning) para tarefas de mineração de dados, disponibilizado gratuitamente pela Waikato University, da Austrália.

Estão disponíveis, no Weka, ferramentas para preparação de dados, classificação, regressão, agrupamento (clustering), associação e visualização [25].

2.10. MATERIAIS UTILIZADOS

Para execução do presente trabalho foram utilizados os seguintes equipamentos:

- Computador equipado com um processador Intel Core i7-2600 @3.4GHz [19], memória RAM de 4GB DDR3 @1333MHz, placa mãe PCWARE IPMH61R3, com fonte de alimentação ALLIED SL-8180 BTX instalado sob corrente de 220V, disponibilizado pelo laboratório (LISHA) para captura de dados e testes das heurísticas.
- Analisador de potência e qualidade de energia Fluke 435 series II [24], para aferições de consumo energético.

O computador utilizado pertence à segunda geração dos processadores Intel Core i7, com arquitetura Sandy Bridge.

3 IMPLEMENTAÇÃO

Esta seção tem como objetivo expor os sistemas desenvolvidos durante a execução do projeto, envolvendo toda a parte de captura dos dados, armazenamento, impressão, transformação/formatação, envio e busca. Esta etapa do trabalho foi desenvolvida em conjunto com o discente José Luis Conradi Hoffmann, de matrícula 15100745, também aluno deste mesmo curso.

Uma descrição mais detalhada da implementação, contendo parte do estudo feito e algumas conclusões que levaram a mudanças e evoluções que trouxeram o sistema até o estado atual, pode ser encontrada em *“Performance Monitoring with EPOS”* [7].

3.1. LEITURA DE TEMPERATURA

De acordo com o manual de desenvolvedores da Intel [4], na arquitetura IA-32, a temperatura de um processador pode ser lida usando o MSR IA32_THERM_STATUS. No tópico 14.7.5.2 “Reading the Digital Sensor” (2018, vol 3B, capítulo 14, p. 2697), foi definido o seguinte método para leitura do valor da temperatura:

Primeiramente, é preciso lembrar que, diferentemente de um leitor analógico de temperatura, a saída do sensor térmico digital é uma temperatura relativa a máxima temperatura permitida para operação do processador:

- Digital Readout (bits 22:16, RO) - Leitura da temperatura digital em 1 grau Celsius relativos ao temperatura de ativação do TCC (Thermal Control Circuitry - circuito de controle térmico).
  - 0: Temperatura de ativação do TCC;
  - 1: (Ativação do TCC - 1), etc.

- Neste sentido, a leitura de um valor baixo do campo Digital Readout (bits 22:16) indica uma temperatura, que na verdade, é alta.

3.2. MODULAÇÃO DE CLOCK

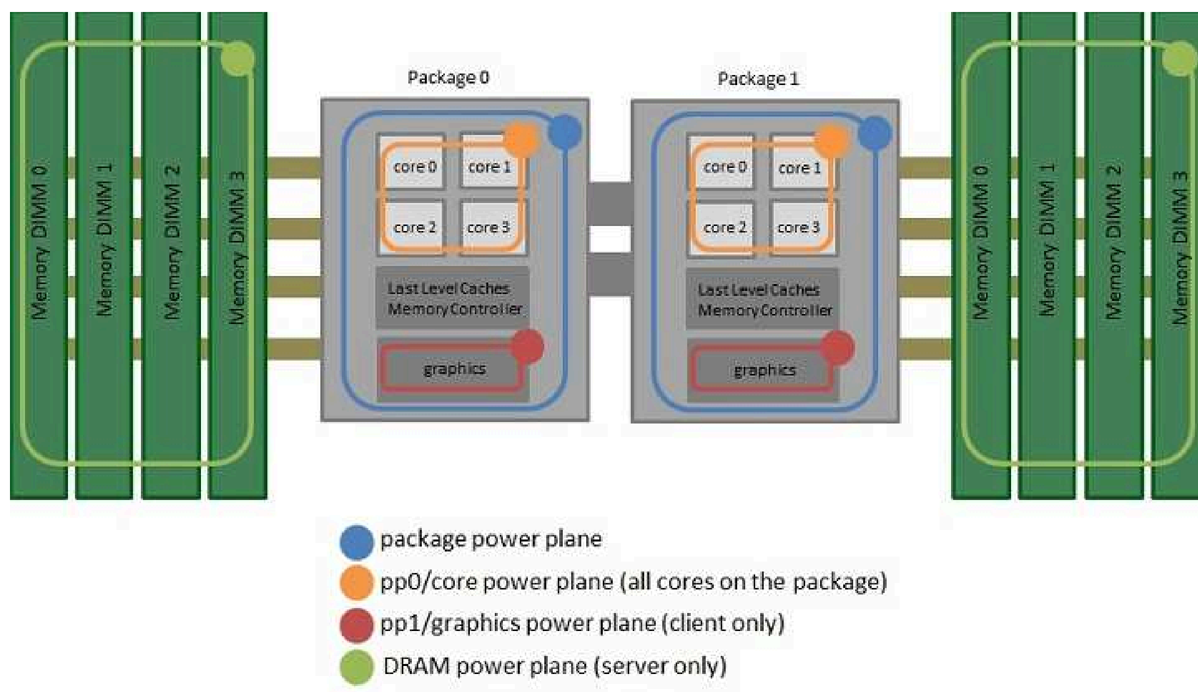
O modelo de modulação de clock implementado segue a descrição do tópico 2.7.1, se utilizando da técnica de duty cycle. O método foi implementado no EPOS especificamente para cpus Intel IA32, sendo que foi optado por utilizar o modelo geral, com variações de 12.5%, não se valendo da extensão para controle com granularidade de 6,25% por questões de garantia de compatibilidade com outros modelos de processadores.

Um aspecto importante a se ressaltar é a thread (tarefa) que executa o método de modulação de clock aplica o duty cycle na cpu sobre a qual está executando.

3.3. MEDIÇÃO DE CONSUMO DE ENERGIA DO PROCESSADOR

Para além da possibilidade de se aferir consumo com uso de ferramentas externas, é possível, ainda, nos processadores Intel, utilizar os registradores da interface RAPL [4], para efetuar “medições” do consumo de forma mais específica. A interface provê suporte para leitura do consumo do pacote (*PKG*), e dos componentes internos ao mesmo (Figura 3.1) e, também, o consumo energético da DRAM.

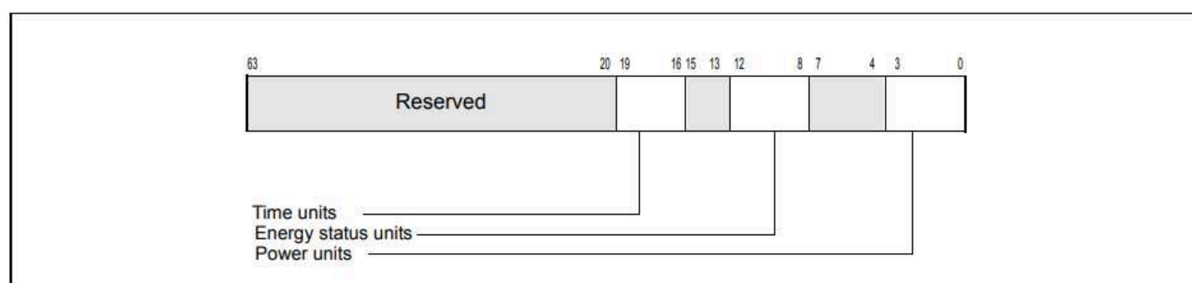
Figura 3.1 – Divisão de componentes utilizada pela Interface RAPL.



Fonte: Intel® Power Governor (2012) [23].

O suporte as medidas, no entanto, varia de acordo com a versão arquitetural do processador. A versão arquitetural Sandy Bridge (utilizada neste trabalho), por exemplo, possui suporte apenas para informações de consumo no nível de PKG, PP0 e PP1 (http://web.eece.maine.edu/~vweaver/projects/rapl/rapl_support.html). Vale ressaltar que o EPOS não inicializa o componente gráfico integrado, não tendo, portanto, suporte ao PP1.

Figura 3.2 – *Layout* do registrador MSR_RAPL_POWER_UNIT.



Fonte: Intel® 64 and IA-32 Architectures Software Developer's Manual (2018)

Figura 3.3 – *Layout* do MSR para leitura dos dados energéticos do Package.

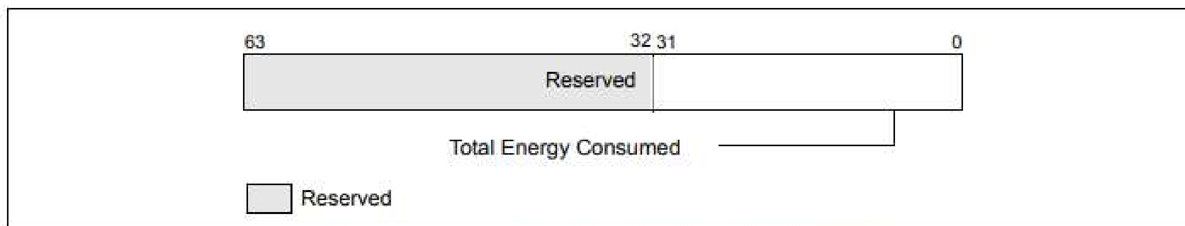


Figure 14-33. MSR_PKG_ENERGY_STATUS MSR

Fonte: Intel® 64 and IA-32 Architectures Software Developer’s Manual (2018)

Figura 3.4 – *Layout* dos MSRs de leitura dos componentes PP0 e PP1.

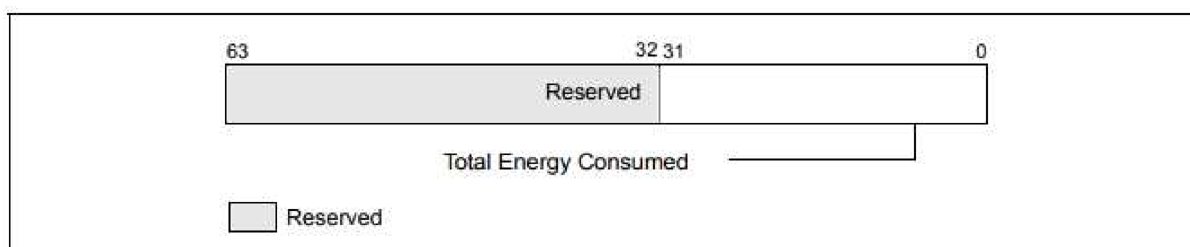


Figure 14-37. MSR_PP0_ENERGY_STATUS/MSR_PP1_ENERGY_STATUS MSR

Fonte: Intel® 64 and IA-32 Architectures Software Developer’s Manual (2018)

Conforme o manual da Intel, o procedimento para aferir consumo energético do Pacote (“Package” ou “PKG”), ou de um de seus componentes, consiste em ler, do registrador referente ao componente a ser aferido, (Figuras 3.3 e 3.4) os 32 bits menos significativos e multiplicar pela unidade de contagem (u), valor calculado a partir dos bits 8 a 12 do registrador MSR_RAPL_POWER_UNIT (Figura 3.2), pela fórmula: $u = 1/2^{ENERGY_STATUS_UNIT}$. O resultado deste cálculo será o valor energético em Joules. Para obter o consumo em Watts, basta dividir o valor encontrado na operação pelo tempo de execução em segundos (Watts = Joules/s).

Uma importante observação é que o valor lido pelos registradores de contagem são referentes ao período desde a última vez que a contagem foi reiniciada, este momento é chamado pela Intel de wraparound (14.9.3 Package RAPL Domain - [4]). A Intel prevê um tempo de 60s para o wraparound quando a carga de trabalho (workload) é alta, mas o tempo pode aumentar conforme a carga diminui. Ainda segundo o manual, as atualizações do valor do registrador são esperadas serem feitas a cada 1 milissegundo.

No EPOS foram implementados apenas os códigos de leitura dos registradores e separação dos valores para unidade e consumo. Os valores de consumo podem ser calculados após a execução, através do log que imprime as capturas.

3.4. SISTEMA NÃO INTRUSIVO DE CAPTURAS

Uma captura tem como objetivo gravar informações do sistema em um determinado momento. Tais informações podem vir a ser utilizadas para uma análise detalhada de uma execução e do comportamento do Software e do Hardware.

Ao efetuar uma captura, o “Momento” atual do sistema é armazenado. Este momento é composto pelo *timestamp* da leitura, a temperatura, os três canais fixos da PMU, os quatro configuráveis, o id, a prioridade e o número de *deadline misses* da thread (que está executando ou passará a executar), o número de *deadline misses* em todo o sistema, o valor do contador de energia do package (pacote onde se encontra os cores e a cache) e contador de energia dos cores.

Para lidar com tais variáveis, foi escolhida uma *struct* em C++ (Figura 3.5), composta por 15 variáveis, que por sua vez, descrevem o momento atual do sistema operacional e do Hardware, de acordo com os critérios configurados na inicialização (canais da PMU monitorados).

Figura 3.5 – Estrutura de um Moment

```
struct Moment {
   unsigned long long _temperature;
   unsigned long long _pmu0;
   unsigned long long _pmu1;
   unsigned long long _pmu2;
   unsigned long long _pmu3;
   unsigned long long _pmu4;
   unsigned long long _pmu5;
   unsigned long long _pmu6;
   unsigned long long _time_stamp;
   unsigned long long _thread_id;
   long long _thread_priority;
   unsigned long long _local_deadline_miss;
   unsigned long long _global_deadline_miss;
   unsigned long long _pkg_energy;
   unsigned long long _pp0_energy;
};
```

As capturas ocorrem sempre que é executado o *dispatch* de uma thread, ou seja, quando uma troca de contexto acontece, ou quando há uma solicitação de execução feita por uma tarefa periódica.

É importante ressaltar que as leituras feitas dos registradores da PMU e de temperatura estão relacionados diretamente ao *Core* que executa o código.

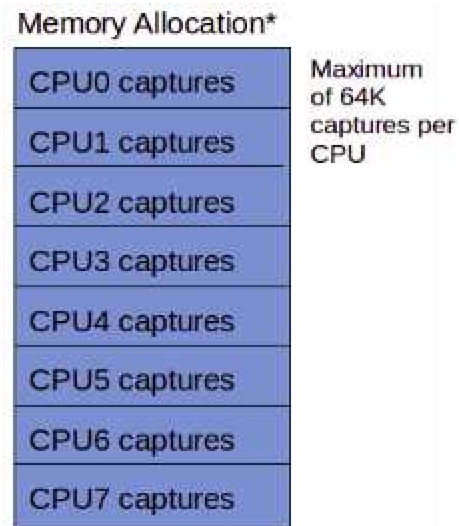
3.4.1. ESTRUTURA DE ARMAZENAMENTO

Dentre as opções disponíveis para armazenar as capturas (buffer consumido em determinados momentos - idle, por exemplo - ou armazenamento total dos dados), a escolha foi por armazenar todas as capturas de uma execução em memória, sem o consumo desses dados até que o sistema finalize sua execução.

A região alocada para o armazenamento foi separada em um vetor de vários elementos da *struct Moment* (ver Figura 3.5). O vetor em questão divide igualmente o

espaço alocado entre todas as CPU's, onde cada uma escreve na sua respectiva "faixa", essa divisão vai da CPU 0 até a CPU 7 (Figura 3.6). Esta divisão ocorre, desta forma, devido ao fato de a máquina disponibilizada para executar os testes ter 8 *Cores* lógicos.

Figura 3.6 – Estrutura de Armazenamento



Cada *Moment* ocupa 120 bytes (15 * o tamanho de um long long na arquitetura IA32 = 15 * 8), sendo assim, o vetor deve ter no mínimo 8 * 120 = 960 bytes (um *moment* por CPU). Após alguns testes, encontrou-se um valor máximo de 64000 capturas por CPU em testes com elevado número de troca de contexto e com período de execução de dois minutos.

Sendo assim, o vetor foi configurado para o tamanho de aproximadamente 61 MB (61440000 B = 61.440000 MB), o que, por sua vez, exigiu um aumento no tamanho padrão da *Heap* do EPOS (configurada no traits da aplicação, arquivo onde são configurados aspectos do programa a ser compilado usando técnica de metaprogramação estática).

3.4.2. NÃO INTRUSIVIDADE

Um dos principais requisitos do sistema de capturas, é que ele não gere interferências no desempenho do sistema como um todo, ou seja, o tempo final da execução deve ser, idealmente, o mesmo do que quando o sistema operacional executa sem o sistema de capturas habilitado. Todavia, manter exatamente o mesmo tempo de execução é impossível, uma vez que, naturalmente, foram adicionadas instruções ao código e, portanto, existe uma pequena tolerância (na grandeza de microsegundos).

Com o objetivo de dar ao sistema uma característica de não intrusão, é necessário uma boa escolha quanto a onde, e quando, as capturas e a inicialização dos componentes ligados a elas devem ocorrer. Com este objetivo, foi realizado um estudo (leitura da documentação e do código) sobre a inicialização e funcionamento do sistema EPOS.

A partir dos resultados dos estudos, foram delimitados locais específicos para realizar estas operações, garantindo um impacto baixíssimo no desempenho do sistema. Sendo assim, o sistema de capturas é inicializado juntamente ao SO, mais especificamente ao inicializar as primeiras threads (idle), evitando qualquer impacto sobre a execução das aplicações. Sobre a questão de quando capturar, foi selecionado o instante da troca de contexto, a fim de evitar uma nova interrupção no sistema.

Outro ponto importante era armazenar as capturas de forma rápida, dado o fato de que as execuções podem gerar um grande volume de dados, deixando inviável realizar a impressão dos dados na tela ou enviá-los via porta serial durante a execução (algo que poderia ocorrer se o buffer alocado fosse menor do que o número de dados alocados, o que geraria a necessidade de esvaziá-lo durante a execução), pois mesmo isso sendo

realizado em um período de *idle* do sistema, dependendo da utilização do task-set sendo executado, pode não haver tempo suficiente para o consumo dos dados sem que ocorra perda.

O paralelismo de escrita também foi focado na implementação, pois se o recurso utilizado não for cercado por uma tranca, as impressões se misturam quando duas ou mais CPUs estão realizando esta tarefa. Logo, deve ser alocado um espaço em memória com capacidade para armazenar todas as capturas durante a execução.

3.4.2.1 VERIFICAÇÃO DA NÃO INTRUSIVIDADE

Para avaliar a não intrusividade do sistema, foi realizado uma comparação de uma execução com e sem o sistema de monitoramento de performance, sendo os tempos analisados medidos ao início e fim da execução do código da aplicação que coordena as tarefas executadas. A verificação apresentou um aumento de aproximadamente 0.000002% em relação a execução com o sistema desabilitado (segundo taskset apresentado na Tabela 3.1, com 15 tarefas projetadas para o escalonador C-EDF). Seguem os cálculos efetuados:

- Tempo de execução sem o sistema de Monitoramento = 122122579 us;
- Tempo de execução com o sistema de Monitoramento = 122122803 us;
- Diferença = 224 us = 0.000224 seg = 0.000001834%.

Tabela 3.1 – Conjunto de Tarefas Utilizado para os testes de não intrusividade.

Período (us)	Deadline (us)	WCET (us)	CPU (Cluster)
4000	4000	435	0
14000	14000	1084	0
20000	20000	1094	0

5000	5000	84	0
23000	23000	2377	1
31000	31000	2513	1
25000	25000	1849	1
25000	25000	2547	2
22000	22000	1838	2
20000	20000	1277	2
30000	30000	440	2
15000	15000	1425	3
6000	6000	556	3
22000	22000	1018	3
23000	23000	810	3

3.4.3. ESCALONADORES PARA MONITORAMENTO

Como já citado, as inicializações dos componentes ligados ao sistema de captura foram feitas juntamente a inicialização do sistema. Para concretizar a inicialização da PMU, portanto, foi adicionada uma chamada de método na inicialização das primeiras threads do sistema.

Tal método foi implementado, originalmente vazio, dentro dos escalonadores, criando uma pequena modificação na estrutura já existente. Assim, para habilitar um escalonador ao monitoramento da PMU, basta preencher este método com os comandos de inicialização e configuração. Além disso, um dos objetivos da implementação era que o monitoramento pudesse ser facilmente ativado ou desativado. Para isso, optou-se por

adicionar uma variável de checagem a ser alterada dentro do método citado, ao qual foi atribuído o nome de init.

Para diferenciar os escalonadores nos quais o método init contém a configuração e inicialização da PMU e a atribuição de valor verdade para variável condiciona, os que foram configurados segundo esse princípio foram renomeados para M<nome do escalonador>.

Todo o código de monitoramento usa estruturas de checagem e, portanto, desativá-lo significa apenas mudar o escalonador no traits antes da compilação, ou simplesmente alterar o valor verdade da variável que libera o monitoramento.

O procedimento citado acima, não altera o funcionamento dos escalonadores, e poderia ser sido replicado em qualquer outro escalonador, a separação foi feita apenas com o intuito de ressaltar as diferenças da configuração padrão do sistema para a configuração para o processo do sistema de monitoramento de performance.

3.4.4. CONFIGURAÇÃO DOS EVENTOS PMU

A partir das configurações citadas no tópico acima, quando um critério de escalonamento que habilita o monitoramento é inicializado (configurados até agora são MGEDF, MCEDF e MPEDF), ele habilita a flag de monitoramento do sistema, além de configurar e inicializar os canais de monitoramento da PMU.

Para selecionar quais canais configuráveis serão monitorados durante a presente execução, uma variável estática foi definida na própria classe PMU do EPOS, com a função de definir a posição do evento escolhido no *enum* presente na classe.

Visando facilitar a configuração do sistema para execuções de testes em batch (lote), foi adotado como padrão que a variável definida na classe PMU representa apenas a posição do primeiro evento. Como consequência disso, ao configurar a PMU na

inicialização do escalonador, se inicializam os eventos configuráveis a partir da posição descrita pela variável da classe PMU e as 3 posições seguintes no vetor.

3.4.5. ENVIO PARA O BANCO

Como já descrito, um dos objetivos desta implementação, era que os dados capturados fossem enviados para plataforma IoT da UFSC. Para tanto foram analisadas as seguintes possibilidades:

- Envio em tempo real:
  - envio do dado assim que capturado;
  - bufferização dos dados para envio em intervalos;
- Envio após execução;

Com a análise das opções e dos impactos de cada uma, e buscando manter a ideia de um sistema de monitoramento com o mínimo possível de jitter/interferência na execução (não intrusividade), foi escolhida a estratégia de envio após a execução.

Partindo da escolha do método de envio, foi implementado um conjunto de aplicações para administrar o funcionamento do EPOS, além de algumas funções para impressão de dados dentro do próprio SO.

3.4.5.1 METADADOS DE ENVIO

O envio dos dados para IoT segue uma estrutura definida por series e smartdatas, na qual as series representam a série temporal sob a qual os dados estarão associados, e as smartdatas representam os dados em si, juntamente com alguns metadados inerentes a eles, que ajudam na identificação da series a qual está associado [6].

A partir disto, são necessários alguns dados do sistema, além dos capturados, para executar o envio a plataforma IoT UFSC [5], são eles:

- Id’s das threads configuradas no teste;
- Series a serem criadas (são geradas antes da execução pois contém os timestamps inicial e final);
- Lista de units (unit é um dos parâmetro das series) utilizadas (varia conforme configuração dos eventos não estáticos);
- Timestamps iniciais e finais.

Destes, os únicos que dependem da execução das tasks e, portanto, são impressos após a execução, são os id’s das threads.

3.4.5.2 SISTEMA EXTERNO

Visando enviar os dados capturados, é necessário, primeiramente, um algoritmo para monitorar a execução do EPOS e salvar o conteúdo impresso em um arquivo, doravante chamado de log, através da leitura constante de uma porta serial durante a execução.

Após o término da execução é preciso, então, de um algoritmo capaz de ler o log e organizar os dados e metadados de maneira a poder realizar o envio das series e das smartdatas.

Por fim para o envio das series e smartdatas são necessários scripts configurados conforme a plataforma.

3.4.5.3 SISTEMA INTERNO

Para possibilitar o envio dos dados, capturados durante a execução, para IoT, foram necessárias as implementações de funções para impressão dos dados, em si, e de alguns metadados para facilitar o processo de envio.

Tendo em mente as questões ligadas ao funcionamento e ao desempenho do sistema, o melhor lugar para realizar a impressão dos metadados e dados relacionados a execução era após a conclusão da mesma, enquanto os dados e metadados independentes da execução são impressos antes do início da execução das tasks. Logo, o único impacto durante a execução é o do processo de captura.

A partir disso, o sistema começa esta etapa de impressão logo após perceber o fim da execução da última thread que, no EPOS, acontece na primeira verificação feita ao entrar em idle, antes do comando que ordena o desligamento do computador.

4 CAPTURA E ANÁLISE DOS DADOS

A partir da implementação realizada, foram iniciados os processos de captura de dados, onde foi testados *task-sets* (conjuntos de tarefas) com diferentes operações explorando operações com inteiros e pontos flutuantes, de modo recursivo e iterativo, e leituras consecutivas de vetores aleatórios em memória) em busca de variáveis que pudessem embasar a escolha de eventos para o desenvolvimento da heurística.

4.1. DEFINIÇÃO DOS TASK-SETS E DAS OPERAÇÕES EXECUTADAS

A geração dos conjuntos de tarefas a serem utilizados para captura dos dados foi feita com o auxílio do conceito de particionamento de tarefas para escalonadores Multi-Core de tempo real descrito por Gracioli em Real-Time Operating System Support for Multicore Applications (2014) [27].

Os conjuntos de tarefas gerados consideraram o escalonador P-EDF, e para a continuação dos trabalhos foram utilizados três grupos de tarefas, descritas nas tabelas 4.1, 4.2 e 4.3. O conjunto de tarefas 1, por ter maior porcentagem de uso, foi utilizada para geração dos dados capturados, enquanto os conjuntos 2 e 3, com menor uso, foram utilizadas para validação da heurística.

Tabela 4.1 - Conjunto de tarefas 1.

Período (us)	Deadline (us)	WCET (us)	CPU
50000	50000	27098	0
25000	25000	5504	1
100000	100000	68919	2
100000	100000	64664	3
50000	50000	9310	4
200000	200000	105758	5

200000	200000	29326	6
200000	200000	67222	7
50000	50000	21151	6
50000	50000	6757	4
50000	50000	34329	1
50000	50000	8203	4
100000	100000	44566	7
25000	25000	8853	4

Tabela 4.2 - Conjunto de tarefas 2.

Período (us)	Deadline (us)	WCET (us)	CPU
50000	50000	24388	0
25000	25000	4954	1
100000	100000	62027	2
100000	100000	58198	3
50000	50000	8379	4
200000	200000	95182	5
200000	200000	26393	6
200000	200000	60500	7
50000	50000	19036	6
50000	50000	6081	4
50000	50000	30896	1
50000	50000	7383	4
100000	100000	40109	7
25000	25000	7968	4

Tabela 4.3 - Conjunto de tarefas 3.

Período (us)	Deadline (us)	WCET (us)	CPU
25000	25000	13958	0
100000	100000	15135	1
200000	200000	136986	2
50000	50000	25923	3
25000	25000	11637	4
100000	100000	20072	5
50000	50000	30484	6
200000	200000	25220	7
200000	200000	23924	7
100000	100000	31920	1
50000	50000	18343	5
50000	50000	19205	7

A Tabela 4.4 apresenta as porcentagens de uso de cada conjunto de tarefas.

Tabela 4.4 - Uso de cada CPU nos Conjuntos de Tarefas 1, 2 e 3.

CPU	Conjunto 1 (Tabela 1)	Conjunto 2 (Tabela 2)	Conjunto 3 (Tabela 3)
0	54,20%	48,78%	55,83%
1	90,67%	81,60%	47,06%
2	68,92%	62,03%	68,49%
3	64,66%	58,19%	51,85%
4	83,95%	75,56%	46,55%
5	52,88%	47,59%	56,76%
6	56,97%	51,27%	60,97%
7	78,18%	70,36%	62,98%

A partir da definição dos conjuntos de tarefas, foi elaborado o conjunto de operações a serem executadas, para cobrir maior número de possibilidades, foram programadas três tipos de operações, sendo elas recursivas, iterativas e operações de leitura e escrita de memória. A base das operações realizadas é descrita abaixo:

- Operação recursiva: consiste na repetição do cálculo da função de Fibonacci para o número 25, com tempo aproximado de 600 us. O número de repetições é determinado pela divisão do WCET da tarefa por 600. Após cada execução do método, uma multiplicação de pontos flutuantes é executada para que se aumente o estresse na máquina e que se aumente a cobertura das funções executadas pelo processador;
- Operação iterativa: execução da função de Fibonacci para um número grande. Tem tempo de execução de 1 us, portanto o número de execuções é WCET;
- Operação de leitura e escrita de memória: executa repetidamente a leitura de regiões aleatórias de um vetor. Tempo médio de 30 us. Repetições são definidas pela divisão de WCET por 30. O tamanho do espaço de memória lido é de 128KB.

Os tempos para execução de cada operação foram calculados em uma execução sem heurística ativa.

4.2. CAPTURA DE DADOS

O total de eventos, possíveis para captura, implementados no EPOS e disponíveis na versão Sandy Bridge da PMU é 218, sendo 3 desses, configurados como contadores

fixos, totalizando 215 eventos configuráveis. Como a versão suporta apenas a contagem simultânea de 4 eventos, foram necessárias 54 execuções por task-set e tipo de execução.

Todas as execuções foram feitas utilizando o escalonador P-EDF (Partitioned EDF). Uma vez que os contadores foram alinhados para uma primeira análise, era necessário que as CPUs tivessem número semelhante de capturas (cada CPU possui uma fila de execução e, pela característica, já citada, de não intrusão do EPOS, as diferentes execuções mantém número semelhante de capturas, mantendo assim um certo determinismo entre as execuções).

Este nível de "determinismo" não aconteceria em execuções G-EDF, uma vez que a fila de execução é única e a distribuição das tarefas entre as CPUs sofre interferência de fatores não determinísticos, e também não seria possível com execuções sobre um escalonador C-EDF (Clustered EDF), uma vez que as filas podem ser compartilhadas por mais de uma CPU, logo as capturas seriam semelhantes entre clusters e não entre CPUs.

Para facilitar o processo de execução, foram criados um arquivo shell-script para compilar e enviar as imagens (alternando os eventos da PMU) e um outro para executar e enviar para a plataforma IoT os logs de execução, fazendo backup dos mesmos em uma mídia removível.

4.3. ANÁLISE

Uma vez realizadas todas as capturas, era necessário realizar uma análise para determinar os eventos a serem utilizados no desenvolvimento da heurística de Power-Cap. Num primeiro momento, porém, era necessário estimar o consumo

energético nas execuções para que se pudesse determinar um rumo quanto as correlações úteis.

4.3.1. ESTIMATIVA DE CONSUMO

A partir dos dados capturados, era necessário fazer estimativas do consumo durante a execução das task-sets antes de estipular-se um ponto de aplicação de Power-Cap.

Esta tarefa foi realizada, neste primeiro momento, com o auxílio da ferramenta Fluke [24], com diagrama de montagem do circuito de captura apresentado na Figura 4.1, utilizando alicates i430flex com limitação de 300 A (amperes) e utilizando o logger (monitor) para capturas a cada 0,25 segundos, ou seja, 250 milissegundos, com a adição dos medidores de consumo energético (nomeado pelo Fluke de Potência ou Power) aos dados a serem capturados.

Figura 4.1 – Diagrama de montagem do circuito de medição - Fluke.

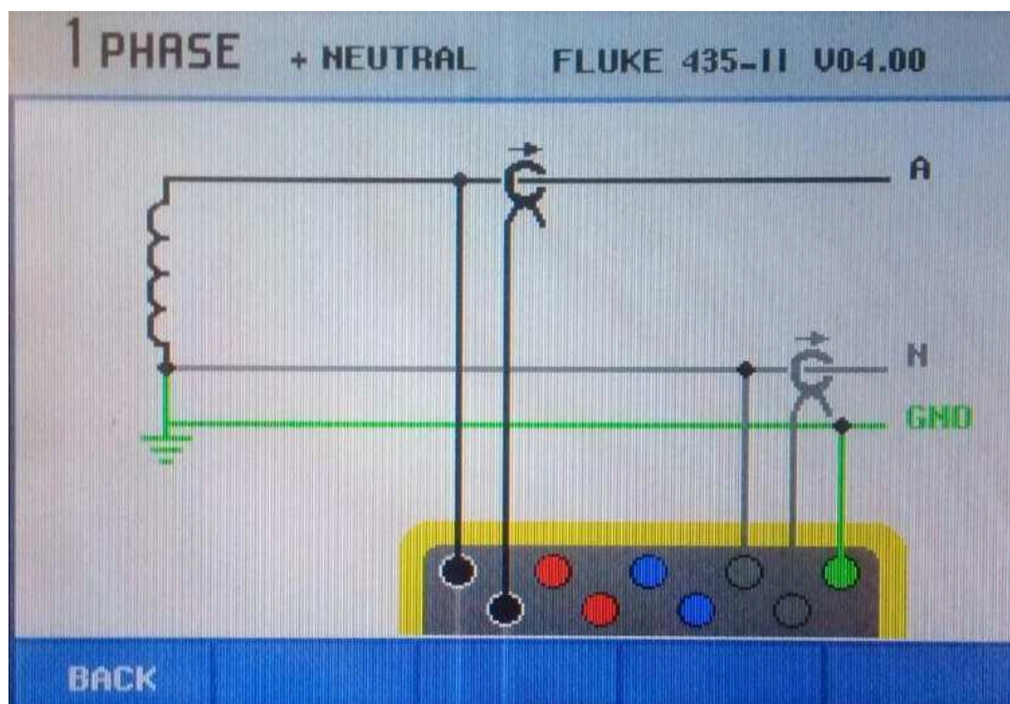


Imagem retirada em forma de fotografia do diagrama apresentado pelo Fluke.

A análise dos dados capturados é feita através do software Power Log, disponibilizado em um CD de instalação que acompanha o aparelho, juntamente com manual de uso em vários idiomas e algumas outras ferramentas, e também disponível no site da Fluke (https://www.fluke.com/pt-br).

Ao se realizar o comparativo com os valores de consumo obtidos a partir da interface RAPL, porém, o valor apontado não era compatível, fato já esperado uma vez que os indicadores da interface RAPL no processador utilizado são capazes de indicar apenas o consumo do pacote ou de um de seus componentes internos, não representando o consumo de energia total da execução das tarefas.

Neste momento se fez necessária uma escolha entre duas alternativas, sendo elas as opções de se trabalhar com a análise de consumo do processador e aplicar Power-Cap apenas sobre seu consumo, ou tratar do Power-Cap sobre todo o sistema.

Tendo em vista a dificuldade de aferir consumo utilizando a interface RAPL para alguns intervalos de tempo (dependendo do tamanho do intervalo de tempo as leituras eram imprecisas, principalmente quando os intervalos de tempo eram menores que um milissegundo), aliada aos fatos de que os dados utilizados para elicitação de eventos correlatos ainda não conterem as leituras dos registradores de consumo da interface e que ao aplicar-se Power-Cap sobre o sistema por completo, automaticamente estaria-se aplicando, também, sobre o processador, foi optado por trabalhar com o desenvolvimento de Power-Cap sobre o sistema por completo.

Uma vez efetuadas as aferições de consumo e levando em conta a escolha adotada, foi buscado um dos eventos da PMU ou dos demais eventos de Software e

Hardware capturados que melhor descrevesse a variação energética. Durante leituras e pesquisas, foi encontrada a relação entre dissipação de energia e temperatura. Para um melhor entendimento dessa relação, foi necessário entender os conceitos de TDP e do Perfil Térmico do processador utilizado.

4.3.1.1 TDP

Potência de Design Térmico (TDP - Thermal Design Power) representa a potência média, em Watts, que o processador dissipa quando operando na Frequência Básica do processador (Processor Base Frequency) com todos os cores ativos sobre uma carga de trabalho de alta complexidade definida pela Intel.

A Frequência Básica do processador descreve a taxa em que os transistores do processador abrem e fecham.

No caso do processador utilizado no desenvolvimento deste trabalho, tanto para captura de dados, quanto para testes, o TDP é fixado em 95W, atingidos com uso de memória RAM DDR3 de 1333MHz.

4.3.1.2 PERFIL TÉRMICO E TTV

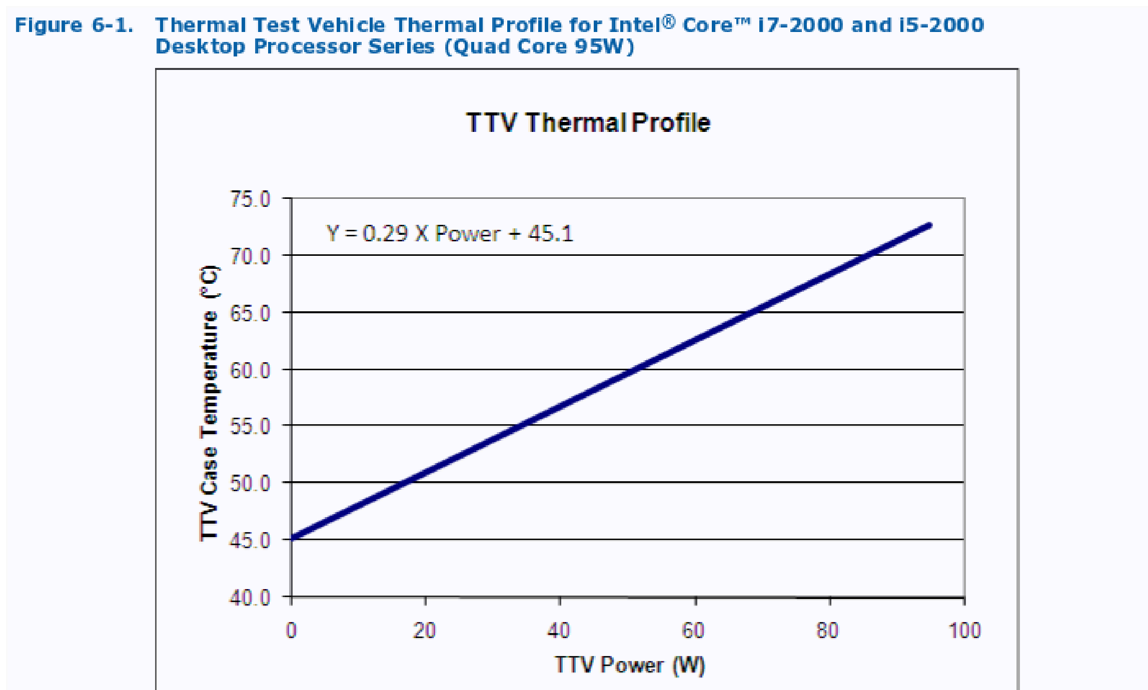
Perfil Térmico é definido pela Intel [20] como a linha que representa a especificação de temperatura do case do TTV num determinado nível de energia consumida, sendo o TTV definido como Veículo de Teste Térmico (Thermal Test Vehicle - TTV), definido como um pacote mecanicamente equivalente que contém um aquecedor resistivo na matriz ("cama" do processador) para avaliar as soluções térmicas.

Para entender melhor esta linha, deve-se entender alguns termos, são eles:

- Psi-CA: definido como parâmetro de caracterização térmica do case para o ambiente. Trata-se de uma medida de performance da solução térmica utilizando a toda a energia do pacote. É calculado como (T_{CASE} - T_{LA}) / Energia Total do Pacote (igual ao TDP);
- T_{CASE}: Temperatura do processador. A temperatura T_{CASE} do TTV é medida no topo do centro geométrico do dissipador de calor do TTV;
- T_{LA}: Temperatura do ambiente, medida nos arredores do processador. Pode ser estimada como a soma da temperatura externa a cpu e o aumento de temperatura típico do chassi do processador;
- $T_{CASE-MAX}$: Maior temperatura conforme especificação de um componente.

O gráfico de perfil térmico para o processador utilizado neste trabalho é fornecido pela Intel e pode ser conferido na Figura 4.2 (parte 'a') com temperatura ambiente de 45.1ºC. O $T_{CASE-MAX}$ para o modelo é de 72.6°C em 95W e o Psi-CA é de 0,29.

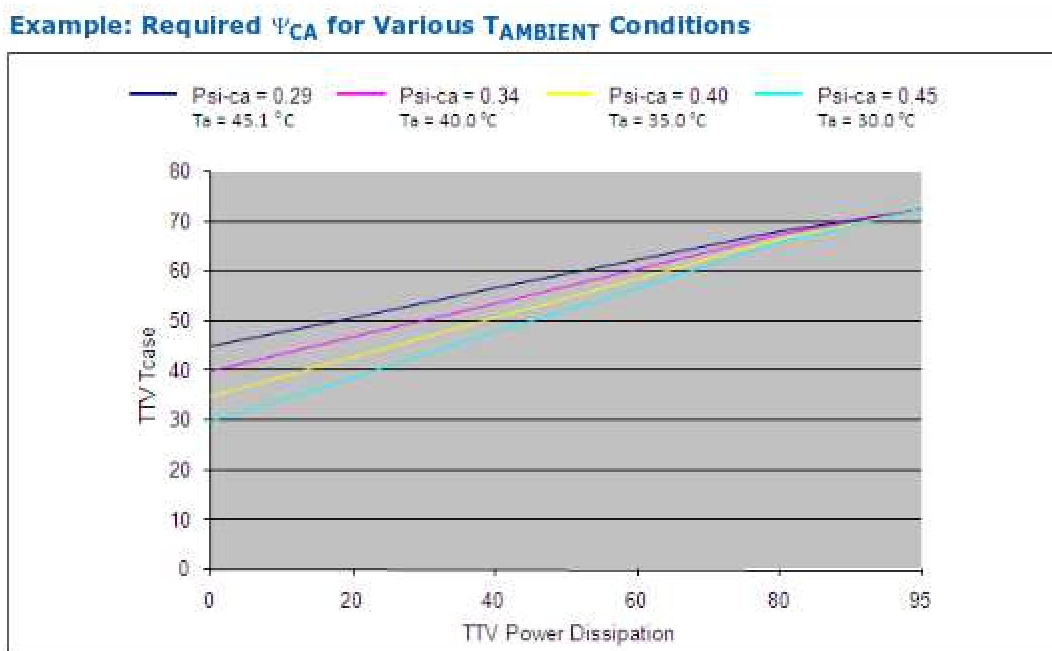
Figura 4.2 a) – Perfil Térmico do Processador Intel® Core^{TM} i7-2600.



Fonte: 2nd Gen Intel® Core™ Processor, LGA1155 Socket: Thermal Guide

Um dos fatores que interfere na relação entre energia dissipada e temperatura é a temperatura ambiente. A Intel disponibiliza, ainda, um gráfico (Figura 4.2 b) que ajuda a entender essa relação e a interferência da temperatura ambiente ($T_{AMBIENT}$).

Figura 4.2 b) – Variação de Psi-CA para vários $T_{AMBIENT}$.



Fonte: 2nd Gen Intel® Core™ Processor, LGA1155 Socket: Thermal Guide

Conforme determinado pela Intel no Guia Térmico do processador [20], quanto mais baixa a temperatura ambiente, mais alto será o Psi-CA requerido, o que significa velocidades mais baixas na ventilação e redução na acústica da solução térmica do processador.

4.3.2. PREPARAÇÃO

Uma vez os dados recolhidos, para aplicação de mineração de dados era necessário um pré-processamento. Tal procedimento foi dividido em etapas de seleção de dados, limpeza e transformação.

Para a análise dos dados capturados, foi optado por trabalhar com arquivos CSVs (comma-separated values). Para transformar os logs gerados em arquivos neste formato, foram utilizados scripts em python.

4.3.2.1 SELEÇÃO DE DADOS

Esta etapa do pré-processamento dos dados tem como objetivo a redução do conjunto de dados a ser trabalhado nas próximas etapas e na mineração em si. Como o conhecimento prévio sobre os dados era pequeno, e os eventos poderiam não ser de fácil interpretação, foi optado por trabalhar com praticamente todo o conjunto de dados, não eliminando nenhum dos eventos capturados. Os únicos dados que foram excluídos da análise foram os identificadores da thread, sua prioridade e os contadores de deadline perdidas (deadline misses).

4.3.2.2 LIMPEZA DE DADOS

Esta etapa do pré-processamento tem como objetivo o tratamento de incompletudes, a remoção de ruídos, através da identificação de outliers, e a correção de inconsistências no conjunto de dados [26].

