MÉTODO PARA DISTRIBUIÇÃO DA CARGA DE TRABALHO DOS SOFTWARES PON EM MULTICORE

QUALIFICAÇÃO DE DOUTORADO

DANILLO LEAL BELMONTE

Orientador: Prof. Dr. Paulo Cézar Stadzisz

Co-orientador: Prof. Dr. Jean Marcelo Simão

AGENDA

- CONTEXTO
- FUNDAMENTOS
- MÉTODO PROPOSTO
- EXPERIMENTOS
- ESFORÇOS
- CRONOGRAMA

AGENDA

- CONTEXTO
- FUNDAMENTOS
- MÉTODO PROPOSTO
- EXPERIMENTOS
- ESFORÇOS
- CRONOGRAMA

CONTEXTO

CONTEXTO DA PESQUISA

- Existe demanda crescente por maior capacidade de processamento devido ao aumento da complexidade e sofisticação dos softwares desenvolvidos e utilizados
- Até 2005 essa demanda foi atendida pelo aumento do clock dos processadores
- Nos últimos anos outra estratégia tem sido largamente empregada que consiste em multiplicar o número de unidades de processamento sem aumento do clock. Neste âmbito, foram desenvolvidos processadores multicore, os quais estão sendo utilizado em larga escala
- A tecnologia multicore consiste na integração de dois ou mais núcleos de processamento em um único processador

CONTEXTO DA PESQUISA

- Os SOs convencionais fazem uso dos processadores multicore dividindo a carga de processamento das aplicações, por meio do escalonamento de tarefas. Entretanto, tal particionamento entre os núcleos não é satisfatório
- Para o desenvolvimento de software para ambiente multicore existem recursos de programação paralela (API PThreads, Cilk, Threading Building Blocks, OpenMP, CUDA). Esses recursos amenizam o trabalho do desenvolvedor, pois a programação paralela exige maior esforço na sua concepção

CONTEXTO DA PESQUISA

- Ao programar para multicore, a programação paralela tem sido frequentemente adotada para o desenvolvimento de aplicações que demandam alto desempenho
- Entretanto, apesar dos recursos de programação disponíveis, para prevalecer-se dos múltiplos núcleos, a implementação de software deveria ser concebida de forma a poder distribuir suas partes “naturalmente” em tempo de execução. Mas isso não acontece
  - Primeiro, pois, para escrever programas que aproveitem efetivamente os múltiplos núcleos de processamento, é necessário que o desenvolvedor despenda maior esforço na programação paralela
  - Segundo, devido ao alto nível de acoplamento de código entre as partes dos objetos, uma das principais deficiências dos atuais paradigmas de programação

CONTEXTO DA PESQUISA

- O aspecto central é, portanto, a forma de distribuição da carga de processamento das aplicações entre os núcleos. Isso pressupõe, porém, que a maioria dos softwares tenham sido projetados para tirar proveito do paralelismo disponível
- De fato, o desempenho obtido por meio da utilização de um processador multicore depende do problema a ser resolvido, bem como da sua implementação em software, ou da maneira como ele é concebido segundo um dado paradigma
- Adicionalmente, o forte acoplamento de código gera dificuldades na partição do software em conjuntos independentes de objetos, usando excessivamente mecanismos de sincronização para manter consistência entre os dados
- O emprego desses mecanismos de sincronização para prevenir o acesso concorrente aos dados deve ser controlado explicitamente pelo desenvolvedor. Essa prática dificulta a programação paralela, gerando muitas falhas de programação devido ao esforço adicional atribuído ao desenvolvedor

CONTEXTO DA PESQUISA

- O Paradigma Orientado a Notificações (PON) apresenta alternativas viáveis para a construção de softwares paralelos, como desacoplamento implícito entre as partes das entidades, incluindo particularmente as entidades lógico-causais
- O uso do PON motiva pesquisas que, a princípio, proverão facilidades na paralelização de código. Entretanto, a aplicação do PON para tal deve ser ainda melhor estudada a fim de prover, por exemplo, soluções efetivas para tratar de estratégias de balanceamento de carga. Ainda, em termos práticos, os conceitos do PON foram inicialmente concebidos sobre o Paradigma Orientado a Objetos (POO), por meio de um framework desenvolvido na linguagem de programação C++, o qual é específico para ambientes monoprocessados

CONTEXTO DA PESQUISA

- Portanto, utilizando-se as atuais linguagens e paradigmas de programação, a grande questão é que os SOs não poderão distribuir a carga de processamento de forma adequada entre os núcleos, pois eles não podem antever a variação dinâmica da carga dos programas ao longo do tempo
- Cabe ao desenvolvedor de cada software construir os algoritmos e a arquitetura que distribua sua carga de processamento
- Deste modo, pesquisas sobre distribuição da carga de processamento em plataformas multicore são necessárias. Essas pesquisas permitirão desenvolver novas técnicas, métodos e paradigmas para a construção de software
- A proposta de pesquisa apresentada nesta tese se insere nesse contexto

PROBLEMA

- A construção de software eficiente e de execução paralela é negligenciada frente ao alto crescimento da indústria de hardware (em especial, da tecnologia multicore)
- A capacidade e produtividade dos desenvolvedores de software não acompanha a ascensão da indústria de hardware
- Há aumento da quantidade de problemas complexos a serem resolvidos computacionalmente
- Ainda, as linguagens e os paradigmas de programação existentes, devido ao forte acoplamento de código, dificultam ainda mais a concepção de software paralelo
- Desta forma, esta tese considera o seguinte problema de base
  - **Como desenvolver software que possa fazer uso das infraestruturas de alta capacidade de processamento oferecida pelos processadores multicore?**

PROBLEMA

- O PON apresenta baixo nível de acoplamento entre as partes das entidades sendo que elas podem ser separadas para executarem paralelamente em diferentes núcleos de processamento
- A concepção de software utilizando o PON torna-o atrativo para sua aplicação no desenvolvimento de software paralelo
- Porém, hoje, o PON, implementado como um framework é específico para ambientes monoprocessados
- Com isso, do problema de base apresentado anteriormente, tem-se o seguinte problema decorrente
  - **Não há um modelo nem técnica para tratar explicitamente e usufruir da programação paralela usando o PON**

PROBLEMA

- Os conceitos do PON foram inicialmente implementados sobre o POO, por meio de um framework desenvolvido na linguagem de programação C++
- A versão prototipal foi concebida por Simão e a versão original, intitulada Framework PON, por Banaszewski [4]
- Ainda, recentemente, o Framework PON sofreu evoluções por meio dos trabalhos de Ronszcka e Valença
- Entretanto, ambas as implementações, versão prototipal e original, são específicas para ambientes monoprocessados
- Como consequência, o desenvolvimento de aplicações PON que façam uso de múltiplos processadores é limitada (se usando threads) ou impraticável com o framework atualmente desenvolvido

OBJETIVOS

- O PON trás elementos promissores para a construção de software paralelo uma vez que ele favorece o desacoplamento dos módulos do software. Esta pesquisa explora esse aspecto e tem por objetivo geral
- **Propor um método para distribuição dinâmica da carga de trabalho dos softwares PON em multicore, contribuindo para o melhor aproveitamento da capacidade de processamento do hardware disponível**
- Trata-se de uma nova abordagem a ser empregada para a resolução de problemas desafiadores ligados à programação paralela em ambientes com vários núcleos. Dentro do objetivo geral enunciado, essa pesquisa tem os seguintes objetivos específicos

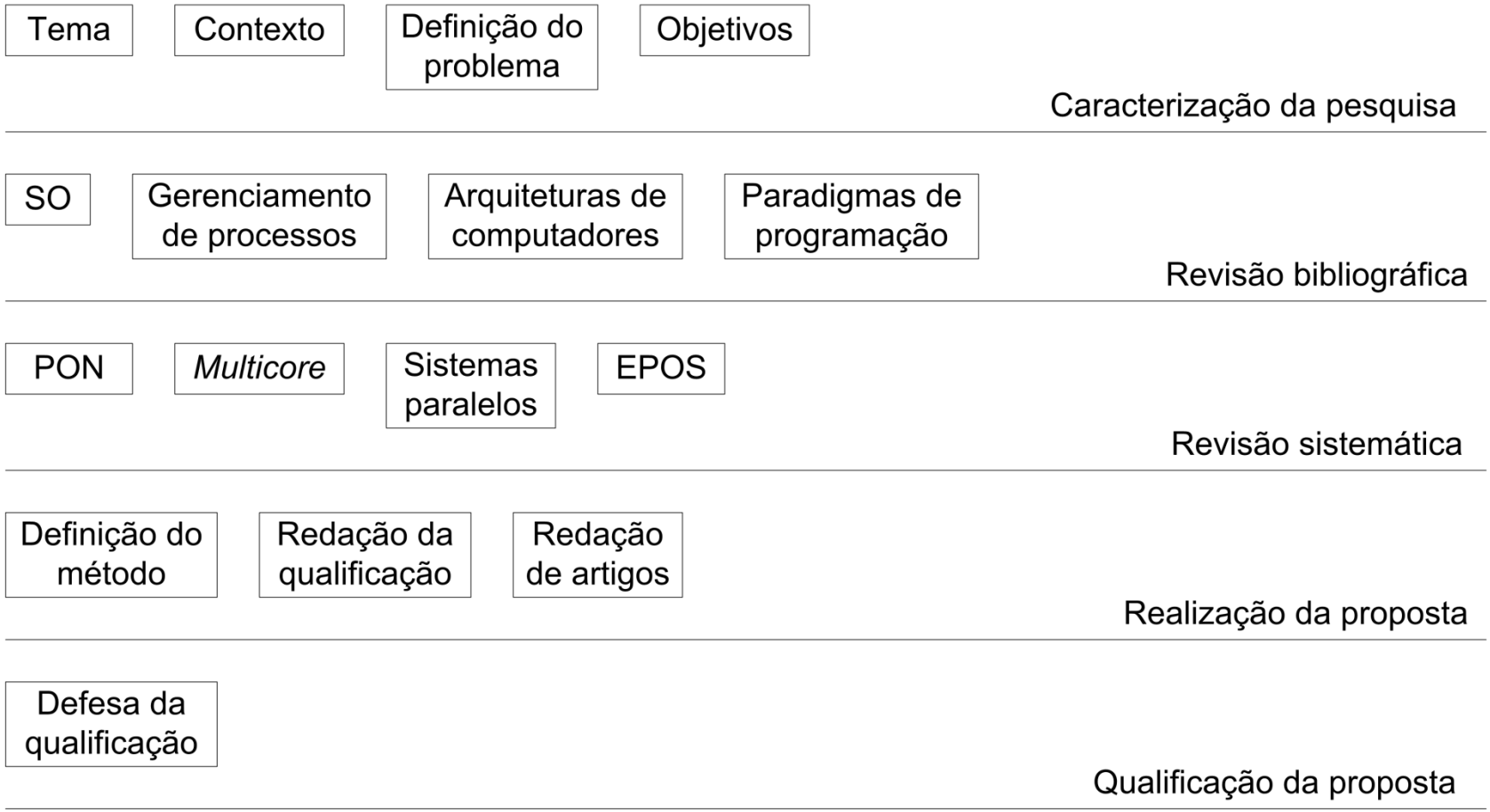
OBJETIVOS

- **Analisar o PON com vistas ao paralelismo de software**
- **Aprofundar o conhecimento sobre a tecnologia multicore visando à concepção de técnicas de paralelismo de software**
- **Propor um conjunto de técnicas para a distribuição da carga de trabalho dos softwares PON em multicore**
- **Avaliar o método proposto por meio de experimentos**

MOTIVAÇÃO

- Percebe-se que existe certa carência nas implementações do PON (frameworks e aplicações), visto que essas não aproveitam os benefícios da possibilidade de processamento paralelo das entidades que compõem um software PON
- Neste sentido, a proposta desse trabalho, além de **viabilizar a distribuição dinâmica da carga de trabalho dos softwares PON em multicore**, **fornece um ambiente de execução** capaz de operacionalizar esse tipo de aplicação

METODO DE PESQUISA ADOTADO



AGENDA

- CONTEXTO
- **FUNDAMENTOS**
- MÉTODO PROPOSTO
- EXPERIMENTOS
- ESFORÇOS
- CRONOGRAMA

FUNDAMENTOS

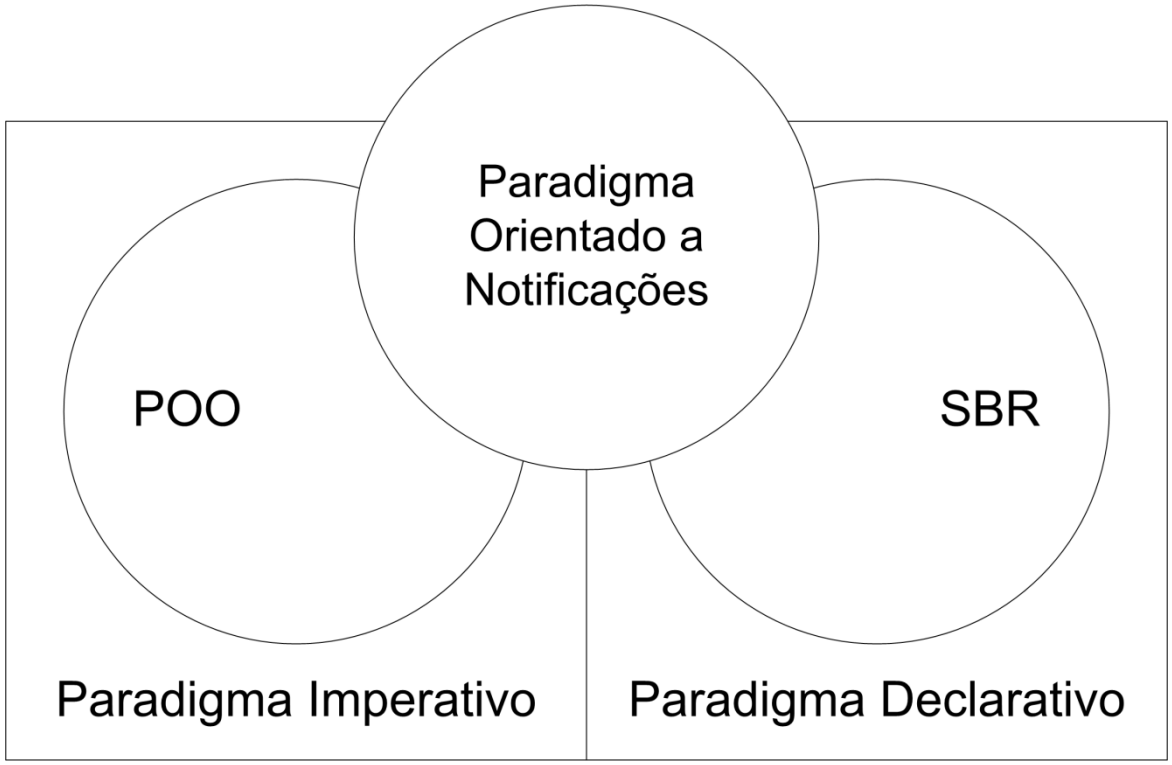
FUNDAMENTAÇÃO TEÓRICA: **PON**

- O Paradigma Orientado a Notificações (PON) foi concebido por Simão (por meio dos seus trabalhos de dissertação e tese), ambos sob a orientação de Stadzisz, no Laboratório de Sistemas Inteligentes de Produção (LSIP) da UTFPR
- O PON foi aprimorado por Banaszewski, Ronszcka e Valença (por meio dos seus trabalhos de dissertação) sob a orientação de Simão e co-orientação de Stadzisz

PON

- O PON se inspira no POO e nos SBRs. Reaproveita os principais conceitos do POO, como a abstração em forma de classes e objetos e a reatividade da programação dirigida a eventos
- Também reaproveita conceitos próprios dos SBR, como a representação do conhecimento como regras e as facilidades da programação declarativa

PON

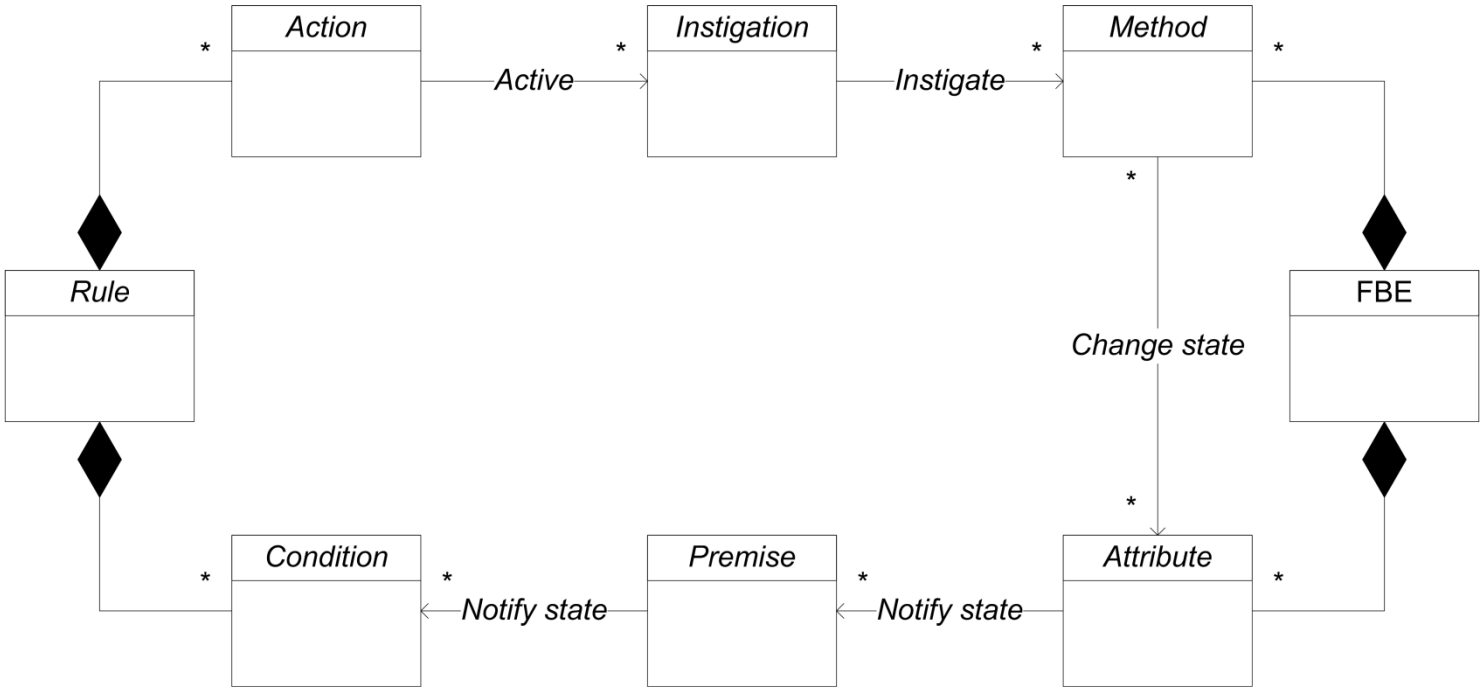


Relação entre o PON, o PI e o PD

Mecanismo de notificações

- O mecanismo de notificações consiste na estrutura interna de execução das instâncias do PON, que determina o fluxo de execução das aplicações
- Por meio desse, as responsabilidades de uma aplicação são divididas entre seus objetos, que cooperam por meio de notificações, informando uns aos outros as suas contribuições para formar o fluxo de execução da aplicação