No caso dos dados trabalhados, as coletas foram feitas em um sistema planejado de modo a evitar incompletudes ou inconsistências, uma vez que todos os valores eram capturados com a devida tipagem em cada uma das capturas.

Quanto a remoção de ruídos, foi optado por ignorar a análise da primeira captura, uma vez que passa-se um período de inicialização onde os contadores demonstram um comportamento não advindo da aplicação e em um período de tempo tipicamente superior aos intervalos entre interrupções e inicializações e/ou finalizações de tarefas.

Outros dados removidos foram as paralisações que expressam o momento em que uma thread é liberada pelo *dispatch* para começar sua execução e o alarme que libera um novo ciclo de execução periódica ainda não foi liberado, logo o intervalo entre a execução dos métodos de *dispatch* e *wait-next* não representa um período de execução das tarefas e pode ter intervalo de tempo de 0 us (microsegundos), comprometendo assim o cálculo de crescimento dos contadores no intervalo.

Com exceção aos aspectos levantados acima, não foram retirados pontos de outlier, pois foi adotado o pressuposto de que o comportamento dos contadores durante a execução possam ser explicados e que as discrepâncias poderiam indicar alguma variação de importância na análise.

4.3.2.3INTEGRAÇÃO DE DADOS

Este procedimento consiste em unir dados de várias fontes para a aplicação de mineração de dados. Num primeiro momento, partindo do pressuposto de que a interferência causada pelo EPOS é baixíssima e que não haviam fatores externos que viessem a interferir na execução dos conjuntos de tarefas (task-sets), foi adotada uma abordagem de unir dados de cada uma das 54 capturas num único arquivo.

Entretanto, a união das capturas acarretou numa pequena perda de precisão (na ordem de alguns microssegundos, ou até um milisegundo, diferença que pode ser causada por aspectos da computação Multi-Core que são, tipicamente, não

determinísticos) e também exigiu a escolha de utilizar média dos contadores fixos, ou ignorar os valores das demais capturas. Para que não houvesse perda de precisão quanto a nenhum contador, nem quanto as variações e as diferenças nos tempos dos intervalos, uma vez que, quando se perde precisão na contagem de tempo, as taxas de crescimento também perdem e os contadores têm seu comportamento distorcido (mesmo que minimamente).

Tendo em vista os problemas encontrados, foi adotada a estratégia de se analisar os comportamentos dos contadores de cada execução separadamente.

4.3.2.4TRANSFORMAÇÃO DOS DADOS

A etapa de transformação dos dados consiste em transformar ou consolidar os dados para que a mineração exercida seja mais eficiente [26]. São estratégias de transformação a aplicação de normalização, que consiste em deixar todos os dados numa igual faixa de variação (geralmente de 0 à 1), agregação, onde vários registros com granularidade maior são agrupados em um único registro com granularidade menor (de dia para mês, por exemplo), discretização, quando atributos numéricos são separados em intervalos e construção de atributos, onde um novo atributo é gerado através dos dados já existentes.

Para este trabalho, como os dados são capturas de contadores digitais e os mesmos são constantemente incrementados, a transformação necessária consistiu em calcular a diferença entre os contadores de modo a determinar o crescimento no atual período de tempo e, também, calcular as médias de crescimento pelo tamanho do intervalo de tempo (crescimento / tempo = unidades_incrementadas/microssegundo).

4.3.3. LEVANTAMENTO DE CORRELAÇÕES

Durante a fase de levantamento de correlações foram utilizadas duas abordagens. A primeira delas buscava correlações sobre crescimentos absolutos de contadores (quanto os contadores cresciam no intervalo entre duas capturas), enquanto a segunda, buscava correlações entre valores médios de crescimento (crescimento no intervalo / tempo). A segunda abordagem contou também com testes analisando variação de temperatura.

Para a primeira abordagem foram usados dois algoritmos disponíveis no Weka, o *PrincipalComponents*, algoritmo que executa PCA (Principal Components Analysis - Análise dos Principais Componentes) e o *CorrelationAttributeEval*, algoritmo que avalia o valor de um atributo medindo a correlação, através do método de Pearson, entre ele e a classe escolhida. Ambos os métodos estão disponíveis por padrão na aba de Seleção de Atributos (Select attributes) do Weka nas suas versões *3.8.2* e *3.9.2*.

Após esta primeira etapa, foi optado por trabalhar com análise de valores médios com a variação de temperatura (segunda abordagem), uma vez que o foco seria identificar quais contadores descreviam melhor uma situação de alta ou de baixa no aquecimento do processador. O processo também se seguiu focando no uso do algoritmo *CorrelationAttributeEval*, uma vez que sua saída é mais facilmente entendida pelo fato de que o algoritmo entrega as correlações de todos os atributos presentes com a classe.

Um modelo de saída do algoritmo para valores relativos de temperatura e média de crescimento dos contadores pode ser encontrado na Figura 4.3.

Figura 4.3 – Aplicação do algoritmo *CorrelationAttributeEval* sobre dados capturados.

```
=== Run information ===

Evaluator:    weka.attributeSelection.CorrelationAttributeEval
Search:       weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1
Relation:     batch34_cpu7_named
Instances:    3245
Attributes:   9
              temperature
              inst_retired
              core_cycle
              ref_clock
              OFFCORE_REQUESTS.DEMAND_DATA_RD
              OFFCORE_REQUESTS.DEMAND_RFO
              OFFCORE_REQUESTS.ALL_DATA_RD
              UOPS_DISPATCHED.CORE
              timestamp
Evaluation mode:    evaluate on all training data



=== Attribute Selection on all input data ===

Search Method:
        Attribute ranking.

Attribute Evaluator (supervised, Class (numeric): 1 temperature):
        Correlation Ranking Filter
Ranked attributes:
 0.50381   8 UOPS_DISPATCHED.CORE
 0.39905   4 ref_clock
 0.39901   3 core_cycle
 0.35247   2 inst_retired
 0.34963   7 OFFCORE_REQUESTS.ALL_DATA_RD
-0.00454   5 OFFCORE_REQUESTS.DEMAND_DATA_RD
-0.00703   9 timestamp
-0.02427   6 OFFCORE_REQUESTS.DEMAND_RFO
```

Os parâmetros utilizados nos métodos aplicados são padrões do Weka, nenhum deles foi alterado.

O procedimento foi repetido para cada uma das 54 execuções usando métodos recursivos e operações de memória, gerando assim, todos os valores de correlação entre cada um dos eventos e a variação de temperatura. Os melhores resultados obtidos são apresentados na Tabela 4.5.

Tabela 4.5 – Tabela das principais correlações com a Variação de Temperatura.

	Exec. Recursiva	Correlação	Exec. Memcpy	Correlação
1	**UOPS_DISPATCHED.THREAD**	0.5	**UOPS_DISPATCHED.THREAD**	0.48
2	**INT_MISC.RECOVERY_CYCLES**	0.42	**ref_clock / core_cycle**	0.35
3	**ref_clock / core_cycle**	0.4	L2_STORE_LOCK_RQSTS.MISS	0.32
4	RESOURCE_STALLS2.OOO_RSRC	0.4	**BR_MISP_RETIRED.NEAR_CALL**	0.31
5	CPL_CYCLES.RING123	0.4	**UOPS_ISSUED.ANY**	0.31
6	**BR_MISP_RETIRED.NEAR_CALL**	0.36	INT_MISC.RAT_STALL_CYCLES	0.3
7	inst_retired	0.35	**INT_MISC.RECOVERY_CYCLES**	0.3
8	Branch Instruction Retired	0.34	LD_BLOCKS_PARTIAL.ADDRESS_ALIAS	0.3
9	IDQ_UOPS_NOT_DELIVERED.CORE	0.34	INSTS_WRITTEN_TO_IQ.INSTS	0.3
10	**UOPS_ISSUED.ANY**	0.25	L1D_PEND_MISS.PENDING	0.3

Durante a geração de correlações, foram obtidos eventos com valores superiores a alguns dos apresentados nesta tabela (Tabela 4.5), porém, ao analisar o log de execução, verificava-se que o comportamento do canal era descrito pelos contadores de *UnHalted Cycles* ou pelo evento *Instruction Retired*. Tais eventos foram removidos da tabela de correlações e ignorados no processo de análise (vide exemplos na Figura 4.4).

Figura 4.4 a) – Exemplo de evento ignorado por semelhança ao UnHalted Cycles.

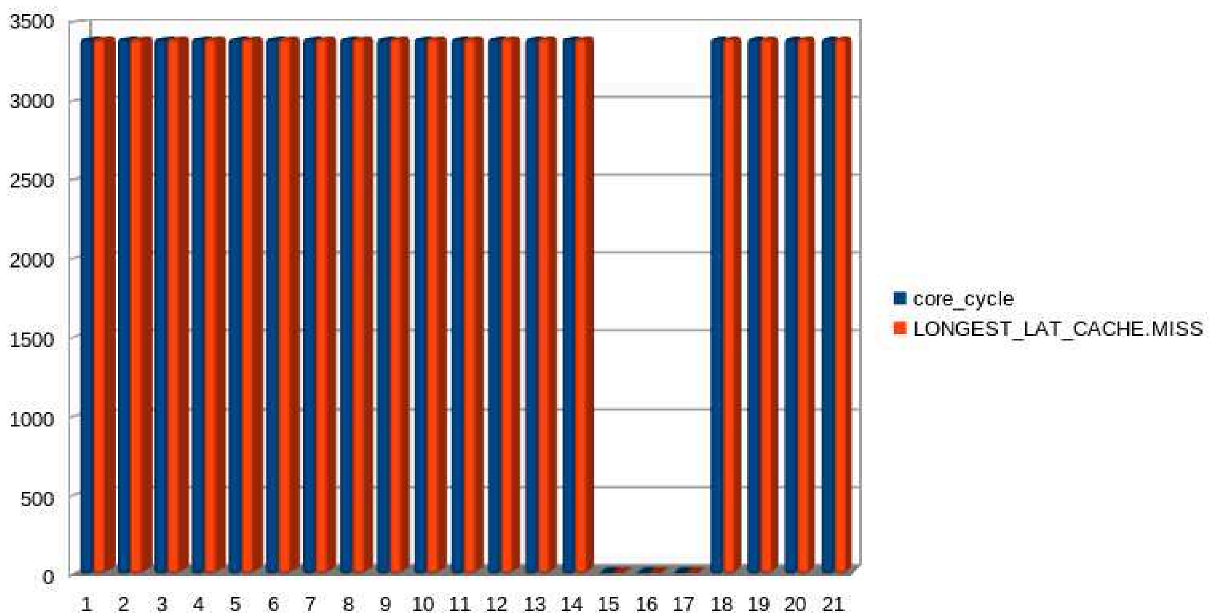
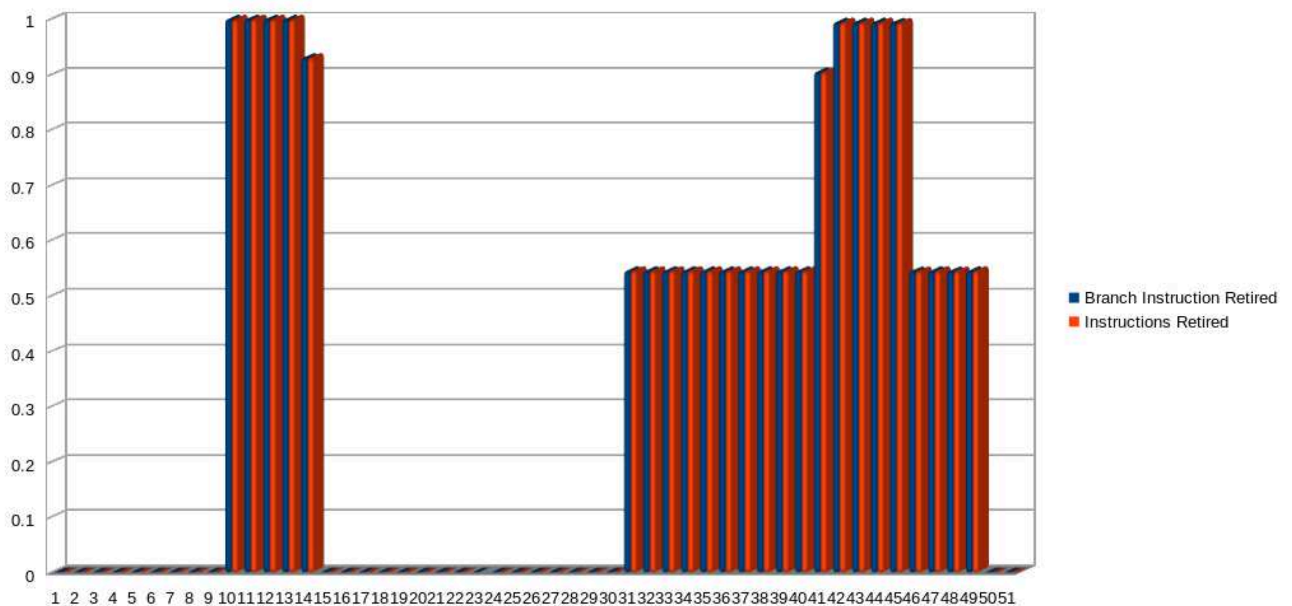


Figura 4.4 b) – Exemplo de evento ignorado por semelhança ao Instructions Retired.



Os resultados obtidos nesta etapa guiaram as opções para o seguimento de todo o trabalho, sendo que após a obtenção das correlações, foi feita ainda uma análise “manual” dos dados para melhor entender os resultados e também para estipular alguns pontos de cortes. Vale ressaltar que dos 5 (cinco) canais encontrados em comum entre os dois tipos de testes, 1 (um) deles é fixo e os demais são configuráveis.

4.3.4. DEFINIÇÃO DO POWER-CAP

Com os resultados obtidos e correlações encontradas, era hora de definir o ponto para aplicação de Power-Cap, em outras palavras, definir o consumo máximo a ser utilizado. Dado que o processador tem TDP de 95W (importante lembrar que TDP não é o maior valor de energia que o processador pode dissipar), foi escolhido um limite de 80W para aplicação da heurística (ponto escolhido para efeitos de teste).

Uma vez escolhido o valor do Power-Cap foi escolhido trabalhar seguindo a função apresentada na Figura 4.2 a, que apresenta o pior caso típico de funcionamento de um sistema com TA de 45.1°C. Logo, a temperatura do case deve ser em torno de 68.3ºC (0.29*80 + 45.1 = 68.3).

Como já citado no tópico “Estimativa de Consumo” (4.3.1), dos dados capturados a temperatura apresenta a melhor ligação com o consumo. Tendo em vista esta ligação e também o fato de o sensor térmico utilizado ser digital e apresentar valores inteiro, o ponto de corte utilizado deverá levar em conta a temperatura de 68°C.

Por efeitos de análise de viabilidade, foi aferido a variação de consumo a 68° entre o ambiente descrito e o ambiente mais frio apresentado no manual da Intel [20, Table 6-6, páginas 44 e 45], onde TA é descrito em 20ºC. Para tal, aplicou-se o cálculo e*0.554 = 68 - 30 (0.554 é o valor encontrado na última coluna da tabela, que é compatível aos valores apresentados na Figura 4.2 a), obtendo o valor de 87W (86.64), sendo a variação considerada aceitável (a temperatura ambiente - TA - deve ser lida na entrada da solução térmica do processador).

4.3.5. USO DE CLASSIFICADORES E ÁRVORES DE DECISÃO

Uma vez definido o consumo máximo, o próximo passo consistiu em utilizar métodos de mineração de dados e aprendizado de máquina para gerar um preditor. Entretanto, era necessário preparar os dados para isso.

4.3.5.1 PREPARAÇÃO E DEFINIÇÃO DE CLASSES

Com o intuito de melhorar os resultados obtidos pelos classificadores, foram feitas as seguintes alterações no arquivo de dados:

- Adição de um atributo com definição de duas classes de temperatura, a primeira delas com um valor “ok”, a ser utilizada quando a temperatura da próxima captura fosse menor ou igual a 68ºC, e a segunda com o valor “passed”, a ser usada quando a temperatura da próxima captura fosse maior que 68°C;
- Alteração de “don’t care” nos eventos fixos de *Instruction Retired* e *UnHalted Cycles* (respectivamente canais 0 e 1 da PMU);
- Redução do conjunto de dados para igualar os registros com temperaturas abaixo e acima da média de forma a não gerar vícios no preditor e evitar situação descrita nas Figuras 4.5 a e b.

Levando em consideração as alterações citadas, uma linha de análise é gerada a partir de 3 capturas, sendo que os valores dos contadores equivalem ao crescimento da primeira para a segunda e o valor da temperatura a ser previsto é o da terceira captura.

4.3.5.2GERAÇÃO DE ÁRVORES

A partir das transformações dos dados e da definição das classes, foram aplicados alguns algoritmos classificadores presentes no Weka. Estes algoritmos tinham como entrada os dados capturados com a concatenação de uma classe (variável a ser prevista) contendo dois valores possíveis, esta classe, então, era determinada a partir da temperatura aferida na captura seguinte registrada no log de execução, sendo “*ok*” para temperaturas menores ou iguais ao limite (aqui, 68°C) e “*passed*” para temperaturas superiores ao limite.

A saída esperada dos algoritmos classificadores, neste caso, era um modelo que baseado nos valores dos contadores fosse capaz de determinar a classe da temperatura

da captura a seguir, funcionando, assim, como um preditor da situação térmica da execução e, por consequência, como um indicador dos níveis de desempenho a serem apontados nos contadores para determinados níveis de aquecimento.

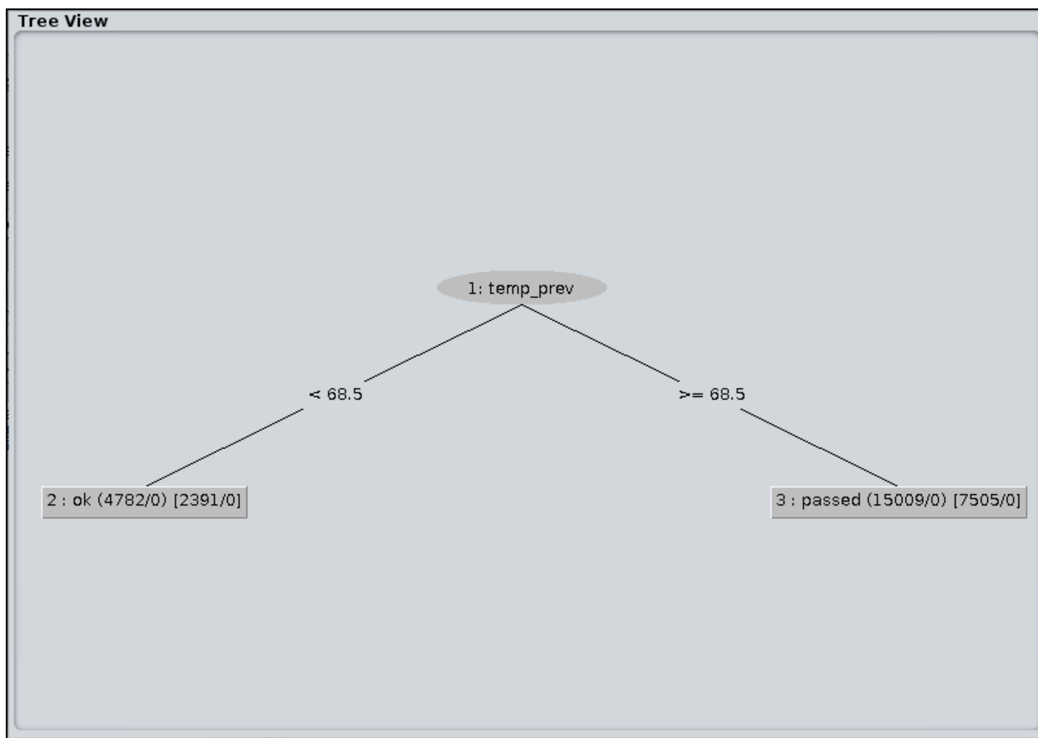
As primeiras tentativas de aplicação de classificadores, no entanto, motivaram mais uma alteração nos registros uma vez que, ao aplicar os métodos sobre todos os dados, as árvores retornadas ignoravam alguns contadores (entre eles o de maior correlação) e utilizavam pontos de corte superiores ao limite de temperatura, ou se tornavam, por vezes, simplistas, apresentando um simples decisor binário, que verificava a temperatura na captura anterior para prever a próxima (Figura 4.5 a e b, respectivamente, descrições textual e gráfica), quando gerada sobre o conjunto total de dados capturados.

Figura 4.5 a) – Árvore de decisão simples - Descrição Textual.

```
REPTree
============

temp_prev < 68.5 : ok (4782/0) [2391/0]
temp_prev >= 68.5 : passed (15009/0) [7505/0]
```

Figura 4.5 b) – Árvore de decisão simples - Descrição Gráfica.



Tendo em vista o problema encontrado, passou-se a ignorar um dos atributos que centralizavam a heurística (Figuras 4.6 a e b, apresentando representações textual e gráfica, respectivamente), buscando obter uma árvore que contempla-se melhor o conjunto avaliado e que correspondesse melhor as análises feitas sem auxílio dos classificadores. O contador ignorado representa o evento INT_MISC.RECOVERY_CYCLES.

Figura 4.6 a) – Árvore de decisão processo 2 - Representação Textual.

```
REPTree
============

temp_prev < 68.5 : ok (2617/74) [1297/22]
temp_prev >= 68.5
|   temp_prev < 69.5
|   |   pmu5 < 4.86
|   |   |   ref_clock < 3393.06
|   |   |   |   ref_clock < 3392.26 : passed (307/46) [150/17]
|   |   |   |   ref_clock >= 3392.26
|   |   |   |   |   timestamp < 4037.5 : passed (50/0) [24/2]
|   |   |   |   |   timestamp >= 4037.5
|   |   |   |   |   |   pmu5 < 4.85
|   |   |   |   |   |   |   pmu6 < 89.82
|   |   |   |   |   |   |   |   pmu6 < 86.84 : passed (15/2) [6/2]
|   |   |   |   |   |   |   |   pmu6 >= 86.84 : ok (3/0) [1/0]
|   |   |   |   |   |   |   pmu6 >= 89.82 : passed (39/2) [15/1]
|   |   |   |   |   |   pmu5 >= 4.85 : passed (23/0) [12/2]
|   |   |   ref_clock >= 3393.06 : passed (12/5) [2/0]
|   |   pmu5 >= 4.86 : passed (98/0) [50/1]
|   temp_prev >= 69.5 : passed (1841/0) [946/0]
```

Figura 4.6 b) – Árvore de decisão processo 2 - Representação Gráfica.

Imagem gerada com o visualizador de árvores do Weka.

A ordenação dos eventos capturados e a relação de evento e nomenclatura utilizada durante a análise é dada na Tabela 4.6.

Tabela 4.6 – Mapeamento de eventos.

Nomenclatura	Descrição
temp_prev	temperatura capturada no ciclo atual
temp_delta	Classe de temperatura prevista
inst_retired	evento INSTRUCTION_RETIRED
core_cycle	evento UNHALTED_CORE_CYCLES
ref_clock	evento UNHALTED_REFERENCE_CYCLES
pmu3	evento UOPS_ISSUED.ANY
pmu4	evento UOPS_DISPATHCED.THREAD
pmu5	evento BR_MISP_RETIRED.NEAR_CALL
pmu6	evento INST_MISC.RECOVERY_CYCLES
timestamp	intervalo de tempo que separa a captura atual da anterior

A alteração realizada fez com que a nova árvore gerada fosse mais complexa e levasse em conta mais contadores, além de apresentar ponto de corte com temperaturas mais baixas (menor que 67 vs maior que 68 da árvore anterior, como a análise se dá em números inteiros, trata-se de 2ºC de espaço), aumentando o tempo hábil para a ação da heurística e sendo, portanto, a árvore escolhida para continuidade do trabalho (Figura 4.7).

Figura 4.7 – Árvore de decisão utilizada - Descrição Textual Completa.

```
REPTree
============

temp_prev < 68.5
|   temp_prev < 67.5 : ok (1285/1) [623/1]
|   temp_prev >= 67.5
|   |   pmu3 < 53.5 : ok (153/26) [82/9]
|   |   pmu3 >= 53.5
|   |   |   timestamp < 4996.5 : ok (74/1) [38/4]
|   |   |   timestamp >= 4996.5
|   |   |   |   ref_clock < 3392.27
|   |   |   |   |   ref_clock < 3392.23 : ok (69/4) [53/7]
|   |   |   |   |   ref_clock >= 3392.23
|   |   |   |   |   |   pmu3 < 75.5
|   |   |   |   |   |   |   pmu3 < 62.5
|   |   |   |   |   |   |   |   pmu4 < 13189.18 : ok (50/9) [18/5]
|   |   |   |   |   |   |   |   pmu4 >= 13189.18 : passed (4/1) [1/0]
|   |   |   |   |   |   |   pmu3 >= 62.5 : ok (103/10) [60/12]
|   |   |   |   |   |   pmu3 >= 75.5 : ok (7/3) [7/0]
|   |   |   |   ref_clock >= 3392.27 : ok (17/0) [13/0]
temp_prev >= 68.5
|   temp_prev < 69.5
|   |   pmu5 < 4.86
|   |   |   pmu4 < 13141.18 : passed (20/0) [6/1]
|   |   |   pmu4 >= 13141.18
|   |   |   |   ref_clock < 3393.12
|   |   |   |   |   ref_clock < 3392.25
|   |   |   |   |   |   pmu5 < 4.85 : passed (13/6) [3/0]
|   |   |   |   |   |   pmu5 >= 4.85
|   |   |   |   |   |   |   pmu5 < 4.85 : passed (168/18) [99/8]
|   |   |   |   |   |   |   pmu5 >= 4.85
|   |   |   |   |   |   |   |   pmu5 < 4.85
|   |   |   |   |   |   |   |   |   pmu4 < 13161.01
|   |   |   |   |   |   |   |   |   |   pmu3 < 75.5
|   |   |   |   |   |   |   |   |   |   |   ref_clock < 3392.18 : passed (2/0) [4/0]
|   |   |   |   |   |   |   |   |   |   |   ref_clock >= 3392.18
|   |   |   |   |   |   |   |   |   |   |   |   pmu4 < 13143.02 : passed (11/5) [5/1]
|   |   |   |   |   |   |   |   |   |   |   |   pmu4 >= 13143.02 : ok (9/2) [1/0]
|   |   |   |   |   |   |   |   |   |   pmu3 >= 75.5 : passed (3/0) [1/0]
|   |   |   |   |   |   |   |   |   pmu4 >= 13161.01 : passed (6/0) [0/0]
|   |   |   |   |   |   |   |   pmu5 >= 4.85 : passed (50/7) [26/5]
|   |   |   |   |   ref_clock >= 3392.25 : passed (155/11) [76/10]
|   |   |   |   ref_clock >= 3393.12 : passed (9/4) [3/0]
|   |   pmu5 >= 4.86 : passed (92/0) [45/0]
|   temp_prev >= 69.5 : passed (4006/0) [1990/0]
```

No entanto, a árvore escolhida apresenta grande complexidade, sendo que sua segunda grande ramificação, para temperatura maior que 68, apresenta uma situação que deve ser evitada, além de apresentar, em suma maioria, pontos em que a temperatura na próxima iteração estará maior que o limite estabelecido. Tendo em vista isto, para

implementação da heurística, este nodo será ignorado, sendo o código gerado analisando apenas o primeiro nodo (Figura 4.8).

Figura 4.8 – Árvore de decisão utilizada - Descrição Textual Sintetizada.

```
REPTree
============

temp_prev < 68.5
|   temp_prev < 67.5 : ok (1285/1) [623/1]
|   temp_prev >= 67.5
|   |   pmu3 < 53.5 : ok (153/26) [82/9]
|   |   pmu3 >= 53.5
|   |   |   timestamp < 4996.5 : ok (74/1) [38/4]
|   |   |   timestamp >= 4996.5
|   |   |   |   ref_clock < 3392.27
|   |   |   |   |   ref_clock < 3392.23 : ok (69/4) [53/7]
|   |   |   |   |   ref_clock >= 3392.23
|   |   |   |   |   |   pmu3 < 75.5
|   |   |   |   |   |   |   pmu3 < 62.5
|   |   |   |   |   |   |   |   pmu4 < 13189.18 : ok (50/9) [18/5]
|   |   |   |   |   |   |   |   pmu4 >= 13189.18 : passed (4/1) [1/0]
|   |   |   |   |   |   |   pmu3 >= 62.5 : ok (103/10) [60/12]
|   |   |   |   |   |   pmu3 >= 75.5 : ok (7/3) [7/0]
|   |   |   |   ref_clock >= 3392.27 : ok (17/0) [13/0]
```

Ao analisar a árvore gerada, possível notar que há definição de pontos de “corte” para timestamp e ciclos de clock. Tais aspectos são possíveis de serem compreendidos a partir de duas especificações da configuração aplicada ao sistema.

A primeira delas é a definição de fatia de tempo do escalonador (time slice), uma vez que o EPOS está configurado para gerar interrupções de reescalonamento a cada 5000 microsegundos, logo se o intervalo entre duas passagens pelo *dispatch* for muito menor que este, a tarefa crítica terminou sua execução antes da próxima parada e o sistema não deve permitir aquecimento com outras tarefas.

A segunda configuração é de cunho estático e refere-se a frequência base do processador, que apesar de ser documentada em 3.4GHz (3400 MHz) tem uma pequena

variação na leitura, ficando próxima à 3392 MHz, indicando que se a frequência de operação for reduzida, a tendência do sistema é o resfriamento.

Todas as árvores apresentadas neste capítulo foram geradas a partir do algoritmo REPTree, disponível por padrão (sem necessidade de se adicionar Softwares ou bibliotecas) no Weka. O algoritmo REPTree consiste em um algoritmo de aprendizado rápido de árvores de decisão que funciona construindo uma árvore de decisão/regressão usando as informações de ganho/variância e podando através da poda com erro-reduzido, utilizando backfitting (a descrição feita consiste em uma livre tradução do texto descrição contido no Weka).

5 DESENVOLVIMENTO DA HEURÍSTICA

Uma vez concluído o processo de análise dos dados e geração de correlações e árvores de decisão, foi iniciado o desenvolvimento da heurística, que, inicialmente foi pensada e codificada baseada na análise manual dos contadores correlatos. Contudo, devido as imprecisões e a não-exploração do potencial de desempenho sob um nível de consumo, os trabalhos de implementação foram focados na árvore de decisão gerada pelo Weka (Figura 4.8).

5.1. AÇÕES DE CONTROLE

A fim de melhorar os recursos de ação, a heurística implementada altera a política de escalonamento (P-EDF) do EPOS, tomando algumas ações conforme o estado dos contadores e da temperatura. Para tal, implementações extras se fizeram necessárias para suspensão controlada de tarefas, lógica de redução de frequência de operação (DVS) e uma lógica de desvio de escalonamento.

5.1.1. SUSPENSÃO CONTROLADA DE TAREFAS

Uma das alternativas encontradas para controle de consumo foi a suspensão de tarefas não críticas, uma vez que a prioridade de execução num sistema de Tempo-Real devem ser as tarefas de maior prioridade (Tempo-Real crítico ou *Hard Real-Time*).

Todavia, ao analisar o código do EPOS, foi verificado que o método de suspensão utilizado é projetado de modo a fornecer o controle das tarefas (*tasks* ou *threads*) suspensas única e exclusivamente ao usuário (programador da aplicação), tirando este controle do sistema operacional.

Para não alterar as escolhas de projeto do EPOS, foi implementada, então, uma nova lógica de suspensão, a ser ativa, única e exclusivamente pelo SO (sistema operacional). Tal lógica consistiu na implementação de dois novos métodos (suspensão e reativamento) nos quais o controle das tarefas suspensas fica em controle do próprio sistema.

Uma vez implementados os métodos, foi criada uma lista contendo as tasks suspensas, sendo esta lista estática e separada por núcleo de processamento lógico, uma vez que para o escalonador P-EDF, cada núcleo é uma partição. Além disso foi também implementado um contador de tarefas suspensas, para auxiliar no processo de tomadas de decisão.

Com toda a infraestrutura de suspensão implementada, os pontos de controle foram colocados em locais estratégicos dentro da implementação da heurística, que por fins de desempenho e não intrusividade, é executada dentro do *dispatch*, i.e. momento reescalonamento.

5.1.2. LÓGICA DE REDUÇÃO DE FREQUÊNCIA DE OPERAÇÃO (DVS)

Última alternativa a ser adotada pela heurística, a redução de frequência de operação dos cores, ou, simplesmente, aplicação de DVS, já fora previamente implementada, restando apenas o controle da diferença na modulação de clock através da inconsistência na configuração dos duty cycles entre cores lógicos de um mesmo núcleo físico.

Para efetuar tal controle, foi adicionado a estrutura de código da Thread (classe Thread), um vetor estático (todas as threads sabem a que frequência estão executando) separado por core físico. O acesso a este vetor utiliza-se de uma função já existente no

EPOS usada para retornar o número do núcleo lógico, sendo o número fisico calculado utilizando a operação modular por 2, observando a numeração atribuída para o padrão de *hyperthreading* para *quad-cores*.

O valor contido no vetor serve para manter as operações sempre coerentes à frequência corrente. A escrita é feita utilizando o core atualmente executando.

5.1.3. LÓGICA DE DESVIO DE ESCALONAMENTO

A operação de desvio de escalonamento consiste em uma alternativa ao uso da suspensão controlada de tarefas. Seu efeito é similar, pois altera a thread a ser escalonada com intuíto de evitar maior aquecimento.

O método foi desenvolvido tendo em vista o momento em que a tarefa a ser escalonada é a mesma que está causando aquecimento (tarefa rodando = tarefa a ser escalonada), porém por se tratar da tarefa em execução, não pode ser suspensa (sob o risco de causar uma falha no sistema operacional por utilizar uma tarefa para suspender a si mesma durante a execução). Deste modo, apenas usa-se um método já existente no EPOS, consistindo em solicitar ao escalonador uma nova tarefa a ser executada.

5.2. IMPLEMENTAÇÃO BASEADA NA ÁRVORE DE DECISÃO

Partindo das ações planejadas, foi dado início a codificação das heurística a partir da árvore gerada, levando em conta, entretanto, apenas os limites de corte inferiores, de modo a dar mais robustez a heurística, uma vez que a geração de árvores de decisão leva em consideração apenas os dados fornecidos e não necessariamente é capaz de abranger os casos mais específicos de todas as execuções, ainda mais ao se levar em conta um conjunto diferente de tarefas (com variações de porcentagem de uso ou de

operações executadas). O resultado desta etapa pode ser conferido através do pseudocódigo exposto na Figura 5.1 (partes a, b, c e d).

Figura 5.1 a) – Pseudocódigo da heurística - parte dispatch.

```
// METODO DISPATCH
01: temperatura = CPU.le_temperatura()
02: deadline_misses_tarefa_atual = tarefa_atual.calcula_deadline_misses()
03: deadline_misses_proxima_tarefa = proxima_tarefa.calcula_deadline_misses()
04: timestamp = timestamp_atual - monitor.tempo_ultima_captura()
05: posicao_vetor_clock = posicao_meu_core( CPU.numero() )
06: ref_clock = pmu.le_canal(2)
07: pmu3 = pmu.le_canal(3)
08: pmu4 = pmu.le_canal(4)
09: pmu5 = pmu.le_canal(5)
10: if ( temperatura <= 65 or execucao_finalizada ):
11:    if ( suspensos[CPU.numero()] > 0 ):
12:       Thread.resumir( suspensos[CPU.numero()] )
13: if ( temperatura <= 64 ):
14:    if ( fator_clock[posicao_vetor_clock] < 8):
15:       fator_clock[posicao_vetor_clock] += 1
16: if (deadline_misses_tarefa_atual or deadline_misses_proxima_tarefa or deadline_misses_sistema):
17:    if ( temperatura < 67 ):
18:       if ( fator_clock[posicao_vetor_clock] < 8):
19:          fator_clock[posicao_vetor_clock] += 1
20:    if ( proxima_tarefa != tarefa_atual and proxima_tarefa.prioridade < CRITICA ):
21:       if ( proxima_tarefa != IDLE and proxima_tarefa != MAIN ):
22:          Thread.suspender(proxima_tarefa)
23:          proxima_tarefa = scheduler.escolha()
24: else:
25:    if ( temperatura < 67 and temperatura > 65 ):
26:       if ( proxima_tarefa != tarefa_atual and proxima_tarefa.prioridade < CRITICA ):
27:          if ( proxima_tarefa != IDLE and proxima_tarefa != MAIN ):
28:             Thread.suspender(proxima_tarefa)
29:             proxima_tarefa = scheduler.escolha()
30:       else if ( proxima_tarefa != IDLE and proxima_tarefa != MAIN ):
31:          if ( proxima_tarefa == tarefa_atual ):
32:             proxima_tarefa = scheduler.escolha_outra()
33:    else:
34:       if ( pmu3 >= 54 and timestamp > 4990 AND ref_clock > 3392 and pmu4 >= 13000 ):
35:          if ( proxima_tarefa != tarefa_atual and proxima_tarefa.prioridade < CRITICA ):
36:             if ( proxima_tarefa != IDLE and proxima_tarefa != MAIN ):
37:                Thread.suspender(proxima_tarefa)
38:                proxima_tarefa = scheduler.escolha()
39:          else if ( proxima_tarefa != IDLE AND proxima_tarefa != MAIN ):
40:             if ( proxima_tarefa == tarefa_atual ):
```

```
41:             proxima_tarefa = scheduler.escolha_outra()
42:       else if ( pmu5 > 4 AND pmu4 > 13000 ):
43:          if ( proxima_tarefa != tarefa_atual and proxima_tarefa.prioridade < CRITICA ):
44:             if ( proxima_tarefa != IDLE and proxima_tarefa != MAIN ):
45:                Thread.suspender(proxima_tarefa)
46:                proxima_tarefa = scheduler.escolha()
47:          else if ( proxima_tarefa != IDLE and proxima_tarefa != MAIN ):
48:             if ( proxima_tarefa == tarefa_atual ):
49:                proxima_tarefa = scheduler.escolha_outra()
50:       else if ( temperatura >= 68 and fator_clock[posicao_vetor_clock] > 4 ):
51:          fator_clock[posicao_vetor_clock] -= 1
```

Ao analisar o pseudocódigo é possível perceber que os pontos definidos pela árvore são encontrados nas linhas 17, 25, 34, 42 e 50. Nestas linhas consistem em:

- linha 17: análise da afirmação contida na árvore de decisão que informa que, para temperaturas menores que 67.5, a temperatura na próxima captura estará dentro do máximo para a faixa de consumo. O limite, porém foi puxado para baixo, uma vez que o leitor digital retorna valores inteiros;
- linha 25: utiliza do mesmo conceito mostrado na árvore de decisão, porém utiliza da análise manual executada para prevenir que tarefas com baixa prioridade e alta necessidade de processamento aja de modo a causar aumento no consumo (esta análise é importante, principalmente quando o sistema de Tempo-Real executa também tarefas de segunda ordem);
- linha 34: aplica o ramo da árvore que leva direto a uma temperatura que indica maior consumo. Ao estar em uma ramificação de *else* é garantido que a temperatura é maior ou igual a 67 (limite é 67.5), verificando assim a primeira condição. A segunda condição é definida caso o valor do evento situado no canal 3 da PMU, vulgo *pmu3*, esteja acima de 54, uma vez que o valor na árvore de decisão é 53.5. A terceira condição diz respeito ao

timestamp, neste caso com valor flexibilizado, trazendo a limite inferior para 4990 (na árvore 4996). A quarta, e última, condição é verificada pelo canal da PMU de número 4, com valor reduzido para 13000 através de análise manual, uma vez que ao verificar outras execuções foi percebido que o ponto de corte em relação ao aumento de temperatura é variável conforme a temperatura atual e também a quantidade de tempo em que o uso se mantém alto e suas variações são amplas, sendo que em reduções de uso o contador chega a valores abaixo de 9000, logo a redução era possível e aumentaria a robustez da heurística (aumentando a faixa de operação), além disso, é necessário um conjunto de 5 variáveis para ativar a heurística;

- linha 42: a fim de verificar a parte desconsiderada da árvore, foi analisado o contador que dá início em conjunto com o contador 4, indicado como maior correlação, efetuando assim uma simplificação na ramificação anteriormente descartada, com intuito de aumentar robustez, sem gerar grandes aumentos na complexidade;
- linha 50: trata a maior das afirmações da árvore, que considera que sempre que a temperatura alcançar seu limite superior, tende a ultrapassá-lo. Num primeiro momento, a verificação chegou a contar com a possibilidade de o sistema entrasse em *idle*, porém esta condição deixou de ser aferida com intuito de cobrir os casos em que a tarefa é só evitada (quando executa-se a ação de desvio de escalonamento) e assim que entrasse na idle, o reescalonamento aconteceria.

Um aspecto importante a se perceber é que foi identificado, com auxílio dos logs, que em temperaturas mais altas a variação de temperatura cai, porém, nesta faixa

trabalhada, esta oscilação demonstrou-se próxima à 3ºC, explicando, portanto, as condições aplicadas nas linhas 10, 13 e 25.

Para entendimento completo da heurística, é necessário, ainda, compreender mais duas regiões de código:

- linha 10: a partir do entendimento de que as variações tendem a ser de no máximo 3ºC e de que o limite térmico para o consumo desejado é de 68ºC, a verificação executada busca, primeiramente, apenas liberar as tarefas suspensas no momento em que não há risco da extrapolação do valor fixado. A segunda verificação feita ocorre devido ao fato de que ao identificar a conclusão das tarefas críticas (sabendo que as mesmas não voltarão a ser executadas) o sistema encerra atividades, finalizando a simulação. No entanto, para que tal fato ocorra, todas as tarefas devem ser finalizadas através de um join, sendo necessário que as threads sejam resumidas;
- linhas 26 a 32 e similares: esta região de código trata das verificações necessárias para se optar entre as alternativas de ação de suspensão controlada de tarefas, mais efetiva, e desvio de escalonamento, menos efetiva, porém necessária para evitar problemas na execução. Este código é repetido dentro de várias das verificações, uma vez que a redução de frequência é feita só em último caso.

Para facilitar a compreensão desta, que é a parte mais importante da heurística, os diagramas de fluxo de execução são apresentados nas Figuras 5.1 (b e c).

Figura 5.1 b) – Diagrama de fluxo de execução - Parte 1.

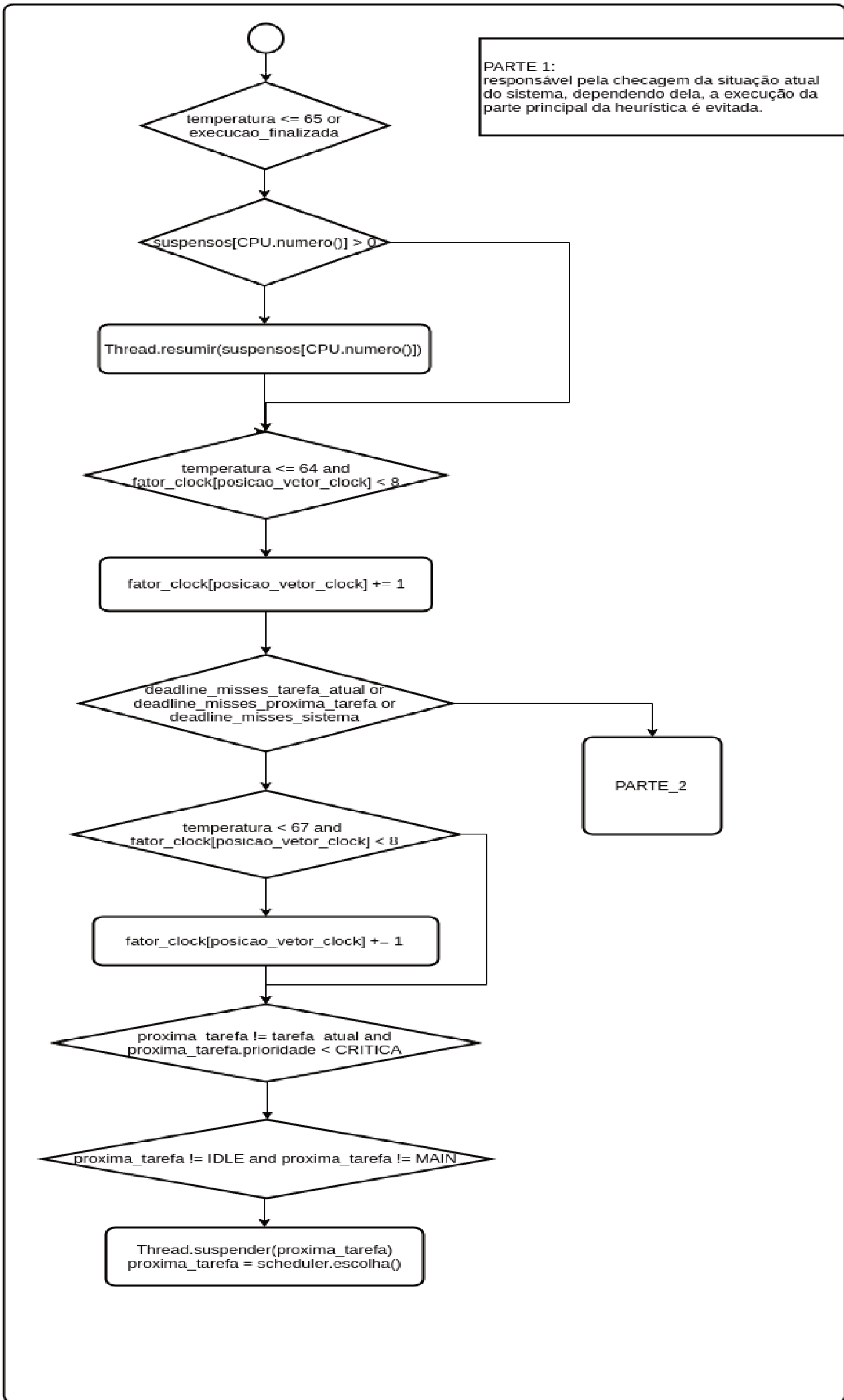
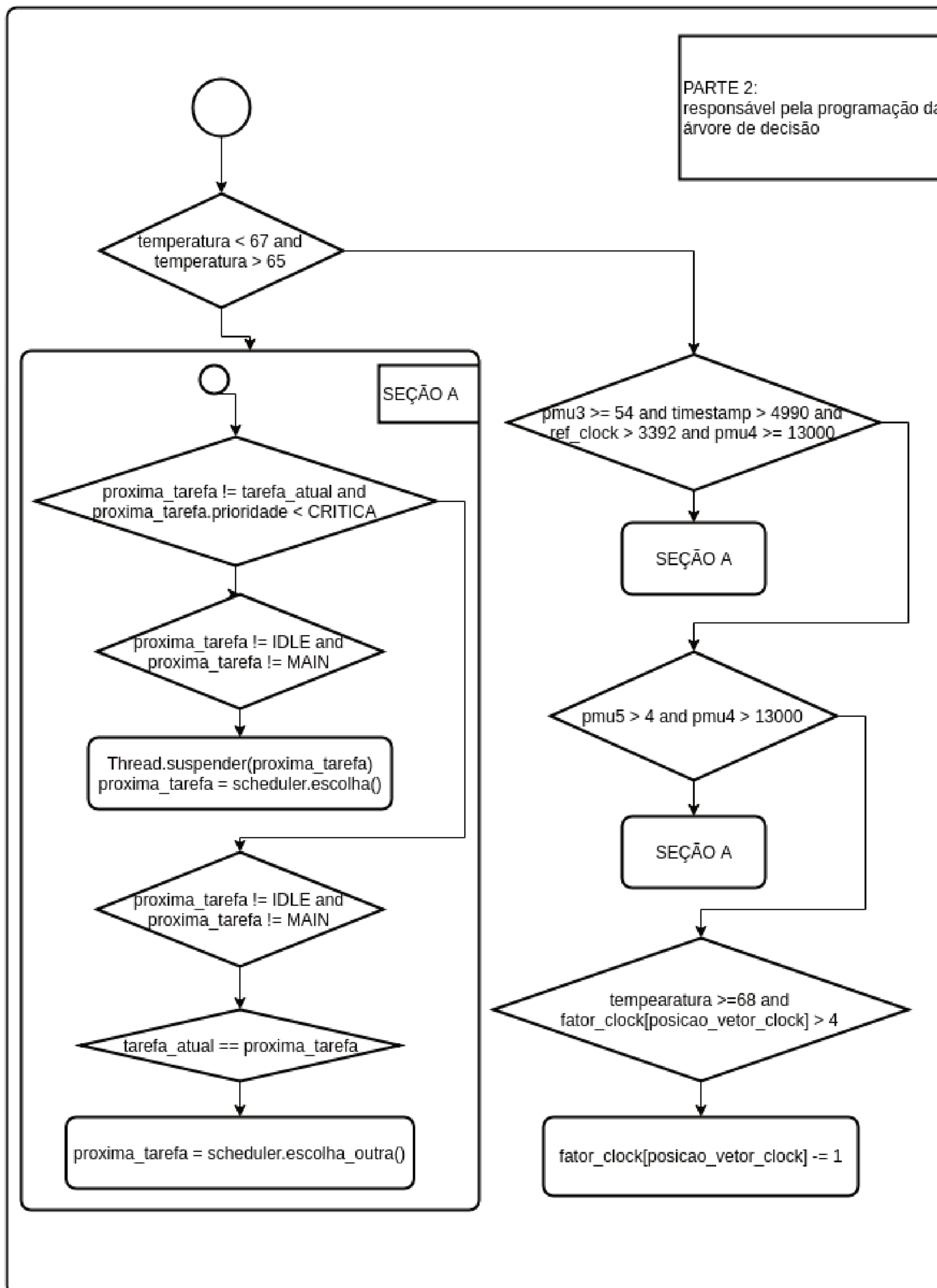


Figura 5.1 c) – Diagrama de fluxo de execução - Parte 2.



Com o entendimento do funcionamento da heurística dentro do dispatch, foi optado pela implementação de uma pequena melhoria, tratando o caso em que o processo de idle esteja sendo executado e que a temperatura já esteja em nível seguro, neste caso as threads suspensas devem ser resumidas e um novo escalonamento deve ser executado. A configuração desta opção pode ser encontrada na Figura 5.1 d.

Figura 5.1 d) – Pseudocódigo da heurística - parte idle.

```
// METODO IDLE
01: if ( suspensos[CPU.numero()] > 0 and temperatura < 67 ):
02:     Thread.resumir(suspensos[CPU.numero()])
```

A partir deste ponto, a heurística está configurada, sendo iniciada a etapa de testes e validação através da análise dos resultados obtidos.

6 RESULTADOS

A partir da configuração da heurística deu-se início aos testes para coleta de resultados, para tanto, porém, era necessário o uso de um conjunto de tarefas críticas (Tempo-Real) que não obtivesse uso maior que o limite para execução sob 80W de consumo médio e a partir dela, através da inserção de tarefas não críticas, aumentar o uso para aferir o consumo da heurística.

Doutro modo, a heurística, pelas configurações entre as linhas 16 e 23 da Figura 5.1 a, manteria o uso no nível necessário para suprir a necessidade do conjunto de tarefas a partir do momento em que fosse detectada a primeira perda de deadline. Esta ocorrência dificultaria a análise da implementação, uma vez que as tarefas tiveram uso mal dimensionado.

6.1. TASK-SETS UTILIZADOS

A partir dos task-sets e operações utilizados na fase de coleta de dados e apresentados anteriormente na seção 4.1 deste documento, foi executado uma pequena redução da carga do conjunto de tarefas de maior uso, foram criados novos conjuntos de operações na forma de tarefas de menor prioridade, executando o mesmo conjunto de operações, porém de maneira contínua (não periódica).

Estas tarefas teriam como função o preenchimento total do uso do processador, uma vez que por terem menor prioridade, seriam executadas nos tempos entre uma e outra execução das críticas. Desta maneira o processador jamais entraria em idle e seu uso seria sempre levado ao limite, porém, sem que se perdessem deadlines devido, justamente, as diferenças de prioridade. As tarefas foram distribuídas uma por CPU (partição do escalonador P-EDF).

Por questões de simulação as tarefas extras foram configuradas para executar continuamente a mesma operação até que as tarefas periódicas (consideradas críticas) finalizassem todas suas execuções. Em outras palavras, podemos descrever as tarefas como um laço de repetição das operações que tem como condição de parada o término das tarefas críticas projetadas inicialmente para geração de dados.

6.2. MÉTODO DE AFERIÇÃO ENERGÉTICA

O método para aferições de consumo utilizado consistiu, novamente, no uso do Fluke conectado à fonte do computador executando aferições a cada 0,25 segundos (250 milissegundos) durante todo o período de execução.

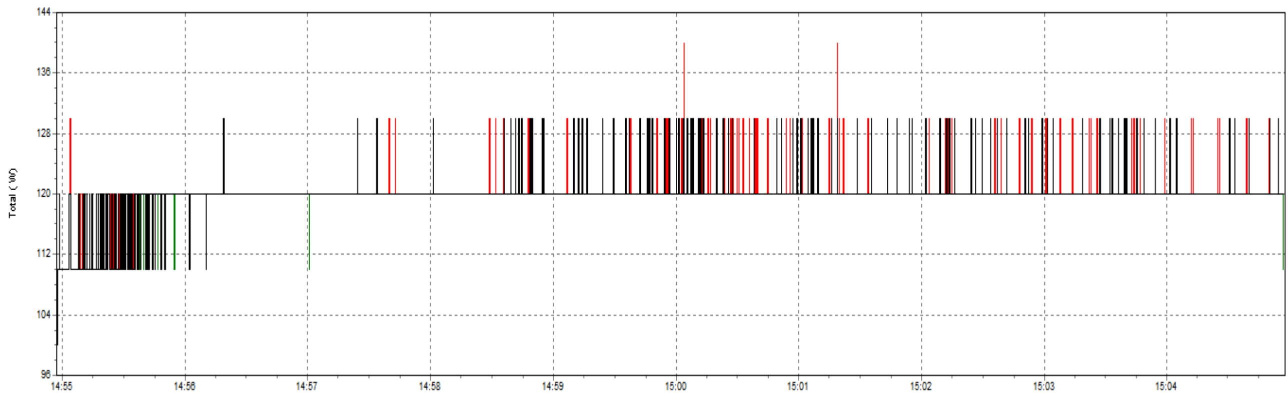
Para cálculo das médias e montagem dos histogramas foram removidos os períodos de inicialização da máquina (carregamento do sistema, momento em que não há execução das tarefas programadas, nem controle da heurística) e após término de execução, uma vez que ao executar o fluke o controle é manual e acaba por existir um intervalo de tempo após o sistema desligar até que as capturas do Fluke sejam interrompidas.

Os testes executados para medir desempenho cobriram três tipos de operações (leituras e escritas de memória, métodos interativos longos e cálculos explorando recursão) e tiveram, no geral, duração de 10 minutos.

6.3. DESEMPENHO COM HEURÍSTICA DESABILITADA

Para as tarefas de cunho recursivo, o desempenho pode ser acompanhado pelo histograma da Figura 6.1, com execução utilizando conjunto de tarefas 2, documentado na seção 4.1. Definição dos Task-Sets e das Operações Executadas deste documento.

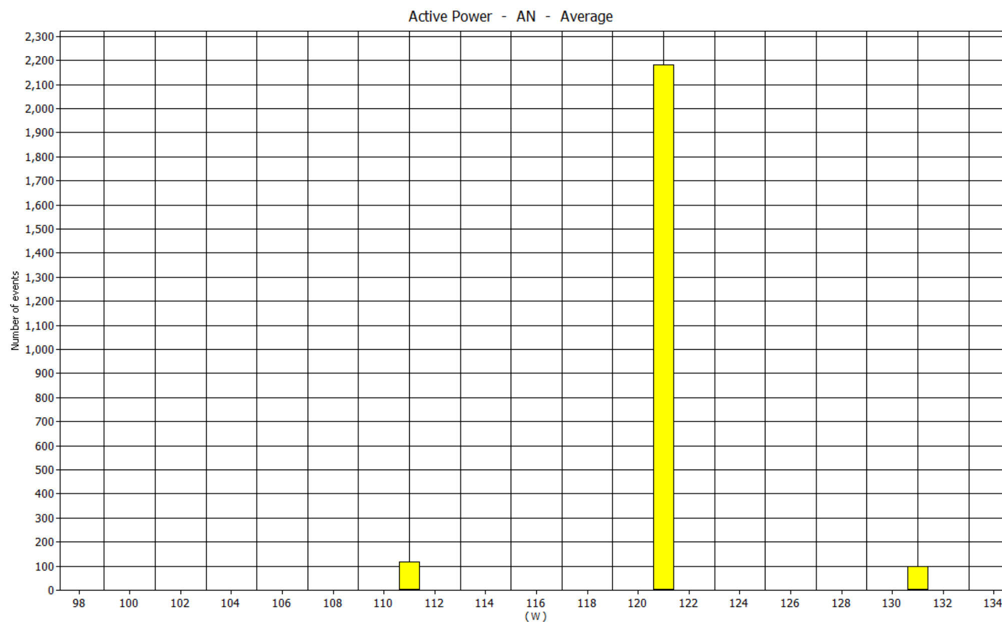
Figura 6.1 – Histograma de consumo - Operações recursivas.



Linhas para 96, 104, 112, 120, 128, 136 e 144W e colunas representando o tempo.

No gráfico de ocorrências, Figura 6.2, é possível reafirmar que o maior número de capturas foi com consumos superiores a 120W.

Figura 6.2 – Frequências de consumo - Operações recursivas.



Consumo no eixo das abscissas e Frequência no eixo das ordenadas.

É possível, ainda, retirar um sumário do consumo na execução (Figura 6.3), entregando dados como média dos valores aferidos (simbolizado pela letra M, no gráfico

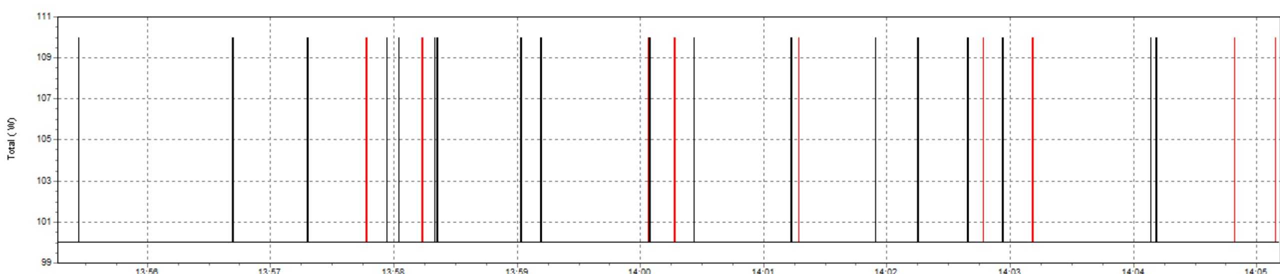
u), valores mínimos (*Minimum value*) e máximos (*Maximum value*), intervalo de tempo (campos *From* e *To*), desvio padrão (símbolo *s*), além de divisões por porcentagem da amostragem.

Figura 6.3 – Sumário de consumo - Operações recursivas.

From	10/4/2018 2:54:57 PM
To	10/4/2018 3:04:57 PM
Maximum value	130 W
At	10/4/2018 2:56:18 PM
Minimum value	100 W
At	10/4/2018 2:54:57 PM
μ	119.9 W
s	3.02726 W
5% percentile	110 W
95% percentile	120 W
% [85% - 110%]	0%
% [90% - 110%]	0 %

Para uma execução iterativa, podemos notar que o histograma (Figura 6.4) apresenta menores zonas de consumo, indicando ser uma tarefa que exige menor poder computacional.

Figura 6.4 – Histograma de consumo - Operações iterativas.



Linhas para 99, 101, 103, 105, 107, 109 e 111W e colunas representando o tempo.

Esta visão também fica clara nas frequências de aferições de consumo, disponível na Figura 6.5, indicando que o potencial de estresse da arquitetura é menor nos testes

com operações iterativas. Por fim, o sumário presente na Figura 6.6 mostra com mais clareza que a variação de consumo é menor do que na execução de operações recursivas.

Figura 6.5 – Frequências de consumo - Operações iterativas.

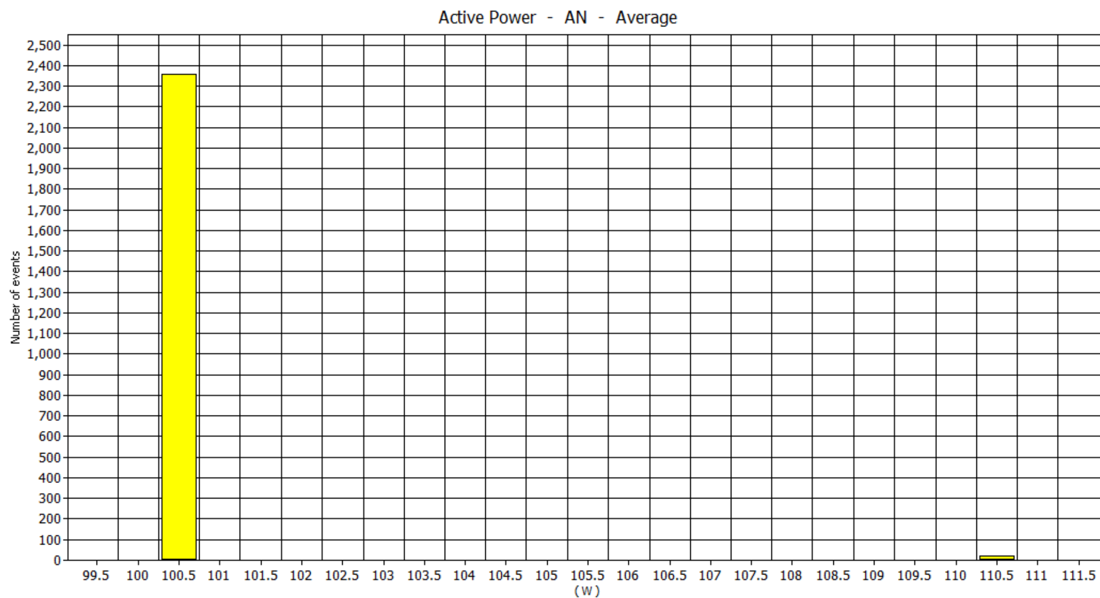


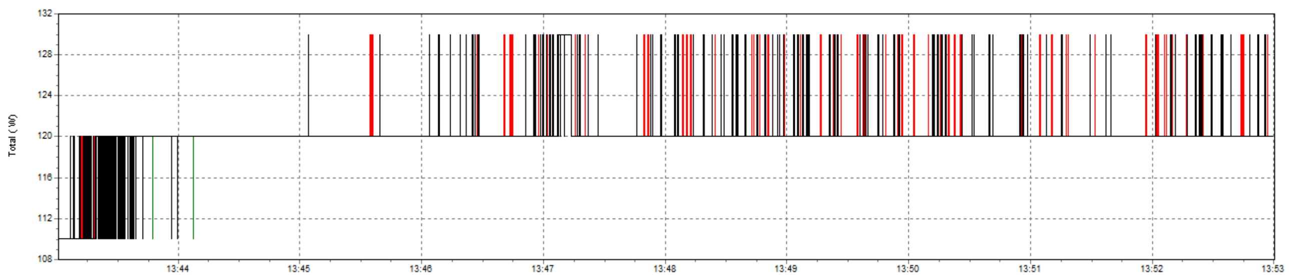
Figura 6.6 – Sumário de consumo - Operações iterativas.

From	10/22/2018 1:55:16 PM
To	10/22/2018 2:05:11 PM
Maximum value	110 W
At	10/22/2018 1:55:26 PM
Minimum value	100 W
At	10/22/2018 1:55:16 PM
μ	100.084 W
s	0.913411 W
5% percentile	100 W
95% percentile	100 W
% [85% - 110%]	0%
% [90% - 110%]	0 %

Na execução com operações de memória, temos um comportamento muito parecido com o visualizado na execução de operações recursivas. No histograma da

Figura 6.7 é possível verificar um período inicial mais baixo e um segundo período onde a execução estabiliza sua faixa térmica e de consumo nas variações de 120 a 130W.

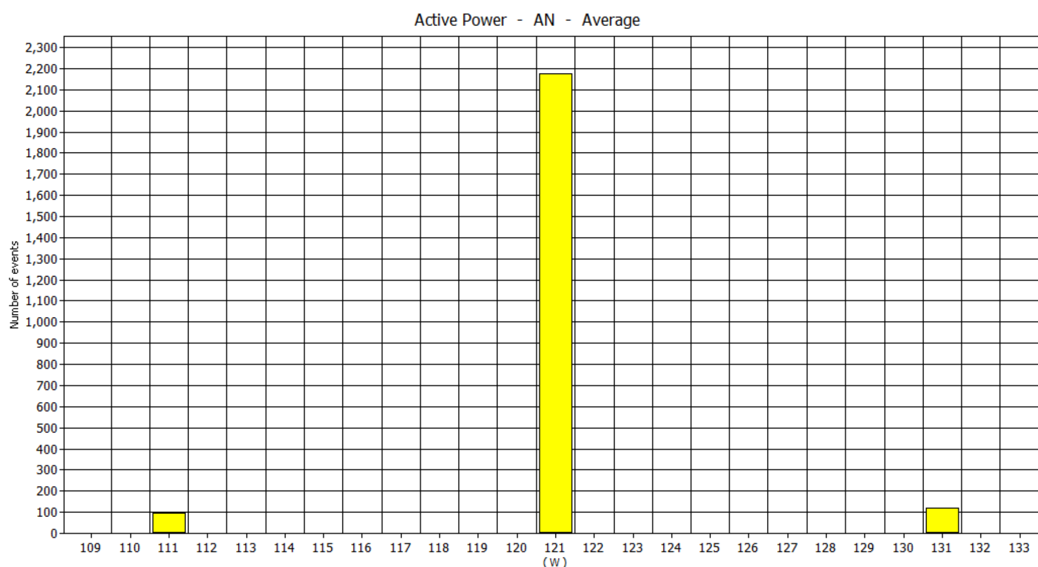
Figura 6.7 – Histograma de consumo - Operações em memória.



Linhas para 108, 112, 116, 120, 124, 128 e 132W e colunas representando o tempo.

Ao analisar o gráfico de frequências de aferições de consumo, presente na Figura 6.8, temos uma noção ainda mais clara da semelhança entre as execuções, com quantidades pequenas de aferições com 110W ou 130W e a grande maioria na faixa de 120W.

Figura 6.8 – Frequências de consumo - Operações em memória.



Por fim, o sumário, apresentado na Figura 6.9, mostra que, em síntese, as execuções foram extremamente semelhantes, apresentando uma divergência de apenas 0,2W na média das aferições e uma diferença de 0,01 no desvio padrão.

Figura 6.9 – Sumário de consumo - Operações em memória.

From	10/22/2018 1:43:01 PM
To	10/22/2018 1:52:59 PM
Maximum value	130 W
At	10/22/2018 1:45:04 PM
Minimum value	110 W
At	10/22/2018 1:43:01 PM
μ	120.109 W
s	3.01631 W
5% percentile	120 W
95% percentile	130 W
% [85% - 110%]	0%
% [90% - 110%]	0 %

Todos os gráficos e imagens apresentadas neste tópico são gerados automaticamente pelo Fluke, sendo assim, uma explicação para os dados neles contidos não será repetida.

6.4. DESEMPENHO COM HEURÌSTICA HABILITADA E COMPARATIVO

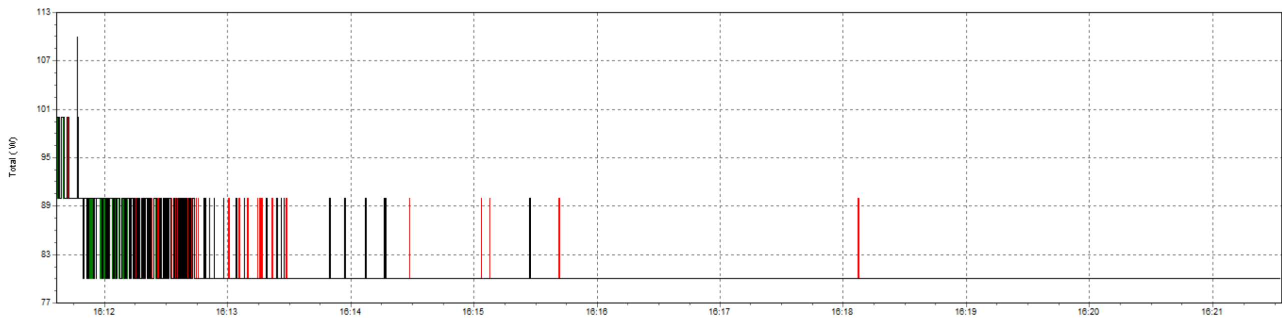
O comportamento apresentado pela heurística, demonstrada neste tópico, reflete novamente a dificuldade de se efetuar o controle de consumo, principalmente ao se contar o consumo de todo o sistema e não somente o do processador, em software.

De modo geral, a heurística não efetua cortes antes que a temperatura se aproxime do modelo expresso pelo TTV, assim sendo, antes que a heurística entre em ação, é necessário um período de aquecimento, que varia conforme a temperatura que o

processador se encontra (não necessariamente temperatura ambiente, uma vez que o processador pode vir de uma reinicialização).

A execução da heurística com conjunto de tarefas 2, usando operações recursivas apresenta este comportamento, conforme pode ser acompanhado na Figura 6.10.

Figura 6.10 – Histograma de consumo - operações recursivas com heurística.

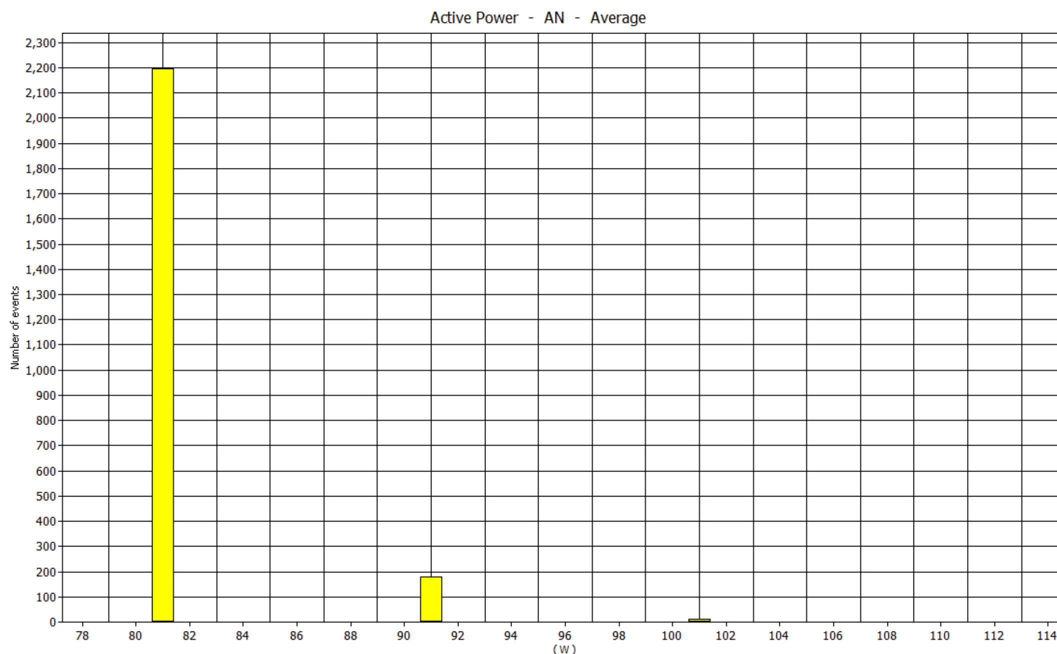


Linhas para 77, 83, 89, 95, 101, 107 e 113W e colunas representando o tempo.

Analisando o comportamento, é possível perceber que o comportamento se estabiliza, apresentando, apenas, alguns momentos de pico de consumo mais afastados, e o tempo para estabilização foi de aproximadamente 1 minuto e meio (90 segundos). A execução possui aproximadamente 10 minutos de duração.

A consistência pós estabilização e o funcionamento também são demonstrados pelos gráficos de frequência das faixas de consumo, apresentado na Figura 6.11.

Figura 6.11 – Frequências de consumo - operações recursivas com heurística.



Uma síntese maior dos resultados pode ser visualizada na Figura 6.12, onde é possível perceber que a média de consumo com uso da heurística parece tender a 80W conforme o tempo de execução aumenta. Nesta execução o consumo médio foi de 80.85W. Além disso, o desvio padrão também diminuiu em relação a execução original. Ainda, o consumo determinado foi alcançado pela primeira vez com 12 segundos de execução.

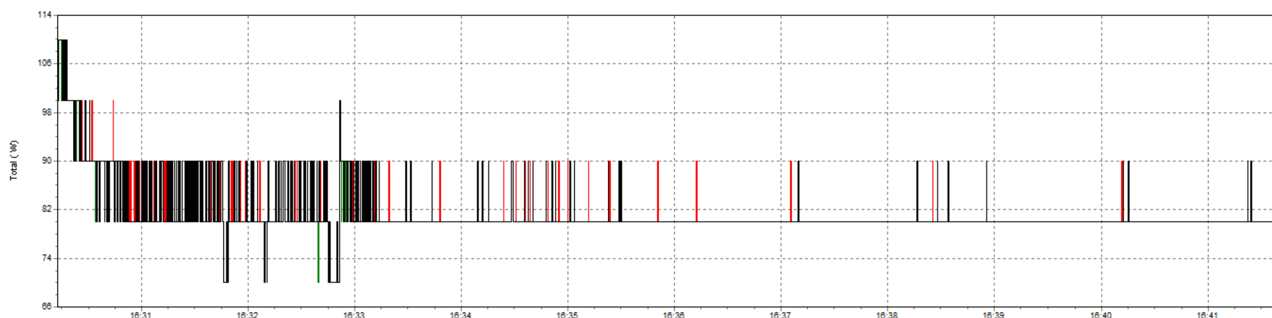
Figura 6.12 – Sumário de consumo - operações recursivas com heurística.

From	10/3/2018 4:11:37 PM
To	10/3/2018 4:21:33 PM
Maximum value	110 W
At	10/3/2018 4:11:46 PM
Minimum value	80 W
At	10/3/2018 4:11:49 PM
μ	80.8504 W
s	2.99298 W
5% percentile	80 W
95% percentile	90 W
% [85% - 110%]	0%
% [90% - 110%]	0 %

Conforme informado anteriormente neste tópico, o tempo de estabilização depende da temperatura do processador no início da execução, para demonstrar tal afirmação, serão apresentados os resultados de duas execuções com mesma configuração (conjunto de tarefas 2 executando com operações de leitura e escrita em memória), porém com diferentes condições térmicas do processador.

Primeiramente, visualizemos os resultados de uma execução com temperatura inicial mais baixa medida na CPU 4 em 55°C (Figura 6.13).

Figura 6.13 – Histograma de consumo - operações em memória com heurística.



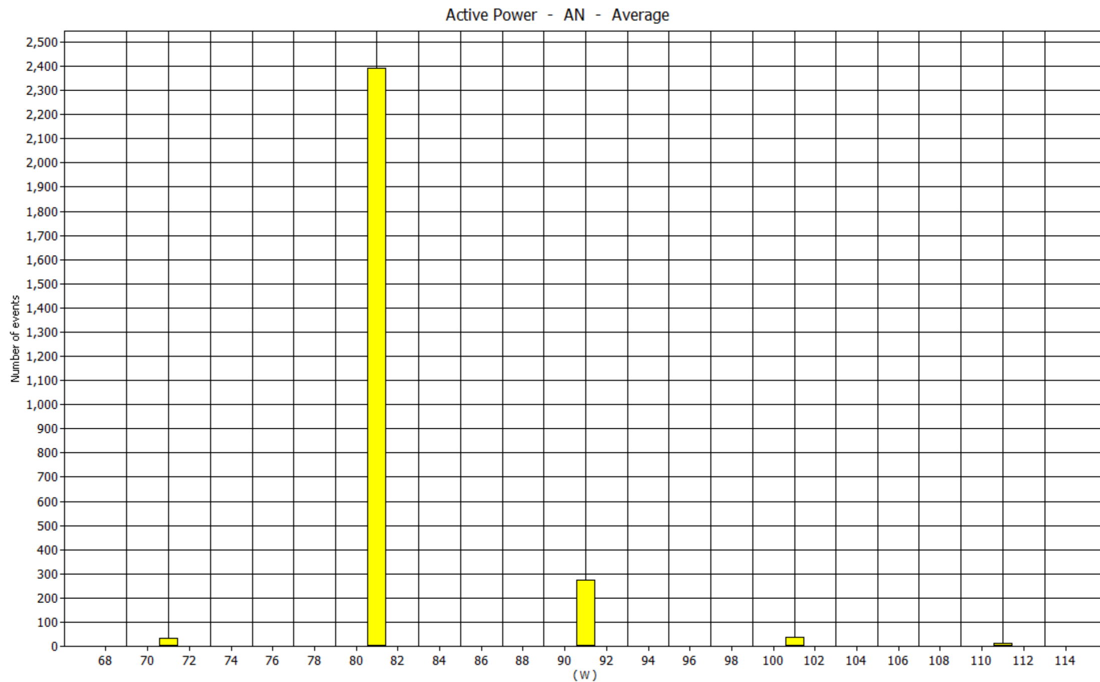
Linhas para 66, 74, 82, 90, 98, 106 e 114W e colunas representando o tempo.

Como a execução começou com uma temperatura mais elevada (execuções em temperatura ambiente e com longos períodos de intervalo de separação tem a primeira leitura abaixo dos 50°C), é possível visualizar que o consumo chegou a ser inferior ao ponto estabelecido antes da estabilização.

O gráfico de frequências de amostragem de consumo (Figura 6.14) mostra que o comportamento não difere muito do apresentado para a aplicação da heurística com execução recursiva, porém apresenta mais picos de consumo. O aumento no número de picos pode ser causado pelo menor aquecimento em operações de memória que em

operações matemáticas recursivas ou, até mesmo, pelo começo já em temperatura elevada que pode ter feito com que a frequência baixasse no princípio da execução.

Figura 6.14 – Frequências de consumo - operações em memória com heurística.



Em uma visão final, o sumário (Figura 6.15) confirma um consumo mais elevado, com média de 81.29W, e apresenta maior desvio padrão.

Figura 6.15 – Sumário de consumo - operações em memória com heurística.

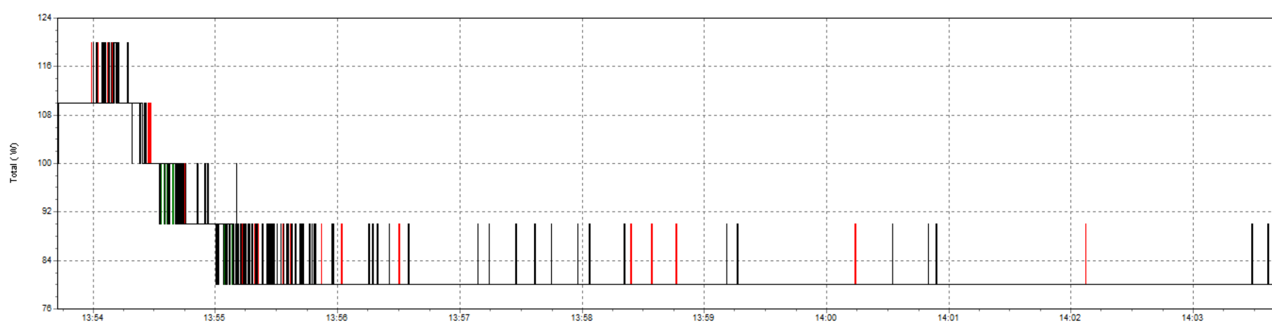
From	10/3/2018 4:30:13 PM
To	10/3/2018 4:41:41 PM
Maximum value	110 W
At	10/3/2018 4:30:13 PM
Minimum value	70 W
At	10/3/2018 4:31:46 PM
μ	81.2941 W
s	4.41457 W
5% percentile	80 W
95% percentile	90 W
% [85% - 110%]	0%
% [90% - 110%]	0 %

Embora não esboçado nas figuras apresentadas, o tempo para registro de 80W a partir do início da execução foi de 22 segundos.

Comparemos, então, com os resultados obtidos sob mesma configuração, porém com temperatura inicial mais baixa de 41°C, aferida na CPU 4.

Ao visualizar o histograma, Figura 6.16, é visível a diferença no tempo necessário para estabilização, foram pouco mais de um minuto para alcançar pela primeira vez o valor estipulado.

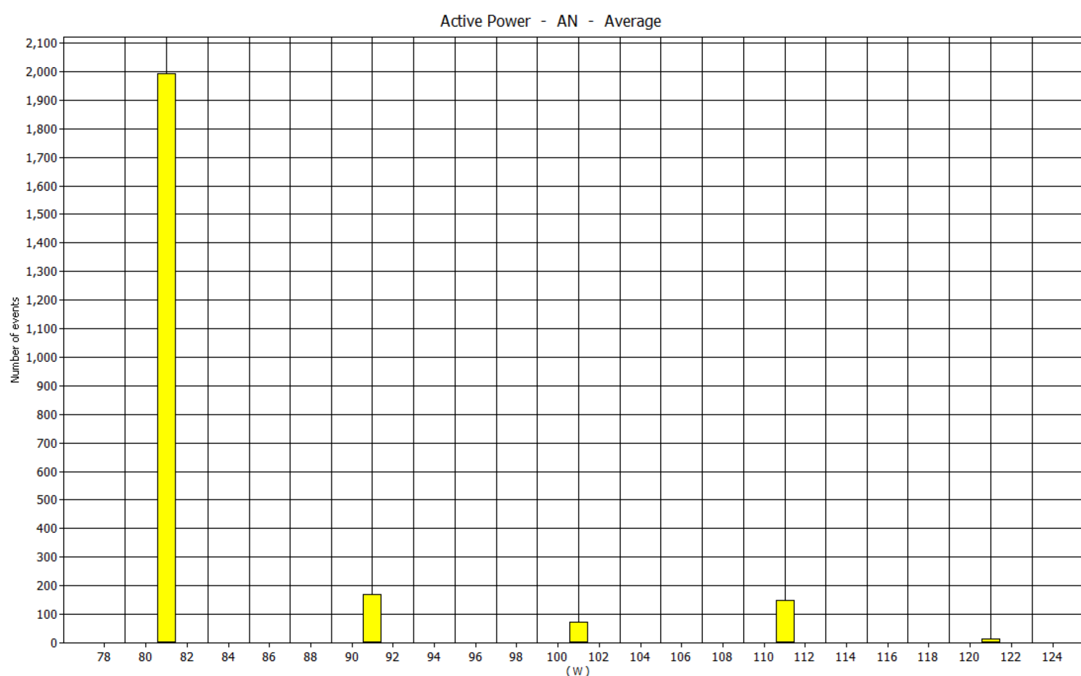
Figura 6.16 – Histograma de consumo - operações em memória com heurística.



Linhas para 76, 84, 92, 100, 108, 116 e 124W e colunas representando o tempo.

Analisando o gráfico de frequências, Figura 6.17, é possível verificar também a menor variabilidade, uma vez que são mais escassas as alterações de consumo.

Figura 6.17 – Frequências de consumo - operações em memória com heurística.



Um último dado a ser apresentado é o gráfico de síntese da execução, disponível na Figura 6.18.

Figura 6.18 – Sumário de consumo - operações em memória com heurística.

From	10/4/2018 1:53:43 PM
To	10/4/2018 2:03:42 PM
Maximum value	120 W
At	10/4/2018 1:54:00 PM
Minimum value	80 W
At	10/4/2018 1:55:00 PM
μ	83.4264 W
s	8.55816 W
5% percentile	80 W
95% percentile	110 W
% [85% - 110%]	0%
% [90% - 110%]	0 %

Analisando o sumário do consumo, visualizamos que a média do consumo é maior, isso se deve, em grande parte, pelo tempo necessário para estabilização da heurística. Entretanto, ao analisar cuidadosamente os sumários, vemos que a execução com

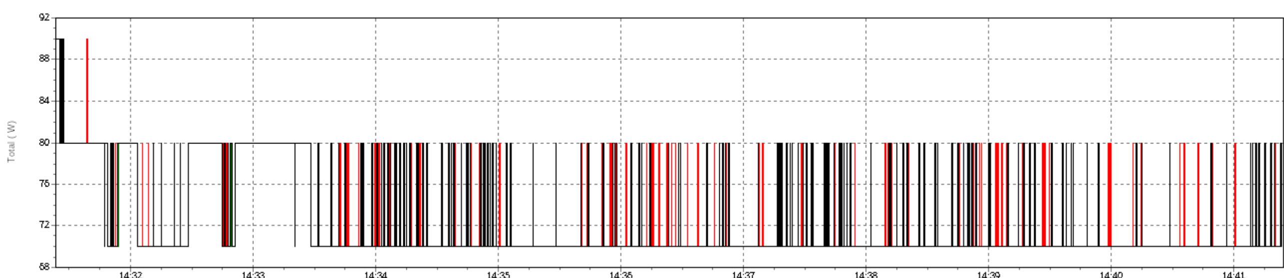
temperatura mais elevada tem duração maior, por mais que configurada para mesmo período de execução (10 minutos).

A diferença nos tempos de execução indica que para não extrapolar o consumo, a heurística fez com que houvesse deadline misses na primeira execução, isso ocorre pois existe um aquecimento mais alto no início de uma execução e a heurística não foi projetada para efetuar ações de emergência (quando o consumo indicado pela temperatura já começa muito acima do ideal) garantindo a execução em tempo das tasks.

Outro aspecto a se analisar, é que a redução de frequência demonstrou afetar mais as operações com leitura e escrita da memória que as operações matemáticas recursivas ou iterativas.

Por fim, podemos analisar o resultado da execução da heurística com operações iterativas, é possível verificar, conforme o histograma da Figura 6.19 que o funcionamento da heurística segue o mesmo padrão, porém, como a execução das operações iterativas delibera menos processamento, ao suspender as tarefas que estressam a arquitetura, o nível de consumo fica abaixo dos 80W.

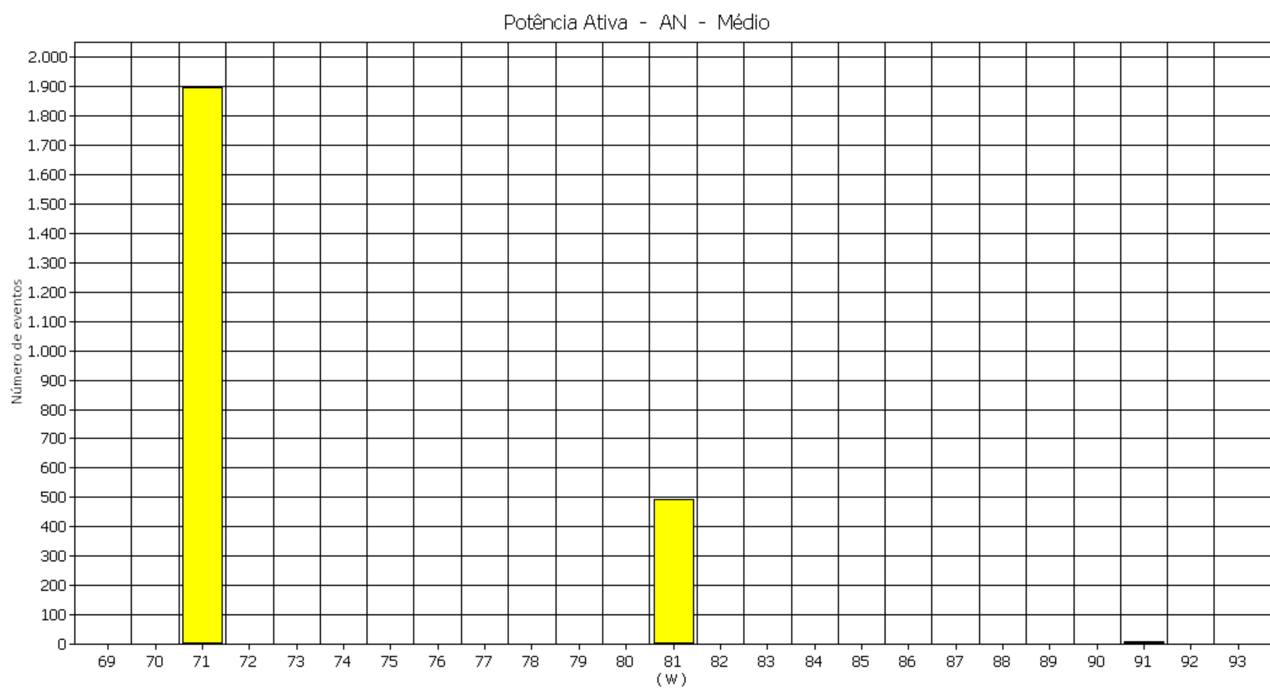
Figura 6.19 – Histograma de consumo - operações iterativas com heurística.