Mecanismo de notificações



Relação entre as entidades PON

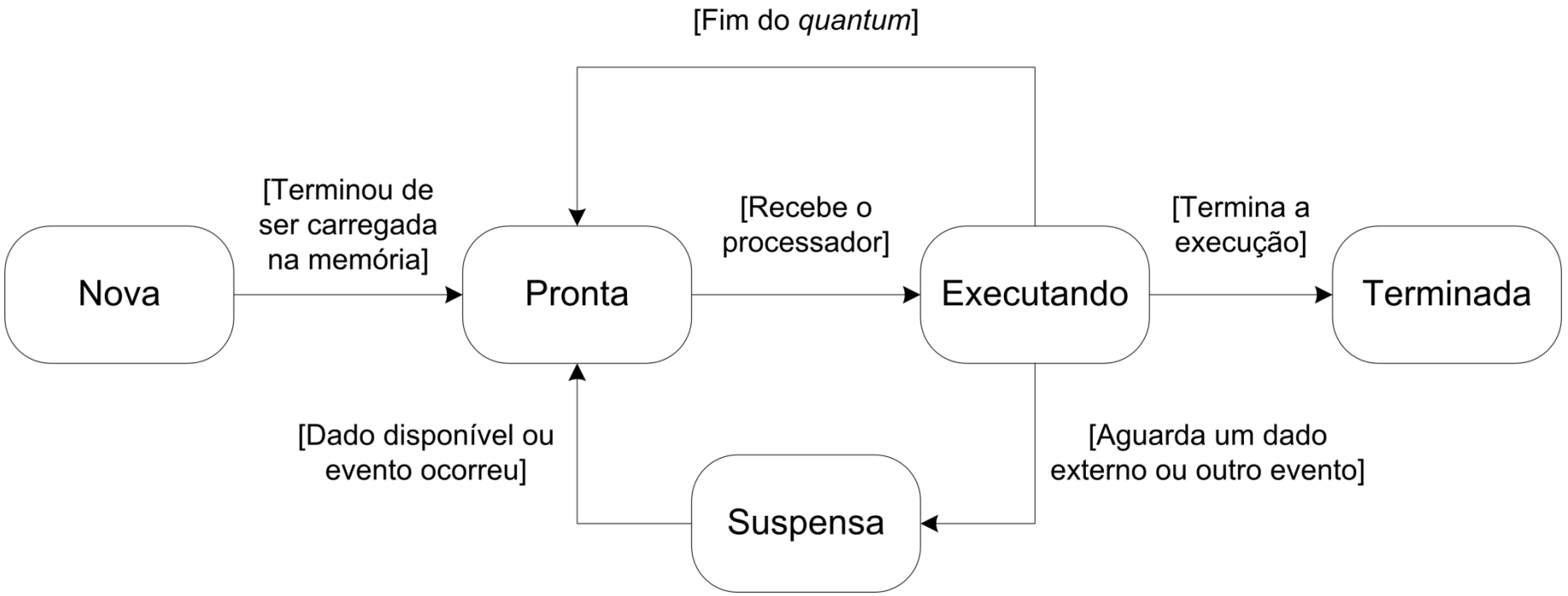
Paralelismo e distribuição no PON

- O PON também pode permitir que o programador obtenha os reais benefícios da computação paralela e distribuída. Esses benefícios se devem a organização da estrutura e comportamento dos componentes do PON, que são alcançáveis de forma transparente pelo programador. Ainda, as características declarativas do PON podem ser adotadas para poupar o programador das particularidades que envolvem a computação multiprocessada
- Os componentes do PON são estruturados e organizados para favorecer a execução paralela. Esses podem ser alocados independentemente a diferentes nós de processamento. Também apresentam comportamento que incentivam a execução nesses ambientes principalmente porque esses se comunicam de forma ativa e pontual por meio de notificações

MEIOS DE ORGANIZAÇÃO DA EXECUÇÃO DE SOFTWARE

- Uma tarefa é definida como sendo a execução de um fluxo sequencial de instruções construído para atender uma finalidade específica
- Possui um estado bem definido a cada instante e, normalmente, são implementadas por threads
- Ainda, as tarefas definem as atividades a serem realizadas dentro do software
- Geralmente, existem mais tarefas a realizar que processadores disponíveis e as tarefas têm diferentes prioridades, desse modo há necessidade de gerenciamento dessas tarefas
- Portanto, a gerência de tarefas tem grande importância dentro de um SO. Cabe ao SO organizar as tarefas definindo uma ordem para executá-las

MEIOS DE ORGANIZAÇÃO DA EXECUÇÃO DE SOFTWARE



Ciclo de vida das tarefas em um sistema de tempo compartilhado

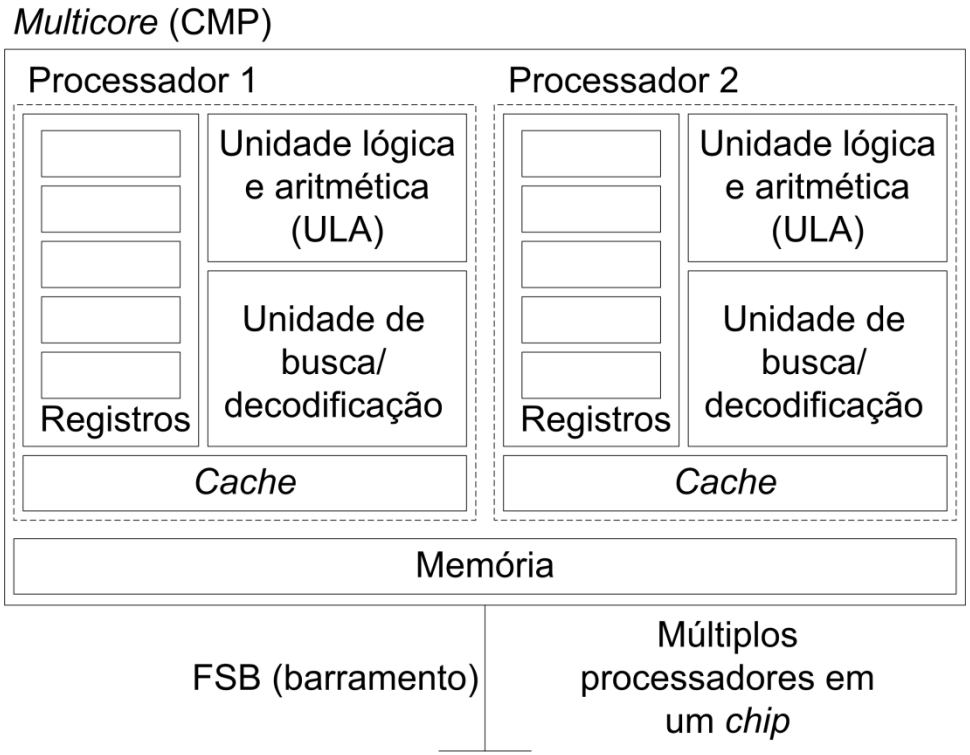
MEIOS DE ORGANIZAÇÃO DA EXECUÇÃO DE SOFTWARE

- Processo é um contêiner de recursos utilizados por uma ou mais tarefas para sua execução e pela própria gerência de tarefas, impedindo que uma tarefa em execução em um determinado processo acesse um recurso atribuído a outro processo
- A necessidade de suportar aplicações com várias threads (i.e. multithreaded) levou os desenvolvedores a incorporar a gerência das threads ao núcleo do SO. O SO realiza o escalonamento de tarefas por meio dos algoritmos de escalonamento

MULTICORE

- Um processador é dito multicore quando possui dois ou mais núcleos de processamento no mesmo chip (que trocam informações via uma memória compartilhada, ou seja, arquitetura SMP)
- Na arquitetura SMP (acrônimo de Symmetric MultiProcessors) os diferentes núcleos são capazes de executar fluxos de instruções de forma independente e compartilham uma área de memória comum
- Essa forma de TLP (acrônimo de Thread Level Parallelism) é também comumente chamada de CMP (acrônimo de Chip-level MultiProcessor). Multicore é um nome popular ao CMP
- Processadores multicore possuem várias configurações sendo que algumas são multithreaded. As abordagens de disposição e comunicação entre os processadores variam entre diferentes implementações

MULTICORE



Arquitetura multicore (CMP)

RECURSOS DE PROGRAMAÇÃO PARA AMBIENTE MULTICORE

- Na **API POSIX Threads** o controle das threads é feito pelo desenvolvedor, que precisa saber gerenciar o código para evitar programações incorretas (e.g. deadlocks). A biblioteca oferece recursos de controle e sincronização das threads. Entretanto, esse controle e sincronização, bem como o balanceamento de carga, disponibilizados pela API, devem ser realizados de forma manual pelo desenvolvedor
- A **linguagem de programação de propósito geral Cilk** apresenta um sistema de execução que se encarrega de fazer o balanceamento de carga e de escalonar as tarefas criadas para que essas executem em paralelo entre os processadores. Porém, isso deve ser informado explicitamente pelo desenvolvedor