Linhas para 68, 72, 76, 80, 84, 88 e 92W e colunas representando o tempo.

Esta observação é mais visível no gráfico de frequências de aferições, Figura 6.20.

Figura 6.20 – Frequências de consumo - operações iterativas com heurística.



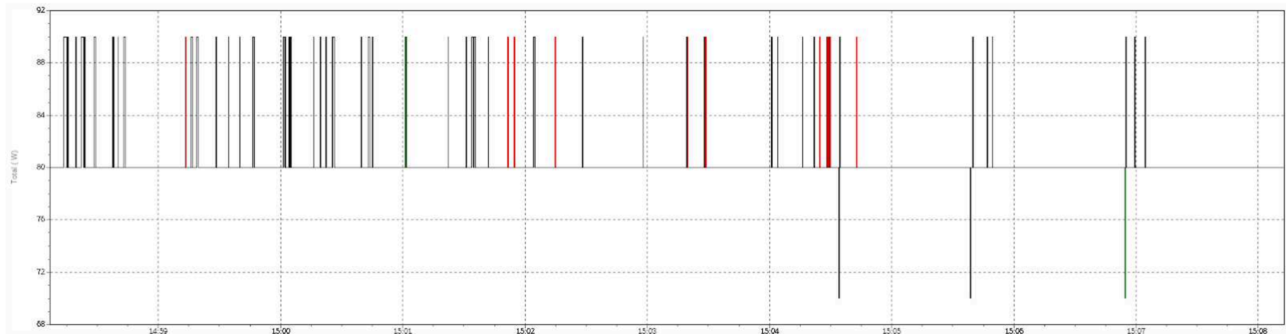
Por fim, a conclusão é reafirmada no sumário da execução, Figura 6.21.

Figura 6.21 – Sumário de consumo - operações iterativas com heurística.

De	23/10/2018 14:31:24
Para	23/10/2018 14:41:23
Valor máximo	90 W
Em	23/10/2018 14:31:24
Valor mínimo	70 W
Em	23/10/2018 14:31:47
µ	72,1301 W
s	4,18581 W
5% percentil	70 W
95% percentil	80 W
% [85% - 110%]	0%
% [90% - 110%]	0 %

Uma última análise pode ser feita usando o conjunto de tarefas 3, cujo uso é um pouco menor, em uma execução pouco maior, foram 10 minutos de execução após estabilização. A Figura 6.22 apresenta o histograma dos últimos 10 minutos de execução.

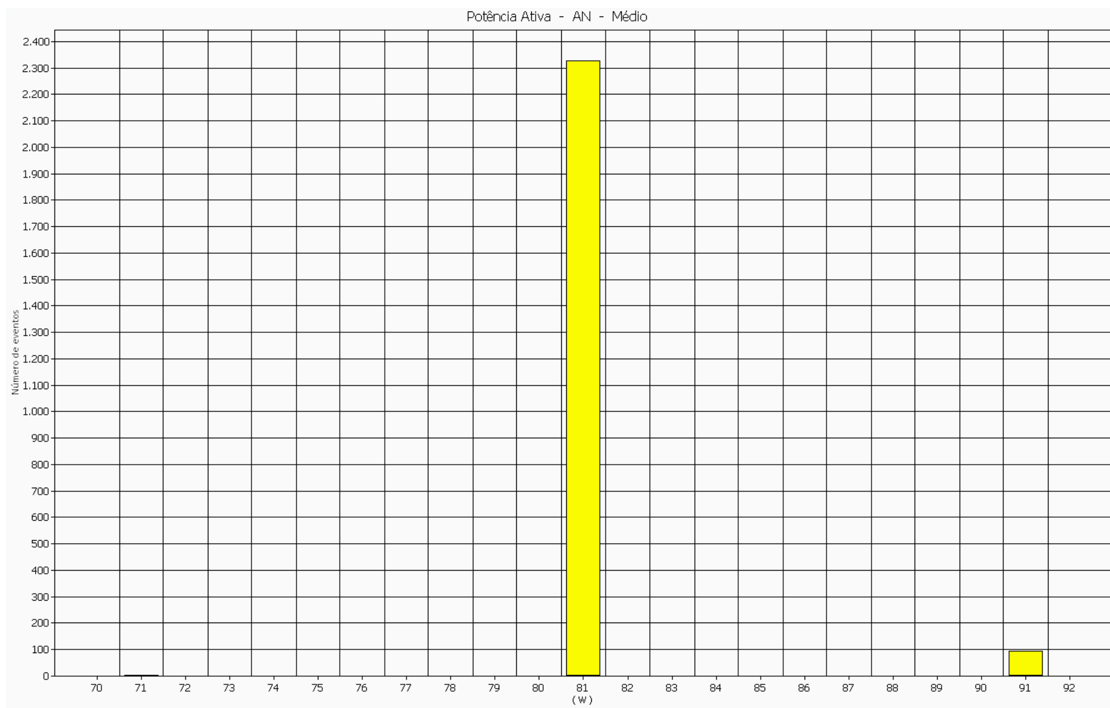
Figura 6.22 – Histograma de consumo - operações recursivas com heurística.



Linhas para 68, 72, 76, 80, 84, 88 e 92W e colunas representando o tempo.

A constância após estabilização pode ser observada também nas frequências de registros de cada faixa de consumo, Figura 6.23.

Figura 6.23 – Frequências de consumo - operações recursivas com heurística.



A média de consumo e o desvio padrão durante a execução podem ser visualizados na Figura 6.24.

Figura 6.24 – Sumário de consumo - operações recursivas com heurística.

De	23/10/2018 14:58:07
Para	23/10/2018 15:08:12
Valor máximo	90 W
Em	23/10/2018 14:58:13
Valor mínimo	70 W
Em	23/10/2018 15:04:34
µ	80,3757 W
s	1,94494 W
5% percentil	80 W
95% percentil	80 W
% [85% - 110%]	0%
% [90% - 110%]	0 %

Novamente, vale lembrar que o computador por vezes vinha de uma execução anterior e, mesmo que se esperasse um intervalo de tempo, a condição térmica não era idêntica a primeira inicialização do mesmo.

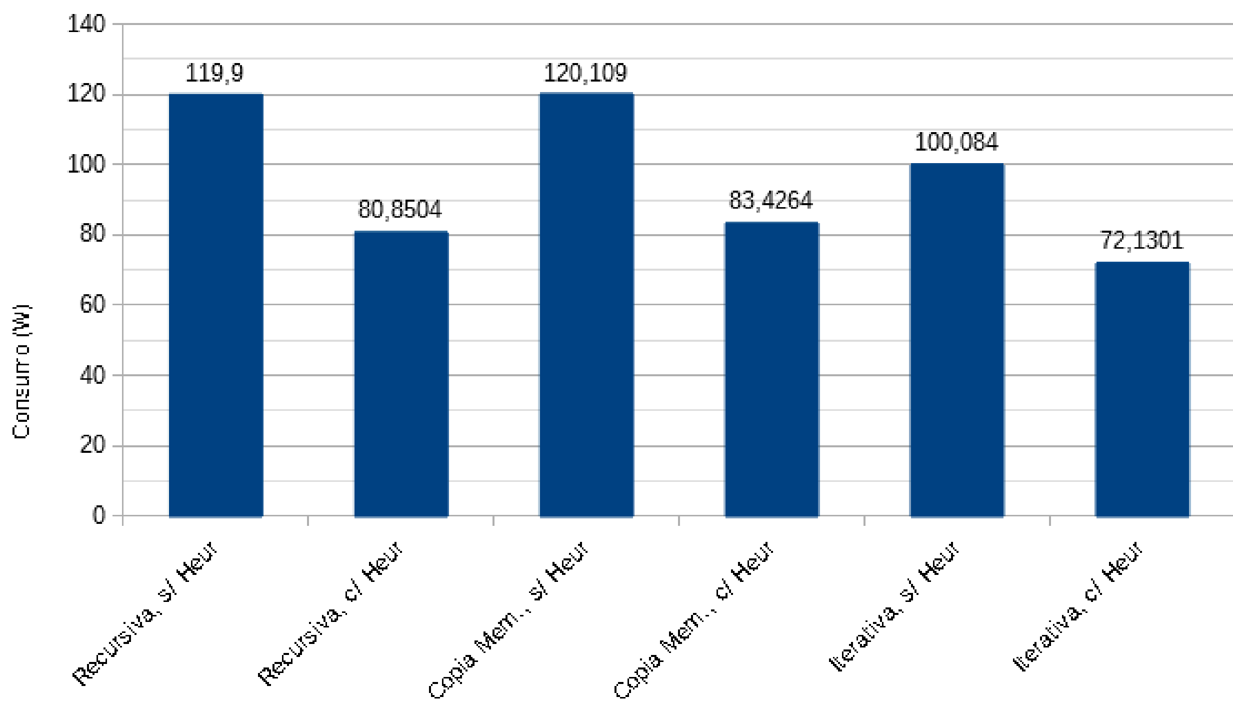
Para concluir a análise dos resultados obtidos, alguns aspectos sobre o funcionamento da heurística devem ser fixados:

- O tempo de convergência da heurística é grande, porém ao levar em consideração uma execução contínua, não necessariamente isso seria encarado como um problema;
- Se a temperatura inicial da execução for muito alta, ou seja, a “cama” do processador já estiver quente, o tempo de convergência será acelerado, porém, cuidados devem ser tomados, pois se a execução começa em temperaturas superiores, a heurística tomará ações mais abruptas, prejudicando, assim, a execução das tarefas críticas e o desempenho do sistema;
- A heurística não suspende tarefas críticas e tem como última alternativa de ação a redução de frequência, logo, perdas de deadline serão evitadas, porém, se as tarefas projetadas não forem capazes de executar sobre as limitações energéticas impostas, o desempenho da heurística será levemente atrapalhado e a performance do sistema será amplamente afetado. O ideal é que as tarefas sejam projetadas com determinada folga;
- A heurística foi gerada para 80W de limitação, para seu funcionamento com outras faixas de consumo, o ideal seria a geração de nova árvore de decisões, o que poderia ser gerado com o mesmo processo. A simples alteração de pontos de temperatura não seria válida, pois o poder

computacional seria alterado, logo o nível de “esforço” expressado pelos contadores mudaria conforme a faixa de atuação.

Além disso, um comparativo dos consumos médios, com e sem heurística ativa, utilizando o conjunto de tarefas 2 como tarefas críticas e com uma tarefa extra, de menor prioridade, criada por cpu (para forçar o aumento de consumo e de aquecimento), é apresentado abaixo, na Figura 6.25.

Figura 6.25 – Consumos Médios Sob Conjunto de Tarefas 2



7 CONCLUSÃO

Este Trabalho de Conclusão de Curso focou-se na implementação de um sistema não intrusivo de monitoramento de performance para o sistema operacional EPOS, adicionando a plataforma o suporte para leitura de sensores térmicos e energéticos e para modulação de clock via duty cycle, com auxílio do também graduando José Luis Conradi Hoffmann.

Com o auxílio do sistema desenvolvido, o presente trabalho aprofundou-se na análise dos eventos e contadores, desenvolvendo, através da documentação do processador fornecida pela Intel (fabricante), uma heurística para Power-Cap capaz de controlar não apenas o processador, mas o consumo de toda a máquina.

O sistema de monitoramento de performance pode ser amplamente utilizado para desenvolvimentos futuros, com ou sem o mesmo enfoque e até mesmo utilizando outras plataformas. A heurística desenvolvida pode ser utilizada como base para o desenvolvimento de novas heurísticas para diferentes computadores e o processo de seu desenvolvimento pode embasar a criação de modelos capazes de se adaptar melhor ao mundo dos sistemas embarcados.

7.1. TRABALHOS FUTUROS

Alguns trabalhos e estudos foram pensados a partir do desenvolvido, dentre eles:

- Utilizando-se do sistema de monitoramento de performance, efetuar novos estudos sobre uma plataforma Multi-core de Tempo-real, a fim de descobrir aspectos de seu funcionamento ainda não explorados;

- Implementar o uso de PEBS, recurso da PMU capaz de causar interrupções ao chegar num determinado valor, para aprimorar os resultados da heurística;
- Utilizar a capacidade de coleta de dados para desenvolver e aplicar uma heurística capaz de trabalhar com o Power-Cap diretamente do processador, adaptando o trabalho ao cenário embarcado;
- Buscar alternativas de implementação da heurística mesclando os controles já desenvolvidos a controladores de consumo de componentes específicos através da interface RAPL, por exemplo.

BIBLIOGRAFIA

[1] **EMBEDDED PARALLEL OPERATING SYSTEM**, SOFTWARE/HARDWARE INTEGRATION LAB. Disponível em: <https://epos.lisha.ufsc.br>. Acesso em: agosto de 2017.

[2] **SOFTWARE/HARDWARE INTEGRATION LAB**. Disponível em: <https://lisha.ufsc.br>. Acesso em: agosto de 2017.

[3] **EPOS 2 USER GUIDE**, EPOS. Disponível em: <https://epos.lisha.ufsc.br/EPOS+2+User+Guide>. Acesso em: agosto de 2017.

[4] **Intel® 64 and IA-32 Architectures Software Developer's Manual**, Disponível em: <https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf>. Acesso: setembro de 2017.

[5] INTERNET OF THINGS AT UFSC. Disponível em: <http://iot.lisha.ufsc.br>. Acesso em: dezembro de 2017.

[6] IOT WITH EPOS, EPOS. Disponível em: <http://epos.lisha.ufsc.br/IoT+with+EPOS>. Acesso em: janeiro de 2018.

[7] HOFFMANN, J. L. C.; HORSTMANN, L. P. "**Performance Monitoring with EPOS**". Disponível em: <http://epos.lisha.ufsc.br/Performance+Monitoring+with+EPOS>. Acesso em: maio 2018.

[8] HOFFMANN, J. L. C.; HORSTMANN, L. P., "**Adaptative DVFS for EPOS Multicore Schedulers**". Disponível em:

<https://epos.lisha.ufsc.br/Adaptive+DVFS+for+EPOS+Multicore+Schedulers>. Acesso em: dezembro de 2017.

[9] KUMAR, V. “**Real-Time Scheduling Algorithms”**. Disponível em: <https://pdfs.semanticscholar.org/549a/2bff5d595e759156fb2cb9bad14d5a2fc892.pdf>. Acesso em: setembro, 2017.

[10] VORA, V.; SOMKUWAR, A. “**Implementation & Performance Analysis of Real Time Scheduling Algorithms for Three Industrial Embedded Applications (IEA)**”. International Journal of Information Technology Convergence and Services (IJITCS) Vol.2, No.6, December 2012.

[11] SULEIMAN, D.; IBRAHIM, M.; HAMARASH, I. “**DYNAMIC VOLTAGE FREQUENCY SCALING (DVFS) FOR MICROPROCESSORS POWER AND ENERGY REDUCTION”**, Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=E6F55A8CE6B176124DC60E15141B65B5?doi=10.1.1.111.1451&rep=rep1&type=pdf>. Acesso em: setembro de 2017.

[12] PILLAI, P.; SHIN, K. G. “**Real-time dynamic voltage scaling for low-power embedded operating systems”**, Disponível em: <http://sosp.org/2001/papers/pillai.pdf>. Acesso em: 15/06/2018.

[13] TANENBAUM, A. S.; BOS, H., “**MODERN OPERATING SYSTEMS”**, Quarta edição, Pearson Education, 2015, Inc., Upper Saddle River, New Jersey, Capítulo 1, Seção 1.3.

[14] MÜCK, T.; SARMA, S.; Dutt, N.; “**Run-DMC: runtime dynamic heterogeneous multicore performance and power estimation for energy efficiency”**, Amsterdam, The Netherlands, IEEE, 2015. Disponível em:

<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7331380>. Acesso em: 29/09/2017.

[15] KANTARDZIC,M. “**Data Mining: Concepts, Models, Methods, and Algorithms”**, Second Edition. 2011. Institute of Electrical and Electronics Engineers. John Wiley & Sons, Inc. Capítulo 1. Disponível em: <https://ieeexplore.ieee.org/xpl/ebooks/bookPdfWithBanner.jsp?fileName=6105629.pdf&bkn=6105606&pdfType=chapter> Acesso em: 18/06/2018

[16] ISLAM, F. M. M. ul; LIN, M. “**Hybrid DVFS Scheduling for Real-Time Systems Based on Reinforcement Learning**”, IEEE SYSTEMS JOURNAL, VOL. 11, NO. 2, JUNE 2017, pg 931 a 940. Disponível em: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7163527>. Acesso em: fevereiro de 2018.

[17] DONYANAVARD, B.; MÜCK, T.; SARMA, S.; DUTT, N. “**SPARTA: Runtime Task Allocation for Energy Efficient Heterogeneous Many-Cores**”. International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2016. IEEE, Pittsburgh, PA, USA. Disponível em: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7750975>. Acesso em: janeiro de 2018.

[18] HARELICK, M.; STOYEN, A. “**CONCEPTS FROM DEADLINE NON-INTRUSIVE MONITORING**”. IFAC Real Time Programming, Schloss Dagstuhl, Germany, 1999. Disponível em: <https://ac.els-cdn.com/S1474667017399640/1-s2.0-S1474667017399640-main.pdf?_tid=

5f258a71-5c3b-4d3f-a754-150e909ef7d4&acdnat=1529944867_773d47342e7cf78f049df14045ed2cde>. Acesso em: junho de 2018.

[19] **Intel Core i7-2600k Processor**. Disponível em: <https://ark.intel.com/products/52214/Intel-Core-i7-2600-Processor-8M-Cache-up-to-3_80-GHz>. Acesso em: maio de 2018.

[20] **Intel® Core™ i7-2000 and i5-2000 Desktop Processor Series (Quad Core 95W) Thermal Profile**. Disponível em: <https://www.intel.com/content/www/us/en/processors/core/2nd-gen-core-lga1155-socket-guide.html> Acesso em: junho de 2018.

[21] ZHANG, Z.; LANG, M.; PAKIN, S.; FU, S. **Trapped capacity: scheduling under a power cap to maximize machine-room throughput**. E2SC'14 Proceedings of the 2nd International Workshop on Energy Efficient Supercomputing. Pág. 41-50. 2014. Disponível em: <https://dl.acm.org/citation.cfm?id=2689715> Acesso em: Setembro de 2018.

[22] ZHANG, H; HOFFMANN, H. **Maximizing Performance Under a Power Cap: A Comparison of Hardware, Software, and Hybrid Techniques**. ASPLOS '16 Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems. Pág. 545-559. 2016. Disponível em: <https://dl.acm.org/citation.cfm?id=2872375> Acesso em: Setembro de 2018.

[23] DIMITROV, M.; STRICKLAND, C.; KIM, S.; KUMAR, K.; DOSHI, K. **"Intel® Power Governor"**, Julho - 2012. Disponível em: <https://software.intel.com/en-us/articles/intel-power-governor >. Acesso em: Julho de 2018.

[24] **“Analisador de energia e potência Fluke 435 série II”,** FLUKE CORPORATION, 2018. Disponível em: <https://www.fluke.com/pt-br/produto/teste-eletrico/os-analisadores-de-qualidade-de-energia/analisadores-da-qualidade-da-energia-trifasica/fluke-435-series-ii>. Acesso em: Julho de 2018.

[25] **“Weka - Waikato Environment for Knowledge Analysis”**. Machine Learning at Waikato University. Disponível em: <https://www.cs.waikato.ac.nz/ml/index.html>. Acesso em: Fevereiro de 2018.

[26] HAN, J.; KAMBER, M.; PEI, J. **Data Mining Concepts and Techniques: 3. ed.** Massachusetts: Morgan Kaufmann - Elsevier. 2012.

[27] GRACIOLI, G. **Real-Time Operating System Support For Multicore Applications.** 2014. Disponivel em: <http://www.lisha.ufsc.br/pub/Gracioli_PHD_2014.pdf>. Acesso em: Outubro de 2018.

ANEXOS

A1. TABELA DE EVENTOS INTEL SANDY BRIDGE NO EPOS

Nome	Nome No EPOS	Nome na Documentação INTEL	Descrição Intel
pmu0	(FIXED) INSTRUCTION_RETIRED	Instructions Retired	Counts when the last uop of an instruction retires.
pmu1	(FIXED) CLOCK	UnHalted Core Cycles	Counts core clock cycles whenever the logical processor is in C0 state (not halted). The frequency of this event varies with state transitions in the core.
pmu2	(FIXED) DVS_CLOCK	UnHalted Reference Cycles1	Counts at a fixed frequency whenever the logical processor is in C0 state (not halted).
pmu3	BRANCH	Branch Instruction Retired	Counts when the last uop of a branch instruction retires.
pmu4	BRANCH_MISS	Branch Misses Retired	Counts when the last uop of a branch instruction retires which corrected misprediction of the branch prediction hardware at execution time.
pmu5	L1_HIT	MEM_LOAD_RETIRED.L1_HIT	Hit accesses to the L1 Cache
pmu6	L2_HIT	MEM_LOAD_RETIRED.L2_HIT	Hit accesses to the L2 Cache
pmu7	L3_HIT / LLC_HIT	MEM_LOAD_RETIRED.L3_HIT	Hit accesses to the L3 Cache
pmu8	L1_MISS	MEM_LOAD_RETIRED.L1_MISS	MISS accesses to the L1 Cache
pmu9	L2_MISS	MEM_LOAD_RETIRED.L2_MISS	MISS accesses to the L2 Cache
pmu10	L3_MISS	MEM_LOAD_RETIRED.L3_MISS	MISS accesses to the L3 Cache
pmu11	LLC_HITM	Hit accesses to the L3 Cache	Hit LLC "Modified"
pmu12	LD_BLOCKS_DATA_UNKNOWN	LD_BLOCKS.DATA_UNKNOWN	Blocked loads due to store buffer blocks with unknown data.
pmu13	LD_BLOCKS_STORE_FORWARD	LD_BLOCKS.STORE_FORWARD	Loads blocked by overlapping with store buffer that cannot be forwarded.

pmu14	LD_BLOCKS_NO_SR	LD_BLOCKS.NO_SR	# of Split loads blocked due to resource not Available
pmu15	LD_BLOCKS_ALL_BLOCK	LD_BLOCKS.ALL_BLOCK	Number of cases where any load is blocked but has no DCU miss.
pmu16	MISALIGN_MEM_REF_LOADS	MISALIGN_MEM_REF.LOADS	Speculative cache-line split load uops dispatched to L1D.
pmu17	MISALIGN_MEM_REF_STORES	MISALIGN_MEM_REF.STORES	Speculative cache-line split Store-address uops dispatched to L1D.
pmu18	LD_BLOCKS_PARTIAL_ADDRESS_ALIAS	LD_BLOCKS_PARTIAL.ADDRESS_ALIAS	False dependencies in MOB due to partial compare on address.
pmu19	LD_BLOCKS_PARTIAL_ALL_STA_BLCOK	LD_BLOCKS_PARTIAL.ALL_STA_BLOCK	The number of times that load operations are temporarily blocked because of older stores, with addresses that are not yet known. A load operation may incur more than one block of this type.
pmu20	DTLB_LOAD_MISSES_MISS_CAUSES_A_WALK	DTLB_LOAD_MISSES.MISS_CAUSES_A_WALK	Misses in all TLB levels that cause a page walk of any page size.
pmu21	DTLB_LOAD_MISSES_MISS_WALK_COMPLETED	DTLB_LOAD_MISSES.WALK_COMPLETED	Misses in all TLB levels that caused page walk completed of any size.
pmu22	DTLB_LOAD_MISSES_MISS_WALK_DURATION	DTLB_LOAD_MISSES.WALK_DURATION	Cycle PMH is busy with a walk.
pmu23	DTLB_LOAD_MISSES_MISS_STLB_HIT	DTLB_LOAD_MISSES.STLB_HIT	Number of cache load STLB hits. No page walk.
pmu24	INT_MISC_RECOVERY_CYCLES	INT_MISC.RECOVERY_CYCLES	Cycles waiting to recover after Machine Clears or JEClear. Set Cmask= 1.
pmu25	INT_MISC_RAT_STALL_CYCLES	INT_MISC.RAT_STALL_CYCLES	Cycles RAT external stall is sent to IDQ for this thread.
pmu26	UOPS_ISSUED_ANY	UOPS_ISSUED.ANY	Increments each cycle the # of Uops issued by the RAT to RS. Set Cmask = 1, Inv = 1, Any= 1to count stalled cycles of this core.

pmu27	FP_COMP_OPS_EXE_X87	FP_COMP_OPS_EXE.X87	Counts number of X87 uops executed
pmu28	FP_COMP_OPS_EXE_SSE_FP_PACKED_DOUBLE	FP_COMP_OPS_EXE.SSE_FP_PACKED_DOUBLE	Counts number of SSE* double precision FP packed uops executed.
pmu29	FP_COMP_OPS_EXE_SSE_FP_SCALAR_SINGLE	FP_COMP_OPS_EXE.SSE_FP_SCALAR_SINGLE	Counts number of SSE* single precision FP scalar uops executed.
pmu30	FP_COMP_OPS_EXE_SSE_PACKED_SINGLE	FP_COMP_OPS_EXE.SSE_PACKED SINGLE	Counts number of SSE* single precision FP packed uops executed.
pmu31	FP_COMP_OPS_EXE_SSE_SCALAR_DOUBLE	FP_COMP_OPS_EXE.SSE_SCALAR_DOUBLE	Counts number of SSE* double precision FP scalar uops executed.
pmu32	SIMD_FP_256_PACKED_SINGLE	SIMD_FP_256.PACKED_SINGLE	Counts 256-bit packed single-precision floatingpoint Instructions.
pmu33	SIMD_FP_256_PACKED_DOUBLE	SIMD_FP_256.PACKED_DOUBLE	Counts 256-bit packed double-precision floatingpoint Instructions.
pmu34	ARITH_FPU_DIV_ACTIVE	ARITH.FPU_DIV_ACTIVE	Cycles that the divider is active, includes INT and FP. Set 'edge =1, cmask=1' to count the number of Divides.
pmu35	INSTS_WRITTEN_TO_IQ_INSTS	INSTS_WRITTEN_TO_IQ.INSTS	Counts the number of instructions written into the IQ every cycle.
pmu36	L2_RQSTS_DEMAND_DATA_RD_HIT	L2_RQSTS.DEMAND_DATA_RD_HIT	Demand Data Read requests that hit L2 cache.
pmu37	L2_RQSTS_ALL_DEMAND_DATA_RD	L2_RQSTS.ALL_DEMAND_DATA_RD	Counts any demand and L1 HW prefetch data load requests to L2.
pmu38	L2_RQSTS_RFO_HITS	L2_RQSTS.RFO_HITS	Counts the number of store RFO requests that hit the L2 cache.
pmu39	L2_RQSTS_RFO_MISS	L2_RQSTS.RFO_MISS	Counts the number of store RFO requests that miss the L2 cache.
pmu40	L2_RQSTS_ALL_RFO	L2_RQSTS.ALL_RFO	Counts all L2 store RFO requests.
pmu41	L2_RQSTS_CODE_RD_HIT	L2_RQSTS.CODE_RD_HIT	Number of instruction fetches that hit the L2 cache.
pmu42	L2_RQSTS_CODE_RD_MISS	L2_RQSTS.CODE_RD_MISS	Number of instruction fetches that missed the L2 cache.

pmu43	L2_RQSTS_ALL_CODE_RD	L2_RQSTS.ALL_CODE_RD	Counts all L2 code requests.
pmu44	L2_RQSTS_PF_HIT	L2_RQSTS.PF_HIT	Requests from L2 Hardware prefetcher that hit L2.
pmu45	L2_RQSTS_PF_MISS	L2_RQSTS.PF_MISS	Requests from L2 Hardware prefetcher that missed L2.
pmu46	L2_RQSTS_ALL_PF	L2_RQSTS.ALL_PF	Any requests from L2 Hardware prefetchers
pmu47	L2_STORE_LOCK_RQSTS_MISS	L2_STORE_LOCK_RQSTS.MISS	RFOs that miss cache lines.
pmu48	L2_STORE_LOCK_RQSTS_HIT_E	L2_STORE_LOCK_RQSTS.HIT_E	RFOs that hit cache lines in E state.
pmu49	L2_STORE_LOCK_RQSTS_HIT_M	L2_STORE_LOCK_RQSTS.HIT_M	RFOs that hit cache lines in M state.
pmu50	L2_STORE_LOCK_RQSTS_ALL	L2_STORE_LOCK_RQSTS.ALL	RFOs that access cache lines in any state.
pmu51	L2_L1D_WB_RQSTS_HIT_E	L2_L1D_WB_RQSTS.HIT_E	Not rejected writebacks from L1D to L2 cache lines in E state.
pmu52	L2_L1D_WB_RQSTS_HIT_M	L2_L1D_WB_RQSTS.HIT_M	Not rejected writebacks from L1D to L2 cache lines in M state.
pmu53	LONGEST_LAT_CACHE_REFERENCE	LONGEST_LAT_CACHE.REFERENCE	This event counts requests originating from the core that reference a cache line in the last level cache.
pmu54	LONGEST_LAT_CACHE_MISS	LONGEST_LAT_CACHE.MISS	This event counts each cache miss condition for references to the last level cache.
pmu55	CPU_CLK_UNHALTED_THREAD_P	CPU_CLK_UNHALTED.THREAD_P	Counts the number of thread cycles while the thread is not in a halt state. The thread enters the halt state when it is running the HLT instruction. The core frequency may change from time to time due to power or thermal throttling.
pmu56	CPU_CLK_THREAD_UNHALTED_REF_XCLK	CPU_CLK_THREAD_UNHALTED_REF_XCLK	Increments at the frequency of XCLK (100 MHz) when not halted.
pmu57	L1D_PEND_MISS_PENDING	L1D_PEND_MISS.PENDING	Increments the number of outstanding L1D misses

			every cycle. Set Cmask = 1 and Edge =1 to count Occurrences
pmu58	DTLB_STORE_MISSES_MISS_CAUSES_A_WALK	DTLB_STORE_MISSES.MISS_CAUSES_A_WALK	Miss in all TLB levels causes a page walk of any page size (4K/2M/4M/1G).
pmu59	DTLB_STORE_MISSES_WALK_COMPLETED	DTLB_STORE_MISSES.WALK_COMPLETED	Miss in all TLB levels causes a page walk that completes of any page size (4K/2M/4M/1G).
pmu60	DTLB_STORE_MISSES_WALK_DURATION	DTLB_STORE_MISSES.WALK_DURATION	Cycles PMH is busy with this walk
pmu61	DTLB_STORE_MISSES_TLB_HIT	DTLB_STORE_MISSES.STLB_HIT	Store operations that miss the first TLB level but hit the second and do not cause page walks.
pmu62	LOAD_HIT_PRE_SW_PF	LOAD_HIT_PRE.SW_PF	Not SW-prefetch load dispatches that hit fill buffer allocated for S/W prefetch.
pmu63	LOAD_HIT_PREHW_PF	LOAD_HIT_PRE.HW_PF	Not SW-prefetch load dispatches that hit fill buffer allocated for H/W prefetch.
pmu64	HW_PRE_REQ_DL1_MISS	HW_PRE_REQ.DL1_MISS	Hardware Prefetch requests that miss the L1D cache. A request is being counted each time it access the cache & miss it, including if a block is applicable or if hit the Fill Buffer for example.
pmu65	L1D_REPLACEMENT	L1D.REPLACEMENT	Counts the number of lines brought into the L1 data Cache
pmu66	L1D_ALLOCATED_IN_M	L1D.ALLOCATED_IN_M	Counts the number of allocations of modified L1D cache lines.
pmu67	L1D_EVICTION	L1D.EVICTION	Counts the number of modified lines evicted from the L1 data cache due to replacement.
pmu68	L1D_ALL_M_REPLACEMENT	L1D.ALL_M_REPLACEMENT	Cache lines in M state evicted out of L1D due to Snoop HitM or dirty line replacement.
pmu69	PARTIAL_RAT_STALLS_FLAGS_MERGE_UOP	PARTIAL_RAT_STALLS.FLAGS_MERGE_UOP	Increments the number of flags-merge uops in flight

			each cycle. Set Cmask = 1 to count cycles.
pmu70	PARTIAL_RAT_STALLS_SLOW_LEA_WINDOW	PARTIAL_RAT_STALLS.SLOW_LEA_WINDOW	Cycles with at least one slow LEA uop allocated.
pmu71	PARTIAL_RAT_STALLS_MUL_SINGLE_UOP	PARTIAL_RAT_STALLS.MUL_SINGLE_UOP	Number of Multiply packed/scalar single precision uops allocated.
pmu72	RESOURCE_STALLS2_ALL_FL_EMPTY	RESOURCE_STALLS2.ALL_FL_EMPTY	Cycles stalled due to free list empty.
pmu73	RESOURCE_STALLS2_ALL_PRF_CONTROL	RESOURCE_STALLS2.ALL_PRF_CONTROL	Cycles stalled due to control structures full for physical registers.
pmu74	RESOURCE_STALLS2_BOB_FULL	RESOURCE_STALLS2.BOB_FULL	Cycles Allocator is stalled due Branch Order Buffer.
pmu75	RESOURCE_STALLS2_OOO_RSRC	RESOURCE_STALLS2.OOO_RSRC	Cycles stalled due to out of order resources full.
pmu76	CPL_CYCLES_RING0	CPL_CYCLES.RING0	Unhalted core cycles when the thread is in ring 0.
pmu77	CPL_CYCLES_RING123	CPL_CYCLES.RING123	Unhalted core cycles when the thread is not in ring 0.
pmu78	RS_EVENTS_EMPTY_CYCLES	RS_EVENTS.EMPTY_CYCLES	Cycles the RS is empty for the thread
pmu79	OFFCORE_REQUESTS_OUTSTANDING_DEMAND_DATA_RD	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_DATA_RD	Offcore outstanding Demand Data Read transactions in SQ to uncore. Set Cmask=1 to count Cycles.
pmu80	OFFCORE_REQUESTS_OUTSTANDING_DEMAND_RFO	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_RFO	Offcore outstanding RFO store transactions in SQ to uncore. Set Cmask=1 to count cycles.
pmu81	OFFCORE_REQUESTS_OUTSTANDING_ALL_DATA_RD	OFFCORE_REQUESTS_OUTSTANDING.ALL_DATA_RD	Offcore outstanding cacheable data read transactions in SQ to uncore. Set Cmask=1 to count Cycles.
pmu82	LOCK_CYCLES_SPLIT_LOCK_UC_LOCK_DURATION	LOCK_CYCLES.SPLIT_LOCK_UC_LOCK_DURATION	Cycles in which the L1D and L2 are locked, due to a UC lock or split lock.
pmu83	LOCK_CYCLES_CACHE_LOCK_DURATION	LOCK_CYCLES.CACHE_LOCK_DURATION	Cycles in which the L1D is locked.
pmu84	IDQ_EMPTY	IDQ.EMPTY	Counts cycles the IDQ is empty.
pmu85	IDQ_MITE_UOPS	IDQ.MITE_UOPS	Increment each cycle # of uops delivered to IDQ

			from MITE path. Set Cmask = 1 to count cycles.
pmu86	IDQ_DSB_UOPS	IDQ.DSB_UOPS	Increment each cycle. # of uops delivered to IDQ from DSB path. Set Cmask = 1 to count cycles
pmu87	IDQ_MS_DSB_UOPS	IDQ.MS_DSB_UOPS	Increment each cycle # of uops delivered to IDQ when MS busy by DSB. Set Cmask = 1 to count cycles MS is busy. Set Cmask=1 and Edge =1 to count MS activations.
pmu88	IDQ_MS_MITE_UOPS	IDQ.MS_MITE_UOPS	Increment each cycle # of uops delivered to IDQ when MS is busy by MITE. Set Cmask = 1 to count Cycles.
pmu89	IDQ_MS_UOPS	IDQ.MS_UOPS	Increment each cycle # of uops delivered to IDQ from MS by either DSB or MITE. Set Cmask = 1 to count cycles.
pmu90	ICACHE_MISSES	ICACHE.MISSES	Number of Instruction Cache, Streaming Buffer and Victim Cache Misses. Includes UC accesses.
pmu91	ITLB_MISSES_MISS_CAUSES_A_WALK	ITLB_MISSES.MISS_CAUSES_A_WALK	Misses in all ITLB levels that cause page walks.
pmu92	ITLB_MISSES_WALK_COMPLETED	ITLB_MISSES.WALK_COMPLETED	Misses in all ITLB levels that cause completed page Walks.
pmu93	ITLB_MISSES_WALK_DURATION	ITLB_MISSES.WALK_DURATION	Cycle PMH is busy with a walk.
pmu94	ITLB_MISSES_STLB_HIT	ITLB_MISSES.STLB_HIT	Number of cache load STLB hits. No page walk.
pmu95	ILD_STALL_LCP	ILD_STALL.LCP	Stalls caused by changing prefix length of the Instruction.
pmu96	ILD_STALL_IQ_FULL	ILD_STALL.IQ_FULL	Stall cycles due to IQ is full.
pmu97	BR_INST_EXEC_COND	BR_INST_EXEC.TAKEN_CONDITIONAL	Taken speculative and retired conditional branches
pmu98	BR_INST_EXEC_DIRECT_JMP	BR_INST_EXEC.TAKEN_DIRECT_JUMP	Taken speculative and retired conditional branches excluding calls and indirects.

pmu99	BR_INST_EXEC_INDIRECT_JMP_NON_CALL_RET	BR_INST_EXEC.TAKEN_INDIRECT_JUMP_NON_CALL_RET	Taken speculative and retired indirect branches excluding calls and returns.
pmu100	BR_INST_EXEC_RETURN_NEAR	BR_INST_EXEC.TAKEN_INDIRECT_NEAR_RETURN	Taken speculative and retired indirect branches that are returns.
pmu101	BR_INST_EXEC_DIRECT_NEAR_CALL	BR_INST_EXEC.TAKEN_DIRECT_NEAR_CALL	Taken speculative and retired direct near calls.
pmu102	BR_INST_EXEC_INDIRECT_NEAR_CALL	BR_INST_EXEC.TAKEN_INDIRECT_NEAR_CALL	Taken speculative and retired indirect near calls.
pmu103	BR_INST_EXEC_NON_TAKEN	BR_INST_EXEC.ALL_INDIRECT_JUMP_NON_CALL_RET	Speculative and retired indirect branches excluding calls and returns.
pmu104	BR_INST_EXEC_TAKEN	BR_INST_EXEC.ALL_INDIRECT_NEAR_RETURN	Speculative and retired indirect branches that are Returns.
pmu105	BR_INST_EXEC_ALL_BRANCHES	BR_INST_EXEC.ALL_BRANCHES	Speculative and retired branches.
pmu106	BR_MISP_EXEC_COND	BR_MISP_EXEC.TAKEN_CONDITIONAL	Taken speculative and retired mispredicted conditional branches
pmu107	BR_MISP_EXEC_INDIRECT_JMP_NON_CALL_RET	BR_MISP_EXEC.TAKEN_INDIRECT_JUMP_NON_CALL_RET	Taken speculative and retired mispredicted indirect branches excluding calls and returns.
pmu108	BR_MISP_EXEC_RETURN_NEAR	BR_MISP_EXEC.TAKEN_RETURN_NEAR	Taken speculative and retired mispredicted indirect branches that are returns.
pmu109	BR_MISP_EXEC_DIRECT_NEAR_CALL	BR_MISP_EXEC.TAKEN_DIRECT_NEAR_CALL	Taken speculative and retired mispredicted direct near calls
pmu110	BR_MISP_EXEC_INDIRECT_NEAR_CALL	BR_MISP_EXEC.TAKEN_INDIRECT_NEAR_CALL	Taken speculative and retired mispredicted indirect near calls.
pmu111	BR_MISP_EXEC_NON_TAKEN	BR_MISP_EXEC.ALL_INDIRECT_JUMP_NON_CALL_RET	Speculative and retired mispredicted indirect branches excluding calls and returns.
pmu112	BR_MISP_EXEC_TAKEN	BR_MISP_EXEC.ALL_NEAR_CALL	Speculative and retired mispredicted direct near Calls.
pmu113	BR_MISP_EXEC_ALL_BRANCHES	BR_MISP_EXEC.ALL_BRANCHES	Speculative and retired mispredicted branches
pmu114	IDQ_UOPS_NOT_DELIVERED_CORE	IDQ_UOPS_NOT_DELIVERED.CORE	Count issue pipeline slots where no uop was delivered from the front end to the back end when

			there is no back-end stall.
pmu115	UOPS_DISPATCHED_PORT_PORT_0	UOPS_DISPATCHED_PORT.PORT_0	Cycles which a Uop is dispatched on port 0.
pmu116	UOPS_DISPATCHED_PORT_PORT_1	UOPS_DISPATCHED_PORT.PORT_1	Cycles which a Uop is dispatched on port 1.
pmu117	UOPS_DISPATCHED_PORT_PORT_2_LD		
pmu118	UOPS_DISPATCHED_PORT_PORT_2_STA		
pmu119	UOPS_DISPATCHED_PORT_PORT_2	UOPS_DISPATCHED_PORT.PORT_2	Cycles which a Uop is dispatched on port 2.
pmu120	UOPS_DISPATCHED_PORT_PORT_3_LD		
pmu121	UOPS_DISPATCHED_PORT_PORT_3_STA		
pmu122	UOPS_DISPATCHED_PORT_PORT_3	UOPS_DISPATCHED_PORT.PORT_3	
pmu123	UOPS_DISPATCHED_PORT_PORT_4	UOPS_DISPATCHED_PORT.PORT_4	Cycles which a Uop is dispatched on port 4.
pmu124	UOPS_DISPATCHED_PORT_PORT_5	UOPS_DISPATCHED_PORT.PORT_5	Cycles which a Uop is dispatched on port 5.
pmu125	RESOURCE_STALLS_ANY	RESOURCE_STALLS.ANY	Cycles Allocation is stalled due to Resource Related reason.
pmu126	RESOURCE_STALLS_LB	RESOURCE_STALLS.LB	Counts the cycles of stall due to lack of load buffers
pmu127	RESOURCE_STALLS_RS	RESOURCE_STALLS.RS	Cycles stalled due to no eligible RS entry available.
pmu128	RESOURCE_STALLS_SB	RESOURCE_STALLS.SB	Cycles stalled due to no store buffers available (not including draining form sync)
pmu129	RESOURCE_STALLS_ROB	RESOURCE_STALLS.ROB	Cycles stalled due to re-order buffer full.
pmu130	RESOURCE_STALLS_FCSW	RESOURCE_STALLS.FCSW	Cycles stalled due to writing the FPU control word.
pmu131	RESOURCE_STALLS_MXCSR		
pmu132	RESOURCE_STALLS_OTHER		
pmu133	DSB2MITE_SWITCHES_COUNT	DSB2MITE_SWITCHES.COUNT	Number of DSB to MITE switches
pmu134	DSB2MITE_SWITCHES_PENALTY_CYCLES	DSB2MITE_SWITCHES.PENALTY_CYCLES	Cycles DSB to MITE switches caused delay

pmu135	DSB_FILL_OTHER_CANCEL	DSB_FILL.OTHER_CANCEL	Cases of cancelling valid DSB fill not because of exceeding way limit.
pmu136	DSB_FILL_EXCEED_DSB_LINES	DSB_FILL.EXCEED_DSB_LINES	DSB Fill encountered > 3 DSB lines.
pmu137	DSB_FILL_ALL_CANCEL		
pmu138	ITLB_ITLB_FLUSH	ITLB.ITLB_FLUSH	Counts the number of ITLB flushes; includes 4k/2M/4M pages.
pmu139	OFFCORE_REQUESTS_DEMAND_DATA_RD	OFFCORE_REQUESTS.DEMAND_DATA_RD	Demand data read requests sent to uncore.
pmu140	OFFCORE_REQUESTS_DEMAND_RFO	OFFCORE_REQUESTS.DEMAND_RFO	Demand RFO read requests sent to uncore, including regular RFOs, locks, ItoM.
pmu141	OFFCORE_REQUESTS_ALL_DATA_RD	OFFCORE_REQUESTS.ALL_DATA_RD	Data read requests sent to uncore (demand and Prefetch).
pmu142	UOPS_DISPATCHED_THREAD	UOPS_DISPATCHED.THREAD	Counts total number of uops to be dispatched perthread each cycle. Set Cmask = 1, INV =1 to count stall cycles.
pmu143	UOPS_DISPATCHED_CORE	UOPS_DISPATCHED.CORE	Counts total number of uops to be dispatched percore each cycle.
pmu144	OFFCORE_REQUESTS_BUFFER_SQ_FULL	OFFCORE_REQUESTS_BUFFER.SQ_FULL	Offcore requests buffer cannot take more entries for this thread core.
pmu145	AGU_BYPASS_CANCEL_COUNT	AGU_BYPASS_CANCEL.COUNT	Counts executed load operations with all the following traits: 1. Addressing of the format [base + offset], 2. The offset is between 1 and 2047, 3. The address specified in the base register is in one page and the address [base+offset] is in another page.
pmu146	OFF_CORE_RESPONSE_0	OFF_CORE_RESPONSE_0	See Section 18.9.5, “Off-core Response Performance Monitoring”.
pmu147	OFF_CORE_RESPONSE_1	OFF_CORE_RESPONSE_1	See Section 18.9.5, “Off-core Response Performance Monitoring”
pmu148	TLB_FLUSH_DTLB_THREAD	TLB_FLUSH.DTLB_THREAD	DTLB flush attempts of the thread-specific entries.

pmu149	TLB_FLUSH_STLB_ANY	TLB_FLUSH.STLB_ANY	Count number of STLB flush attempts.
pmu150	L1D_BLOCKS_BANK_CONFLICT_CYCLES	L1D_BLOCKS.BANK_CONFLICT_CYCLES	Cycles when dispatched loads are cancelled due to L1D bank conflicts with other load ports.
pmu151	INST_RETIRED_ANY_P	INST_RETIRED.ANY_P	Number of instructions at retirement.
pmu152	INST_RETIRED_PREC_DIST	INST_RETIRED.PREC_DIST	Precise instruction retired event with HW to reduce effect of PEBS shadow in IP distribution
pmu153	OTHER_ASSISTS_ITLB_MISS_RETIRED	OTHER_ASSISTS.ITLB_MISS_RETIRED	Instructions that experienced an ITLB miss.
pmu154	OTHER_ASSISTS_AVX_STORE	OTHER_ASSISTS.AVX_STORE	Number of assists associated with 256-bit AVX store operations.
pmu155	OTHER_ASSISTS_AVX_TO_SSE	OTHER_ASSISTS.AVX_TO_SSE	Number of transitions from AVX-256 to legacy SSE when penalty applicable.
pmu156	OTHER_ASSISTS_SSE_TO_AVX	OTHER_ASSISTS.SSE_TO_AVX	Number of transitions from SSE to AVX-256 when penalty applicable.
pmu157	UOPS_RETIRED_ALL	UOPS_RETIRED.ALL	Counts the number of micro-ops retired, Use cmask=1 and invert to count active cycles or stalled Cycles.
pmu158	UOPS_RETIRED_RETIRE_SLOTS	UOPS_RETIRED.RETIRE_SLOTS	Counts the number of retirement slots used each Cycle.
pmu159	MACHINE_CLEARS_MEMORY_ORDERING	MACHINE_CLEARS.MEMORY_ORDERING	Counts the number of machine clears due to memory order conflicts.
pmu160	MACHINE_CLEARS_SMC	MACHINE_CLEARS.SMC	Counts the number of times that a program writes to a code section.
pmu161	MACHINE_CLEARS_MASKMOV	MACHINE_CLEARS.MASKMOV	Counts the number of executed AVX masked load operations that refer to an illegal address range with the mask bits set to 0.
pmu162	BR_INST_RETIRED_ALL_BRANCHES_ARCH	BR_INST_RETIRED.ALL_BRANCHES	Branch instructions at retirement.
pmu163	BR_INST_RETIRED_CONDITIONAL	BR_INST_RETIRED.CONDITIONAL	Counts the number of conditional branch instructions retired.

pmu164	BR_INST_RETIRED_NEAR_CALL	BR_INST_RETIRED.NEAR_CALL	Direct and indirect near call instructions retired.
pmu165	BR_INST_RETIRED_ALL_BRANCHES	BR_INST_RETIRED.ALL_BRANCHES	Counts the number of branch instructions retired.
pmu166	BR_INST_RETIRED_NEAR_RETURN	BR_INST_RETIRED.NEAR_RETURN	Counts the number of near return instructions Retired.
pmu167	BR_INST_RETIRED_NOT_TAKEN	BR_INST_RETIRED.NOT_TAKEN	Counts the number of not taken branch instructions Retired.
pmu168	BR_INST_RETIRED_NEAR_TAKEN	BR_INST_RETIRED.NEAR_TAKEN	Number of near taken branches retired.
pmu169	BR_INST_RETIRED_FAR_BRANCH	BR_INST_RETIRED.FAR_BRANCH	Number of far branches retired.
pmu170	BR_MISP_RETIRED_ALL_BRANCHES_ARCH	BR_MISP_RETIRED.ALL_BRANCHES	Mispredicted branch instructions at retirement.
pmu171	BR_MISP_RETIRED_CONDITIONAL	BR_MISP_RETIRED.CONDITIONAL	Mispredicted conditional branch instructions retired.
pmu172	BR_MISP_RETIRED_NEAR_CALL	BR_MISP_RETIRED.NEAR_CALL	Direct and indirect mispredicted near call instructions retired.
pmu173	BR_MISP_RETIRED_ALL_BRANCHES	BR_MISP_RETIRED.ALL_BRANCHES	Mispredicted macro branch instructions retired.
pmu174	BR_MISP_RETIRED_NOT_TAKEN	BR_MISP_RETIRED.NOT_TAKEN	Mispredicted not taken branch instructions retired.
pmu175	BR_MISP_RETIRED_TAKEN	BR_MISP_RETIRED.TAKEN	Mispredicted taken branch instructions retired.
pmu176	FP_ASSIST_X87_OUTPUT	FP_ASSIST.X87_OUTPUT	Number of X87 assists due to output value
pmu177	FP_ASSIST_X87_INPUT	FP_ASSIST.X87_INPUT	Number of X87 assists due to input value.
pmu178	FP_ASSIST_SIMD_OUTPUT	FP_ASSIST.SIMD_OUTPUT	Number of SIMD FP assists due to output values.
pmu179	FP_ASSIST_SIMD_INPUT	FP_ASSIST.SIMD_INPUT	Number of SIMD FP assists due to input values.
pmu180	FP_ASSIST_ANY	FP_ASSIST.ANY	Cycles with any input/output SSE* or FP assists.
pmu181	ROB_MISC_EVENTS_LBR_INSERTS	ROB_MISC_EVENTS.LBR_INSERTS	Count cases of saving new LBR records by Hardware.
pmu182	MEM_TRANS_RETIRED_LOAD_LATENCY	MEM_TRANS_RETIRED.LOAD_LATENCY	Randomly sampled loads whose latency is above a user defined threshold. A small fraction of the

			overall loads are sampled due to randomization. PMC3 only.
pmu183	MEM_TRANS_RETIRED_PRECISE_STORE	MEM_TRANS_RETIRED.PRECISE_STORE	Sample stores and collect precise store operation via PEBS record. PMC3 only.
pmu184	MEM_UOP_RETIRED_LOADS		
pmu185	MEM_UOP_RETIRED_STORES		
pmu186	MEM_UOP_RETIRED_STLB_MISS	MEM_UOPS_RETIRED.STLB_MISS_LOADS	Retired load uops that miss the STLB.
pmu187	MEM_UOP_RETIRED_LOCK	MEM_UOPS_RETIRED.LOCK_LOADS	Retired load uops with locked access.
pmu188	MEM_UOP_RETIRED_SPLIT	MEM_UOPS_RETIRED.SPLIT_LOADS	Retired load uops that split across a cacheline Boundary.
pmu189	MEM_UOP_RETIRED_ALL		
pmu190	MEM_UOPS_RETIRED_ALL_LOADS	MEM_UOPS_RETIRED.ALL_LOADS	All retired load uops
pmu191	MEM_LOAD_UOPS_RETIRED_L1_HIT	MEM_LOAD_UOPS_RETIRED.L1_HIT	Retired load uops with L1 cache hits as data Sources.
pmu192	MEM_LOAD_UOPS_RETIRED_L2_HIT	MEM_LOAD_UOPS_RETIRED.L2_HIT	Retired load uops with L2 cache hits as data Sources
pmu193	MEM_LOAD_UOPS_RETIRED_L3_HIT		
pmu194	MEM_LOAD_UOPS_RETIRED_HIT_LFB	MEM_LOAD_UOPS_RETIRED.HIT_LFB	Retired load uops which data sources were load uops missed L1 but hit FB due to preceding miss to the same cache line with data not ready.
pmu195	XSNP_MISS	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_MISS	Retired load uops whose data source was an onpackage core cache LLC hit and cross-core snoop Missed.
pmu196	XSNP_HIT	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HIT	Retired load uops whose data source was an onpackage LLC hit and cross-core snoop hits.

pmu197	XSNP_HITM	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM	Retired load uops whose data source was an onpackage core cache with HitM responses.
pmu198	XSNP_NONE	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_NONE	Retired load uops whose data source was LLC hit with no snoop required.
pmu199	MEM_LOAD_UOPS_MISC_RETIRED_LLC_MISS	MEM_LOAD_UOPS_RETIRED.LLC_MISS	Retired load uops which data sources were data missed LLC (excluding unknown data source).
pmu200	L2_TRANS_DEMAND_DATA_RD	L2_TRANS.DEMAND_DATA_RD	Demand Data Read requests that access L2 cache
pmu201	L2_TRANS_RFO	L2_TRANS.RFO	RFO requests that access L2 cache.
pmu202	L2_TRANS_CODE_RD	L2_TRANS.CODE_RD	L2 cache accesses when fetching instructions
pmu203	L2_TRANS_ALL_PF	L2_TRANS.ALL_PF	L2 or LLC HW prefetches that access L2 cache. //including rejects
pmu204	L2_TRANS_L1D_WB	L2_TRANS.L1D_WB	L1D writebacks that access L2 cache.
pmu205	L2_TRANS_L2_FILL	L2_TRANS.L2_FILL	L2 fill requests that access L2 cache.
pmu206	L2_TRANS_L2_WB	L2_TRANS.L2_WB	L2 writebacks that access L2 cache.
pmu207	L2_TRANS_ALL_REQ_UESTS	L2_TRANS.ALL_REQUESTS	Transactions accessing L2 pipe.
pmu208	L2_LINES_IN_I	L2_LINES_IN.I	L2 cache lines in I state filling L2
pmu209	L2_LINES_IN_S	L2_LINES_IN.S	L2 cache lines in S state filling L2.
pmu210	L2_LINES_IN_E	L2_LINES_IN.E	L2 cache lines in E state filling L2.
pmu211	L2_LINES_IN_ALL	L2_LINES_IN.ALL	L2 cache lines filling L2.
pmu212	L2_LINES_OUT_DEMAND_CLEAN	L2_LINES_OUT.DEMAND_CLEAN	Clean L2 cache lines evicted by demand.
pmu213	L2_LINES_OUT_DEMAND_DIRTY	L2_LINES_OUT.DEMAND_DIRTY	Dirty L2 cache lines evicted by demand.
pmu214	L2_LINES_OUT_DEMAND_PF_CLEAN	L2_LINES_OUT.PF_CLEAN	Clean L2 cache lines evicted by L2 prefetch
pmu215	L2_LINES_OUT_DEMAND_PF_DIRTY	L2_LINES_OUT.PF_DIRTY	Dirty L2 cache lines evicted by L2 prefetch.
pmu216	L2_LINES_OUT_DEMAND_DIRTY_ALL	L2_LINES_OUT.DIRTY_ALL	Dirty L2 cache lines filling the L2. //Counting does not cover Rejects.
pmu217	SQ_MISC_SPLIT_LOCK	SQ_MISC.SPLIT_LOCK	Split locks in SQ.

A2. CÓDIGO DO SISTEMA DE MONITORAMENTO DE PERFORMANCE

Segue o link para o repositório GitHub onde se encontra o código referente ao sistema de monitoramento de performance. Vale lembrar que este código foi desenvolvido em conjunto com o discente José Luis Conradi Hoffmann, visto a utilidade do código para a pesquisa e desenvolvimento do TCC de ambos.

Por fim, o código pode ser encontrado no seguinte link:

- https://github.com/LeonardoHorstmann/Performance-Monitoring-on-EPOS

Junto do mesmo, se encontra um arquivo README que descreve as alterações realizadas.

Os arquivos alterados em relação ao código do EPOS são apresentados abaixo.

app/test_pedf.cc (código da aplicação de teste)

```
#include <periodic_thread.h>
#include <utility/random.h>
#include <clock.h>
///*
#include <machine/pc/rtc.h>
#include <chronometer.h>
#include <utility/ostream.h>
#include <tsc.h>
//*/

using namespace EPOS;


#define BASE_MEMCPY_T 30
#define BASE_RECURSIVE_FIB_T 600
#define THREADS 12 //number of real-time tasks, usually, threads_parameters size
#define SECONDS 600 //execution time in seconds

OStream cout;
// period (microsecond), deadline, execution time (microsecond), cpu (partitioned)

unsigned int iterations[THREADS];
/*
unsigned int threads_parameters[][4] = { //heavy weight taskset
{ 50000 , 50000 , 27098 , 0 },
{ 25000 , 25000 , 5504 , 1 },
{ 100000 , 100000 , 68919 , 2 },
```

```
{ 100000 , 100000 , 64664 , 3 },
{ 50000 , 50000 , 9310 , 4 },
{ 200000 , 200000 , 105758 , 5 },
{ 200000 , 200000 , 29326 , 6 },
{ 200000 , 200000 , 67222 , 7 },
{ 50000 , 50000 , 21151 , 6 },
{ 50000 , 50000 , 6757 , 4 },
{ 50000 , 50000 , 34329 , 1 },
{ 50000 , 50000 , 8203 , 4 },
{ 100000 , 100000 , 44566 , 7 },
{ 25000 , 25000 , 8853 , 4 }
};
//*/
//*
unsigned int threads_parameters[][4] = { //light weight taskset
{ 25000 , 25000 , 13958 , 0 },
{ 100000 , 100000 , 15135 , 1 },
{ 200000 , 200000 , 136986 , 2 },
{ 50000 , 50000 , 25923 , 3 },
{ 25000 , 25000 , 11637 , 4 },
{ 100000 , 100000 , 20072 , 5 },
{ 50000 , 50000 , 30484 , 6 },
{ 200000 , 200000 , 25220 , 7 },
{ 200000 , 200000 , 23924 , 7 },
{ 100000 , 100000 , 31920 , 1 },
{ 50000 , 50000 , 18343 , 5 },
{ 50000 , 50000 , 19205 , 7 } };
//*/

//calculates number of iterations to be executed to accomplish execution time in SECONDS
int calc_iterations() {
    int sum = 0;
    //float base = 0;
    for (int i = 0; i < THREADS; i++) {
        if (iterations[i] == 0) {
    //          cout<<i<<" "<<threads_parameters[i][0]<<endl;
            iterations[i] = (SECONDS * 1000000) / threads_parameters[i][0];
        }
        sum+=iterations[i];
        cout<<threads_parameters[i][0]<<" "<<iterations[i]<<endl;
    }
    return sum;
}

//fibonacci recursive method
int fib(int pos) {
    if (pos == 1 || pos == 0) {
        return 1;
    } else {
        return (fib(pos - 1) + fib(pos - 2));
    }

}
/* //This is used to collect information about the time one execution */
```

```
/*unsigned int chronos[THREADS][ITERATIONS]; //capture time of each iteration
float mean_chronos[THREADS];
unsigned int pos_chronos[THREADS];
unsigned int total_chronos[THREADS];
unsigned int id_Thread[THREADS];

void mean() {
       cout << "Mean Execution Time per Iteration: " << endl;
       int greater = 0;
    for (int i = 0; i < THREADS; i++) {
       cout << "Thread[" << i << "] " << " - Cluster = " << threads_parameters[i][3]*2 << endl;
        mean_chronos[i] = 0;
        for (int j = 0; j < pos_chronos[i]; j++) {
              if (chronos[i][j] > greater)
                     greater = chronos[i][j];
                     cout << "Iter[" << j << "] = " << chronos[i][j] << endl;
            mean_chronos[i] += chronos[i][j];
        }
        mean_chronos[i] /= pos_chronos[i];
        cout << "Mean      = " << mean_chronos[i] << "us; Worst Case = " << greater << "us" <<
endl;
        cout << "Hard Mean = " << total_chronos[i]/ITERATIONS << "us" << endl;
        cout << "Max iter  = " << (threads_parameters[i][2]/BASE_MEMCPY_T)+1 << endl;
        cout << "---------------------------------------" << endl;
    }


}
//*/

//memcpy vector
int vectors[Traits<Build>::CPUS][32768];

//high priority threads vector (REAL-TIME)
Periodic_Thread * threads[THREADS];

//low priority threads vector
Thread *low_Threads[8];

//definition of clock variable used to calculate random seed
Clock clock;

//thread id vector
unsigned int threads_id[THREADS];

//method that configures iterative fibonacci execution
int iterative_fib_test (int id) {
    float ret = 1.33;
    int fib = 1;
    int temp = 1;
    int prev = 1;
    int max = (int) ((int(threads_parameters[id][2])));
    for (int i = 0; i < iterations[id]; i++) {
```

```
        Periodic_Thread::wait_next();
        for (int x = 0; x < max; x++) {
            fib = 1;
            prev = 1;
            for (int j = 1; j < 1000; j++) {
                temp = prev+fib;
                prev = fib;
                fib = temp;
            }
            ret *= fib;
        }
    }
    return int(ret);
}

//method that runs fibonacci recursively
int fib_test (int id) {
    float ret = 1.33;
    int max = (int) ((int(threads_parameters[id][2])/BASE_RECURSIVE_FIB_T));
    for (int i = 0; i < iterations[id]; i++) {
        Periodic_Thread::wait_next();
        for (int x = 0; x < max; x++) {
            ret *= fib(25);
        }
    }
    return int(ret);
}

//method that runs memcpy method
int test(int id) {
    Random * rand;
    float ret = 1.33;
    int max = (int) ((threads_parameters[id][2]/BASE_MEMCPY_T));
    for (int i = 0; i < iterations[id]; i++) {
        Periodic_Thread::wait_next();
        rand->seed(clock.now());
        for (int x = 0; x < max; x++) {
            memcpy(reinterpret_cast<void *>(&vectors[(x*3)%3]),reinterpret_cast<void
*>(&vectors[rand->random() % Machine::n_cpus()]),sizeof(vectors[0]));
        }
    }
    return int(ret);
}

//method to execute low_priority tasks, uncomment what you want to be executed
int low_priority() {
    int x = 0;

//*
    float result = 1.33;
    //cout << Machine::cpu_id() << endl;
    while (!Thread::_end_capture) {
        result = result*fib(25);
        x++;
```

```
    }
    return int(result); //*/
/*
    Random * rand;
    while (!Thread::_end_capture) {
        memcpy(reinterpret_cast<void *>(&vectors[(x*3)%3]),reinterpret_cast<void
*>(&vectors[rand->random() % Machine::n_cpus()]),sizeof(vectors[0]));
        memcpy(reinterpret_cast<void *>(&vectors[(x*2)%3]),reinterpret_cast<void
*>(&vectors[rand->random() % Machine::n_cpus()]),sizeof(vectors[0]));
        memcpy(reinterpret_cast<void *>(&vectors[(x)%3]),reinterpret_cast<void
*>(&vectors[rand->random() % Machine::n_cpus()]),sizeof(vectors[0]));
        x++;
    }
    return 0; //*/

/*
    int fib = 1;
    int prev = 1;
    int temp = 1;
    while (!Thread::_end_capture) {
        temp = fib+prev;
        prev = fib;
        fib = temp;
    }

    return fib; //*/
}


int main()
{
    Thread::_end_capture = true;
       // initializing the vectors used to stress CPU
    for (int m = 0; m < Machine::n_cpus(); m++) {
        for (int i = 0; i < 32768; i++) {
            vectors[m][i] = i * m - 2 * m;
        }
    }

    int numero = calc_iterations();
    Thread::_end_capture = false;
    // start
    unsigned long long init_time = ((TSC::time_stamp() * 1000000)/TSC::frequency());
    /*
    *task configuration, to execute memcpy operations change "fib_test" to "test",
    *for iterative method replace to "iterative_fib_test"
    */
    for(int i=0;i<THREADS;i++){
        threads[i] = new Periodic_Thread(RTConf(threads_parameters[i][0], iterations[i],
Thread::READY ,
            Thread::Criterion((RTC::Microsecond) threads_parameters[i][0], (RTC::Microsecond)
threads_parameters[i][1],
            (RTC::Microsecond) int(threads_parameters[i][2]), (threads_parameters[i][3]))),
&fib_test, i);
```

```
        threads_id[i] = reinterpret_cast<volatile unsigned int>(threads[i]);
    }
    /*
    for(int i=0;i<8;i++){
        Thread::Configuration   conf(Thread::READY,
                                Thread::Criterion(Thread::LOW,i)
                                );
        low_Threads[i] = new Thread(conf, &low_priority);
    }
    //*/
    // sync
    for(int i=0; i<THREADS;i++) {
        threads[i]->join();
    }

    //mean(); //use to calculate execution mean time

    // stop capturing
    Thread::_end_capture = true;

    /* uncomment this and line 238
    for(int i = 0; i < 8; i++)
       low_Threads[i]->join();
    //*/

    // used to print a series to current deadline misses, wich is thread bound, not cpu bound.
    cout<<"time = "<<(((TSC::time_stamp() * 1000000)/TSC::frequency()) - init_time)<<endl;

    //used to print thread_ids for sending algorithms
    /*
    cout << "<begin_tseries>" << endl;
    cout << "<";
    for (int i = 0; i < THREADS-1; i++)
        cout << threads_id[i] << ",";
    cout << threads_id[THREADS-1] << ">" << endl;
    cout << "<end_tseries>" << endl;
    //end */

    for(int i = 0; i < THREADS; i++)
        delete threads[i];

    return 0;
}
```

app/test_pedf_traits.h (configuração da aplicação de teste)

```
#ifndef __traits_h
#define __traits_h

#include <system/config.h>

__BEGIN_SYS
```

```
// Global Configuration
template<typename T>
struct Traits
{
    static const bool enabled = true;
    static const bool debugged = true;
    static const bool hysterically_debugged = false;
    typedef TLIST<> ASPECTS;
};

template<> struct Traits<Build>
{
    enum {LIBRARY, BUILTIN, KERNEL};
    static const unsigned int MODE = LIBRARY;

    enum {IA32, ARMv7};
    static const unsigned int ARCHITECTURE = IA32;

    enum {PC, Cortex};
    static const unsigned int MACHINE = PC;

    enum {Legacy_PC, eMote3, LM3S811, Zynq};
    static const unsigned int MODEL = Legacy_PC;

    static const unsigned int CPUS = 8;
    static const unsigned int NODES = 1; // > 1 => NETWORKING
};


// Utilities
template<> struct Traits<Debug>
{
    static const bool error   = true;
    static const bool warning = true;
    static const bool info    = false;
    static const bool trace   = false;
};

template<> struct Traits<Lists>: public Traits<void>
{
    static const bool debugged = hysterically_debugged;
};

template<> struct Traits<Spin>: public Traits<void>
{
    static const bool debugged = hysterically_debugged;
};

template<> struct Traits<Heaps>: public Traits<void>
{
    static const bool debugged = hysterically_debugged;
};
```

```
// System Parts (mostly to fine control debugging)
template<> struct Traits<Boot>: public Traits<void>
{
};

template<> struct Traits<Setup>: public Traits<void>
{
};

template<> struct Traits<Init>: public Traits<void>
{
};

// template<> struct Traits<Monitoring>: public Traits<void>
// {
// };

// Mediators
template<> struct Traits<Serial_Display>: public Traits<void>
{
    static const bool enabled = false;
    enum {UART, USB};
    static const int ENGINE = UART;
    static const int COLUMNS = 80;
    static const int LINES = 24;
    static const int TAB_SIZE = 8;
};

__END_SYS

#include __ARCH_TRAITS_H
#include __MACH_TRAITS_H

__BEGIN_SYS

// Components
template<> struct Traits<Application>: public Traits<void>
{
    static const unsigned int STACK_SIZE = Traits<Machine>::STACK_SIZE;
    static const unsigned int HEAP_SIZE = Traits<Machine>::HEAP_SIZE;
    static const unsigned int MAX_THREADS = Traits<Machine>::MAX_THREADS;
};

template<> struct Traits<System>: public Traits<void>
{
    static const unsigned int mode = Traits<Build>::MODE;
    static const bool multithread = (Traits<Application>::MAX_THREADS > 1);
    static const bool multitask = (mode != Traits<Build>::LIBRARY);
    static const bool multicore = (Traits<Build>::CPUS > 1) && multithread;
    static const bool multiheap = (mode != Traits<Build>::LIBRARY) ||
Traits<Scratchpad>::enabled;

    enum {FOREVER = 0, SECOND = 1, MINUTE = 60, HOUR = 3600, DAY = 86400, WEEK = 604800, MONTH
```

```
= 2592000, YEAR = 31536000};
    static const unsigned long LIFE_SPAN = 1 * HOUR; // in seconds

    static const bool reboot = true;

    static const unsigned int STACK_SIZE = Traits<Machine>::STACK_SIZE;
    //configuration of heap was changed to storing data captured
    static const unsigned int HEAP_SIZE = (6 * 8 * 4 * 1024 * 1024) +
(Traits<Application>::MAX_THREADS + 1) * Traits<Application>::STACK_SIZE;
};

template<> struct Traits<Task>: public Traits<void>
{
    static const bool enabled = Traits<System>::multitask;
};

template<> struct Traits<Thread>: public Traits<void>
{
    static const bool smp = Traits<System>::multicore;

    typedef Scheduling_Criteria::MPEDF Criterion;
    static const unsigned int QUANTUM = 5000; // us

    static const bool trace_idle = hysterically_debugged;
};

template<> struct Traits<Scheduler<Thread> >: public Traits<void>
{
    static const bool debugged = Traits<Thread>::trace_idle || hysterically_debugged;
};

template<> struct Traits<Periodic_Thread>: public Traits<void>
{
    static const bool simulate_capacity = false;
};

template<> struct Traits<Address_Space>: public Traits<void>
{
    static const bool enabled = Traits<System>::multiheap;
};

template<> struct Traits&lt;Segment&gt;: public Traits<void>
{
    static const bool enabled = Traits<System>::multiheap;
};

template<> struct Traits<Alarm>: public Traits<void>
{
    static const bool visible = hysterically_debugged;
};

template<> struct Traits<Synchronizer>: public Traits<void>
{
    static const bool enabled = Traits<System>::multithread;
```

```
};

template<> struct Traits<Network>: public Traits<void>
{
    static const bool enabled = (Traits<Build>::NODES > 1);

    static const unsigned int RETRIES = 3;
    static const unsigned int TIMEOUT = 10; // s

    // This list is positional, with one network for each NIC in Traits<NIC>::NICS
    typedef LIST<IP> NETWORKS;
};

//template<> struct Traits<ELP>: public Traits<Network>
//{
//    static const bool enabled = NETWORKS::Count<ELP>::Result;
//
//    static const bool acknowledged = true;
//};

template<> struct Traits<TSTP>: public Traits<Network>
{
    static const bool enabled = NETWORKS::Count<TSTP>::Result;
    static const unsigned int KEY_SIZE = 16;
};

// template<> template <typename S> struct Traits<Smart_Data<S>>: public Traits<Network>
// {
//     static const bool enabled = NETWORKS::Count<TSTP>::Result;
// };

template<> struct Traits<IP>: public Traits<Network>
{
    static const bool enabled = NETWORKS::Count<IP>::Result;

    enum {STATIC, MAC, INFO, RARP, DHCP};

    struct Default_Config {
        static const unsigned int  TYPE    = DHCP;
        static const unsigned long ADDRESS = 0;
        static const unsigned long NETMASK = 0;
        static const unsigned long GATEWAY = 0;
    };

    template<unsigned int UNIT>
    struct Config: public Default_Config {};

    static const unsigned int TTL  = 0x40; // Time-to-live
};

template<> struct Traits<IP>::Config<0> //: public Traits<IP>::Default_Config
{
    static const unsigned int  TYPE      = MAC;
    static const unsigned long ADDRESS   = 0x0a000100;  // 10.0.1.x x=MAC[5]
```

```
    static const unsigned long NETMASK   = 0xffffff00;  // 255.255.255.0
    static const unsigned long GATEWAY   = 0;           // 10.0.1.1
};

template<> struct Traits<IP>::Config<1>: public Traits<IP>::Default_Config
{
};

template<> struct Traits<UDP>: public Traits<Network>
{
    static const bool checksum = true;
};

template<> struct Traits<TCP>: public Traits<Network>
{
    static const unsigned int WINDOW = 4096;
};

template<> struct Traits<DHCP>: public Traits<Network>
{
};


__END_SYS

#endif
```

```
include/architecture/ia32/cpu.h:
novas funções para ler MSRs e para configurar a modulação de clock;
 - pp1_energy_status()
 - pp0_energy_status()
 - pkg_energy_status()
 - dram_energy_status()
 - rapl_power_unit()
 - rapl_energy_unit()
     Documentação da leitura de msrs em 14.9.3 “Package RAPL Domain” do Intel SDM
 - temperature()
 - clock (Reg64 clock)
```

```
// EPOS IA32 CPU Mediator Declarations

#ifndef __ia32_h
#define __ia32_h

#include <cpu.h>

__BEGIN_SYS

class CPU: public CPU_Common
{
...
    //check Intel SDM, 14.9.3 "Package RAPL Domain"
    static unsigned long long rapl_energy_unit() {
        Reg64 rapl_energy_read = rdmsr(0x606);
```

```
        rapl_energy_read >>= 8;
        rapl_energy_read &= (1ULL << 5) - 1;
        return (rapl_energy_read);
    }
    //indicates the value to be used while measuring energy
    static unsigned long long rapl_power_unit() {
        Reg64 rapl_power_read = rdmsr(0x606);
        rapl_power_read &= (1ULL << 4) - 1;
        return (rapl_power_read);
    }
    //energy consumed by the entire package
    static unsigned long long pkg_energy_status() {
        Reg64 pkg_energy_read = rdmsr(0x611);
        pkg_energy_read &= (1ULL << 32) - 1;
        return (pkg_energy_read);
    }
    //energy consumed by the dram, check for compatibility
    static unsigned long long dram_energy_status() {
        Reg64 dram_energy_read = rdmsr(0x619);
        dram_energy_read &= (1ULL << 32) - 1;
        return (dram_energy_read);
    }
    //energy consumed by the cores
    static unsigned long long pp0_energy_status() {
        Reg64 pp0_energy_read = rdmsr(0x639);
        pp0_energy_read &= (1ULL << 32) - 1;
        return (pp0_energy_read);
    }
    //energy on cache and integrated GPU, check for compatibility
    static unsigned long long pp1_energy_status() {
        Reg64 pp1_energy_read = rdmsr(0x641);
        pp1_energy_read &= (1ULL << 32) - 1;
        return (pp1_energy_read);
    }
...
   static unsigned int temperature() {
        Reg64 therm_read = rdmsr(THERM_STATUS);
        Reg64 temp_target_read = rdmsr(TEMPERATURE_TARGET);
        //temperature is measured by taking from the target the value on status
        int bits = 22 - 16 + 1;
        therm_read >>= 16;
        therm_read &= (1ULL << bits) - 1;
        bits = 23 - 16 + 1;
        temp_target_read >>= 16;
        temp_target_read &= (1ULL << bits) - 1;
        return (temp_target_read - therm_read);
    }
...
   static void clock(Reg64 clock) {
        // clock must be taken as Reg64 because Hertz is Reg32 is some configurations and
that's not enough for the comparisons bellow
        unsigned int dc;
        Reg64 cpu_clock_aux = _cpu_clock;
        if(clock <= cpu_clock_aux * 1875 / 10000)
```

```
            dc = 0b10011;   // Minimum duty cycle of 12.5 %
        else if(clock >= cpu_clock_aux * 9375 / 10000)
            dc = 0b01001;   // Disable duty cycling and operate at full speed
        else
            dc = 0b10001 | ((clock * 10000 / cpu_clock_aux + 625) / 625); // Dividing by 625
instead of 1250 eliminates the shift left
        wrmsr(CLOCK_MODULATION, dc);
    }
...
}
```

include/architecture/ia32/monitoring_capture.h (definicao do sistema de monitoramento)

```
#ifndef __ia32_monitoring_capture_h
#define __ia32_monitoring_capture_h

#include <machine.h>
#include <pmu.h>
#include <nic.h>
#include <tsc.h>

__BEGIN_SYS
struct Moment {
    unsigned long long _temperature;
    unsigned long long _pmu0;
    unsigned long long _pmu1;
    unsigned long long _pmu2;
    unsigned long long _pmu3;
    unsigned long long _pmu4;
    unsigned long long _pmu5;
    unsigned long long _pmu6;
    unsigned long long _time_stamp;
    unsigned long long _thread_id;
    long long _thread_priority;
    unsigned long long _deadline;
    unsigned long long _global_deadline;
    unsigned long long _pkg_energy;
    unsigned long long _pp0_energy;

    //"constructor"
    Moment() {}

    // used to make a copy of values
    // static void copy (Moment from, Moment *to) {
    //     to->_temperature = from._temperature;
    //     to->_pmu0 = from._pmu0;
    //     to->_pmu1 = from._pmu1;
    //     to->_pmu2 = from._pmu2;
    //     to->_pmu3 = from._pmu3;
    //     to->_pmu4 = from._pmu4;
    //     to->_pmu5 = from._pmu5;
    //     to->_pmu6 = from._pmu6;
```

```
    //      to->_time_stamp = from._time_stamp;
    //      to->_thread_id = from._thread_id;
    //      to->_thread_priority = from._thread_priority;
    //      to->_deadline = from._deadline;
    //      to->_global_deadline = from._global_deadline;

    // }
};

class Monitoring_Capture {

public:
       Moment * _mem_moment; //pointer to vector of moments
       unsigned int _init_pos[Traits<Build>::CPUS]; //initial positions of each cpu
    unsigned int _mem_pos[Traits<Build>::CPUS]; // number of stored captures of each cpu
    int _max_size; //max size of the cpu subvectors
       unsigned int _over[Traits<Build>::CPUS]; //number of captures unstored per cpu

    // smart_data_info
    unsigned long long _t0; //initial execution time
    unsigned long long _t1; //final execution time
    NIC::Address _mac; //mac address -> not used anymore
    unsigned long long _tsc_base; //base tsc time
    unsigned int _units[12]; //units vector (ignoring rapl)
    //coordinates
    int _x;
    int _y;
    int _z;
    unsigned int _r; //radius - distance from the central point (x,y,z)
    unsigned int _errorsmart; //smardata error value
    unsigned int _confidence; //confidence smartdata value

public:

    //constructor
    Monitoring_Capture(int size, Moment * init) {
       //storing config
        _max_size = size;
       for (unsigned int i = 0; i < Traits<Build>::CPUS; i++) {
               _mem_pos[i] = i * size;
               _init_pos[i] = _mem_pos[i];
               _over[i] = 0;
       }
        _mem_moment = init;
        //metadata config
        _t0 = RTC::seconds_since_epoch() * 1000000;
        _t1 = _t0 + (5*60*1000000);
        _tsc_base = _t0 - (TSC::time_stamp() * 1000000 / TSC::frequency());
        NIC nic;
        _mac = nic.address();
        for (int i = 0; i < 8; i++)
            _units[i] = (i+1) << 16 | 8;
        _units[8] = ((PMU::_channel_3+8)+1) << 16 | 8;
        _units[9] = ((PMU::_channel_4+8)+1) << 16 | 8;
```

```
        _units[10] = ((PMU::_channel_5+8)+1) << 16 | 8;
        _units[11] = ((PMU::_channel_6+8)+1) << 16 | 8;
        _x =  3746654;//3765.829 * 100000;
        _y = -4237592;
        _z = -293735;
        _r = 0;
        _errorsmart = 0;
        _confidence = 1;
        //print basic metadata before system execution
        print_smart_params();
        series();
    }

    //basic destructor
    ~Monitoring_Capture () {
        delete _mem_moment;
    }

    //stores a capture using the params
    void capture(unsigned int temperature, unsigned long long pmu0, unsigned long long pmu1,
unsigned long long pmu2, unsigned long long pmu3, unsigned long long pmu4, unsigned long long
pmu5,
                unsigned long long pmu6, unsigned int thread_id, long long thread_priority,
unsigned int cpu_id, unsigned int deadline, unsigned int global_deadline, unsigned long long
pkg, unsigned long long pp0) {
        if (_mem_pos[cpu_id] < ((cpu_id+1) * _max_size)) {
            unsigned int pos = _mem_pos[cpu_id];
            _mem_moment[pos]._temperature = (unsigned long long)temperature;
            _mem_moment[pos]._pmu0 = pmu0;
            _mem_moment[pos]._pmu1 = pmu1;
            _mem_moment[pos]._pmu2 = pmu2;
            _mem_moment[pos]._pmu3 = pmu3;
            _mem_moment[pos]._pmu4 = pmu4;
            _mem_moment[pos]._pmu5 = pmu5;
            _mem_moment[pos]._pmu6 = pmu6;
            _mem_moment[pos]._time_stamp = _tsc_base + (TSC::time_stamp() * 1000000 /
TSC::frequency());
            _mem_moment[pos]._thread_id = (unsigned long long)thread_id;
            _mem_moment[pos]._thread_priority = (long long)thread_priority;
            _mem_moment[pos]._deadline = (unsigned long long)deadline;
            _mem_moment[pos]._global_deadline = (unsigned long long)global_deadline;
            _mem_moment[pos]._pkg_energy = pkg;
            _mem_moment[pos]._pp0_energy = pp0;
            //Moment::copy(m, &_mem_moment[_mem_pos[cpu_id]]);
            _mem_pos[cpu_id]++;
           // if (!_circular_cpu_data[cpu_id]->next()) {
           //      send(cpu_id);
           // }
           // _circular_cpu_data[cpu_id]->insert(m);
        }
         /*else {
            _over[cpu_id]++;
        }//*/
    }
```

```
    //returns time of the system (reuse of the values)
    unsigned long long time () {
        return _tsc_base + (TSC::time_stamp() * 1000000 / TSC::frequency());
    }

    //returns the value of the last capture of a channel/event on a cpu
    unsigned long long last_capture(unsigned int cpu_id, unsigned int channel) {
        unsigned int pos = _mem_pos[cpu_id] - 1;
        if (pos < 0) {
            return 0;
        }
        switch (channel) {
            case 0:
                return _mem_moment[pos]._pmu0;
            case 1:
                return _mem_moment[pos]._pmu1;
            case 2:
                return _mem_moment[pos]._pmu2;
            case 3:
                return _mem_moment[pos]._pmu3;
            case 4:
                return _mem_moment[pos]._pmu4;
            case 5:
                return _mem_moment[pos]._pmu5;
            case 6:
                return _mem_moment[pos]._pmu6;
            default:
                return _mem_moment[pos]._time_stamp;
        }
    }

    void series();

    void datas();

    void print_smart_params();

    //returns the position of the storing of a cpu
    unsigned int  get_mem_pos(int cpu_id) {
        return _mem_pos[cpu_id];
    }

};

__END_SYS

#endif //monitoring_capture_h
```