RECURSOS DE PROGRAMAÇÃO PARA AMBIENTE MULTICORE

- A **biblioteca Threading Building Blocks** torna a concepção de software paralelo mais fácil, tirando a responsabilidade do desenvolvedor na programação das threads. Mas o que deve ser executado de forma paralela deve ser informado pelo desenvolvedor
- O **pragma de compilador Open MultiProcessing** admite a paralelização de código sequencial por meio de threads. **CUDA** é uma arquitetura de computação paralela e também um pragma de compilador que possibilita aumentos significativos no desempenho de computação pelo aproveitamento da potência da GPU. Em ambos os casos, Open MultiProcessing e CUDA, threads são concebidas para a execução concorrente de códigos. Tais threads devem ser indicadas para serem executadas de forma paralela, por meio de pragmas. Ainda, para que isso ocorra, o compilador deve suportar para pragmas

EPOS

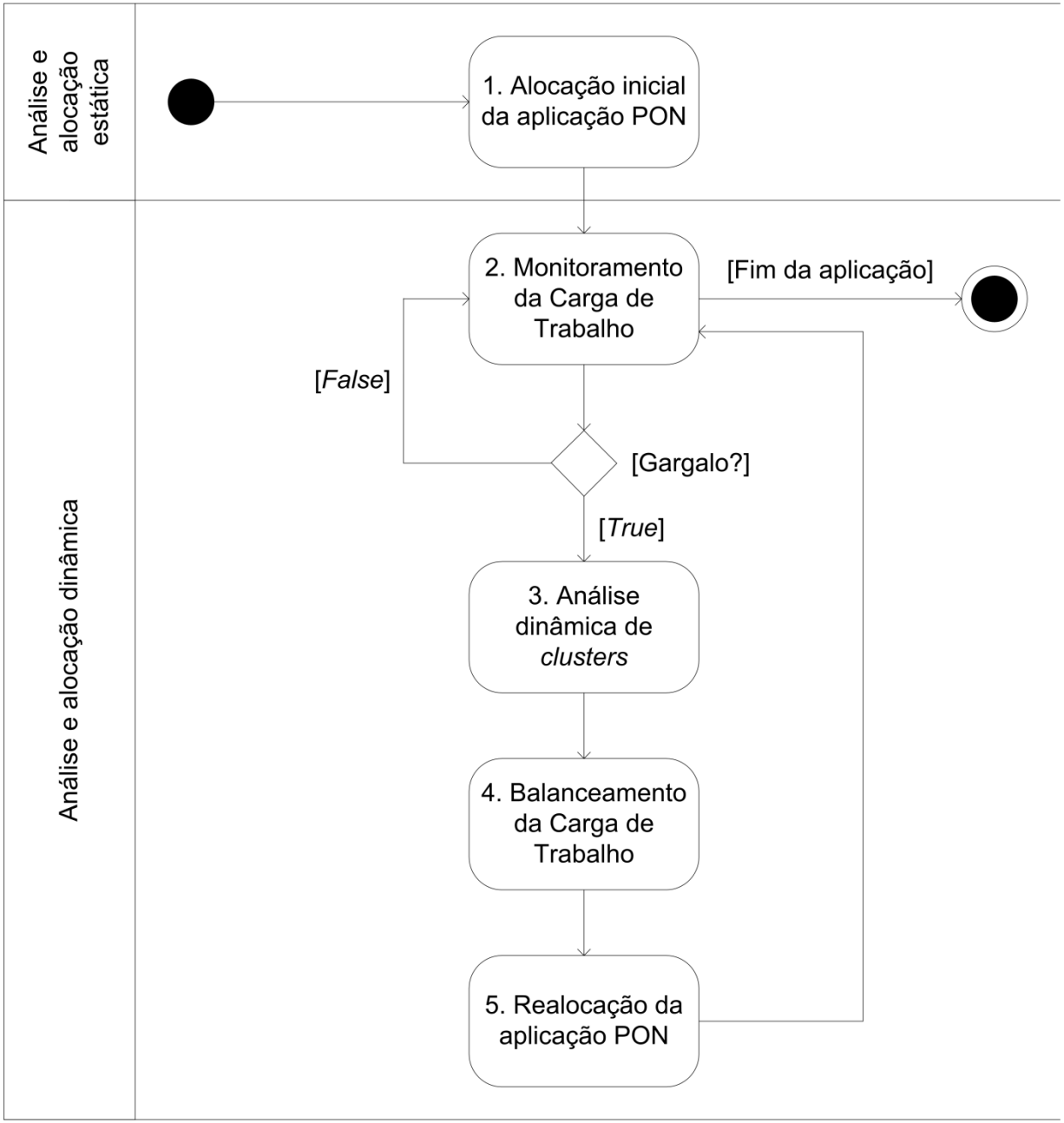
- O Embedded Parallel Operating System (EPOS) é um SO paralelo para plataformas embarcadas baseado em componentes que possui uma arquitetura escalável. É um framework para geração de sistemas que provê suporte a softwares dedicados e independentes de plataforma. Ele implementa aspectos por meio de adaptadores de cenários, que quando combinados com outros componentes do sistema, formam diferentes arquiteturas de software
- O EPOS foi implementado por meio da linguagem de programação C++. Os componentes formam a base e suporte necessário ao sistema, tanto em software quanto em hardware, originados posteriormente a um processo de engenharia de domínio. Tal engenharia de domínio consiste em um desenvolvimento sistemático de um modelo e sua implementação

AGENDA

- CONTEXTO
- FUNDAMENTOS
- **MÉTODO PROPOSTO**
- EXPERIMENTOS
- ESFORÇOS
- CRONOGRAMA

MÉTODO PROPOSTO

- Esforço principal desta pesquisa, a qual visa contribuir para a distribuição da carga de trabalho de software por meio da proposta de um método para distribuição dinâmica da carga de trabalho dos softwares PON em multicore, contribuindo para o melhor aproveitamento da capacidade de processamento do hardware disponível


Análise e alocação estática
Análise e alocação dinâmica
1. Alocação inicial da aplicação PON
2. Monitoramento da Carga de Trabalho
[Fim da aplicação]
[False]
[Gargalo?]
[True]
3. Análise dinâmica de clusters
4. Balanceamento da Carga de Trabalho
5. Realocação da aplicação PON

ETAPA 1
Alocação inicial da aplicação PON

- Etapa responsável por distribuir o software PON (partes do software PON) aos núcleos do processador para que ele seja executado
- Tal distribuição se dá de acordo com as dependências e índices de acoplamento entre as entidades PON, estabelecendo-se grupos (i.e. clusters) de regras PON (Rules)
- Essa etapa inicia-se com a definição de casos de uso. Cada caso de uso provido pela aplicação tem uma colaboração para realizá-lo e, da mesma forma, cada colaboração pode ser “composta” pela execução de um conjunto de Rules que realiza essa colaboração
- A etapa 1 envolve as seguintes atividades

Atividade i

Levantar os casos de uso que compõem a aplicação e estimar a frequência de uso de cada Use Case, empiricamente

- Essa estimativa é um parâmetro configurável para cada caso de uso

Atividade ii

Enumerar e definir quais Rules realizam uma determinada colaboração que, por sua vez, realizam um Use Case

- A atividade ii, assim como a i, deve ser configurável (deve-se informar as Rules que colaboram para as realizações dos casos de uso)
- Para realizar a atividade ii, primeiramente, é necessário estabelecer conjuntos de entidades PON para cada Rule
- Informadas as Rules, pode-se associá-las aos casos de uso definidos na atividade i

Atividade iii

Descobrir a quantidade de núcleos existentes no processador

- Para a realização da atividade iii o método utilizará bibliotecas específicas do SO

Atividade iv
Averiguar a taxa de utilização do processador e, então, alocar o software PON

- Para a realização da verificação da taxa de utilização do processador e, finalmente, alocação do software PON, deve-se: levantar a taxa de utilização dos núcleos, calcular a taxa de disponibilidade e alocar a aplicação (software PON) no núcleo com maior taxa de disponibilidade
- **Nessa primeira etapa não há distribuição da aplicação PON e sim sua alocação integral a apenas um núcleo. Isso se deve ao fato de não se saber se sua capacidade será ou não suficiente para o processamento da aplicação PON**

ETAPA 2
Monitoramento da Carga de Trabalho

- Nesta etapa inicia-se a análise e alocação dinâmica por meio do monitoramento da carga de trabalho. Essa atividade de monitoramento será realizada por um módulo de software responsável por monitorar dinamicamente as cargas de trabalho do processador, denominado MCT
- Em tal monitoramento, caso seja identificado algum gargalo em um determinado intervalo de tempo, avança-se à etapa 3. Caso contrário, o MCT continua o monitoramento dinâmico das cargas de trabalho do software PON que está em execução nos núcleos. Ainda, caso a aplicação seja encerrada, o processo de análise e alocação dinâmica é finalizado
- Essa etapa envolve a seguinte atividade

Atividade v

Analisar a evolução da carga de trabalho dos núcleos, verificar a proximidade de gargalo segundo algum limite e produzir a detecção de gargalo

- O intervalo de tempo de monitoramento deverá ser configurável para cada tipo de aplicação
- Para a análise da evolução da carga de trabalho dos núcleos serão utilizadas bibliotecas específicas do SO
- Baseando-se na taxa de ocupação, ao se chegar ao valor de proximidade de gargalo informado (e.g. 95%), do(s) núcleo(s) do processador, produz-se a sua detecção propriamente dita
- **Nesse estudo considera-se apenas um software PON em execução (i.e. "premissa de processo PON único"). De outra forma, seria necessário trabalhar no gerenciamento de múltiplas aplicações**

ETAPA 3
Análise dinâmica de clusters

- Essa etapa é responsável pela análise e estabelecimento de clusters de entidades PON, levando em consideração a dependência entre essas entidades e seus respectivos índices de acoplamento
- São estabelecidos clusters de Rules do software PON em execução
- Assim, essa etapa envolve as seguintes atividades

Atividade vi
Determinar os índices de acoplamento interno e externo entre as Rules

- Inicialmente é necessária a criação de uma sociomatriz, que tem por finalidade os cálculos IC e EC. As ligações na sociomatriz dão-se entre entidades PON e elas são representadas por meio do conceito de díades (ligações bidimensionais). Tais díades constituem relações de "notificações" (i.e. possíveis notificações) entre essas entidades
  - IC é medido por meio das díades entre as entidades PON que compõem uma mesma Rule
  - EC é calculado por meio das díades entre entidades PON que não pertencem a uma mesma Rule. Além disso, entidades PON que são compartilhadas por duas ou mais Rules também são consideradas no seu cálculo
- Ainda, a estratégia adota pelo método, independentemente da quantidade de núcleos disponíveis no processador é: **"sempre que um determinado núcleo apresentar sobrecarga, a parte do software PON que nele executa poderá ser dividida em duas partes, estabelecendo assim uma relação entre dois novos clusters de Rules. Tais clusters podem ser compostos por uma ou mais Rules"**

Atividade vii
Examinar a taxa de utilização do processador

- Para realizar a verificação da taxa de utilização do processador (dinamicamente), deve-se: levantar novamente a taxa de ocupação e calcular a taxa de disponibilidade.
- Com isso, a próxima etapa do método, após os cálculos dos IC, EC e a verificação da taxa de utilização do processador, é adotar uma estratégia para balancear a carga de trabalho

ETAPA 4
Balanceamento de Carga de Trabalho

- Essa etapa é responsável por definir uma estratégia de distribuição do software PON. Para tanto, utilizam-se os resultados da análise da carga de trabalho do processador, em tempo de execução
- Com isso, por meio do Módulo de Distribuição (MD), essa etapa é responsável por decidir em quais núcleos as entidades PON serão executadas
- O MD é o módulo de software responsável pela aplicação de um algoritmo de balanceamento de carga. Assim sendo, o MD utilizará uma estratégia para a distribuição da aplicação
- Essa etapa envolve as seguintes atividades

Atividade viii

Avaliar as notificações entre entidades PON internas a uma mesma Rule e os overheads entre Rules em diferentes núcleos

- Os custos por notificação (entre entidades PON internas a uma mesma Rule) e do overhead (entre Rules em diferentes núcleos) devem ser estimados (informados) na primeira vez em que o software PON estiver em execução
- Ao final da etapa 4, tais valores (notificação e overhead) podem ser atualizados com os devidos cálculos efetivos (e não mais estimados), pois já terá ocorrido a distribuição das partes do software PON nos núcleos

Atividade ix

Calcular os custos de processamento por intervalo de tempo de monitoramento por Rule e entre Rules em diferentes núcleos

- Para a realização do cálculo do custo de processamento por intervalo de tempo de monitoramento por Rule deve-se:
  - Multiplicar o índice de acoplamento interno por Rule pelo custo por notificação (entre entidades PON internas a uma mesma Rule) para cada Rule
  - Dividir o intervalo de tempo de monitoramento pela frequência de uso de cada Use Case e, assim, obter o número de execuções de cada Rule
  - Multiplicar os resultados dos cálculos supracitados e, assim, obter o custo total de processamento por Rule
- Para a realização do cálculo do custo de processamento por intervalo de tempo de monitoramento entre Rules em diferentes núcleos deve-se:
  - Multiplicar o índice de acoplamento externo entre Rules em diferentes núcleos pelo custo do overhead (entre núcleos) para cada relação entre Rules e, assim, obter o custo do acoplamento externo
  - Multiplicar o número de execuções do software PON pelo custo do acoplamento externo
  - Dividir o resultado da multiplicação dos cálculos supracitados por dois, conforme estratégia adota pelo método

Atividade x

Distribuir as partes do software PON dentre as possibilidades de clusters, por meio das relações utilizadas para o cálculo do acoplamento externo e também por meio do cálculo do acoplamento interno de cada Rule

- Para distribuir as Rules dentre as possibilidades de clusters por meio das relações utilizadas para o cálculo do acoplamento externo, com base na atividade ix, deve-se:
  - Separar em dois clusters (estratégia adota pelo método na atividade vi) cada relação entre Rules
  - Ponderá-las por meio dos custos de processamento total dos índices de acoplamento interno e externo

Atividade xi

Escolher a melhor relação custo e benefício (i.e. melhor distribuição) das Rules nos clusters dentre as opções apresentadas na atividade x

- Por fim, para escolher a melhor distribuição de Rules nos clusters (dentre as opções definidas na atividade x), deve-se calcular:
  - A razão de disponibilidade entre os núcleos
  - A razão de custo de processamento entre os clusters
  - O módulo (ou valor absoluto) da subtração entre as razões de disponibilidade entre os núcleos e de custo de processamento entre os clusters. Com isso, serão obtidos os valores finais de estratégia
- **Cabe ressaltar que só haverá estratégia se for possível, ou seja, se houver um cluster que puder ser redistribuído. Senão, será verificado se o cluster que mais processa poderia ser ele movido para outro núcleo, pois existem outros processos usando-o também e não se pode interferir neles segundo a premissa de processo PON único e de não intervenção nos processos não PON (i.e. "premissa de não intervenção nos processos não PON")**

ETAPA 5
Realocação da aplicação PON

- Essa etapa é responsável pela realocação da aplicação e, para isso, baseia-se na etapa 4, mais especificamente na estratégia de distribuição adotada
- Essa etapa envolve a seguinte atividade

Atividade xii
Reposicionar as partes do software PON (i.e. Rules) nos núcleos redefinidos

- Para a realização da atividade xii, modificações na estrutura interna do Framework PON se farão necessárias, de modo que ao serem notificadas, as entidades PON não executem imediatamente seu código de controle. Ao invés disso, as entidades se registram em controladores de núcleos, os quais armazenam referências para essas, de modo a gerenciar a execução paralela do sistema, definindo em quais núcleos do processador tais entidades executarão
- Basicamente, os controladores de núcleos representam uma espécie de escalonador de entidades. Ainda, serão instanciados um controlador para cada núcleo presente no processador do respectivo ambiente multiprocessado. Sendo assim, cada entidade ao receber uma notificação se registra no respectivo controlador de núcleo, baseado na estratégia definida pelo método em questão
- Por fim, os controladores de núcleos executam as entidades no momento em que estiverem livres, mantendo a execução do sistema de maneira paralela, sempre que existirem entidades nos núcleos
- Finalmente, volta-se à etapa 2, na qual o módulo de software continua o monitoramento dinâmico das cargas de trabalho do processador

AGENDA

- CONTEXTO
- FUNDAMENTOS
- MÉTODO PROPOSTO
- **EXPERIMENTOS**
- ESFORÇOS
- CRONOGRAMA

EXPERIMENTOS

- O caso de estudo consistiu na implementação de um simulador para o **controle de operação de um avião**
  - **IMPLEMENTAÇÃO NO POO EM MULTICORE COM THREADS NO LINUX**
  - **IMPLEMENTAÇÃO NO PON MONOPROCESSADO NO LINUX**
  - **IMPLEMENTAÇÃO NO POO COM THREADS NO EPOS**

Prova de conceito

- Um toy problem foi implementado, com vistas a avaliar o método, ou seja, verificar que o conceito em questão é suscetível de ser explorado de uma maneira útil
- Basicamente, a aplicação é constituída de um semáforo, que possui dois estados, aberto e fechado, e de carros, que possuem as funcionalidades acelerar e frear

Prova de conceito

- Desta forma, quando o semáforo estiver com o estado aberto, os carros devem acelerar. Caso contrário, ou seja, o semáforo estiver com o estado fechado, os carros devem frear
- Portanto, em se tratando do PON, os **FBEs** são o **Semaphore** e os **Car** e, as **Rules**, **SemaphoreClosed**, **SemaphoreOpened** e **CarBrake** e **CarAccelerate**

Prova de conceito

- Há dois tipos de carros: Maserati e Lamborghini, sendo que o Maserati é 2x mais potente (potência do motor) que o Lamborghini (“peso” do Method, impacto no IC)
- A potência do motor (i.e. POWER) está relacionada às Rules. A CarAccelerate executa um Method POWER e a CarBrake Method POWER/2
- Existem premissas compartilhadas que são as relacionadas ao SemaphoreOpened -> CarAccelerate e SemaphoreClosed -> CarBrake

Prova de conceito

- As execuções dos experimentos foram realizadas em um PC com:
  - **Processador AMD Turion 64 X2 Dualcore 2.0GHz**
  - **Memória RAM de 3GB**
  - **SO Linux (Debian)**
    - Visando evitar resultados imprecisos em relação a preempções