src/architecture/ia32/monitoring_capture_send.cc (métodos de impressão dos dados)

```
#include <architecture/ia32/monitoring_capture.h>
```

```
#include <utility/ostream.h>
__BEGIN_SYS

OStream cout;
//printing captured data -> comment line 9 to print all the data
void Monitoring_Capture::datas() {
        unsigned long long final_time = RTC::seconds_since_epoch() *1000000;
        /*
        cout << "<end_capture>" << endl;
        for (unsigned int i = 0; i < Machine::n_cpus(); i++) {
                cout << "init_temp" << i << ": " << _mem_moment[_init_pos[i]]._temperature <<
endl;
                cout << "cap  CPU" << i << ": " << _mem_pos[i] - _init_pos[i] << endl;
                cout << "over CPU" << i << ": " << _over[i] << endl;
                cout << "deadlines " << i << ": " <<_mem_moment[_mem_pos[i]-1]._deadline <<
endl;
        }
        cout << "global_ddl_m" << ": " <<_mem_moment[_mem_pos[0]-1]._global_deadline << endl;
        //cout << "final time | Elapsed time: " << final_time << " | " << (final_time -
_t0)/1000000 << endl;
        while(1) cout<<"simulation ended"<<endl;
        //*/
        cout << "<begin_capture>" << endl;
        Moment m;
        for (unsigned int i = 0; i < Machine::n_cpus(); i++) {
                //continue; //discomment if you don't want to print the collected data
                for (unsigned int j = _init_pos[i]; j < _mem_pos[i]; j++) {
                        m = _mem_moment[j];
                        unsigned long long value[] = {m._temperature, m._pmu0, m._pmu1, m._pmu2,
m._pmu3, m._pmu4, m._pmu5, m._pmu6,
                                                    m._time_stamp, m._thread_id, m._thread_priority, i,
m._deadline, m._global_deadline, m._pkg_energy, m._pp0_energy};
                        cout << "<";
                        for (int i = 0; i < 15; i++) {
                                cout << value[i] << ",";
                        }
                        cout << value[15] << ">" << endl;
                }
                //break;
        }
        cout << "<end_capture>" << endl;
        for (unsigned int i = 0; i < Machine::n_cpus(); i++) {
                cout << "cap  CPU" << i << ": " << _mem_pos[i] - _init_pos[i] << endl;
                cout << "over CPU" << i << ": " << _over[i] << endl;
        }
        cout << "final time | Elapsed time: " << final_time << " | " << (final_time -
_t0)/1000000 << endl;
        while(1) cout<<"simulation ended"<<endl;
}

//print series(metadata)
void Monitoring_Capture::series() {
    //*
    cout << "<begin_series>" << endl;
```

```
	for (unsigned int cpu = 0; cpu < Machine::n_cpus(); cpu++)
		for (int i = 0; i < 12; i++) {
			if (i != 4) {
				cout << "+\n"
				<< "{\n"
				<< "\t\"series\" :\n"
				<< "\t{\n"
				<< "\t\t\"version\" : \"1.1\",\n"
				<< "\t\t\"unit\" : " << _units[i] << ",\n"
				<< "\t\t\"x\" : " << _x + cpu*10 << ",\n"
				<< "\t\t\"y\" : " << _y << ",\n"
				<< "\t\t\"z\" : " << _z << ",\n"
				<< "\t\t\"r\" : " << _r << ",\n"
				<< "\t\t\"t0\" : " << _t0 << ",\n"
				<< "\t\t\"t1\" : " << _t1 << ",\n"
				<< "\t\t\"dev\" : " << cpu << "\n"
				<< "\t}\n"
				<< "}\n"
				<< "+" << endl;
			}
		}
	cout << "<end_series>" << endl;//*/
}
//print the parameters to mount the smartdatas
void Monitoring_Capture::print_smart_params() {
	//*
	cout << "<begin_params>" << endl;
	cout << "<" << _mac << ",";
	for (int i = 0; i < 12; i++)
		cout << _units[i] << ",";
	cout << _t0 << ",";
	cout << _t1 << ">" << endl;
	cout << "<end_params>" << endl;
	//*/
}

__END_SYS
```

```
include/pmu.h
 - Adicionado a configuração dos eventos da PMU versão Sandy Bridge ao enum Event;
 - Adicionado variaveis para seleção dos canais da PMU a partir do enum Event;
```

```
// EPOS PMU Mediator Common Package

#ifndef __pmu_h
#define __pmu_h

#include <system/config.h>

__BEGIN_SYS

class PMU_Common
```

```
{
public:
    typedef unsigned int Channel;
    typedef long long int Count;

    enum Event {
        CLOCK, //0
        DVS_CLOCK, //1
        INSTRUCTION, //2
        BRANCH, //3
        BRANCH_MISS, //4
        L1_HIT, //5
        L2_HIT, //6
        L3_HIT, //7
        LLC_HIT = L3_HIT,
        CACHE_HIT = LLC_HIT,
        L1_MISS, //8
        L2_MISS, //9
        L3_MISS, //10
        LLC_MISS = L3_MISS,
        CACHE_MISS = LLC_MISS,
        LLC_HITM, //11
        //Sandy_Bridge_events
        LD_BLOCKS_DATA_UNKNOWN_C, //#12
        LD_BLOCKS_STORE_FORWARD_C,
        LD_BLOCKS_NO_SR_C,
        LD_BLOCKS_ALL_BLOCK_C,
        MISALIGN_MEM_REF_LOADS_C,
        MISALIGN_MEM_REF_STORES_C,
        LD_BLOCKS_PARTIAL_ADDRESS_ALIAS_C,
        LD_BLOCKS_PARTIAL_ALL_STA_BLCOK_C,
        DTLB_LOAD_MISSES_MISS_CAUSES_A_WALK_C,
        DTLB_LOAD_MISSES_MISS_WALK_COMPLETED_C,
        DTLB_LOAD_MISSES_MISS_WALK_DURATION_C,
        DTLB_LOAD_MISSES_MISS_STLB_HIT_C,
        INT_MISC_RECOVERY_CYCLES_C,
        INT_MISC_RAT_STALL_CYCLES_C,
        UOPS_ISSUED_ANY_C,
        FP_COMP_OPS_EXE_X87_C,
        FP_COMP_OPS_EXE_SSE_FP_PACKED_DOUBLE_C,
        FP_COMP_OPS_EXE_SSE_FP_SCALAR_SINGLE_C,
        FP_COMP_OPS_EXE_SSE_PACKED_SINGLE_C,
        FP_COMP_OPS_EXE_SSE_SCALAR_DOUBLE_C,
        SIMD_FP_256_PACKED_SINGLE_C,
        SIMD_FP_256_PACKED_DOUBLE_C,
        ARITH_FPU_DIV_ACTIVE_C,
        INSTS_WRITTEN_TO_IQ_INSTS_C,
        L2_RQSTS_DEMAND_DATA_RD_HIT_C,
        L2_RQSTS_ALL_DEMAND_DATA_RD_C,
        L2_RQSTS_RFO_HITS_C,
        L2_RQSTS_RFO_MISS_C,
        L2_RQSTS_ALL_RFO_C,
        L2_RQSTS_CODE_RD_HIT_C,
        L2_RQSTS_CODE_RD_MISS_C,
```

```
L2_RQSTS_ALL_CODE_RD_C,
L2_RQSTS_PF_HIT_C,
L2_RQSTS_PF_MISS_C,
L2_RQSTS_ALL_PF_C,
L2_STORE_LOCK_RQSTS_MISS_C,
L2_STORE_LOCK_RQSTS_HIT_E_C,
L2_STORE_LOCK_RQSTS_HIT_M_C,
L2_STORE_LOCK_RQSTS_ALL_C,
L2_L1D_WB_RQSTS_HIT_E_C,
L2_L1D_WB_RQSTS_HIT_M_C,
LONGEST_LAT_CACHE_REFERENCE_C, //table 19-1 architectural event
LONGEST_LAT_CACHE_MISS_C, //table 19-1 architectural event
CPU_CLK_UNHALTED_THREAD_P_C, //table 19-1 architectural event
CPU_CLK_THREAD_UNHALTED_REF_XCLK_C, //table 1901 architectural event
L1D_PEND_MISS_PENDING_C, //counter 2 only - set cmask = 1 to count cycles
DTLB_STORE_MISSES_MISS_CAUSES_A_WALK_C,
DTLB_STORE_MISSES_WALK_COMPLETED_C,
DTLB_STORE_MISSES_WALK_DURATION_C,
DTLB_STORE_MISSES_TLB_HIT_C,
LOAD_HIT_PRE_SW_PF_C,
LOAD_HIT_PREHW_PF_C,
HW_PRE_REQ_DL1_MISS_C,
L1D_REPLACEMENT_C,
L1D_ALLOCATED_IN_M_C,
L1D_EVICTION_C,
L1D_ALL_M_REPLACEMENT_C,
PARTIAL_RAT_STALLS_FLAGS_MERGE_UOP_C,
PARTIAL_RAT_STALLS_SLOW_LEA_WINDOW_C,
PARTIAL_RAT_STALLS_MUL_SINGLE_UOP_C,
RESOURCE_STALLS2_ALL_FL_EMPTY_C,
RESOURCE_STALLS2_ALL_PRF_CONTROL_C,
RESOURCE_STALLS2_BOB_FULL_C,
RESOURCE_STALLS2_OOO_RSRC_C,
CPL_CYCLES_RING0_C, //use edge to count transition
CPL_CYCLES_RING123_C,
RS_EVENTS_EMPTY_CYCLES_C,
OFFCORE_REQUESTS_OUTSTANDING_DEMAND_DATA_RD_C,
OFFCORE_REQUESTS_OUTSTANDING_DEMAND_RFO_C,
OFFCORE_REQUESTS_OUTSTANDING_ALL_DATA_RD_C,
LOCK_CYCLES_SPLIT_LOCK_UC_LOCK_DURATION_C,
LOCK_CYCLES_CACHE_LOCK_DURATION_C,
IDQ_EMPTY_C,
IDQ_MITE_UOPS_C,
IDQ_DSB_UOPS_C,
IDQ_MS_DSB_UOPS_C,
IDQ_MS_MITE_UOPS_C,
IDQ_MS_UOPS_C,
ICACHE_MISSES_C,
ITLB_MISSES_MISS_CAUSES_A_WALK_C,
ITLB_MISSES_WALK_COMPLETED_C,
ITLB_MISSES_WALK_DURATION_C,
ITLB_MISSES_STLB_HIT_C,
ILD_STALL_LCP_C,
ILD_STALL_IQ_FULL_C,
```

```
BR_INST_EXEC_COND_C,
BR_INST_EXEC_DIRECT_JMP_C,
BR_INST_EXEC_INDIRECT_JMP_NON_CALL_RET_C,
BR_INST_EXEC_RETURN_NEAR_C,
BR_INST_EXEC_DIRECT_NEAR_CALL_C,
BR_INST_EXEC_INDIRECT_NEAR_CALL_C,
BR_INST_EXEC_NON_TAKEN_C,
BR_INST_EXEC_TAKEN_C,
BR_INST_EXEC_ALL_BRANCHES_C,
BR_MISP_EXEC_COND_C,
BR_MISP_EXEC_INDIRECT_JMP_NON_CALL_RET_C,
BR_MISP_EXEC_RETURN_NEAR_C,
BR_MISP_EXEC_DIRECT_NEAR_CALL_C,
BR_MISP_EXEC_INDIRECT_NEAR_CALL_C,
BR_MISP_EXEC_NON_TAKEN_C,
BR_MISP_EXEC_TAKEN_C,
BR_MISP_EXEC_ALL_BRANCHES_C,
IDQ_UOPS_NOT_DELIVERED_CORE_C,
UOPS_DISPATCHED_PORT_PORT_0_C,
UOPS_DISPATCHED_PORT_PORT_1_C,
UOPS_DISPATCHED_PORT_PORT_2_LD_C,
UOPS_DISPATCHED_PORT_PORT_2_STA_C,
UOPS_DISPATCHED_PORT_PORT_2_C,
UOPS_DISPATCHED_PORT_PORT_3_LD_C,
UOPS_DISPATCHED_PORT_PORT_3_STA_C,
UOPS_DISPATCHED_PORT_PORT_3_C,
UOPS_DISPATCHED_PORT_PORT_4_C,
UOPS_DISPATCHED_PORT_PORT_5_C,
RESOURCE_STALLS_ANY_C,
RESOURCE_STALLS_LB_C,
RESOURCE_STALLS_RS_C,
RESOURCE_STALLS_SB_C,
RESOURCE_STALLS_ROB_C,
RESOURCE_STALLS_FCSW_C,
RESOURCE_STALLS_MXCSR_C,
RESOURCE_STALLS_OTHER_C,
DSB2MITE_SWITCHES_COUNT_C,
DSB2MITE_SWITCHES_PENALTY_CYCLES_C,
DSB_FILL_OTHER_CANCEL_C,
DSB_FILL_EXCEED_DSB_LINES_C,
DSB_FILL_ALL_CANCEL_C,
ITLB_ITLB_FLUSH_C,
OFFCORE_REQUESTS_DEMAND_DATA_RD_C,
OFFCORE_REQUESTS_DEMAND_RFO_C,
OFFCORE_REQUESTS_ALL_DATA_RD_C,
UOPS_DISPATCHED_THREAD_C,
UOPS_DISPATCHED_CORE_C,
OFFCORE_REQUESTS_BUFFER_SQ_FULL_C,
AGU_BYPASS_CANCEL_COUNT_C,
OFF_CORE_RESPONSE_0_C,
OFF_CORE_RESPONSE_1_C,
TLB_FLUSH_DTLB_THREAD_C,
TLB_FLUSH_STLB_ANY_C,
L1D_BLOCKS_BANK_CONFLICT_CYCLES_C,
```

```
INST_RETIRED_ANY_P_C, //table 19-1 architectural event
INST_RETIRED_PREC_DIST_C, //PMC1 only; must quiesce other PMCs
OTHER_ASSISTS_ITLB_MISS_RETIRED_C,
OTHER_ASSISTS_AVX_STORE_C,
OTHER_ASSISTS_AVX_TO_SSE_C,
OTHER_ASSISTS_SSE_TO_AVX_C,
UOPS_RETIRED_ALL_C,
UOPS_RETIRED_RETIRE_SLOTS_C,
MACHINE_CLEARS_MEMORY_ORDERING_C,
MACHINE_CLEARS_SMC_C,
MACHINE_CLEARS_MASKMOV_C,
BR_INST_RETIRED_ALL_BRANCHES_ARCH_C, //table 19-1
BR_INST_RETIRED_CONDITIONAL_C,
BR_INST_RETIRED_NEAR_CALL_C,
BR_INST_RETIRED_ALL_BRANCHES_C,
BR_INST_RETIRED_NEAR_RETURN_C,
BR_INST_RETIRED_NOT_TAKEN_C,
BR_INST_RETIRED_NEAR_TAKEN_C,
BR_INST_RETIRED_FAR_BRANCH_C,
BR_MISP_RETIRED_ALL_BRANCHES_ARCH_C, //table 19-1
BR_MISP_RETIRED_CONDITIONAL_C,
BR_MISP_RETIRED_NEAR_CALL_C,
BR_MISP_RETIRED_ALL_BRANCHES_C,
BR_MISP_RETIRED_NOT_TAKEN_C,
BR_MISP_RETIRED_TAKEN_C,
FP_ASSIST_X87_OUTPUT_C,
FP_ASSIST_X87_INPUT_C,
FP_ASSIST_SIMD_OUTPUT_C,
FP_ASSIST_SIMD_INPUT_C,
FP_ASSIST_ANY_C,
ROB_MISC_EVENTS_LBR_INSERTS_C,
MEM_TRANS_RETIRED_LOAD_LATENCY_C, //specify threshold in MSR 0x3F6
MEM_TRANS_RETIRED_PRECISE_STORE_C, //see section 18.8.4.3
MEM_UOP_RETIRED_LOADS_C,
MEM_UOP_RETIRED_STORES_C,
MEM_UOP_RETIRED_STLB_MISS_C,
MEM_UOP_RETIRED_LOCK_C,
MEM_UOP_RETIRED_SPLIT_C,
MEM_UOP_RETIRED_ALL_C,
MEM_UOPS_RETIRED_ALL_LOADS_C, // Supports PEBS. PMC0-3 only regardless HTT.
MEM_LOAD_UOPS_RETIRED_L1_HIT_C,
MEM_LOAD_UOPS_RETIRED_L2_HIT_C,
MEM_LOAD_UOPS_RETIRED_L3_HIT_C,
MEM_LOAD_UOPS_RETIRED_HIT_LFB_C,
XSNP_MISS_C,
XSNP_HIT_C,
XSNP_HITM_C,
XSNP_NONE_C,
MEM_LOAD_UOPS_MISC_RETIRED_LLC_MISS_C,
L2_TRANS_DEMAND_DATA_RD_C,
L2_TRANS_RFO_C,
L2_TRANS_CODE_RD_C,
L2_TRANS_ALL_PF_C,
L2_TRANS_L1D_WB_C,
```

```
        L2_TRANS_L2_FILL_C,
        L2_TRANS_L2_WB_C,
        L2_TRANS_ALL_REQ_UESTS_C,
        L2_LINES_IN_I_C,
        L2_LINES_IN_S_C,
        L2_LINES_IN_E_C,
        L2_LINES_IN_ALL_C,
        L2_LINES_OUT_DEMAND_CLEAN_C,
        L2_LINES_OUT_DEMAND_DIRTY_C,
        L2_LINES_OUT_DEMAND_PF_CLEAN_C,
        L2_LINES_OUT_DEMAND_PF_DIRTY_C, // last execution end
        L2_LINES_OUT_DEMAND_DIRTY_ALL_C, //#216 should not be used as parameter at _channel_3
-> use 214
        SQ_MISC_SPLIT_LOCK_C, //217
        EVENTS
    };

    enum Flags {
        NONE,
        INT
    };
    //first test 12. last normal test 212. last test 214
    static const unsigned int _channel_3 = 26; // sum 4 after each execution
    static const unsigned int _channel_4 = _channel_3+1;
    static const unsigned int _channel_5 = _channel_3+2;
    static const unsigned int _channel_6 = _channel_3+3;

protected:
    PMU_Common() {}
};

__END_SYS

#ifdef __PMU_H
#include __PMU_H
#endif

#endif
```

include/scheduler.h
- Configuração dos escalonadores com monitoramento habilitado;
- método init

```
// EPOS Scheduler Component Declarations

#ifndef __scheduler_h
#define __scheduler_h

#include <utility/list.h>
#include <cpu.h>
#include <machine.h>
#include <pmu.h>
```

```
__BEGIN_SYS

// All scheduling criteria, or disciplines, must define operator int() with
// the semantics of returning the desired order of a given object within the
// scheduling list
namespace Scheduling_Criteria
{
    // Priority (static and dynamic)
    class Priority
    {
        friend class _SYS::RT_Thread;

    public:
        enum {
            MAIN   = 0,
            HIGH   = 1,
            NORMAL = (unsigned(1) << (sizeof(int) * 8 - 1)) - 3,
            LOW    = (unsigned(1) << (sizeof(int) * 8 - 1)) - 2,
            IDLE   = (unsigned(1) << (sizeof(int) * 8 - 1)) - 1
        };

        static const bool timed = false;
        static const bool dynamic = false;
        static const bool preemptive = true;
        static const bool monitoring = false; // enables monitoring

    public:
        Priority(int p = NORMAL): _priority(p) {}

        operator const volatile int() const volatile { return _priority; }
        static void init() {} //created to configure and initialize pmu
        void update() {}
        unsigned int queue() const { return 0; }

    protected:
        volatile int _priority;
    };
...
     class MGEDF : public GEDF
      {
      public:
          static const bool monitoring = true;
      public:
          static void init() {
            PMU::stop(0);
            PMU::stop(1);
            PMU::stop(2);
            PMU::stop(3);
            PMU::stop(4);
            PMU::stop(5);
            PMU::stop(6);

            PMU::reset(0);
```

```
            PMU::reset(1);
            PMU::reset(2);
            PMU::reset(3);
            PMU::reset(4);
            PMU::reset(5);
            PMU::reset(6);

            PMU::write(0, 0);
            PMU::write(1, 0);
            PMU::write(2, 0);
            PMU::write(3, 0);
            PMU::write(4, 0);
            PMU::write(5, 0);
            PMU::write(6, 0);

            PMU::config(3, (PMU::Event)PMU::_channel_3);
            PMU::config(4, (PMU::Event)PMU::_channel_4);
            PMU::config(5, (PMU::Event)PMU::_channel_5);
            PMU::config(6, (PMU::Event)PMU::_channel_6);

            PMU::start(0);
            PMU::start(1);
            PMU::start(2);
            PMU::start(3);
            PMU::start(4);
            PMU::start(5);
            PMU::start(6);
         }
          MGEDF(int p = APERIODIC)
          : GEDF(p) {} // Aperiodic

          MGEDF(const Microsecond & d, const Microsecond & p = SAME, const Microsecond & c =
UNKNOWN, int cpu = ANY)
          : GEDF(d, p, c, cpu) {}

          static unsigned int queue() { return current_head(); }
          static unsigned int current_head() { return Machine::cpu_id(); }
      };
...
     class MPEDF : public PEDF
      {
       enum { ANY = Variable_Queue::ANY };
      public:
          static const bool monitoring = true;
          static const bool power_cap = true;
      public:
          static void init() {
            PMU::stop(0);
            PMU::stop(1);
            PMU::stop(2);
            PMU::stop(3);
            PMU::stop(4);
            PMU::stop(5);
            PMU::stop(6);
```

```
            PMU::reset(0);
            PMU::reset(1);
            PMU::reset(2);
            PMU::reset(3);
            PMU::reset(4);
            PMU::reset(5);
            PMU::reset(6);

            PMU::write(0, 0);
            PMU::write(1, 0);
            PMU::write(2, 0);
            PMU::write(3, 0);
            PMU::write(4, 0);
            PMU::write(5, 0);
            PMU::write(6, 0);

            PMU::config(3, (PMU::Event)PMU::_channel_3);
            PMU::config(4, (PMU::Event)PMU::_channel_4);
            PMU::config(5, (PMU::Event)PMU::_channel_5);
            PMU::config(6, (PMU::Event)PMU::_channel_6);

            PMU::start(0);
            PMU::start(1);
            PMU::start(2);
            PMU::start(3);
            PMU::start(4);
            PMU::start(5);
            PMU::start(6);
         }
          MPEDF(int p = APERIODIC)
          : PEDF(p) {} // Aperiodic

          MPEDF(const Microsecond & d, const Microsecond & p = SAME, const Microsecond & c =
UNKNOWN, int cpu = ANY)
          : PEDF(d, p, c, cpu) {}

          using Variable_Queue::queue;

          static unsigned int current_head() { return Machine::cpu_id(); }
      };
...
     class MCEDF : public CEDF
      {
        enum { ANY = Variable_Queue::ANY };
      public:
          static const bool monitoring = true;
      public:
          static void init() {
            PMU::stop(0);
            PMU::stop(1);
            PMU::stop(2);
            PMU::stop(3);
            PMU::stop(4);
```

```
            PMU::stop(5);
            PMU::stop(6);

            PMU::reset(0);
            PMU::reset(1);
            PMU::reset(2);
            PMU::reset(3);
            PMU::reset(4);
            PMU::reset(5);
            PMU::reset(6);

            PMU::write(0, 0);
            PMU::write(1, 0);
            PMU::write(2, 0);
            PMU::write(3, 0);
            PMU::write(4, 0);
            PMU::write(5, 0);
            PMU::write(6, 0);

            PMU::config(3, (PMU::Event)PMU::_channel_3);
            PMU::config(4, (PMU::Event)PMU::_channel_4);
            PMU::config(5, (PMU::Event)PMU::_channel_5);
            PMU::config(6, (PMU::Event)PMU::_channel_6);

            PMU::start(0);
            PMU::start(1);
            PMU::start(2);
            PMU::start(3);
            PMU::start(4);
            PMU::start(5);
            PMU::start(6);
         }
          MCEDF(int p = APERIODIC)
          : CEDF(p) {} // Aperiodic

          MCEDF(const Microsecond & d, const Microsecond & p = SAME, const Microsecond & c =
UNKNOWN, int cpu = ANY)
          : CEDF(d, p, c, cpu) {}
      };
}
...
template<typename T>
class Scheduling_Queue<T, Scheduling_Criteria::MGEDF>:
public Multihead_Scheduling_List<T> {};

template<typename T>
class Scheduling_Queue<T, Scheduling_Criteria::MPEDF>:
public Scheduling_Multilist<T> {};

template<typename T>
class Scheduling_Queue<T, Scheduling_Criteria::MCEDF>:
public Multihead_Scheduling_Multilist<T> {};
```

```
include/thread.h:
- adição de variáveis para o funcionamento do sistema de capturas;
```

```
// EPOS Thread Component Declarations

#ifndef __thread_h
#define __thread_h

#include <utility/queue.h>
#include <utility/handler.h>
#include <cpu.h>
#include <machine.h>
#include <system.h>
#include <scheduler.h>
#include &lt;segment.h&gt;
#include <architecture/ia32/monitoring_capture.h>
extern "C" { void __exit(); }

__BEGIN_SYS

class Thread
{
...
public:
    volatile int _missed_deadlines; //it's not unsigned because of the calculating process
    int _times_p_count; //counts the number of executions to be used on deadline misses calc
    static Monitoring_Capture* _thread_monitor; //system monitor
    static volatile bool _end_capture; //stops capture when True
    static unsigned int _global_deadline_misses;
    static unsigned int _prev_global_deadline_misses; //used to calc the difference when using
heuristics

};
...
```

```
include/periodic_thread.h:
- calculates _times_p_count, ou seja, o número de vezes que já terminou uma execução.
- captura de um momento dentro do wait_next()
```

```
// EPOS Periodic Thread Component Declarations

// Periodic threads are achieved by programming an alarm handler to invoke
// p() on a control semaphore after each job (i.e. task activation). Base
// threads are created in BEGINNING state, so the scheduler won't dispatch
// them before the associate alarm and semaphore are created. The first job
// is dispatched by resume() (thus the _state = SUSPENDED statement)

#ifndef __periodic_thread_h
#define __periodic_thread_h

#include <utility/handler.h>
#include <thread.h>
#include <alarm.h>
```

```
__BEGIN_SYS

// Aperiodic Thread
typedef Thread Aperiodic_Thread;

// Periodic Thread
class Periodic_Thread: public Thread
{
...
template<typename ... Tn>
    Periodic_Thread(const Configuration & conf, int (* entry)(Tn ...), Tn ... an)
    : Thread(Thread::Configuration(SUSPENDED, (conf.criterion != NORMAL) ? conf.criterion :
Criterion(conf.period), conf.color, conf.task, conf.stack_size), entry, an ...),
      _semaphore(0), _handler(&_semaphore, this), _alarm(conf.period, &_handler, conf.times) {
        if (Criterion::monitoring) {
            _times_p_count = conf.times;
            _alarm_times = &_alarm;
        }
        if((conf.state == READY) || (conf.state == RUNNING)) {
            _state = SUSPENDED;
            resume();
        } else
            _state = conf.state;
    }

    const Microsecond & period() const { return _alarm.period(); }
    void period(const Microsecond & p) { _alarm.period(p); }

    static volatile bool wait_next() {

        Periodic_Thread * t = reinterpret_cast<Periodic_Thread *>(running());
        //calc ddl misses and capture
        if (Criterion::monitoring && !_end_capture) {
            unsigned int entry_temp = CPU::temperature();
            t->_missed_deadlines = t->_times_p_count - (t->_alarm_times->_times);

            unsigned long long pkg = CPU::pkg_energy_status();
            unsigned long long pp0 = CPU::pp0_energy_status();
            _thread_monitor->capture(entry_temp, PMU::read(0), PMU::read(1), PMU::read(2),
PMU::read(3), PMU::read(4), PMU::read(5),
                PMU::read(6), reinterpret_cast<volatile unsigned int>(t), t->priority(),
Machine::cpu_id(), t->_missed_deadlines, t->_global_deadline_misses, pkg, pp0);
        }
        //end capture code

        if(t->_alarm._times) {
            t->_semaphore.p();
            //db<Thread>(WRN)<<"times: "<<t->_alarm._times<<endl;
            t->_times_p_count--;
        }

        return t->_alarm._times;
    }
```

```
...
};
```

```
include/system/types.h:
- adicionados os escalonadores criados como um tipo.
```

```
// EPOS Internal Type Management System

typedef __SIZE_TYPE__ size_t;

#ifndef __types_h
#define __types_h
...
template<typename> class Scheduler;
namespace Scheduling_Criteria
{
    class Priority;
    class FCFS;
    class RR;
    class RM;
    class DM;
    class EDF;
    class GRR;
    class CPU_Affinity;
    class GEDF;
    class MGEDF; //added scheduler entry
    class PEDF;
    class MPEDF; //added scheduler entry
    class CEDF;
    class MCEDF; //added scheduler entry
    class PRM;
};
...
#endif
```

```
src/component/thread.cc:
- Captura de um momento dentro do dispatch;
- Mudança do código de finalização do sistema no método idle para imprimir os dados antes de
desligar o computador;
```

```
// EPOS Thread Component Implementation

#include <machine.h>
#include <system.h>
#include <thread.h>
#include <alarm.h> // for FCFS

// This_Thread class attributes
__BEGIN_UTIL
bool This_Thread::_not_booting;
__END_UTIL
```

```
__BEGIN_SYS

// Class attributes
Monitoring_Capture* Thread::_thread_monitor;
unsigned int Thread::_global_deadline_misses;
unsigned int Thread::_prev_global_deadline_misses;
volatile unsigned int Thread::_thread_count;
volatile bool Thread::_end_capture;
...
void Thread::dispatch(Thread * prev, Thread * next, bool charge)
{

    if(charge) {
        if(Criterion::timed)
            _timer->reset();

        //Monitoring Capture
        if(Criterion::monitoring && !_end_capture) {
            unsigned int entry_temp = CPU::temperature();
            if (prev->priority() != IDLE && prev->priority() != MAIN) {
                prev->_missed_deadlines = prev->_times_p_count - (prev->_alarm_times->_times +
1);
            } else
                 prev->_missed_deadlines = 0;

            if (next->priority() != IDLE && next->priority() != MAIN) {
                next->_missed_deadlines = next->_times_p_count - (next->_alarm_times->_times +
1);
            } else
                next->_missed_deadlines = 0;

            if (prev->_missed_deadlines < 0)
                    prev->_missed_deadlines = 0;

            if (next->_missed_deadlines < 0)
                    next->_missed_deadlines = 0;

            unsigned long long ts = _thread_monitor->last_capture(Machine::cpu_id(), 7);
            unsigned long long ts_dif = _thread_monitor->time() - ts;
            if (ts == 0) {
                ts_dif = 5000;
            }
            unsigned int cpu = Machine::cpu_id();
            if (cpu % 2) {
                cpu--;
            }
            unsigned long long pkg = CPU::pkg_energy_status();
            unsigned long long pp0 = CPU::pp0_energy_status();
            //unsigned long long energy_unit = CPU::rapl_energy_unit(); // use only to know the
power unit value
            _thread_monitor->capture(entry_temp, PMU::read(0), PMU::read(1), PMU::read(2),
PMU::read(3), PMU::read(4), PMU::read(5),
            PMU::read(6), reinterpret_cast<volatile unsigned int>(prev), prev->priority(),
```

```
Machine::cpu_id(), prev->_missed_deadlines, prev->_global_deadline_misses, pkg, pp0);
            // capture next -> not necessary, it's possible to infer most of the data
            // _thread_monitor->capture(entry_temp, PMU::read(0), PMU::read(1), PMU::read(2),
PMU::read(3), PMU::read(4), PMU::read(5),
            // PMU::read(6), reinterpret_cast<volatile unsigned int>(next), next->priority(),
Machine::cpu_id(), next->_missed_deadlines, next->_global_deadline_misses);
        }
    }

    if(prev != next) {
        if(prev->_state == RUNNING)
            prev->_state = READY;
        next->_state = RUNNING;

        db<Thread>(TRC) << "Thread::dispatch(prev=" << prev << ",next=" << next << ")" << endl;
        db<Thread>(INF) << "prev={" << prev << ",ctx=" << *prev->_context << "}" << endl;
        db<Thread>(INF) << "next={" << next << ",ctx=" << *next->_context << "}" << endl;

        if(smp)
            _lock.release();

        if(multitask && (next->_task != prev->_task))
            next->_task->activate();

        CPU::switch_context(&prev->_context, next->_context);
    } else
        if(smp)
            _lock.release();

    // TODO: could this be moved to right after the switch_context?
    CPU::int_enable();
}

int Thread::idle()
{
    while(_thread_count > Machine::n_cpus()) { // someone else besides idles
        if(Traits<Thread>::trace_idle)
            db<Thread>(TRC) << "Thread::idle(CPU=" << Machine::cpu_id() << ",this=" <<
running() << ")" << endl;
        CPU::int_enable();
        CPU::halt();
        if(_scheduler.schedulables() > 0)// A thread might have been woken up by another CPU
            yield();
    }

    CPU::int_disable();
    if(Machine::cpu_id() == 0) {
        if (Criterion::monitoring) {
            db<Thread>(WRN) << "temp: "<< CPU::temperature() <<endl;
            _thread_monitor->datas();
        }
        db<Thread>(WRN) << "The last thread has exited!" << endl;

        if(reboot) {
```

```
            db<Thread>(WRN) << "Rebooting the machine ..." << endl;
            Machine::reboot();
        } else
            db<Thread>(WRN) << "Halting the machine ..." << endl;
    }
    CPU::halt();

    return 0;
}
...
```

```
src/component/thread_init.cc:
- aloca memória e inicializa o sistema de capturas
```

```
// EPOS Thread Component Initialization

#include <system.h>
#include <thread.h>
#include <alarm.h>
#include <clock.h>
#include &lt;segment.h&gt;
#include <mmu.h>

__BEGIN_SYS

void Thread::init()
{
    // The installation of the scheduler timer handler must precede the
    // creation of threads, since the constructor can induce a reschedule
    // and this in turn can call timer->reset()
    // Letting reschedule() happen during thread creation is harmless, since
    // MAIN is created first and dispatch won't replace it nor by itself
    // neither by IDLE (which has a lower priority)
    if(Criterion::timed && (Machine::cpu_id() == 0))
        _timer = new (SYSTEM) Scheduler_Timer(QUANTUM, time_slicer);

    // Install an interrupt handler to receive forced reschedules
    if(smp) {
        if(Machine::cpu_id() == 0)
            db<Thread>(WRN)<<"DATE:: "<<RTC::date()<<endl;
        if(Machine::cpu_id() == 0)
            IC::int_vector(IC::INT_RESCHEDULER, rescheduler);
        IC::enable(IC::INT_RESCHEDULER);
    }
    // Checks if monitoring execution is enable
    if (Criterion::monitoring) {
        // Only one CPU needs to initialize the Monitoring Capture System
        if (Machine::cpu_id() == 0) {
            _end_capture = true;
            // Allocating a memory region (117MB) for store the capture
            // For this to work, we increased heap size in 120MB. --> This can handle 1024000
captures
```

```
            _thread_monitor = new Monitoring_Capture(128000, new (SYSTEM) Moment[1024000]);
        }
    }

    Criterion::init();
}

__END_SYS
```

```
- src/component/semaphore.cc:
       - register global_deadline_misses
```

```
// EPOS Semaphore Component Implementation

#include <semaphore.h>

__BEGIN_SYS

Semaphore::Semaphore(int v): _value(v)
{
    db<Synchronizer>(TRC) << "Semaphore(value=" << _value << ") => " << this << endl;
}


Semaphore::~Semaphore()
{
    db<Synchronizer>(TRC) << "~Semaphore(this=" << this << ")" << endl;
}


void Semaphore::p()
{
    db<Synchronizer>(TRC) << "Semaphore::p(this=" << this << ",value=" << _value << ")" <<
endl;

    begin_atomic();
    if(fdec(_value) < 1)
        sleep(); // implicit end_atomic()
    else
        end_atomic();
}


void Semaphore::v()
{
    db<Synchronizer>(TRC) << "Semaphore::v(this=" << this << ",value=" << _value << ")" <<
endl;

    begin_atomic();
    if(finc(_value) < 0)
        wakeup();  // implicit end_atomic()
    else {
```

```
        if (Thread::Criterion::monitoring)
            if(_value > 1) // if a daadline miss has occour in this alarm (one alarm per
thread)
                Thread::self()->_global_deadline_misses++; // increase the count of global
deadline misses (deadline misses in all threads)
        end_atomic();
    }
}

__END_SYS
```