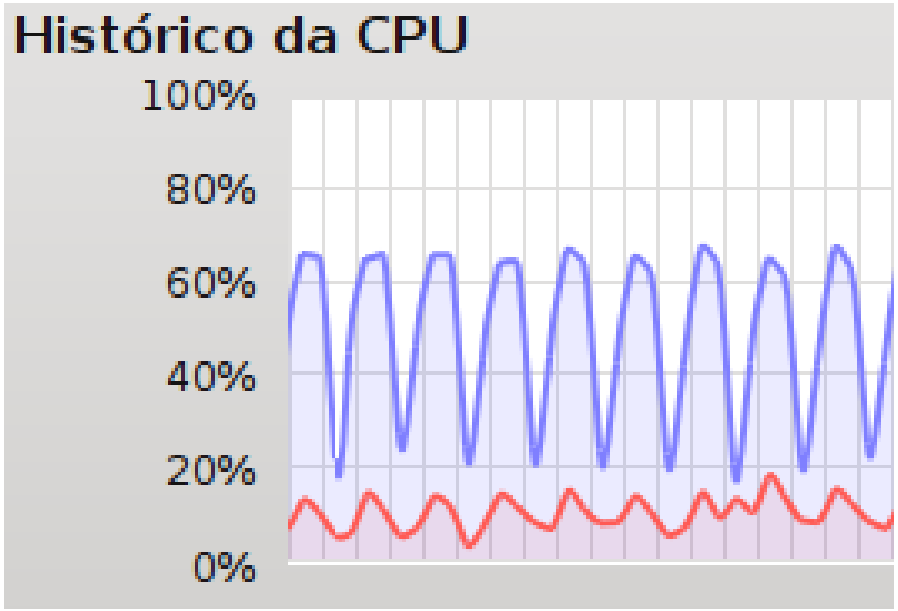
Prova de conceito

- Iniciando o método proposto...
  - Use Case **CarController** é realizado pelas Rules CarBrake e CarAccelerate
  - Use Case **SemaphoreController** é realizado pelas Rules SemaphoreClosed e SemaphoreOpened
  - SemaphoreController = 1000ms
  - TT = 2000ms

Cenário 1

- FBE (1 Semaphore e 15 Car):
  - 10 Maserati e 5 Lamborghini
  - Valor de proximidade de gargalo = 80%

```
Core 1: 91.00% available.
Core 2: 90.38% available.
Allocating NOP software on core 1
Processor: 25.98% occupied.
Core 1: 44.00% occupied.
Core 2: 8.65% occupied.
Processor: 25.37% occupied.
Core 1: 43.00% occupied.
Core 2: 8.00% occupied.
Processor: 26.60% occupied.
Core 1: 42.00% occupied.
Core 2: 11.54% occupied.
Processor: 25.62% occupied.
Core 1: 43.56% occupied.
Core 2: 8.74% occupied.
Processor: 32.84% occupied.
Core 1: 46.00% occupied.
Core 2: 19.61% occupied.
Processor: 27.59% occupied.
Core 1: 44.44% occupied.
Core 2: 9.80% occupied.
Processor: 27.80% occupied.
Core 1: 46.00% occupied.
Core 2: 9.71% occupied.
Processor: 27.45% occupied.
Core 1: 45.00% occupied.
Core 2: 10.48% occupied.
Processor: 28.22% occupied.
```



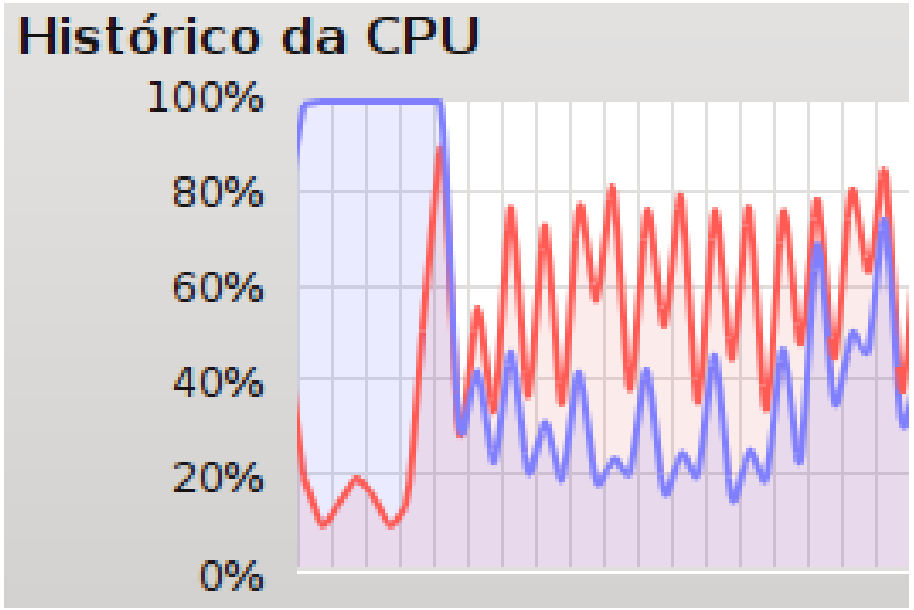
- Alocou o software PON no núcleo 1 e não distribuiu mais, porque a média não atingiu 80% em nenhum dos núcleos

- A linha vermelha (e valores) do núcleo 2 representam apenas a execução do SO. O núcleo 1 representa a execução de todo o software PON e do SO

Cenário 2

- FBE (1 Semaphore e 25 Car):
  - 15 Maserati e 10 Lamborghini
  - Valor de proximidade de gargalo = 80%

```
Core 1: 90.91% available.
Core 2: 56.10% available.
Allocating NOP software on core 1
Processor: 58.71% occupied.
Core 1: 100.00% occupied.
Workload balancing of the core 1
Processor: 92.00% occupied.
Core 1: 100.00% occupied.
Workload balancing of the core 1
Processor: 47.29% occupied.
Core 1: 35.00% occupied.
Core 2: 58.25% occupied.
Processor: 44.28% occupied.
Core 1: 32.00% occupied.
Core 2: 57.43% occupied.
Processor: 45.54% occupied.
Core 1: 34.65% occupied.
Core 2: 56.86% occupied.
Processor: 45.27% occupied.
Core 1: 35.35% occupied.
Core 2: 55.45% occupied.
Processor: 44.55% occupied.
Core 1: 35.35% occupied.
Core 2: 53.92% occupied.
Processor: 50.50% occupied.
Core 1: 39.39% occupied.
Core 2: 61.39% occupied.
Processor: 48.02% occupied.
```



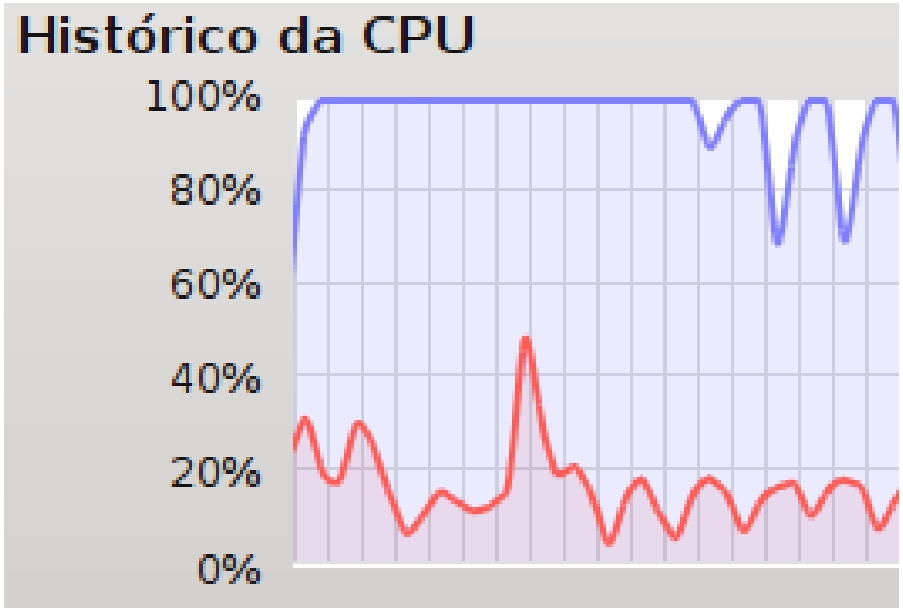
- O núcleo 2 já estava ocupado pelo SO, então o método proposto alocou o software PON no núcleo 1 que apresentava-se mais disponível

- Houveram dois balanceamentos para atingir o ponto de equilíbrio

Cenário 3

- FBE (1 Semaphore e 25 Car):
  - 15 Maserati e 10 Lamborghini
  - Valor de proximidade de gargalo = 80%
    - Utilizando-se apenas um núcleo, ou seja, o balanceamento de carga de trabalho foi desativado para o mesmo cenário anterior

```
Core 1: 86.14% available.
Allocating NOP software on core 1
Processor: 64.18% occupied.
Core 1: 100.00% occupied.
Processor: 57.07% occupied.
Core 1: 100.00% occupied.
Processor: 56.50% occupied.
Core 1: 100.00% occupied.
Processor: 59.50% occupied.
Core 1: 100.00% occupied.
Processor: 57.29% occupied.
Core 1: 100.00% occupied.
Processor: 55.50% occupied.
Core 1: 93.00% occupied.
Processor: 47.00% occupied.
Core 1: 78.79% occupied.
Processor: 48.00% occupied.
Core 1: 79.00% occupied.
Processor: 49.00% occupied.
```



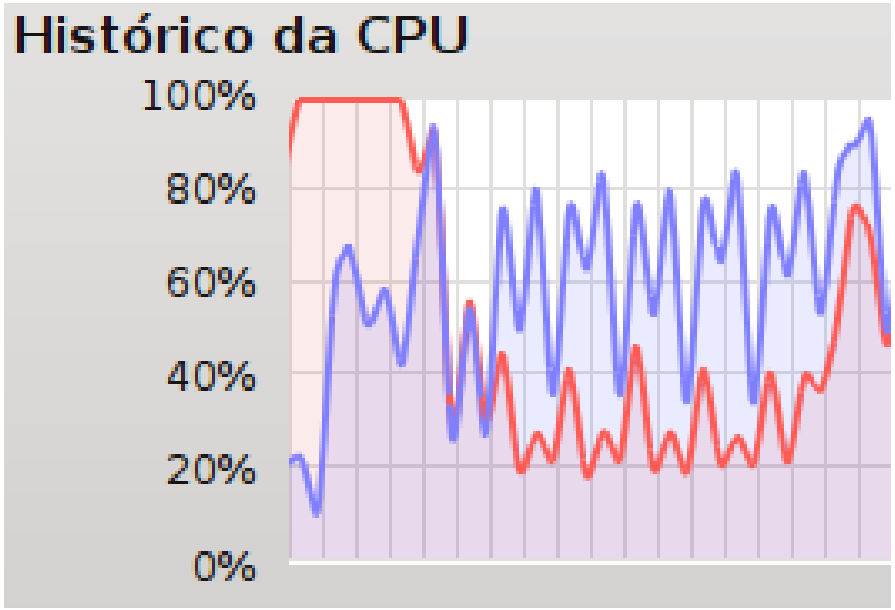
- O núcleo 2 já estava ocupado pelo SO, então o todo o software PON foi alocado no núcleo 1

- Nesse caso o todo o software PON ficará somente no núcleo 1, pois o balanceamento de carga de trabalho foi desativado

Cenário 4

- FBE (1 Semaphore e 25 Car):
  - 15 Maserati e 10 Lamborghini
  - Valor de proximidade de gargalo = 80%
    - Similar ao cenário 2, entretanto, após ponto de equilíbrio, é executado o software youtube e o método começa a realização de novo balanceamento de carga de trabalho

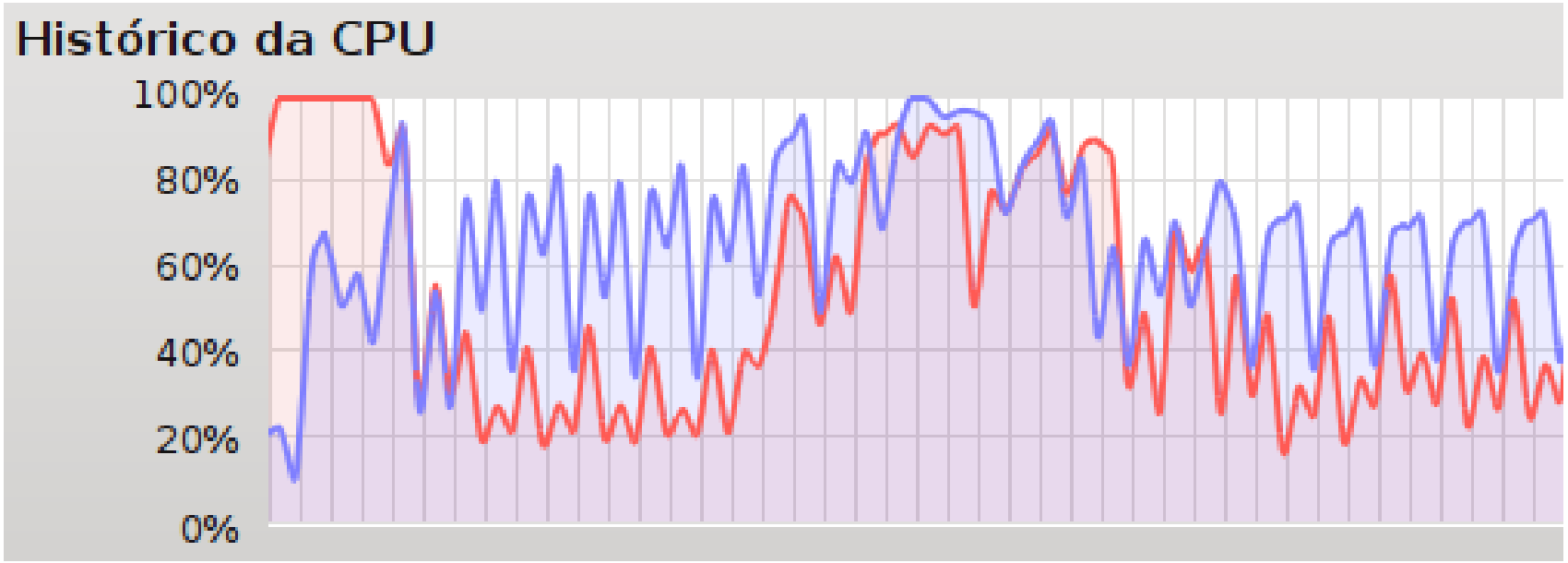
```
Core 1: 50.00% available.
Core 2: 48.00% available.
Allocating NOP software on core 1
Processor: 72.67% occupied.
Core 1: 100.00% occupied.
Available core 1: 0.00% available.
Available core 2: 64.38% available.
Workload balancing of the core 1
Processor: 88.56% occupied.
Core 1: 100.00% occupied.
Available core 1: 0.00% available.
Available core 2: 22.22% available.
Workload balancing of the core 1
Processor: 48.77% occupied.
Core 1: 40.59% occupied.
Core 2: 56.31% occupied.
Processor: 50.75% occupied.
Core 1: 42.42% occupied.
Core 2: 58.42% occupied.
Processor: 48.28% occupied.
Core 1: 40.00% occupied.
Core 2: 56.73% occupied.
Processor: 48.51% occupied.
Core 1: 39.00% occupied.
Core 2: 58.42% occupied.
Processor: 48.76% occupied.
Core 1: 39.00% occupied.
Core 2: 58.00% occupied.
Processor: 47.52% occupied.
Core 1: 37.00% occupied.
Core 2: 57.84% occupied.
Processor: 48.26% occupied.
```



- O núcleo 2 já estava ocupado pelo SO, então o método proposto alocou o software PON no núcleo 1 que apresentava-se mais disponível

- Houveram dois balanceamentos para atingir o ponto de equilíbrio

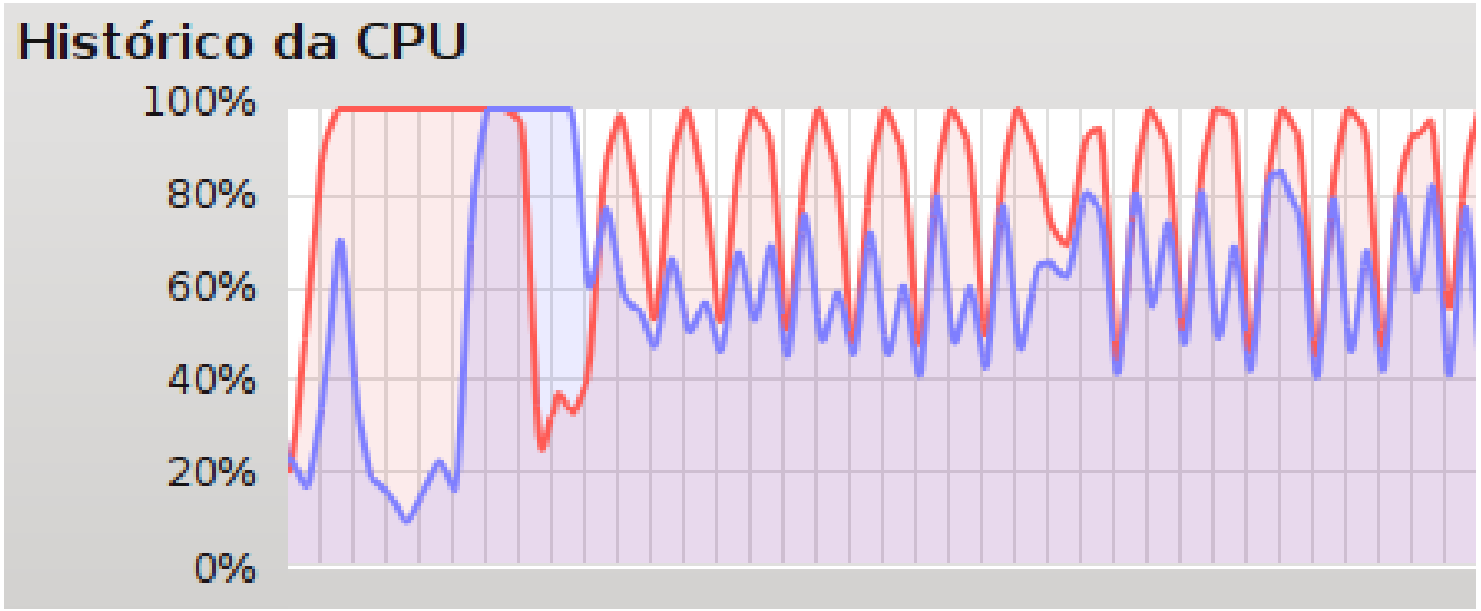
- Abriu-se o youtube e então iniciou-se o balanceamento de carga do software PON...