src/architecture/ia32/pmu.cc:
- Configuração dos eventos númerados no vetor:
 - CPU::Reg32 Intel_Sandy_Bridge_PMU::_events[PMU_Common::EVENTS]

```
// Adds the address of the events in Intel Sandy Bredige Processors to the enum Events
const CPU::Reg32 Intel_Sandy_Bridge_PMU::_events[PMU_Common::EVENTS] = {
                        /* CLOCK              */ UNHALTED_CORE_CYCLES,
                        /* DVS_CLOCK          */ UNHALTED_REFERENCE_CYCLES,
                        /* INSTRUCTIONS       */ INSTRUCTIONS_RETIRED,
                        /* BRANCHES           */ BRANCH_INSTRUCTIONS_RETIRED,
                        /* BRANCH_MISSES      */ BRANCH_MISSES_RETIRED,
                        /* L1_HIT             */ 0,
                        /* L2_HIT             */ 0,
                        /* L3_HIT             */ LLC_REFERENCES,
                        /* L1_MISS            */ 0,
                        /* L2_MISS            */ 0,
                        /* L3_MISS            */ LLC_MISSES,
                        /*LD_BLOCKS_DATA_UNKNOWN_C*/
Intel_Sandy_Bridge_PMU::LD_BLOCKS_DATA_UNKNOWN,
                                            /*LD_BLOCKS_STORE_FORWARD_C*/
Intel_Sandy_Bridge_PMU::LD_BLOCKS_STORE_FORWARD,
                                            /*LD_BLOCKS_NO_SR_C*/
Intel_Sandy_Bridge_PMU::LD_BLOCKS_NO_SR,
                                            /*LD_BLOCKS_ALL_BLOCK_C*/
Intel_Sandy_Bridge_PMU::LD_BLOCKS_ALL_BLOCK,
                                            /*MISALIGN_MEM_REF_LOADS_C*/
Intel_Sandy_Bridge_PMU::MISALIGN_MEM_REF_LOADS,
                                            /*MISALIGN_MEM_REF_STORES_C*/
Intel_Sandy_Bridge_PMU::MISALIGN_MEM_REF_STORES,
                                            /*LD_BLOCKS_PARTIAL_ADDRESS_ALIAS_C*/
Intel_Sandy_Bridge_PMU::LD_BLOCKS_PARTIAL_ADDRESS_ALIAS,
                                            /*LD_BLOCKS_PARTIAL_ALL_STA_BLCOK_C*/
Intel_Sandy_Bridge_PMU::LD_BLOCKS_PARTIAL_ALL_STA_BLCOK,
                                            /*DTLB_LOAD_MISSES_MISS_CAUSES_A_WALK_C*/
Intel_Sandy_Bridge_PMU::DTLB_LOAD_MISSES_MISS_CAUSES_A_WALK,
                                            /*DTLB_LOAD_MISSES_MISS_WALK_COMPLETED_C*/
Intel_Sandy_Bridge_PMU::DTLB_LOAD_MISSES_MISS_WALK_COMPLETED,
                                            /*DTLB_LOAD_MISSES_MISS_WALK_DURATION_C*/
Intel_Sandy_Bridge_PMU::DTLB_LOAD_MISSES_MISS_WALK_DURATION,
                                            /*DTLB_LOAD_MISSES_MISS_STLB_HIT_C*/
Intel_Sandy_Bridge_PMU::DTLB_LOAD_MISSES_MISS_STLB_HIT,
```

```
                                            /*INT_MISC_RECOVERY_CYCLES_C*/
Intel_Sandy_Bridge_PMU::INT_MISC_RECOVERY_CYCLES,
                                            /*INT_MISC_RAT_STALL_CYCLES_C*/
Intel_Sandy_Bridge_PMU::INT_MISC_RAT_STALL_CYCLES,
                                            /*UOPS_ISSUED_ANY_C*/
Intel_Sandy_Bridge_PMU::UOPS_ISSUED_ANY,
                                            /*FP_COMP_OPS_EXE_X87_C*/
Intel_Sandy_Bridge_PMU::FP_COMP_OPS_EXE_X87,
                                            /*FP_COMP_OPS_EXE_SSE_FP_PACKED_DOUBLE_C*/
Intel_Sandy_Bridge_PMU::FP_COMP_OPS_EXE_SSE_FP_PACKED_DOUBLE,
                                            /*FP_COMP_OPS_EXE_SSE_FP_SCALAR_SINGLE_C*/
Intel_Sandy_Bridge_PMU::FP_COMP_OPS_EXE_SSE_FP_SCALAR_SINGLE,
                                            /*FP_COMP_OPS_EXE_SSE_PACKED_SINGLE_C*/
Intel_Sandy_Bridge_PMU::FP_COMP_OPS_EXE_SSE_PACKED_SINGLE,
                                            /*FP_COMP_OPS_EXE_SSE_SCALAR_DOUBLE_C*/
Intel_Sandy_Bridge_PMU::FP_COMP_OPS_EXE_SSE_SCALAR_DOUBLE,
                                            /*SIMD_FP_256_PACKED_SINGLE_C*/
Intel_Sandy_Bridge_PMU::SIMD_FP_256_PACKED_SINGLE,
                                            /*SIMD_FP_256_PACKED_DOUBLE_C*/
Intel_Sandy_Bridge_PMU::SIMD_FP_256_PACKED_DOUBLE,
                                            /*ARITH_FPU_DIV_ACTIVE_C*/
Intel_Sandy_Bridge_PMU::ARITH_FPU_DIV_ACTIVE,
                                            /*INSTS_WRITTEN_TO_IQ_INSTS_C*/
Intel_Sandy_Bridge_PMU::INSTS_WRITTEN_TO_IQ_INSTS,
                                            /*L2_RQSTS_DEMAND_DATA_RD_HIT_C*/
Intel_Sandy_Bridge_PMU::L2_RQSTS_DEMAND_DATA_RD_HIT,
                                            /*L2_RQSTS_ALL_DEMAND_DATA_RD_C*/
Intel_Sandy_Bridge_PMU::L2_RQSTS_ALL_DEMAND_DATA_RD,
                                            /*L2_RQSTS_RFO_HITS_C*/
Intel_Sandy_Bridge_PMU::L2_RQSTS_RFO_HITS,
                                            /*L2_RQSTS_RFO_MISS_C*/
Intel_Sandy_Bridge_PMU::L2_RQSTS_RFO_MISS,
                                            /*L2_RQSTS_ALL_RFO_C*/
Intel_Sandy_Bridge_PMU::L2_RQSTS_ALL_RFO,
                                            /*L2_RQSTS_CODE_RD_HIT_C*/
Intel_Sandy_Bridge_PMU::L2_RQSTS_CODE_RD_HIT,
                                            /*L2_RQSTS_CODE_RD_MISS_C*/
Intel_Sandy_Bridge_PMU::L2_RQSTS_CODE_RD_MISS,
                                            /*L2_RQSTS_ALL_CODE_RD_C*/
Intel_Sandy_Bridge_PMU::L2_RQSTS_ALL_CODE_RD,
                                            /*L2_RQSTS_PF_HIT_C*/
Intel_Sandy_Bridge_PMU::L2_RQSTS_PF_HIT,
                                            /*L2_RQSTS_PF_MISS_C*/
Intel_Sandy_Bridge_PMU::L2_RQSTS_PF_MISS,
                                            /*L2_RQSTS_ALL_PF_C*/
Intel_Sandy_Bridge_PMU::L2_RQSTS_ALL_PF,
                                            /*L2_STORE_LOCK_RQSTS_MISS_C*/
Intel_Sandy_Bridge_PMU::L2_STORE_LOCK_RQSTS_MISS,
                                            /*L2_STORE_LOCK_RQSTS_HIT_E_C*/
Intel_Sandy_Bridge_PMU::L2_STORE_LOCK_RQSTS_HIT_E,
                                            /*L2_STORE_LOCK_RQSTS_HIT_M_C*/
Intel_Sandy_Bridge_PMU::L2_STORE_LOCK_RQSTS_HIT_M,
                                            /*L2_STORE_LOCK_RQSTS_ALL_C*/
Intel_Sandy_Bridge_PMU::L2_STORE_LOCK_RQSTS_ALL,
```

```
                                            /*L2_L1D_WB_RQSTS_HIT_E_C*/
Intel_Sandy_Bridge_PMU::L2_L1D_WB_RQSTS_HIT_E,
                                            /*L2_L1D_WB_RQSTS_HIT_M_C*/
Intel_Sandy_Bridge_PMU::L2_L1D_WB_RQSTS_HIT_M,
                                            /*LONGEST_LAT_CACHE_REFERENCE_C*/
Intel_Sandy_Bridge_PMU::LONGEST_LAT_CACHE_REFERENCE,
                                            /*LONGEST_LAT_CACHE_MISS_C*/
Intel_Sandy_Bridge_PMU::LONGEST_LAT_CACHE_MISS,
                                            /*CPU_CLK_UNHALTED_THREAD_P_C*/
Intel_Sandy_Bridge_PMU::CPU_CLK_UNHALTED_THREAD_P,
                                            /*CPU_CLK_THREAD_UNHALTED_REF_XCLK_C*/
Intel_Sandy_Bridge_PMU::CPU_CLK_THREAD_UNHALTED_REF_XCLK,
                                            /*L1D_PEND_MISS_PENDING_C*/
Intel_Sandy_Bridge_PMU::L1D_PEND_MISS_PENDING,
                                            /*DTLB_STORE_MISSES_MISS_CAUSES_A_WALK_C*/
Intel_Sandy_Bridge_PMU::DTLB_STORE_MISSES_MISS_CAUSES_A_WALK,
                                            /*DTLB_STORE_MISSES_WALK_COMPLETED_C*/
Intel_Sandy_Bridge_PMU::DTLB_STORE_MISSES_WALK_COMPLETED,
                                            /*DTLB_STORE_MISSES_WALK_DURATION_C*/
Intel_Sandy_Bridge_PMU::DTLB_STORE_MISSES_WALK_DURATION,
                                            /*DTLB_STORE_MISSES_TLB_HIT_C*/
Intel_Sandy_Bridge_PMU::DTLB_STORE_MISSES_TLB_HIT,
                                            /*LOAD_HIT_PRE_SW_PF_C*/
Intel_Sandy_Bridge_PMU::LOAD_HIT_PRE_SW_PF,
                                            /*LOAD_HIT_PREHW_PF_C*/
Intel_Sandy_Bridge_PMU::LOAD_HIT_PREHW_PF,
                                            /*HW_PRE_REQ_DL1_MISS_C*/
Intel_Sandy_Bridge_PMU::HW_PRE_REQ_DL1_MISS,
                                            /*L1D_REPLACEMENT_C*/
Intel_Sandy_Bridge_PMU::L1D_REPLACEMENT,
                                            /*L1D_ALLOCATED_IN_M_C*/
Intel_Sandy_Bridge_PMU::L1D_ALLOCATED_IN_M,
                                            /*L1D_EVICTION_C*/
Intel_Sandy_Bridge_PMU::L1D_EVICTION,
                                            /*L1D_ALL_M_REPLACEMENT_C*/
Intel_Sandy_Bridge_PMU::L1D_ALL_M_REPLACEMENT,
                                            /*PARTIAL_RAT_STALLS_FLAGS_MERGE_UOP_C*/
Intel_Sandy_Bridge_PMU::PARTIAL_RAT_STALLS_FLAGS_MERGE_UOP,
                                            /*PARTIAL_RAT_STALLS_SLOW_LEA_WINDOW_C*/
Intel_Sandy_Bridge_PMU::PARTIAL_RAT_STALLS_SLOW_LEA_WINDOW,
                                            /*PARTIAL_RAT_STALLS_MUL_SINGLE_UOP_C*/
Intel_Sandy_Bridge_PMU::PARTIAL_RAT_STALLS_MUL_SINGLE_UOP,
                                            /*RESOURCE_STALLS2_ALL_FL_EMPTY_C*/
Intel_Sandy_Bridge_PMU::RESOURCE_STALLS2_ALL_FL_EMPTY,
                                            /*RESOURCE_STALLS2_ALL_PRF_CONTROL_C*/
Intel_Sandy_Bridge_PMU::RESOURCE_STALLS2_ALL_PRF_CONTROL,
                                            /*RESOURCE_STALLS2_BOB_FULL_C*/
Intel_Sandy_Bridge_PMU::RESOURCE_STALLS2_BOB_FULL,
                                            /*RESOURCE_STALLS2_OOO_RSRC_C*/
Intel_Sandy_Bridge_PMU::RESOURCE_STALLS2_OOO_RSRC,
                                            /*CPL_CYCLES_RING0_C*/
Intel_Sandy_Bridge_PMU::CPL_CYCLES_RING0,
                                            /*CPL_CYCLES_RING123_C*/
Intel_Sandy_Bridge_PMU::CPL_CYCLES_RING123,
```

```
                                            /*RS_EVENTS_EMPTY_CYCLES_C*/
Intel_Sandy_Bridge_PMU::RS_EVENTS_EMPTY_CYCLES,
                                            /*OFFCORE_REQUESTS_OUTSTANDING_DEMAND_DATA_RD_C*/
Intel_Sandy_Bridge_PMU::OFFCORE_REQUESTS_OUTSTANDING_DEMAND_DATA_RD,
                                            /*OFFCORE_REQUESTS_OUTSTANDING_DEMAND_RFO_C*/
Intel_Sandy_Bridge_PMU::OFFCORE_REQUESTS_OUTSTANDING_DEMAND_RFO,
                                            /*OFFCORE_REQUESTS_OUTSTANDING_ALL_DATA_RD_C*/
Intel_Sandy_Bridge_PMU::OFFCORE_REQUESTS_OUTSTANDING_ALL_DATA_RD,
                                            /*LOCK_CYCLES_SPLIT_LOCK_UC_LOCK_DURATION_C*/
Intel_Sandy_Bridge_PMU::LOCK_CYCLES_SPLIT_LOCK_UC_LOCK_DURATION,
                                            /*LOCK_CYCLES_CACHE_LOCK_DURATION_C*/
Intel_Sandy_Bridge_PMU::LOCK_CYCLES_CACHE_LOCK_DURATION,
                                            /*IDQ_EMPTY_C*/ Intel_Sandy_Bridge_PMU::IDQ_EMPTY,
                                            /*IDQ_MITE_UOPS_C*/
Intel_Sandy_Bridge_PMU::IDQ_MITE_UOPS,
                                            /*IDQ_DSB_UOPS_C*/
Intel_Sandy_Bridge_PMU::IDQ_DSB_UOPS,
                                            /*IDQ_MS_DSB_UOPS_C*/
Intel_Sandy_Bridge_PMU::IDQ_MS_DSB_UOPS,
                                            /*IDQ_MS_MITE_UOPS_C*/
Intel_Sandy_Bridge_PMU::IDQ_MS_MITE_UOPS,
                                            /*IDQ_MS_UOPS_C*/
Intel_Sandy_Bridge_PMU::IDQ_MS_UOPS,
                                            /*ICACHE_MISSES_C*/
Intel_Sandy_Bridge_PMU::ICACHE_MISSES,
                                            /*ITLB_MISSES_MISS_CAUSES_A_WALK_C*/
Intel_Sandy_Bridge_PMU::ITLB_MISSES_MISS_CAUSES_A_WALK,
                                            /*ITLB_MISSES_WALK_COMPLETED_C*/
Intel_Sandy_Bridge_PMU::ITLB_MISSES_WALK_COMPLETED,
                                            /*ITLB_MISSES_WALK_DURATION_C*/
Intel_Sandy_Bridge_PMU::ITLB_MISSES_WALK_DURATION,
                                            /*ITLB_MISSES_STLB_HIT_C*/
Intel_Sandy_Bridge_PMU::ITLB_MISSES_STLB_HIT,
                                            /*ILD_STALL_LCP_C*/
Intel_Sandy_Bridge_PMU::ILD_STALL_LCP,
                                            /*ILD_STALL_IQ_FULL_C*/
Intel_Sandy_Bridge_PMU::ILD_STALL_IQ_FULL,
                                            /*BR_INST_EXEC_COND_C*/
Intel_Sandy_Bridge_PMU::BR_INST_EXEC_COND,
                                            /*BR_INST_EXEC_DIRECT_JMP_C*/
Intel_Sandy_Bridge_PMU::BR_INST_EXEC_DIRECT_JMP,
                                            /*BR_INST_EXEC_INDIRECT_JMP_NON_CALL_RET_C*/
Intel_Sandy_Bridge_PMU::BR_INST_EXEC_INDIRECT_JMP_NON_CALL_RET,
                                            /*BR_INST_EXEC_RETURN_NEAR_C*/
Intel_Sandy_Bridge_PMU::BR_INST_EXEC_RETURN_NEAR,
                                            /*BR_INST_EXEC_DIRECT_NEAR_CALL_C*/
Intel_Sandy_Bridge_PMU::BR_INST_EXEC_DIRECT_NEAR_CALL,
                                            /*BR_INST_EXEC_INDIRECT_NEAR_CALL_C*/
Intel_Sandy_Bridge_PMU::BR_INST_EXEC_INDIRECT_NEAR_CALL,
                                            /*BR_INST_EXEC_NON_TAKEN_C*/
Intel_Sandy_Bridge_PMU::BR_INST_EXEC_NON_TAKEN,
                                            /*BR_INST_EXEC_TAKEN_C*/
Intel_Sandy_Bridge_PMU::BR_INST_EXEC_TAKEN,
                                            /*BR_INST_EXEC_ALL_BRANCHES_C*/
```

```
Intel_Sandy_Bridge_PMU::BR_INST_EXEC_ALL_BRANCHES,
                                            /*BR_MISP_EXEC_COND_C*/
Intel_Sandy_Bridge_PMU::BR_MISP_EXEC_COND,
                                            /*BR_MISP_EXEC_INDIRECT_JMP_NON_CALL_RET_C*/
Intel_Sandy_Bridge_PMU::BR_MISP_EXEC_INDIRECT_JMP_NON_CALL_RET,
                                            /*BR_MISP_EXEC_RETURN_NEAR_C*/
Intel_Sandy_Bridge_PMU::BR_MISP_EXEC_RETURN_NEAR,
                                            /*BR_MISP_EXEC_DIRECT_NEAR_CALL_C*/
Intel_Sandy_Bridge_PMU::BR_MISP_EXEC_DIRECT_NEAR_CALL,
                                            /*BR_MISP_EXEC_INDIRECT_NEAR_CALL_C*/
Intel_Sandy_Bridge_PMU::BR_MISP_EXEC_INDIRECT_NEAR_CALL,
                                            /*BR_MISP_EXEC_NON_TAKEN_C*/
Intel_Sandy_Bridge_PMU::BR_MISP_EXEC_NON_TAKEN,
                                            /*BR_MISP_EXEC_TAKEN_C*/
Intel_Sandy_Bridge_PMU::BR_MISP_EXEC_TAKEN,
                                            /*BR_MISP_EXEC_ALL_BRANCHES_C*/
Intel_Sandy_Bridge_PMU::BR_MISP_EXEC_ALL_BRANCHES,
                                            /*IDQ_UOPS_NOT_DELIVERED_CORE_C*/
Intel_Sandy_Bridge_PMU::IDQ_UOPS_NOT_DELIVERED_CORE,
                                            /*UOPS_DISPATCHED_PORT_PORT_0_C*/
Intel_Sandy_Bridge_PMU::UOPS_DISPATCHED_PORT_PORT_0,
                                            /*UOPS_DISPATCHED_PORT_PORT_1_C*/
Intel_Sandy_Bridge_PMU::UOPS_DISPATCHED_PORT_PORT_1,
                                            /*UOPS_DISPATCHED_PORT_PORT_2_LD_C*/
Intel_Sandy_Bridge_PMU::UOPS_DISPATCHED_PORT_PORT_2_LD,
                                            /*UOPS_DISPATCHED_PORT_PORT_2_STA_C*/
Intel_Sandy_Bridge_PMU::UOPS_DISPATCHED_PORT_PORT_2_STA,
                                            /*UOPS_DISPATCHED_PORT_PORT_2_C*/
Intel_Sandy_Bridge_PMU::UOPS_DISPATCHED_PORT_PORT_2,
                                            /*UOPS_DISPATCHED_PORT_PORT_3_LD_C*/
Intel_Sandy_Bridge_PMU::UOPS_DISPATCHED_PORT_PORT_3_LD,
                                            /*UOPS_DISPATCHED_PORT_PORT_3_STA_C*/
Intel_Sandy_Bridge_PMU::UOPS_DISPATCHED_PORT_PORT_3_STA,
                                            /*UOPS_DISPATCHED_PORT_PORT_3_C*/
Intel_Sandy_Bridge_PMU::UOPS_DISPATCHED_PORT_PORT_3,
                                            /*UOPS_DISPATCHED_PORT_PORT_4_C*/
Intel_Sandy_Bridge_PMU::UOPS_DISPATCHED_PORT_PORT_4,
                                            /*UOPS_DISPATCHED_PORT_PORT_5_C*/
Intel_Sandy_Bridge_PMU::UOPS_DISPATCHED_PORT_PORT_5,
                                            /*RESOURCE_STALLS_ANY_C*/
Intel_Sandy_Bridge_PMU::RESOURCE_STALLS_ANY,
                                            /*RESOURCE_STALLS_LB_C*/
Intel_Sandy_Bridge_PMU::RESOURCE_STALLS_LB,
                                            /*RESOURCE_STALLS_RS_C*/
Intel_Sandy_Bridge_PMU::RESOURCE_STALLS_RS,
                                            /*RESOURCE_STALLS_SB_C*/
Intel_Sandy_Bridge_PMU::RESOURCE_STALLS_SB,
                                            /*RESOURCE_STALLS_ROB_C*/
Intel_Sandy_Bridge_PMU::RESOURCE_STALLS_ROB,
                                            /*RESOURCE_STALLS_FCSW_C*/
Intel_Sandy_Bridge_PMU::RESOURCE_STALLS_FCSW,
                                            /*RESOURCE_STALLS_MXCSR_C*/
Intel_Sandy_Bridge_PMU::RESOURCE_STALLS_MXCSR,
                                            /*RESOURCE_STALLS_OTHER_C*/
```

```
Intel_Sandy_Bridge_PMU::RESOURCE_STALLS_OTHER,
                                            /*DSB2MITE_SWITCHES_COUNT_C*/
Intel_Sandy_Bridge_PMU::DSB2MITE_SWITCHES_COUNT,
                                            /*DSB2MITE_SWITCHES_PENALTY_CYCLES_C*/
Intel_Sandy_Bridge_PMU::DSB2MITE_SWITCHES_PENALTY_CYCLES,
                                            /*DSB_FILL_OTHER_CANCEL_C*/
Intel_Sandy_Bridge_PMU::DSB_FILL_OTHER_CANCEL,
                                            /*DSB_FILL_EXCEED_DSB_LINES_C*/
Intel_Sandy_Bridge_PMU::DSB_FILL_EXCEED_DSB_LINES,
                                            /*DSB_FILL_ALL_CANCEL_C*/
Intel_Sandy_Bridge_PMU::DSB_FILL_ALL_CANCEL,
                                            /*ITLB_ITLB_FLUSH_C*/
Intel_Sandy_Bridge_PMU::ITLB_ITLB_FLUSH,
                                            /*OFFCORE_REQUESTS_DEMAND_DATA_RD_C*/
Intel_Sandy_Bridge_PMU::OFFCORE_REQUESTS_DEMAND_DATA_RD,
                                            /*OFFCORE_REQUESTS_DEMAND_RFO_C*/
Intel_Sandy_Bridge_PMU::OFFCORE_REQUESTS_DEMAND_RFO,
                                            /*OFFCORE_REQUESTS_ALL_DATA_RD_C*/
Intel_Sandy_Bridge_PMU::OFFCORE_REQUESTS_ALL_DATA_RD,
                                            /*UOPS_DISPATCHED_THREAD_C*/
Intel_Sandy_Bridge_PMU::UOPS_DISPATCHED_THREAD,
                                            /*UOPS_DISPATCHED_CORE_C*/
Intel_Sandy_Bridge_PMU::UOPS_DISPATCHED_CORE,
                                            /*OFFCORE_REQUESTS_BUFFER_SQ_FULL_C*/
Intel_Sandy_Bridge_PMU::OFFCORE_REQUESTS_BUFFER_SQ_FULL,
                                            /*AGU_BYPASS_CANCEL_COUNT_C*/
Intel_Sandy_Bridge_PMU::AGU_BYPASS_CANCEL_COUNT,
                                            /*OFF_CORE_RESPONSE_0_C*/
Intel_Sandy_Bridge_PMU::OFF_CORE_RESPONSE_0,
                                            /*OFF_CORE_RESPONSE_1_C*/
Intel_Sandy_Bridge_PMU::OFF_CORE_RESPONSE_1,
                                            /*TLB_FLUSH_DTLB_THREAD_C*/
Intel_Sandy_Bridge_PMU::TLB_FLUSH_DTLB_THREAD,
                                            /*TLB_FLUSH_STLB_ANY_C*/
Intel_Sandy_Bridge_PMU::TLB_FLUSH_STLB_ANY,
                                            /*L1D_BLOCKS_BANK_CONFLICT_CYCLES_C*/
Intel_Sandy_Bridge_PMU::L1D_BLOCKS_BANK_CONFLICT_CYCLES,
                                            /*INST_RETIRED_ANY_P_C*/
Intel_Sandy_Bridge_PMU::INST_RETIRED_ANY_P,
                                            /*INST_RETIRED_PREC_DIST_C*/
Intel_Sandy_Bridge_PMU::INST_RETIRED_PREC_DIST,
                                            /*OTHER_ASSISTS_ITLB_MISS_RETIRED_C*/
Intel_Sandy_Bridge_PMU::OTHER_ASSISTS_ITLB_MISS_RETIRED,
                                            /*OTHER_ASSISTS_AVX_STORE_C*/
Intel_Sandy_Bridge_PMU::OTHER_ASSISTS_AVX_STORE,
                                            /*OTHER_ASSISTS_AVX_TO_SSE_C*/
Intel_Sandy_Bridge_PMU::OTHER_ASSISTS_AVX_TO_SSE,
                                            /*OTHER_ASSISTS_SSE_TO_AVX_C*/
Intel_Sandy_Bridge_PMU::OTHER_ASSISTS_SSE_TO_AVX,
                                            /*UOPS_RETIRED_ALL_C*/
Intel_Sandy_Bridge_PMU::UOPS_RETIRED_ALL,
                                            /*UOPS_RETIRED_RETIRE_SLOTS_C*/
Intel_Sandy_Bridge_PMU::UOPS_RETIRED_RETIRE_SLOTS,
                                            /*MACHINE_CLEARS_MEMORY_ORDERING_C*/
```

```
Intel_Sandy_Bridge_PMU::MACHINE_CLEARS_MEMORY_ORDERING,
                                            /*MACHINE_CLEARS_SMC_C*/
Intel_Sandy_Bridge_PMU::MACHINE_CLEARS_SMC,
                                            /*MACHINE_CLEARS_MASKMOV_C*/
Intel_Sandy_Bridge_PMU::MACHINE_CLEARS_MASKMOV,
                                            /*BR_INST_RETIRED_ALL_BRANCHES_ARCH_C*/
Intel_Sandy_Bridge_PMU::BR_INST_RETIRED_ALL_BRANCHES_ARCH,
                                            /*BR_INST_RETIRED_CONDITIONAL_C*/
Intel_Sandy_Bridge_PMU::BR_INST_RETIRED_CONDITIONAL,
                                            /*BR_INST_RETIRED_NEAR_CALL_C*/
Intel_Sandy_Bridge_PMU::BR_INST_RETIRED_NEAR_CALL,
                                            /*BR_INST_RETIRED_ALL_BRANCHES_C*/
Intel_Sandy_Bridge_PMU::BR_INST_RETIRED_ALL_BRANCHES,
                                            /*BR_INST_RETIRED_NEAR_RETURN_C*/
Intel_Sandy_Bridge_PMU::BR_INST_RETIRED_NEAR_RETURN,
                                            /*BR_INST_RETIRED_NOT_TAKEN_C*/
Intel_Sandy_Bridge_PMU::BR_INST_RETIRED_NOT_TAKEN,
                                            /*BR_INST_RETIRED_NEAR_TAKEN_C*/
Intel_Sandy_Bridge_PMU::BR_INST_RETIRED_NEAR_TAKEN,
                                            /*BR_INST_RETIRED_FAR_BRANCH_C*/
Intel_Sandy_Bridge_PMU::BR_INST_RETIRED_FAR_BRANCH,
                                            /*BR_MISP_RETIRED_ALL_BRANCHES_ARCH_C*/
Intel_Sandy_Bridge_PMU::BR_MISP_RETIRED_ALL_BRANCHES_ARCH,
                                            /*BR_MISP_RETIRED_CONDITIONAL_C*/
Intel_Sandy_Bridge_PMU::BR_MISP_RETIRED_CONDITIONAL,
                                            /*BR_MISP_RETIRED_NEAR_CALL_C*/
Intel_Sandy_Bridge_PMU::BR_MISP_RETIRED_NEAR_CALL,
                                            /*BR_MISP_RETIRED_ALL_BRANCHES_C*/
Intel_Sandy_Bridge_PMU::BR_MISP_RETIRED_ALL_BRANCHES,
                                            /*BR_MISP_RETIRED_NOT_TAKEN_C*/
Intel_Sandy_Bridge_PMU::BR_MISP_RETIRED_NOT_TAKEN,
                                            /*BR_MISP_RETIRED_TAKEN_C*/
Intel_Sandy_Bridge_PMU::BR_MISP_RETIRED_TAKEN,
                                            /*FP_ASSIST_X87_OUTPUT_C*/
Intel_Sandy_Bridge_PMU::FP_ASSIST_X87_OUTPUT,
                                            /*FP_ASSIST_X87_INPUT_C*/
Intel_Sandy_Bridge_PMU::FP_ASSIST_X87_INPUT,
                                            /*FP_ASSIST_SIMD_OUTPUT_C*/
Intel_Sandy_Bridge_PMU::FP_ASSIST_SIMD_OUTPUT,
                                            /*FP_ASSIST_SIMD_INPUT_C*/
Intel_Sandy_Bridge_PMU::FP_ASSIST_SIMD_INPUT,
                                            /*FP_ASSIST_ANY_C*/
Intel_Sandy_Bridge_PMU::FP_ASSIST_ANY,
                                            /*ROB_MISC_EVENTS_LBR_INSERTS_C*/
Intel_Sandy_Bridge_PMU::ROB_MISC_EVENTS_LBR_INSERTS,
                                            /*MEM_TRANS_RETIRED_LOAD_LATENCY_C*/
Intel_Sandy_Bridge_PMU::MEM_TRANS_RETIRED_LOAD_LATENCY,
                                            /*MEM_TRANS_RETIRED_PRECISE_STORE_C*/
Intel_Sandy_Bridge_PMU::MEM_TRANS_RETIRED_PRECISE_STORE,
                                            /*MEM_UOP_RETIRED_LOADS_C*/
Intel_Sandy_Bridge_PMU::MEM_UOP_RETIRED_LOADS,
                                            /*MEM_UOP_RETIRED_STORES_C*/
Intel_Sandy_Bridge_PMU::MEM_UOP_RETIRED_STORES,
                                            /*MEM_UOP_RETIRED_STLB_MISS_C*/
```

```
Intel_Sandy_Bridge_PMU::MEM_UOP_RETIRED_STLB_MISS,
                                            /*MEM_UOP_RETIRED_LOCK_C*/
Intel_Sandy_Bridge_PMU::MEM_UOP_RETIRED_LOCK,
                                            /*MEM_UOP_RETIRED_SPLIT_C*/
Intel_Sandy_Bridge_PMU::MEM_UOP_RETIRED_SPLIT,
                                            /*MEM_UOP_RETIRED_ALL_C*/
Intel_Sandy_Bridge_PMU::MEM_UOP_RETIRED_ALL,
                                            /*MEM_UOPS_RETIRED_ALL_LOADS_C*/
Intel_Sandy_Bridge_PMU::MEM_UOPS_RETIRED_ALL_LOADS,
                                            /*MEM_LOAD_UOPS_RETIRED_L1_HIT_C*/
Intel_Sandy_Bridge_PMU::MEM_LOAD_UOPS_RETIRED_L1_HIT,
                                            /*MEM_LOAD_UOPS_RETIRED_L2_HIT_C*/
Intel_Sandy_Bridge_PMU::MEM_LOAD_UOPS_RETIRED_L2_HIT,
                                            /*MEM_LOAD_UOPS_RETIRED_L3_HIT_C*/
Intel_Sandy_Bridge_PMU::MEM_LOAD_UOPS_RETIRED_L3_HIT,
                                            /*MEM_LOAD_UOPS_RETIRED_HIT_LFB_C*/
Intel_Sandy_Bridge_PMU::MEM_LOAD_UOPS_RETIRED_HIT_LFB,
                                            /*XSNP_MISS_C*/ Intel_Sandy_Bridge_PMU::XSNP_MISS,
                                            /*XSNP_HIT_C*/ Intel_Sandy_Bridge_PMU::XSNP_HIT,
                                            /*XSNP_HITM_C*/ Intel_Sandy_Bridge_PMU::XSNP_HITM,
                                            /*XSNP_NONE_C*/ Intel_Sandy_Bridge_PMU::XSNP_NONE,
                                            /*MEM_LOAD_UOPS_MISC_RETIRED_LLC_MISS_C*/
Intel_Sandy_Bridge_PMU::MEM_LOAD_UOPS_MISC_RETIRED_LLC_MISS,
                                            /*L2_TRANS_DEMAND_DATA_RD_C*/
Intel_Sandy_Bridge_PMU::L2_TRANS_DEMAND_DATA_RD,
                                            /*L2_TRANS_RFO_C*/
Intel_Sandy_Bridge_PMU::L2_TRANS_RFO,
                                            /*L2_TRANS_CODE_RD_C*/
Intel_Sandy_Bridge_PMU::L2_TRANS_CODE_RD,
                                            /*L2_TRANS_ALL_PF_C*/
Intel_Sandy_Bridge_PMU::L2_TRANS_ALL_PF,
                                            /*L2_TRANS_L1D_WB_C*/
Intel_Sandy_Bridge_PMU::L2_TRANS_L1D_WB,
                                            /*L2_TRANS_L2_FILL_C*/
Intel_Sandy_Bridge_PMU::L2_TRANS_L2_FILL,
                                            /*L2_TRANS_L2_WB_C*/
Intel_Sandy_Bridge_PMU::L2_TRANS_L2_WB,
                                            /*L2_TRANS_ALL_REQ_UESTS_C*/
Intel_Sandy_Bridge_PMU::L2_TRANS_ALL_REQ_UESTS,
                                            /*L2_LINES_IN_I_C*/
Intel_Sandy_Bridge_PMU::L2_LINES_IN_I,
                                            /*L2_LINES_IN_S_C*/
Intel_Sandy_Bridge_PMU::L2_LINES_IN_S,
                                            /*L2_LINES_IN_E_C*/
Intel_Sandy_Bridge_PMU::L2_LINES_IN_E,
                                            /*L2_LINES_IN_ALL_C*/
Intel_Sandy_Bridge_PMU::L2_LINES_IN_ALL,
                                            /*L2_LINES_OUT_DEMAND_CLEAN_C*/
Intel_Sandy_Bridge_PMU::L2_LINES_OUT_DEMAND_CLEAN,
                                            /*L2_LINES_OUT_DEMAND_DIRTY_C*/
Intel_Sandy_Bridge_PMU::L2_LINES_OUT_DEMAND_DIRTY,
                                            /*L2_LINES_OUT_DEMAND_PF_CLEAN_C*/
Intel_Sandy_Bridge_PMU::L2_LINES_OUT_DEMAND_PF_CLEAN,
                                            /*L2_LINES_OUT_DEMAND_PF_DIRTY_C*/
```

```
Intel_Sandy_Bridge_PMU::L2_LINES_OUT_DEMAND_PF_DIRTY,
                                            /*L2_LINES_OUT_DEMAND_DIRTY_ALL_C*/
Intel_Sandy_Bridge_PMU::L2_LINES_OUT_DEMAND_DIRTY_ALL,
                                            /*SQ_MISC_SPLIT_LOCK_C*/
Intel_Sandy_Bridge_PMU::SQ_MISC_SPLIT_LOCK
};
```

A3. CÓDIGO DA HEURÍSTICA DESENVOLVIDA

Abaixo segue o link para o repositório do GitHub contendo o código referente a Heurística desenvolvida durante o TCC:

- https://github.com/LeonardoHorstmann/Power-Cap-Heuristic-for-Multicore-Real-Time-Systems

Os arquivos que apresentam diferenças em relação ao código original do sistema de monitoramento de performance são apresentados abaixo.

```
include/scheduler.h
- Criada a variável "power_cap", que representa a condição de ativação da heurística.
```

```
// EPOS Scheduler Component Declarations

#ifndef __scheduler_h
#define __scheduler_h

#include <utility/list.h>
#include <cpu.h>
#include <machine.h>
#include <pmu.h>

__BEGIN_SYS

// All scheduling criteria, or disciplines, must define operator int() with
// the semantics of returning the desired order of a given object within the
// scheduling list
namespace Scheduling_Criteria
{
    // Priority (static and dynamic)
    class Priority
    {
        friend class _SYS::RT_Thread;

    public:
        enum {
```

```
            MAIN   = 0,
            HIGH   = 1,
            NORMAL = (unsigned(1) << (sizeof(int) * 8 - 1)) - 3,
            LOW    = (unsigned(1) << (sizeof(int) * 8 - 1)) - 2,
            IDLE   = (unsigned(1) << (sizeof(int) * 8 - 1)) - 1
        };

        static const bool timed = true;
        static const bool dynamic = false;
        static const bool preemptive = true;
        static const bool monitoring = true; // enables monitoring
        static const bool power_cap = true; // enables power_capping

    public:
        Priority(int p = NORMAL): _priority(p) {}

        operator const volatile int() const volatile { return _priority; }
        static void init() {} //created to configure and initialize pmu
        void update() {}
        unsigned int queue() const { return 0; }

    protected:
        volatile int _priority;
    };
...
```

include/thread,h
- Criado o vetor "_clock_factor", utilizado pela heurística para controlar o duty cycle aplicado na modulação de clock.
- Criada a variável “_prev_global_deadline_misses” para controle da variação no número de deadlines perdidas.
- Criado o vetor “_power_suspended” para controlar as tarefas suspensas pela heurística.
- Criado o vetor “_power_suspended_count” para controlar o número de suspensões feitas pela heurística. Facilita checagem por suspensos.

```
// EPOS Thread Component Declarations

#ifndef __thread_h
#define __thread_h

#include <utility/queue.h>
#include <utility/handler.h>
#include <cpu.h>
#include <machine.h>
#include <system.h>
#include <scheduler.h>
#include &lt;segment.h&gt;
#include <architecture/ia32/monitoring_capture.h>
extern "C" { void __exit(); }

__BEGIN_SYS

class Thread
```

```
{
    friend class Init_First;
    friend class Init_System;
    friend class System;
    friend class Scheduler<Thread>;
    friend class Synchronizer_Common;
    friend class Alarm;
    friend class Task;
    friend class Agent;

protected:
    static const bool smp = Traits<Thread>::smp;
    static const bool preemptive = Traits<Thread>::Criterion::preemptive;
    static const bool multitask = Traits<System>::multitask;
    static const bool reboot = Traits<System>::reboot;

    static const unsigned int QUANTUM = Traits<Thread>::QUANTUM;
    static const unsigned int STACK_SIZE = multitask ? Traits<System>::STACK_SIZE :
Traits<Application>::STACK_SIZE;
    static const unsigned int USER_STACK_SIZE = Traits<Application>::STACK_SIZE;

    typedef CPU::Log_Addr Log_Addr;
    typedef CPU::Context Context;

public:
    // Thread State
    enum State {
        RUNNING,
        READY,
        SUSPENDED,
        WAITING,
        FINISHING
    };

    // Thread Priority
    typedef Scheduling_Criteria::Priority Priority;

    // Thread Scheduling Criterion
    typedef Traits<Thread>::Criterion Criterion;
    enum {
        HIGH    = Criterion::HIGH,
        NORMAL  = Criterion::NORMAL,
        LOW     = Criterion::LOW,
        MAIN    = Criterion::MAIN,
        IDLE    = Criterion::IDLE
    };

    // Thread Configuration
    // t = 0 => Task::self()
    // ss = 0 => user-level stack on an auto expand segment
    struct Configuration {
        Configuration(const State & s = READY, const Criterion & c = NORMAL, const Color & a =
WHITE, Task * t = 0, unsigned int ss = STACK_SIZE)
        : state(s), criterion(c), color(a), task(t), stack_size(ss) {}
```

```
        State state;
        Criterion criterion;
        Color color;
        Task * task;
        unsigned int stack_size;
    };

    // Thread Queue
    typedef Ordered_Queue<Thread, Criterion, Scheduler<Thread>::Element> Queue;

public:
    template<typename ... Tn>
    Thread(int (* entry)(Tn ...), Tn ... an);
    template<typename ... Tn>
    Thread(const Configuration & conf, int (* entry)(Tn ...), Tn ... an);
    ~Thread();

    const volatile State & state() const { return _state; }

    const volatile Priority & priority() const { return _link.rank(); }
    void priority(const Priority & p);

    Task * task() const { return _task; }

    int join();
    void pass();
    void suspend() { suspend(false); }
    void resume();

    static Thread * volatile self() { return running(); }
    static void yield();
    static void exit(int status = 0);

protected:
    void constructor_prologue(const Color & color, unsigned int stack_size);
    void constructor_epilogue(const Log_Addr & entry, unsigned int stack_size);

    static Thread * volatile running() { return _scheduler.chosen(); }

    Queue::Element * link() { return &_link; }

    Criterion & criterion() { return const_cast<Criterion &>(_link.rank()); }

    static void lock() {
        CPU::int_disable();
        if(smp)
            _lock.acquire();
    }

    static void unlock() {
        if(smp)
            _lock.release();
        CPU::int_enable();
```

```
    }

    static bool locked() { return CPU::int_disabled(); }

    void suspend(bool locked);

    static void sleep(Queue * q);
    static void wakeup(Queue * q);
    static void wakeup_all(Queue * q);

    static void suspend(Thread * t, Queue* q);
    static void resume(Queue* q);
    static void reschedule();
    static void reschedule(unsigned int cpu);
    static void rescheduler(const IC::Interrupt_Id & interrupt);
    static void time_slicer(const IC::Interrupt_Id & interrupt);

    static void dispatch(Thread * prev, Thread * next, bool charge = true);

    static int idle();

private:
    static void init();

protected:
    Task * _task;
    Segment * _user_stack;

    char * _stack;
    Context * volatile _context;
    volatile State _state;
    Queue * _waiting;
    Thread * volatile _joining;
    Queue::Element _link;
    Alarm* _alarm_times;

    static volatile unsigned int _thread_count;
    static volatile unsigned int _power_suspended_count[Traits<Build>::CPUS]; //counts
power-cap suspended
    static Queue * _power_suspended[Traits<Build>::CPUS]; //controls power-cap suspensions
    static Scheduler_Timer * _timer;
    static Scheduler<Thread> _scheduler;
    static Spin _lock;
public:
    volatile int _missed_deadlines; //it's not unsigned because of the calculating process
    int _times_p_count; //counts the number of executions to be used on deadline misses calc
    static Monitoring_Capture* _thread_monitor; //system monitor
    static volatile bool _end_capture; //stops capture when True
    static unsigned int _global_deadline_misses; //counts total amount of deadline misses
    static unsigned int _prev_global_deadline_misses; //used to calc the difference when using
heuristics
    static volatile unsigned int _clock_factor[Traits<Build>::CPUS]; //stores clock factors
};
...
```

```
include/pmu.h
- Configurados os canais para monitorar os eventos necessários.
```

```cpp
// EPOS PMU Mediator Common Package

#ifndef __pmu_h
#define __pmu_h

#include <system/config.h>

__BEGIN_SYS

class PMU_Common
{
Public:
...
    static const unsigned int _channel_3 = 26; // sum 4 after each execution
    // static const unsigned int _channel_4 = _channel_3+1;
    // static const unsigned int _channel_5 = _channel_4+1;
    // static const unsigned int _channel_6 = _channel_5+1;

    static const unsigned int _channel_4 = 142;
    static const unsigned int _channel_5 = 172;
    static const unsigned int _channel_6 = 24;

...
};
```

```
src/component/thread.cc:
- dispatch() -> Adição do código da heurística.
- Adição de métodos de suspend() e resume() focados na heurística.
- Alteração do método “constructor_prologue” para inicialização dos vetores de suspensão
```

```cpp
// EPOS Thread Component Implementation

#include <machine.h>
#include <system.h>
#include <thread.h>
#include <alarm.h> // for FCFS

// This_Thread class attributes
__BEGIN_UTIL
bool This_Thread::_not_booting;
__END_UTIL

__BEGIN_SYS

// Class attributes
Monitoring_Capture* Thread::_thread_monitor;
Thread::Queue * Thread::_power_suspended[Traits<Build>::CPUS];
unsigned int Thread::_global_deadline_misses;
```

```
unsigned int Thread::_prev_global_deadline_misses;
volatile unsigned int Thread::_thread_count;
volatile unsigned int Thread::_power_suspended_count[Traits<Build>::CPUS];
volatile bool Thread::_end_capture;
volatile unsigned int Thread::_clock_factor[Traits<Build>::CPUS];

Scheduler_Timer * Thread::_timer;
Scheduler<Thread> Thread::_scheduler;
Spin Thread::_lock;

// Methods
void Thread::constructor_prologue(const Color & color, unsigned int stack_size)
{
    lock();

    _thread_count++;
    _scheduler.insert(this);

    if (Machine::cpu_id() == 0) {
        for (unsigned int i = 0; i < Machine::n_cpus(); i++) {
            _power_suspended[i] = new Thread::Queue();
            _power_suspended_count[i] = 0;
        }
    }

    if(Traits<MMU>::colorful && color != WHITE)
        _stack = new (color) char[stack_size];
    else
        _stack = new (SYSTEM) char[stack_size];
}


void Thread::constructor_epilogue(const Log_Addr & entry, unsigned int stack_size)
{
    db<Thread>(TRC) << "Thread(task=" << _task
                    << ",entry=" << entry
                    << ",state=" << _state
                    << ",priority=" << _link.rank()
                    << ",stack={b=" << reinterpret_cast<void *>(_stack)
                    << ",s=" << stack_size
                    << "},context={b=" << _context
                    << "," << *_context << "}) => " << this << "@" << _link.rank().queue() <<
endl;

    if(multitask)
        _task->insert(this);

    if((_state != READY) && (_state != RUNNING))
        _scheduler.suspend(this);

    if(preemptive && (_state == READY) && (_link.rank() != IDLE))
        reschedule(_link.rank().queue());
    else
        unlock();
```

```
    _missed_deadlines = 0;
    _times_p_count = 0;
    _alarm_times = 0;
}


Thread::~Thread()
{
    lock();

    db<Thread>(TRC) << "~Thread(this=" << this
                    << ",state=" << _state
                    << ",priority=" << _link.rank()
                    << ",stack={b=" << reinterpret_cast<void *>(_stack)
                    << ",context={b=" << _context
                    << "," << *_context << "})" << endl;

    // The running thread cannot delete itself!
    assert(_state != RUNNING);

    switch(_state) {
    case RUNNING:  // For switch completion only: the running thread would have deleted itself!
Stack wouldn't have been released!
        exit(-1);
        break;
    case READY:
        _scheduler.remove(this);
        _thread_count--;
        break;
    case SUSPENDED:
        _scheduler.resume(this);
        _scheduler.remove(this);
        _thread_count--;
        break;
    case WAITING:
        _waiting->remove(this);
        _scheduler.resume(this);
        _scheduler.remove(this);
        _thread_count--;
        break;
    case FINISHING: // Already called exit()
        break;
    }

    if(multitask) {
        _task->remove(this);
        delete _user_stack;
    }

    if(_joining)
        _joining->resume();

    unlock();
```

```
    delete _stack;
}


void Thread::priority(const Priority & c)
{
    lock();

    db<Thread>(TRC) << "Thread::priority(this=" << this << ",prio=" << c << ")" << endl;

    unsigned int old_cpu = _link.rank().queue();

    _link.rank(Criterion(c));

    if(_state != RUNNING) {
        _scheduler.remove(this);
        _scheduler.insert(this);
    }

    if(preemptive) {
        reschedule(old_cpu);
        if(smp) {
            lock();
            reschedule(_link.rank().queue());
        }
    }
}


int Thread::join()
{
    lock();

    db<Thread>(TRC) << "Thread::join(this=" << this << ",state=" << _state << ")" << endl;

    // Precondition: no Thread::self()->join()
    assert(running() != this);

    // Precondition: a single joiner
    assert(!_joining);

    if(_state != FINISHING) {
        _joining = running();
        _joining->suspend(true);
    } else
        unlock();

    return *reinterpret_cast<int *>(_stack);
}


void Thread::pass()
{
```

```
    lock();

    db<Thread>(TRC) << "Thread::pass(this=" << this << ")" << endl;

    Thread * prev = running();
    Thread * next = _scheduler.choose(this);

    if(next)
        dispatch(prev, next, false);
    else {
        db<Thread>(WRN) << "Thread::pass => thread (" << this << ") not ready!" << endl;
        unlock();
    }
}

void Thread::suspend(Thread * t, Queue* q)
{
    // if(!locked)
    //     lock();

    // Thread * prev = running();

    _scheduler.suspend(t);
    t->_state = SUSPENDED;
    q->insert(&t->_link);
    _power_suspended_count[Machine::cpu_id()]++;

    // Thread * next = running();

    // dispatch(prev, next);
}


void Thread::resume(Queue* q)
{
    // assert(locked());

    if(!q->empty()) {
        Thread * t = q->remove()->object();
        t->_state = READY;
        t->_waiting = 0;
        _scheduler.resume(t);
        _power_suspended_count[Machine::cpu_id()]--;

        //if(preemptive)
        //     reschedule(t->_link.rank().queue());
    } //else
        //unlock();

}

void Thread::suspend(bool locked)
{
    if(!locked)
```

```
        lock();

    db<Thread>(TRC) << "Thread::suspend(this=" << this << ")" << endl;

    Thread * prev = running();

    _scheduler.suspend(this);
    _state = SUSPENDED;

    Thread * next = running();

    dispatch(prev, next);
}


void Thread::resume()
{
    lock();

    db<Thread>(TRC) << "Thread::resume(this=" << this << ")" << endl;

    if(_state == SUSPENDED) {
        _state = READY;
        _scheduler.resume(this);

        if(preemptive)
            reschedule(_link.rank().queue());
    } else {
        db<Thread>(WRN) << "Resume called for unsuspended object!" << endl;

        unlock();
    }
}


// Class methods
void Thread::yield()
{
    lock();

    db<Thread>(TRC) << "Thread::yield(running=" << running() << ")" << endl;

    Thread * prev = running();
    Thread * next = _scheduler.choose_another();

    dispatch(prev, next);
}


void Thread::exit(int status)
{
    lock();

    db<Thread>(TRC) << "Thread::exit(status=" << status << ") [running=" << running() << "]" <<
```

```
endl;

    Thread * prev = running();
    _scheduler.remove(prev);
    *reinterpret_cast<int *>(prev->_stack) = status;
    prev->_state = FINISHING;

    _thread_count--;

    if(prev->_joining) {
        prev->_joining->_state = READY;
        _scheduler.resume(prev->_joining);
        prev->_joining = 0;
    }

    dispatch(prev, _scheduler.choose());
}


void Thread::sleep(Queue * q)
{
    db<Thread>(TRC) << "Thread::sleep(running=" << running() << ",q=" << q << ")" << endl;

    // lock() must be called before entering this method
    assert(locked());

    Thread * prev = running();
    _scheduler.suspend(prev);
    prev->_state = WAITING;
    q->insert(&prev->_link);
    prev->_waiting = q;

    dispatch(prev, _scheduler.chosen());
}


void Thread::wakeup(Queue * q)
{
    db<Thread>(TRC) << "Thread::wakeup(running=" << running() << ",q=" << q << ")" << endl;

    // lock() must be called before entering this method
    assert(locked());

    if(!q->empty()) {
        Thread * t = q->remove()->object();
        t->_state = READY;
        t->_waiting = 0;
        _scheduler.resume(t);

        if(preemptive)
            reschedule(t->_link.rank().queue());
    } else
        unlock();
}
```

```
void Thread::wakeup_all(Queue * q)
{
    db<Thread>(TRC) << "Thread::wakeup_all(running=" << running() << ",q=" << q << ")" << endl;

    // lock() must be called before entering this method
    assert(locked());

    if(!q->empty())
        while(!q->empty()) {
            Thread * t = q->remove()->object();
            t->_state = READY;
            t->_waiting = 0;
            _scheduler.resume(t);

            if(preemptive) {
                reschedule(t->_link.rank().queue());
                lock();
            }
         }
    else
        unlock();
}


void Thread::reschedule()
{
    db<Scheduler<Thread> >(TRC) << "Thread::reschedule()" << endl;

    // lock() must be called before entering this method
    assert(locked());

    Thread * prev = running();
    Thread * next = _scheduler.choose();

    dispatch(prev, next);
}


void Thread::reschedule(unsigned int cpu)
{
    if(!smp || (cpu == Machine::cpu_id()))
        reschedule();
    else {
        db<Scheduler<Thread> >(TRC) << "Thread::reschedule(cpu=" << cpu << ")" << endl;
        IC::ipi_send(cpu, IC::INT_RESCHEDULER);
        unlock();
    }
}


void Thread::rescheduler(const IC::Interrupt_Id & interrupt)
{
```

```
    lock();

    reschedule();
}


void Thread::time_slicer(const IC::Interrupt_Id & i)
{
    lock();

    reschedule();
}


void Thread::dispatch(Thread * prev, Thread * next, bool charge)
{

    if(charge) {
        if(Criterion::timed)
            _timer->reset();

        //Monitoring Capture
        if(Criterion::monitoring && !_end_capture) {
            unsigned int entry_temp = CPU::temperature();
            if (prev->priority() != IDLE && prev->priority() != MAIN) {
                prev->_missed_deadlines = prev->_times_p_count - (prev->_alarm_times->_times +
1);
            } else
                 prev->_missed_deadlines = 0;

            if (next->priority() != IDLE && next->priority() != MAIN) {
                next->_missed_deadlines = next->_times_p_count - (next->_alarm_times->_times +
1);
            } else
                next->_missed_deadlines = 0;

            if (prev->_missed_deadlines < 0)
                    prev->_missed_deadlines = 0;

            if (next->_missed_deadlines < 0)
                    next->_missed_deadlines = 0;

            unsigned long long ts = _thread_monitor->last_capture(Machine::cpu_id(), 7);
            unsigned long long ts_dif = _thread_monitor->time() - ts;
            if (ts == 0) {
                ts_dif = 5000;
            }
            unsigned int cpu = Machine::cpu_id();
            if (cpu % 2) {
                cpu--;
            }
            unsigned long long pkg = CPU::pkg_energy_status();
            unsigned long long pp0 = CPU::pp0_energy_status();
            //unsigned long long energy_unit = CPU::rapl_energy_unit(); // use only to know the
```

```
power unit value
            // unsigned long long channel_0 = PMU::read(0) -
_thread_monitor->last_capture(Machine::cpu_id(), 0);
            //unsigned long long channel_1 = PMU::read(1) -
_thread_monitor->last_capture(Machine::cpu_id(), 1);
            unsigned long long channel_2 = PMU::read(2) -
_thread_monitor->last_capture(Machine::cpu_id(), 2);
            unsigned long long channel_3 = PMU::read(3) -
_thread_monitor->last_capture(Machine::cpu_id(), 3);
            unsigned long long channel_4 = PMU::read(4) -
_thread_monitor->last_capture(Machine::cpu_id(), 4);
            unsigned long long channel_5 = PMU::read(5) -
_thread_monitor->last_capture(Machine::cpu_id(), 5);
            // unsigned long long channel_6 = PMU::read(6) -
_thread_monitor->last_capture(Machine::cpu_id(), 6);
            if (Criterion::power_cap) {
                      //pre-heuristic
                long long normal_priority = static_cast<long long>(Thread::NORMAL);
                normal_priority = static_cast<long long>(normal_priority >> 10); //divids per
1024 instead of using dividing per 1000 to avoid float point operation
                if (_thread_monitor->last_capture(Machine::cpu_id(), 2) == 0) {
                    channel_2 = 0;
                }

                if ((next->priority() == IDLE && entry_temp <= 65 &&
_power_suspended_count[Machine::cpu_id()]) || _end_capture) {
                    if (_power_suspended_count[Machine::cpu_id()]) {
                        running()->resume(_power_suspended[Machine::cpu_id()]);
                    }
                }

                if (entry_temp <= 64) {
                    if (_clock_factor[cpu] < 8) {
                        _clock_factor[cpu]+=1;
                        CPU::clock(CPU::clock()/8*_clock_factor[cpu]);
                    }
                }
                //heuristic - part 1
                if ((next->_missed_deadlines > 0 || prev->_missed_deadlines > 0 ||
_global_deadline_misses > 0)) {
                    _global_deadline_misses=0;
                    if (entry_temp <= 66) {
                        if (_clock_factor[cpu] < 8) {
                            _clock_min[cpu] = _clock_factor[cpu]+1;
                            _clock_factor[cpu]+=1;
                            CPU::clock(CPU::clock()/8*_clock_factor[cpu]);
                        }
                    }
                    if (reinterpret_cast<volatile unsigned int>(next) !=
reinterpret_cast<volatile unsigned int>(prev) &&
                                        next->priority() > normal_priority && (next->priority() !=
IDLE && next->priority() != MAIN)) {
                        running()->suspend(next, _power_suspended[Machine::cpu_id()]);
                        next = _scheduler.choose();
```

```
                    }
                }
                /*heuristic - part 2*/
                else if (entry_temp < 67 && entry_temp > 65) {
                    if (reinterpret_cast<volatile unsigned int>(next) !=
reinterpret_cast<volatile unsigned int>(prev) &&
                                    next->priority() > normal_priority && (next->priority() !=
IDLE && next->priority() != MAIN)) {
                        running()->suspend(next, _power_suspended[Machine::cpu_id()]);
                        next = _scheduler.choose();
                    } else if (reinterpret_cast<volatile unsigned int>(next) ==
reinterpret_cast<volatile unsigned int>(prev)
                                    && next->priority() != IDLE && next->priority() >
normal_priority && !_end_capture) {
                        next = _scheduler.choose_another();
                    }
                } else {
                    if ( (channel_3 >= 54) && (ts_dif > 4990) && (channel_2 >= 3392*ts_dif)  &&
(channel_4 >= 13000*ts_dif) ) {
                        if (reinterpret_cast<volatile unsigned int>(next) !=
reinterpret_cast<volatile unsigned int>(prev) &&
                                        next->priority() > normal_priority && (next->priority()
!= IDLE && next->priority() != MAIN)) {
                            running()->suspend(next, _power_suspended[Machine::cpu_id()]);
                            next = _scheduler.choose();
                        } else if (reinterpret_cast<volatile unsigned int>(next) ==
reinterpret_cast<volatile unsigned int>(prev)
                                        && next->priority() != IDLE && next->priority() >
normal_priority && !_end_capture) {
                            next = _scheduler.choose_another();
                        } else if ( (channel_5 >= 4.85*ts_dif) && _clock_factor[cpu] >= 5 &&
next->priority() != IDLE ) {
                            _clock_factor[cpu]-=1;
                            CPU::clock(CPU::clock()/8*_clock_factor[cpu]);
                        }
                    } else {
                        if ( (channel_5 >= 4*ts_dif) && (channel_4 >= 13000*ts_dif) ) {
                            if (reinterpret_cast<volatile unsigned int>(next) !=
reinterpret_cast<volatile unsigned int>(prev) &&
                                            next->priority() > normal_priority &&
(next->priority() != IDLE && next->priority() != MAIN)) {
                                running()->suspend(next, _power_suspended[Machine::cpu_id()]);
                                next = _scheduler.choose();
                            } else if (reinterpret_cast<volatile unsigned int>(next) ==
reinterpret_cast<volatile unsigned int>(prev)
                                            && next->priority() != IDLE && next->priority() >
normal_priority && !_end_capture) {
                                next = _scheduler.choose_another();
                            } else if ( _clock_factor[cpu] >= 5 && next->priority() != IDLE ) {
                                _clock_factor[cpu]-=1;
                                CPU::clock(CPU::clock()/8*_clock_factor[cpu]);
                            }
                        } else {
                            if ( (entry_temp >= 68 && _clock_factor[cpu] >= 5 &&
```

```
next->priority() != IDLE) || (entry_temp > 68 && next->priority() == IDLE) ) {
                                _clock_factor[cpu]-=1;
                                CPU::clock(CPU::clock()/8*_clock_factor[cpu]);
                            }
                        }
                    }
                }
            }
            _thread_monitor->capture(entry_temp, PMU::read(0), PMU::read(1), PMU::read(2),
PMU::read(3), PMU::read(4), PMU::read(5),
            PMU::read(6), reinterpret_cast<volatile unsigned int>(prev), prev->priority(),
Machine::cpu_id(), prev->_missed_deadlines, prev->_global_deadline_misses, pkg, pp0);
        }
    }

    if(prev != next) {
        if(prev->_state == RUNNING)
            prev->_state = READY;
        next->_state = RUNNING;

        db<Thread>(TRC) << "Thread::dispatch(prev=" << prev << ",next=" << next << ")" << endl;
        db<Thread>(INF) << "prev={" << prev << ",ctx=" << *prev->_context << "}" << endl;
        db<Thread>(INF) << "next={" << next << ",ctx=" << *next->_context << "}" << endl;

        if(smp)
            _lock.release();

        if(multitask && (next->_task != prev->_task))
            next->_task->activate();

        CPU::switch_context(&prev->_context, next->_context);
    } else
        if(smp)
            _lock.release();

    // TODO: could this be moved to right after the switch_context?
    CPU::int_enable();
}


int Thread::idle()
{
    while(_thread_count > Machine::n_cpus()) { // someone else besides idles
        if(Traits<Thread>::trace_idle)
            db<Thread>(TRC) << "Thread::idle(CPU=" << Machine::cpu_id() << ",this=" <<
running() << ")" << endl;
        CPU::int_enable();
        if (_power_suspended_count[Machine::cpu_id()] > 0 && CPU::temperature() < 67)
            resume(_power_suspended[Machine::cpu_id()]);
        //if (_power_suspended_count[Machine::cpu_id()] == 0 || CPU::temperature() >= 67) {
       CPU::int_enable();
       CPU::halt();
        if(_scheduler.schedulables() > 0)// A thread might have been woken up by another CPU
            yield();
```

```
    }

    CPU::int_disable();
    if(Machine::cpu_id() == 0) {
        if (Criterion::monitoring) {
            db<Thread>(WRN) << "temp: "<< CPU::temperature() <<endl;
            _thread_monitor->datas();
        }
        db<Thread>(WRN) << "The last thread has exited!" << endl;

        if(reboot) {
            db<Thread>(WRN) << "Rebooting the machine ..." << endl;
            Machine::reboot();
        } else
            db<Thread>(WRN) << "Halting the machine ..." << endl;
    }
    CPU::halt();

    return 0;
}

__END_SYS

// Id forwarder to the spin lock
__BEGIN_UTIL
unsigned int This_Thread::id()
{
    return _not_booting ? reinterpret_cast<volatile unsigned int>(Thread::self()) :
Machine::cpu_id() + 1;
}
__END_UTIL
```