... até atingimento de novo ponto de equilíbrio

Cenário 5

- FBE (1 Semaphore e 36 Car):
  - 18 Maserati e 18 Lamborghini
  - Valor de proximidade de gargalo = 80%



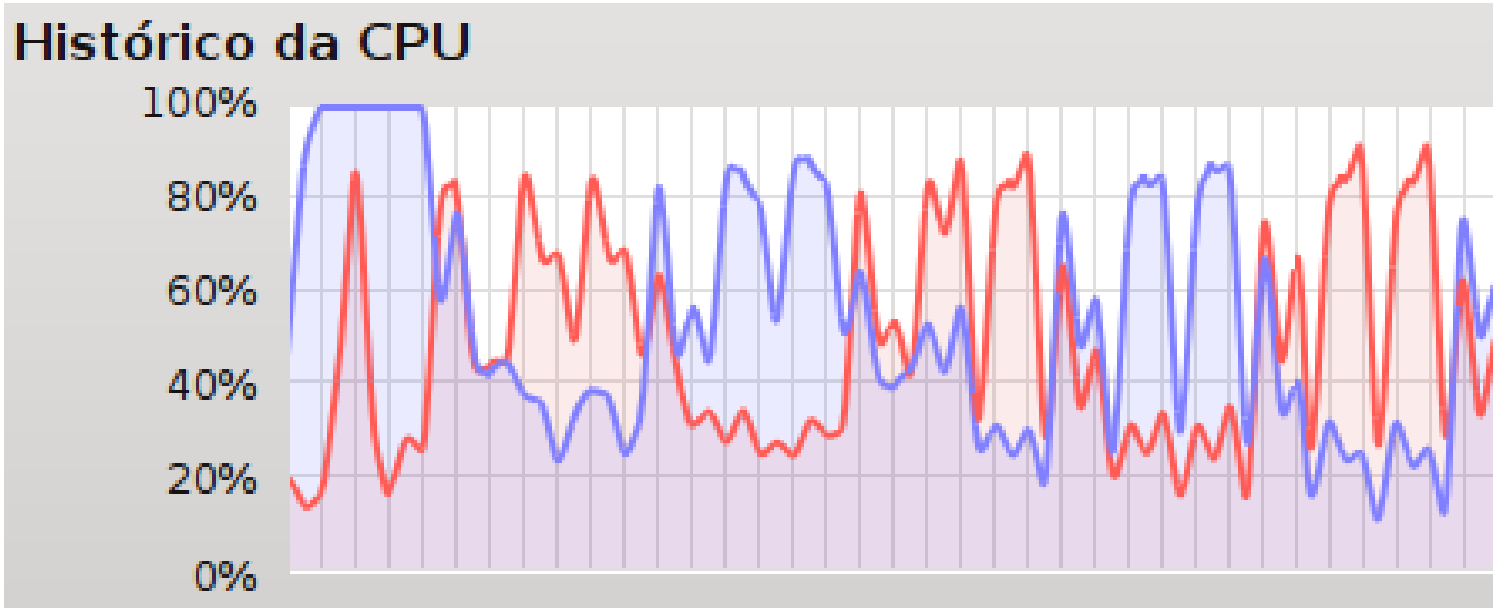
- O núcleo 1 já estava ocupado pelo SO, então o método proposto alocou o software PON no núcleo 2 que apresentava-se mais disponível

- Houveram quatro balanceamentos para atingir o ponto de equilíbrio

- Apesar de ambos os núcleos estarem utilizando boa parte de sua capacidade, eles não chegaram a atingir o valor de proximidade de gargalo, mantendo a execução estável, sem novos balanceamentos de carga de trabalho

Cenário 6

- FBE (1 Semaphore e 25 Car):
  - 15 Maserati e 10 Lamborghini
  - Valor de proximidade de gargalo = 50%
    - O objetivo aqui foi tentar chegar em um cenário estável, ou seja, manter a média de utilização dos núcleos em uma taxa de ocupação de aproximadamente 50% de processamento, oscilando o balanceamento de carga de trabalho entre os núcleos disponíveis (nesse caso, núcleos 1 e 2)



- Sempre, após o atingimento do valor de proximidade de gargalo, que é de 50%, é realizado um novo balanceamento de carga de trabalho entre os núcleos disponíveis

AGENDA

- CONTEXTO
- FUNDAMENTOS
- MÉTODO PROPOSTO
- EXPERIMENTOS
- **ESFORÇOS**
- CRONOGRAMA

ESFORÇOS

- 2009 e 2010: Realização dos créditos do doutorado em quatro trimestres
  - Coeficiente de rendimento 10,0
  - 48 créditos exigidos completados
- 2009: Realização da proficiência em língua inglesa no primeiro ano
- 2010: Estratégia de escrita de três artigos para:
  - XI Workshop em Sistemas Computacionais de Alto Desempenho (WSCAD 2010)
  - XIV Congresso Ibero-Americano em Engenharia de Software (CIbSE 2011)
  - I Workshop on Autonomic Distributed Systems (WoSIDA 2011)

ESFORÇOS

- 2010: Co-orientação de IC (Iniciação Científica):
  - Acadêmico de Engenharia Eletrônica Lucas Weber
  - Atividades de pesquisa:
    - Estado da arte (confecção de relatório) e da técnica
    - Trabalho: Viabilidade do Controle Orientado a Notificações (CON) em ambiente concorrente
  - Esforços na orientação e escrita de artigo para:
    - XV Seminário de Iniciação Científica e Tecnológica da UTFPR (SICITE 2010)

ESFORÇOS

- 2011: Esforços na orientação e escrita de artigos para:
  - Mestrandos Ronszcka (CPGEI) e Venâncio (PPGCA)
    - III Congreso Internacional de Computación y Telecomunicaciones (COMTEL 2011)
    - IEEE CS International Conference on Software Science, Technology, and Engineering (SwSTE 2012)

ESFORÇOS

- Artigos completos publicados em periódicos (2012):
  1. **BELMONTE, D.**; SIMÃO, J.; STADZISZ, P. **Proposta de um Método para Distribuição da Carga de Trabalho usando o Paradigma Orientado a Notificações (PON)**. Revista SODEBRAS, v. 7, p. 10-17, 2012.
  2. SIMÃO, J.; **BELMONTE, D.**; VALENÇA, G.; BATISTA, M.; LINHARES, R.; BANASZEWSKI, R.; FABRO, J.; TACLA, C.; STADZISZ, P.; RONSZCKA, A. **A Game Comparative Study: Object-Oriented Paradigm and Notification-Oriented Paradigm**. Journal of Software Engineering and Applications (JSEA), v. 5, p. 722-736, 2012.
  3. SIMÃO, J.; **BELMONTE, D.**; RONSZCKA, A.; LINHARES, R.; VALENÇA, G.; BANASZEWSKI, R.; FABRO, J.; TACLA, C.; STADZISZ, P.; BATISTA, M. **Notification Oriented and Object Oriented Paradigms comparison via Sale System**. Journal of Software Engineering and Applications (JSEA), v. 5, p. 695-710, 2012.

ESFORÇOS

- Trabalhos completos publicados em anais de congressos (2010, 2011 e 2012):
  4. SIMÃO, J.; STADZISZ, P.; TACLA, C.; LINHARES, R.; BELMONTE, D.; BANASZEWSKI, R. **Comparações entre duas materializações do Paradigma Orientado a Notificações (PON): Framework PON Prototipal versus Framework PON Primário**. In: IV Congreso Internacional de Computación y Telecomunicaciones (COMTEL). Lima, 2012.
  5. RONSZCKA, A.; **BELMONTE, D.**; VALENÇA, G.; BATISTA, M.; LINHARES, R.; TACLA, C.; STADZISZ, P.; SIMÃO, J. **Comparações quantitativas e qualitativas entre o Paradigma Orientado a Objetos e o Paradigma Orientado a Notificações sobre um simulador de jogo**. In: III Congreso Internacional de Computación y Telecomunicaciones (COMTEL). Lima, 2011.
  6. WEBER, L.; **BELMONTE, D.**; STADZISZ, P.; SIMÃO, J. **Viabilidade de Controle Orientado a Notificações (CON) em ambiente concorrente baseado em threads**. In: XV Seminário de Iniciação Científica e Tecnológica (SICITE). Cornélio Procópio, 2010.

AGENDA

- CONTEXTO
- FUNDAMENTOS
- MÉTODO PROPOSTO
- EXPERIMENTOS
- ESFORÇOS
- **CRONOGRAMA**

CRONOGRAMA

- No caso de estudo implementação parcial do simulador para o controle de operação de um avião no POO com threads no EPOS, emulado no Linux por meio do QEMU, apesar da implementação realizada inicialmente no POO (em multicore com threads), a adaptação para o EPOS não se deu naturalmente, devido ao fato do EPOS também ter sido concebido por meio de um framework (mesmo que em linguagem de programação C++). Tal framework não usa, por exemplo, a estrutura vector, apresentando uma implementação própria. O mesmo ocorre com o uso de threads e outras bibliotecas para distribuição em ambiente multicore
  - **Servidor Dell PowerEdge R710**
  - **Dois processadores Intel Xeon X5675 Sixcore 3.07GHz**
  - **Software de Virtualização VMWare ESXi 5.1.0**

CRONOGRAMA

- Será virtualizado um ambiente para testes executando um servidor virtual sobre um servidor físico. Tal ambiente permitirá maior utilização de recursos de hardware, tais como **memória, processador e espaço em disco**, enquanto garantirá que o isolamento e a segurança sejam mantidos. Isso prevenirá que uma aplicação afete outra quando se fizer uma atualização ou mudança. Ainda, o **ambiente único do SO será não preemptivo**
  - **Processador Intel Xeon X5675 de oito núcleos 3.07GHz**
  - **Memória RAM de 12GB**
  - **SO Debian Kernel Linux 2.6.32-5-amd64 GNOME 2.30.2**

CRONOGRAMA

- Essa pesquisa qualifica um trabalho de tese
- Desta forma, **ajustar as considerações a serem feitas pela banca** e **realizar mais experimentos**, agora:
  - Com o caso de estudo do simulador de controle de operação de um avião e
  - Com a nova (super) máquina virtualizada (portanto, testes mais robustos, fidedignos)
- **Conceber um artigo para um Journal nas Engenharias IV**

DÚVIDAS, PERGUNTAS?

Obrigado pela atenção!