A4. ARTIGO PADRÃO SBC

Heurística De Power-cap Em Sistemas Multicore De Tempo-real Com Uso De Monitoramento De Performance

Leonardo Passig Horstmann

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brazil

{leonardo.horstmann}@grad.ufsc.br

Abstract. *With the increasing use of embedded systems and mobile devices powered by a battery and the computational power growth and their needs, it has become more necessary to study techniques for better use of energy resources in order to maximize the working time of a system. One of the alternatives for a better exploration of the energy source is the application of Power-Cap, limiting the energy consumption in order not to extrapolate the desired level. The use of Power-Cap in Real-Time systems, however, must take into account the existence of critical tasks that should not lose their deadlines. This paper describes the development of a Power-Cap heuristic in Real-Time Multicore Systems based on data extracted through Performance Monitoring and Software and Hardware event studies.*

Resumo. *Com a crescente no uso de sistemas embarcados e dispositivos móveis alimentados a bateria e o aumento do poder computacional e das necessidades dos mesmos, têm-se feito cada vez mais necessário o estudo de técnicas para melhor utilização dos recursos energéticos de modo a maximizar o tempo de funcionamento de um sistema. Uma das alternativas para uma melhor exploração da fonte energética é a aplicação de Power-Cap, limitando o consumo energético de maneira a não extrapolar o nível desejado. O uso de Power-Cap em sistemas de Tempo-Real, no entanto, deve levar em conta a existência de tarefas críticas que não devem perder seus tempos limite para execução (deadlines). Tendo em vista os aspectos citados, o presente artigo descreve o desenvolvimento de uma heurística de Power-Cap em Sistemas Multicore de Tempo-Real tendo como base dados extraídos através de Monitoramento de Performance e estudos de eventos de Software e Hardware.*

Palavras-chave: Multicore, Tempo-Real, Power-Cap, DVFS, PMU, Performance Monitoring

1. Introdução

Nos dias de hoje, os dispositivos móveis e sistemas embarcados, que, em sua maioria, operam tendo como fonte de alimentação uma bateria, representam a maior parte dos sistemas computacionais desenvolvidos. Isto traz à tona, ainda mais, a importância da aplicação de Power-Cap (Tampa de Energia ou, em outras palavras, um limite máximo para consumo energético).

Existem basicamente duas maneiras de se aplicar Power-Capping, a primeira delas busca uma abordagem a nível de Software e a segunda delas a nível de Hardware. Segundo Zhang e Hoffmann (2016), as abordagens de Software apresentam flexibilidade, permitindo a coordenação de múltiplos recursos de hardware, porém apresentam lentidão para alcançar o objetivo, requerendo

muito tempo para convergir para o Power-Cap. Abordagens de Hardware, por sua vez, tendem a convergir muito rapidamente, mas controlam apenas voltagem e frequência de operação, limitando assim a performance num todo.

Dentre as opções para Power-Capping a nível de Software, destacam-se o controle da frequência de clock (modulação de clock) através de DVS e o desenvolvimento de estratégias de escalonamento focadas no consumo energético (power-aware).

Seguindo a ideia de ZHANG, LANG, PAKIN e FU (2014), o desenvolvimento de escalonadores de tarefas paralelas focados em consumo energético tem sido reconhecido como uma demanda para a computação de alto desempenho (high performance computing - HPC).

Ainda neste cenário, o uso de técnicas de DVS tende a aumentar a duração e a vida útil de suas baterias, uma vez que, conforme frizado por Islam e Lin (2017), dentre os vários componentes num dispositivo computacional, o processador é um dos maiores consumidores de energia, sendo sua performance diretamente relacionada com sua dissipação energética, e consome aproximadamente de 18% a 30% de toda energia consumida pelo dispositivo.

Por outro lado, o monitoramento de performance tem tomado um importante papel no que se refere, principalmente, a predição de eventos, permitindo operar o mecanismo de DVS no melhor momento. Mück, Sarma e Dutt (2015), ao descreverem o modelo "Run-DMC: Runtime Dynamic Heterogeneous Multicore Performance and Power Estimation for Energy Efficiency", previam uma etapa de coleta de dados (por eles chamada de "sensing"), na qual eram realizadas leituras de contadores de performance de Hardware (HPCs - Hardware Performance Counter).

No modelo proposto, bem como no presente trabalho, as coletas de dados eram realizadas na ordem de trocas de contexto, de modo a ser possível manter as métricas de performance separadas por threads.

Tendo em mente o desenvolvimento já feito, o foco do presente trabalho é se utilizar de um sistema operacional embarcado de baixa interferência (EPOS) para coletar dados de execução provenientes da Unidade de Monitoramento de Performance da Intel (Performance Monitoring Unit - PMU), tornando possível o correlacionamento dos diferentes contadores de performance e a busca de novas variáveis de interesse para o desenvolvimento de Heurísticas de Power-Cap para sistemas Multi-Core de Tempo-Real.

2. Materiais

Os dados analisados foram capturados através de sensores e contadores de software e hardware, sendo eles vindos dos contadores da Unidade de Monitoramento de Performance da Intel (PMU) e dos sensores térmicos e energéticos, com leituras dos MSRs ligados a temperatura (IA32_THERM_STATUS e IA32_TEMPERATURE_TARGET) e da Interface RAPL (MSR_RAPL_POWER_UNIT, MSR_PKG_ENERGY_STATUS e MSR_PP0_ENERGY_STATUS), utilizada para controle e aferição de consumo energético.

O computador utilizado durante a fase de testes, geração de dados e configuração da heurística contava com um processador Intel® Core™ i7-2600, de arquitetura Sandy-Bridge, com 4 GB de memória RAM DDR3 de 1333MHz, placa mãe PCWARE IPMH61R3, com fonte de alimentação ALLIED SL-8180 BTX instalado sob corrente de 220V, disponibilizado pelo laboratório (LISHA) para captura de dados e testes das heurísticas.

Para aferições energéticas, além da interface RAPL, foi utilizado um analisador de potência e qualidade de energia Fluke 435 series II, para aferições de consumo energético.

Para as análises de Mineração de Dados através da aplicação de algoritmos de correlação de atributos, fora utilizado o software WEKA (https://www.cs.waikato.ac.nz/ml/weka/), um software de código aberto desenvolvido pelo grupo de Machine Learning da universidade de waikato.

3. Sistema De Monitoramento De Performance Não Intrusivo

Para coleta dos dados de performance, foi desenvolvido, em conjunto com o discente do curso de Ciências da Computação, na UFSC, José Luis Conradi Hoffmann, de matrícula 15100745 , um sistema de monitoramento de performance não intrusivo, que armazena dados capturados durante a execução como um retrato de um momento da execução. Cada captura serve para adquirir informações sobre eventos de hardware e variáveis de sistema que possam ser úteis para a análise da execução.

Em síntese, uma captura é definida como uma descrição de um determinado momento do sistema, contendo: Os valores dos 7 contadores de eventos de Hardware (3 fixos e 4 configuráveis); O tempo em que a captura foi realizada (em formato Unix Time); A temperatura da CPU em que a captura foi realizada; A Thread que estava executando, sua prioridade e a quantidade de Deadlines perdidas por ela; A quantidade de Deadlines perdidas até o momento pelo sistema como um todo; A contagem de energia consumida pela interface RAPL, trazendo o consumo do PKG e do PP0.

```
01: struct Moment {
02:     unsigned long long _temperature;
03:     unsigned long long _pmu0;
04:     unsigned long long _pmu1;
05:     unsigned long long _pmu2;
06:     unsigned long long _pmu3;
07:     unsigned long long _pmu4;
08:     unsigned long long _pmu5;
09:     unsigned long long _pmu6;
10:     unsigned long long _time_stamp;
11:     unsigned long long _thread_id;
12:     long long _thread_priority;
13:     unsigned long long _deadline;
14:     unsigned long long _global_deadline;
15:     unsigned long long _pkg_energy;
16:     unsigned long long _pp0_energy;
};
```

Memory Allocation*
CPU0 captures
CPU1 captures
CPU2 captures
CPU3 captures
CPU4 captures
CPU5 captures
CPU6 captures
CPU7 captures

Figura 1. Estrutura de um Momento de captura e organização da alocação de memória.

Essas capturas são armazenadas em uma região da memória para cada uma das CPUs e consumidas somente no final da execução. Isto elimina a intrusividade de captura e consumo dos dados, visto que a escrita dos dados é paralelizada entre as CPUs e o consumo não tem influência na execução. Outro fator importante para evitar a intrusividade é realizar capturas somente em momentos de não intrusão, com estudos realizados em cima da estrutura do sistema EPOS, foi possível identificar os pontos de troca de contexto das threads como uma região propícia para realizar as capturas sem gerar aumento no tempo de execução. O sistema de captura funciona da seguinte maneira:

1. O sistema de captura é configurado junto a inicialização do sistema de Threads do EPOS.
2. A PMU é configurada no início da execução, junto da inicialização do sistema de escalonamento;
3. As capturas são habilitadas assim que a aplicação inicia sua execução, e também são desabilitadas ao fim da aplicação.

4. Sempre que o sistema reescalona uma Thread, uma captura daquele momento é realizada e armazenada.
5. Antes do sistema operacional finalizar, os momentos armazenados são enviados via porta serial para outro computador.
6. Neste outro computador um log é gerado, e em sequência é consumido pelo algoritmo de envio dos dados para uma plataforma IoT.

4. Coleta De Dados

A partir do desenvolvimento do sistema de monitoramento de performance, foram gerados conjuntos de operações focados em estimular o processador e os contadores de eventos da PMU. As operações utilizadas basearam-se em uma variação do algoritmo de Fibonacci Recursivo com operações de ponto flutuante e em um algoritmo de Cópia de Regiões de Memória.

Após a escolha das tarefas a serem executadas, foram gerados conjuntos de tarefas (task-sets) para execução, utilizando o método descrito por Gracioli em Real-Time Operating System Support for Multicore Applications (2014) para particionamento de tarefas. Tal método gera conjuntos de tarefas para Sistemas de Tempo-Real em arquiteturas Multicore, levando em conta dados arquiteturais e o tipo de escalonador desejado.

Por fim, o escalonador escolhido para geração dos dados foi o Partitioned-EDF (P-EDF), devido a possibilidade de manter o comportamento das capturas do sistema de monitoramento nas CPUs em diferentes execuções, que utilizassem o mesmo task-set. Um comportamento estável foi visado para facilitar a análise dos comportamentos dos contadores, ligados à CPU, durante as execuções, principalmente devido ao fato de que existem mais de 200 eventos monitoráveis, ao passo que apenas 4 eventos podem ser configurados para cada execução. Os conjuntos de tarefas gerados podem ser encontrados na Tabela 1.

Tabela 1. Uso total de cada CPU para os conjuntos de tarefas utilizadas para geração e análise dos dados.

CPU	Uso Conjunto 1	Uso Conjunto 2	Uso Conjunto 3
0	54,20%	48,78%	55,83%
1	90,67%	81,60%	47,06%
2	68,92%	62,03%	68,49%
3	64,66%	58,19%	51,85%
4	83,95%	75,56%	46,55%
5	52,88%	47,59%	56,76%
6	56,97%	51,27%	60,97%
7	78,18%	70,36%	62,98%

5. Análise dos Contadores

Uma vez capturados os dados e feito o pré-processamento, era necessário relacioná-los ao consumo energético. Esta comparação, no entanto, demonstrou-se inviável com os métodos utilizados para análise de consumo, uma vez que o Fluke, apesar da grande precisão, apresentava leituras intervaladas em 250 ms, um empecilho levando em conta a granularidade do sistema na ordem de microsegundos e com capturas espaçadas por um período máximo de 5 ms, perdendo assim granularidade nas variações de comportamento e, as aferições feitas utilizando-se a Interface RAPL demonstram uma imprecisão inerente a interface, que apresenta atualizações intervaladas por 1 ms, gerando, portanto, capturas com atualizações em momentos iniciais e capturas que são executadas próximas ao ponto de atualização.

Tendo em vista as dificuldades citadas, foram analisadas as especificações térmicas e mecânicas do processador (Intel® Core™ i7-2600 Thermal Profile), onde foi visualizada a relação de consumo energético e dissipação de calor, demonstrado através do Perfil Térmico, conforme Figura 2.

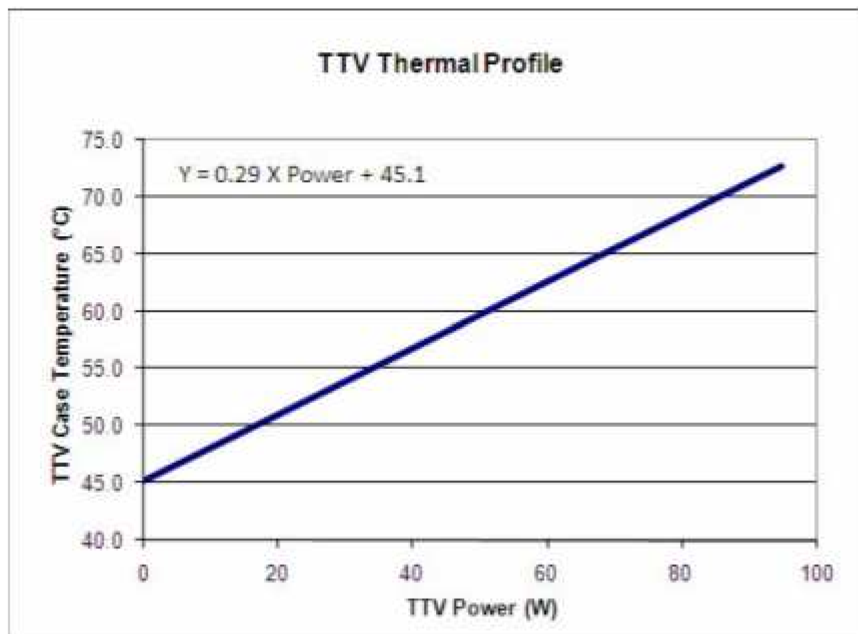


Figura 2. Representação da relação de temperatura e consumo energético.

Diante a relação estabelecida, os dados capturados foram relacionados à temperatura. As correlações foram calculadas utilizando o método Correlation Attribute Evaluator disponível por padrão no Weka na versão 3.9.2.

Após esta etapa de levantamento de correlações entre os eventos, foram eliminados da análise os eventos que se equivalessem aos eventos fixos em valores absolutos (Figura 3) ou em proporção, verificada através da normalização (Figura 4).

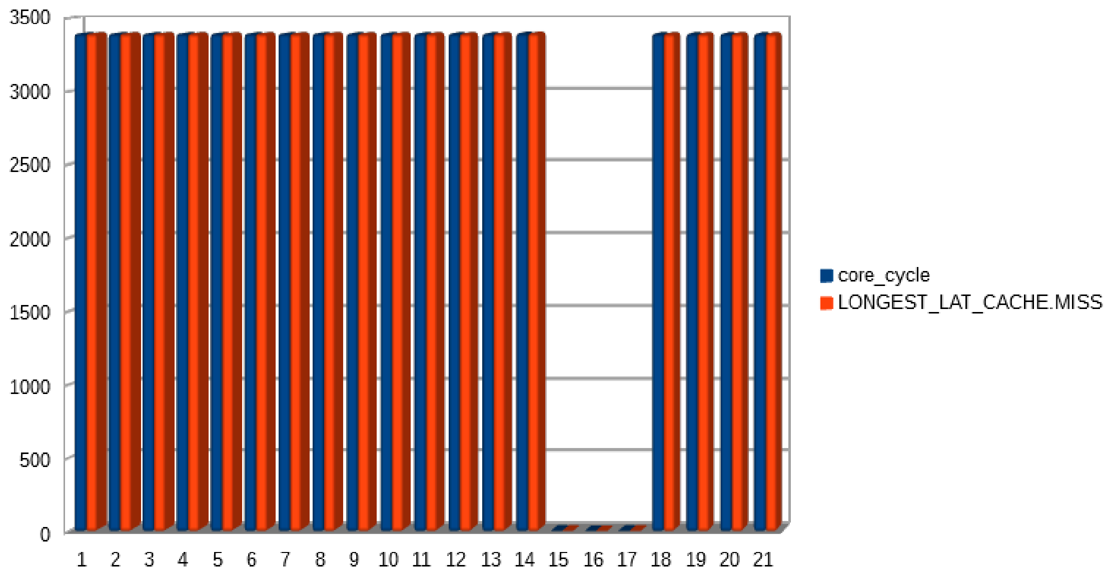


Figura 3. Verificação da equivalência de valores entre LONGEST_LAT_CACHE.MISS e CORE_CYCLE - evento de monitoramento fixo da arquitetura.

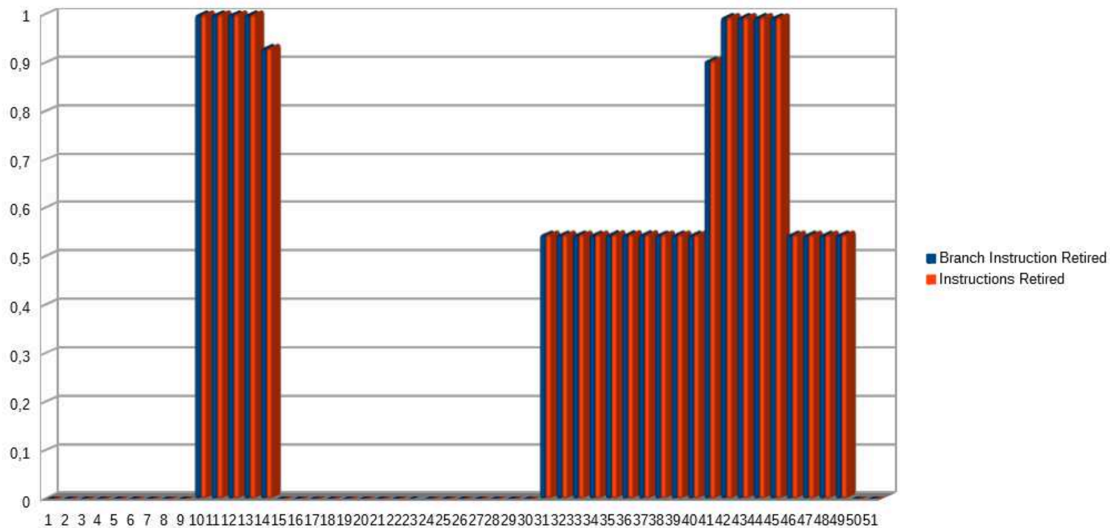


Figura 4. Verificação da equivalência proporcional entre Branch Instruction Retired e Instruction Retired - evento de monitoramento fixo da arquitetura.

Concluída a etapa de eliminação dos eventos "redundantes", foi gerada uma tabela com as dez maiores correlações para os dois tipos de testes usados (Tabela 2), sendo os eventos em comum colocados em destaque e selecionados para desenvolvimento da heurística.

Tabela 2. Principais eventos correlatos com a Temperatura conforme execução.

	Exec. Recursiva	Correl.	Exec. Memcpy	Correl.
1	UOPS_DISPATCHED.THREAD	0.5	UOPS_DISPATCHED.THREAD	0.48
2	INT_MISC.RECOVERY_CYCLES	0.42	ref_clock / core_cycle	0.35
3	ref_clock / core_cycle	0.4	L2_STORE_LOCK_RQSTS.MISS	0.32
4	RESOURCE_STALLS2.OOO_RSRC	0.4	BR_MISP_RETIRED.NEAR_CALL	0.31
5	CPL_CYCLES.RING123	0.4	UOPS_ISSUED.ANY	0.31
6	BR_MISP_RETIRED.NEAR_CALL	0.36	INT_MISC.RAT_STALL_CYCLES	0.3
7	INST_RETIRED	0.35	INT_MISC.RECOVERY_CYCLES	0.3
8	BRANCH_INSTRUCTION_RETIRED	0.34	LD_BLOCKS_PARTIAL_ADDRESS_ALIAS	0.3
9	IDQ_UOPS_NOT_DELIVERED.CORE	0.34	INSTS_WRITTEN_TO_IQ.INSTS	0.3
10	UOPS_ISSUED.ANY	0.25	L1D_PEND_MISS.PENDING	0.3

A partir da definição dos eventos correlatos, foram iniciados os processos para geração das árvores de decisão com o classificador REPTree, também disponível no Weka, com intuito principal de facilitar a visualização de pontos de corte para as variáveis encontradas.

Antes da aplicação do classificador, porém, era necessário a geração de classes a serem previstas e, por consequência, a definição do ponto de Power-Cap. Ao analisar o Design Térmico e Mecânico do processador utilizado, então, foi detectado um TDP de 95W e, a partir da capacidade máxima projetada, foi estabelecido um ponto de Power-Cap experimental de 80W que, ao comparar ao Perfil Térmico, indicou um limite de temperatura de 68.3°C. Esse procedimento determinou as classes a serem previstas. O classificador teria como entrada as atuais taxas de crescimento dos contadores e, teria como função prever se a temperatura num próximo momento de captura seria menor ou igual a 68 (o valor indicado nos sensores é digital e inteiro) ou maior do que o limite estabelecido.

6. Heurística Desenvolvida

A partir das árvores geradas foram retirados os pontos de corte, sendo possível dar início a modelagem do algoritmo da heurística. Num primeiro momento, então, foi decidido pela adição de tarefas de baixa prioridade, para aumentar os pontos de ação da heurística, sem prejudicar a execução das tarefas críticas.

Nesse cenário foram traçadas as operações de controle da heurística, a primeira delas seria a suspensão de tarefas não críticas e, a segunda, a ser utilizada em casos em que a suspensão de tarefas não seja suficiente para manter temperatura e consumo sob os limites, a modulação de frequência do processador, feita através da configuração de duty cycles.

Apesar do intuito principal de manter as restrições térmicas e energéticas, a heurística também foi pensada para manter o maior desempenho possível sob as limitações traçadas. Com este objetivo, foi desenvolvida uma análise prévia a heurística.

Para entender a análise, é necessário entender a separação de faixas de temperatura, indicadas pelos comportamentos visualizados nos logs de execução, que demonstram que na faixa térmica trabalhada e com intervalos máximos entre duas capturas de 5 ms, a variação térmica alterna entre 1 e 3°C, com maiores probabilidades para valores próximos a 1°C. Tendo em mente o comportamento citado, foram separadas 3 faixas de temperatura: Boa (temperaturas abaixo de 65°C), Ok (entre 65 e 67°C) e em Risco (maiores que 67°C).

A análise executada antes da heurística com intuito de aproveitar ao máximo o desempenho disponível pode ser visualizada na Figura 5.

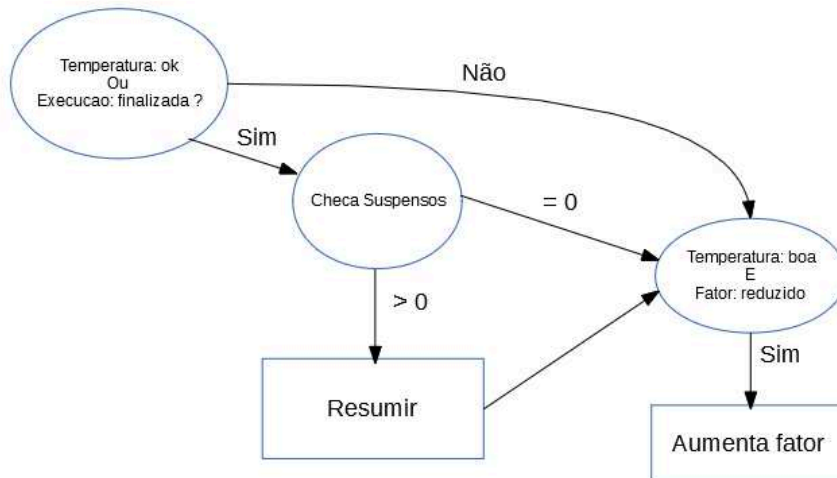


Figura 5. Verificação anterior à heurística.

Como já citado, um dos objetivos da heurística era não prejudicar a execução de tarefas críticas, sendo assim, antes de executar as operações de controle, é verificada a condição de não existirem deadlines perdidas por tarefas críticas. Tal verificação é representada na Figura 6.

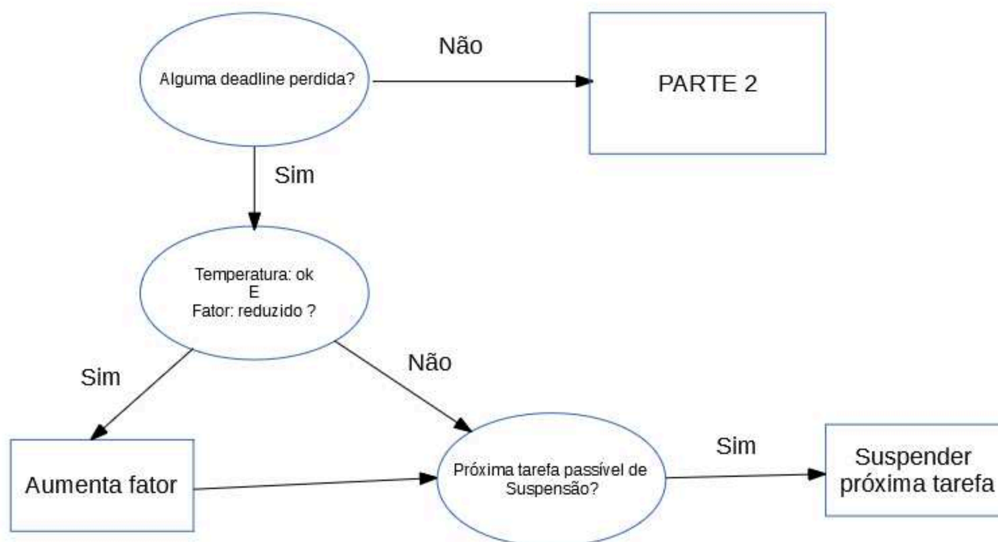


Figura 6. Primeira parte da heurística - Verificação do status das tarefas críticas.

Por fim, o algoritmo da heurística foi modelado tendo em vista a temperatura atual e o desempenho potencial demonstrado pelos contadores de monitoramento de eventos, com pontos de cortes definidos durante geração dos classificadores. A modelagem da parte de controle da heurística pode ser encontrada na Figura 7.

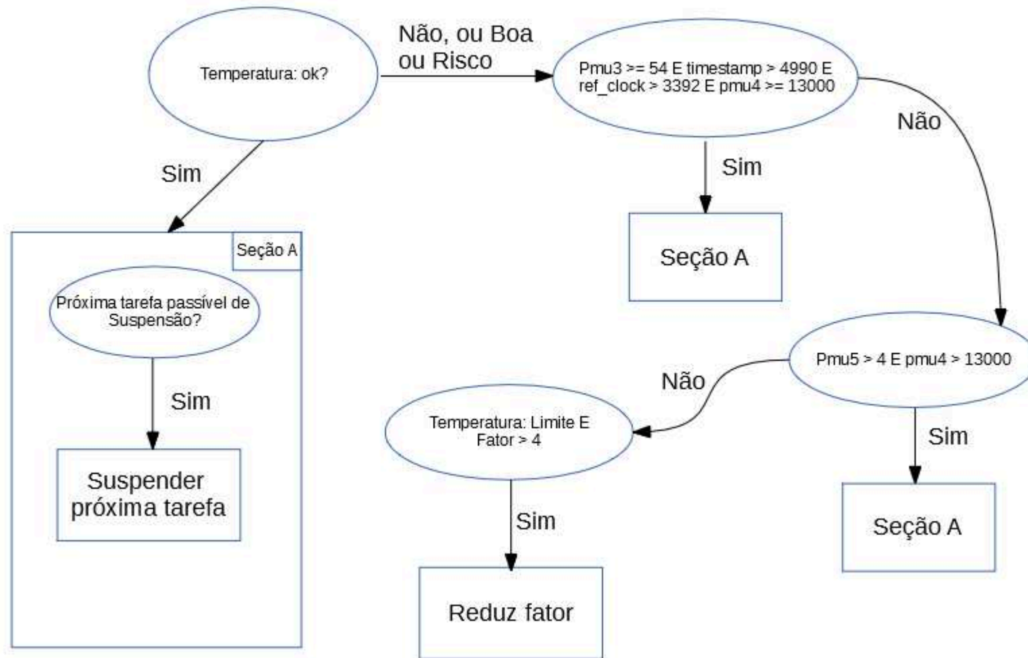


Figura 7. Segunda parte da heurística - Mecanismo de controle.

7. Resultados

Para efeitos de visualização dos resultados e do impacto da heurística, foram geradas execuções com e sem o uso da mesma. Em execuções com operações aritméticas recursivas e de ponto flutuante, sem o uso da heurística, é possível notar que o consumo que se inicia entre 90 e 110 W se estabiliza entre 120 e 130 W, conforme Figura 8.

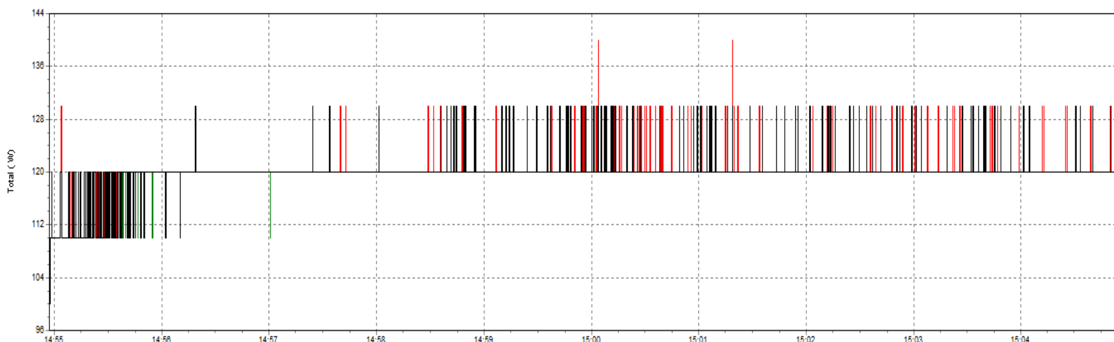


Figura 8. Execução de 10 minutos com uso de operações aritméticas recursivas com ponto flutuante e sem uso da heurística com estabilização entre 120 e 130 W.

A mesma execução foi repetida com uso da heurística, onde é possível perceber que o movimento, neste caso, é oposto, após um período de estabilização (tempo de convergência) o consumo fica em 80 W com poucas variações para 90. Este tempo de estabilização é variável conforme fatores como temperatura inicial da execução e potencial de consumo e aquecimento das operações e/ou do taskset utilizados. O histórico de consumo durante a execução com heurística ativa é encontrado na Figura 9.

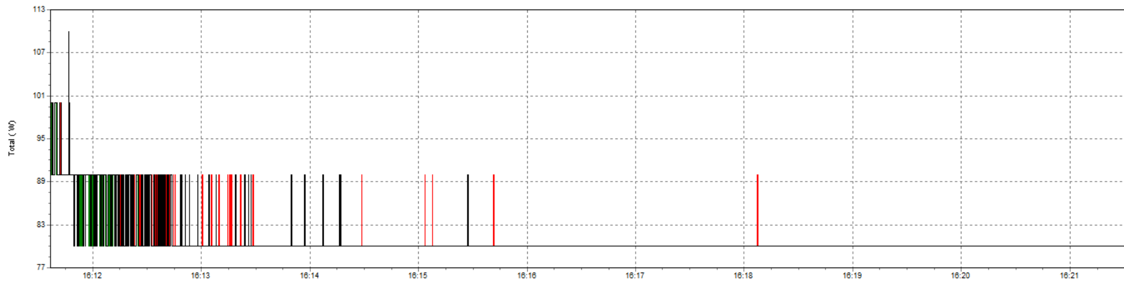


Figura 9. Execução de 10 minutos com uso de operações aritméticas recursivas com ponto flutuante e com uso da heurística com estabilização entre 80 e 90 W.

Um sumário das médias de consumo em execuções como a apresentada acima e algumas outras é apresentado no sumário da Figura 10, as identificações mostram o tipo de execução e o uso (ou não) da heurística e os valores são referentes as médias de consumo de cada execução. Consumos são expressos em Watts (W).

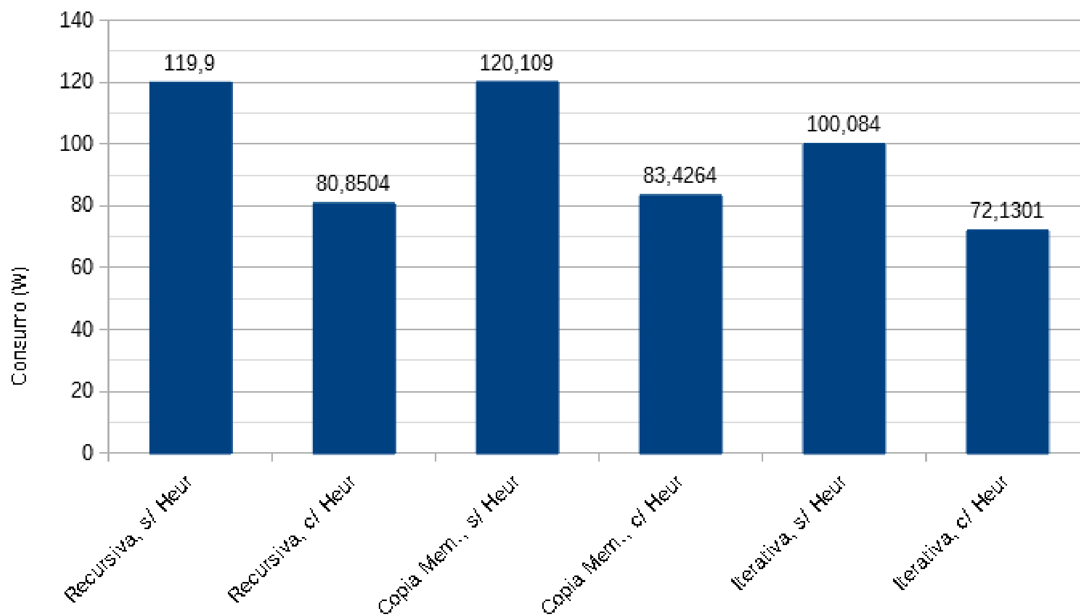


Figura 10. Sumário comparativo das médias de consumo em execuções com período de 10 minutos.

É possível perceber que em tarefas iterativas, que tem menor consumo sem heurística, obteve consumo até menor do que o limite e que operações com cópia de memória obtiveram resultado médio pouco maior que o valor estabelecido. Um dos motivos é o maior tempo de convergência demonstrado por este tipo de operação.

8. Conclusões e Trabalhos Futuros

O presente artigo descreveu o desenvolvimento de um Sistema de Monitoramento de Performance para sistemas Multicore de Tempo-Real (conjuntamente ao discente do curso de Ciências da Computação na UFSC, José Luis Conradi Hoffmann, de matrícula 15100745), bem como o desenvolvimento de uma heurística de Power-Cap baseada na aplicação de técnicas de Data-Mining e de estudos feitos sobre os dados coletados pelo monitoramento.

O sistema de monitoramento de performance pode ser amplamente utilizado para desenvolvimentos futuros, com ou sem o mesmo enfoque e até mesmo utilizando outras plataformas. A heurística desenvolvida, apesar de limitada para uma faixa de consumo e uma arquitetura em específico, pode ser utilizada como base para o desenvolvimento de novas heurísticas para diferentes computadores e o processo de seu desenvolvimento pode embasar a criação de modelos capazes de se adaptar melhor ao mundo dos sistemas embarcados.

Trabalhos futuros podem ser realizados tendo como enfoque a generalização da heurística desenvolvida para novas arquiteturas e novas restrições energéticas e térmicas, podendo-se analisar a possibilidade de aplicação de redes neurais durante o processo de generalização. Outros enfoques podem estar na migração do sistema de capturas para novos ambientes e a aplicação dos estudos possíveis em novas aplicações.

Referências

Intel® 64 and IA-32 Architectures Software Developer's Manual, https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf, Dezembro 2017.

Embedded Parallel Operating System, LISHA, 2018. https://epos.lisha.ufsc.br/HomePage, Junho, 2018.

Software/Hardware integration Laboratory, LISHA, 2018. https://lisha.ufsc.br, Janeiro, 2018.

HAMMOND, L., NAYFEH, B. A., OLU-KOTUN, K., "A Single-Chip Multiprocessor", Computer, vol. 30, no. 9, pp. 79-85, Setembro. 1997. https://ieeexplore.ieee.org/document/612253/, Junho 2018.

DONYANAVARD, B.; MÜCK, T.; SARMA, S.; DUTT, N., "SPARTA: Runtime task allocation for energy efficient heterogeneous manycores", Department of Computer Science, University of California, Irvine, USA, IEEE, 2016. http://ieeexplore.ieee.org/document/7750975, Setembro 2017.

MÜCK, T.; SARMA, S.; DUTT, N.; "Run-DMC: runtime dynamic heterogeneous multicore performance and power estimation for energy efficiency", Amsterdam, The Netherlands, IEEE, 2015. https://dl.acm.org/citation.cfm?id=2830859, Setembro, 2017.

GRACIOLI, G., FRÖHLICH, A. A., "On the Design and Evaluation of a Real-Time Operating System for Cache-Coherent Multicore Architectures", ACM SIGOPS Operating Systems Review - Special Topics, vol. 49, no. 2, pág 2-16, Dezembro - 2015. https://dl.acm.org/citation.cfm?id=2883594, Setembro 2017.

SULEIMAN, D., IBRAHIM, M., HAMARASH, I., "Dynamic Voltage Frequency Scaling (Dvfs) For Microprocessors Power And Energy Reduction", http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=E6F55A8CE6B176124DC60E15141B65B5?doi=10.1.1.111.1451&rep=rep1&type=pdf, Agosto 2017.

Intel® Core™ i7-2000 and i5-2000 Desktop Processor Series (Quad Core 95W) Thermal Profile, https://www.intel.com/content/www/us/en/processors/core/2nd-gen-core-lga1155-socket-guide.html, Dezembro 2017.