

Modelos de Programação Linear Inteira para Geração de Executivos Cíclicos Multicore com Dependências Entre Tarefas

Bruno Nogueira¹, Cleiber de Meireles Silva Junnior¹, Pedro Rocha¹, Rodrigo Paes¹, Alfredo Lima², Eduardo Tavares³, Francisco Airtton Silva⁴

¹Universidade Federal de Alagoas (UFAL)

Instituto de Computação, Av. Lourival Melo Mota, s/n, Tabuleiro dos Martins,
CEP: 57072-900, Maceió, AL, Brasil

²Universidade Federal Fluminense (UFF)

Rua Passo da Pátria, 156, CEP: 24210-240, Niterói, RJ, Brasil

³Universidade Federal de Pernambuco (UFPE)

Av. Jornalista Aníbal Fernandes, s/n, CEP: 50740-560, Recife, PE, Brasil

⁴Universidade Federal do Piauí (UFPI)

R. Cícero Duarte, 905, CEP: 64607-670, Picos, PI, Brasil

{bruno, cmsj, phbr, rodrigo}@ic.ufal.br, alfredolima@id.uff.br,
eagt@cin.ufpe.br, faps@ufpi.edu.br

RESUMO

A geração de executivos cíclicos (ECs) em plataformas *multicore* amplia desafios antes restritos a sistemas *unicore*. Neste trabalho, propomos modelos baseados em Programação Linear Inteira (ILP) para construção de ECs em ambientes *multicore*, integrando explicitamente relações de precedência e exclusão entre tarefas, aspectos críticos negligenciados em trabalhos anteriores. Os modelos propostos permitem gerar ECs preemptivos e não preemptivos sob esquemas de alocação particionada e global, garantindo previsibilidade temporal mesmo em cenários com interdependências complexas. Os resultados experimentais demonstram que os modelos produzem executivos cíclicos ótimos para diversos conjuntos de tarefas, comprovando sua eficácia em cenários de pequena e média escala.

PALAVRAS CHAVE. Escalonamento. Sistemas de tempo real. Sistemas embarcados. Programação linear inteira.

Eixos temáticos: OA – Outras Aplicações em PO. OD – Otimização Discreta

ABSTRACT

The implementation of Cyclic Executives (CEs) on multicore platforms extends challenges previously confined to uniprocessor systems. In this work, we propose Integer Linear Programming models for constructing CEs in multicore environments, explicitly integrating precedence and exclusion relationships between tasks, critical aspects neglected in previous works. The proposed models allow the generation of both preemptive and non-preemptive CEs under partitioned and global allocation schemes, ensuring temporal predictability even in scenarios with complex interdependencies. Experimental results show that the models produce optimal cyclic executives for various task sets, demonstrating their effectiveness in small- and medium-scale scenarios.

KEYWORDS. Scheduling. Real-time systems. Embedded systems. Integer linear programming.

Thematic axes: OA – Other Applications in OR. OD – Discrete Optimization.

1. Introdução

Em ambientes críticos, onde até mesmo um microssegundo de atraso pode comprometer vidas ou operações, a previsibilidade oferecida pelos sistemas de tempo real torna-se não apenas desejável, mas obrigatória. O escalonamento por meio de Executivos Cíclicos (EC) é uma técnica de projeto para sistemas de tempo real que define, de maneira *offline*, uma escala periódica para a execução de tarefas. Essa abordagem assegura o atendimento dos períodos e *deadlines* de cada tarefa [Baker e Shaw, 1989; Burns et al., 1995; Liu, 2000; Erciyes e Erciyes, 2019]. A escala repete-se a cada *hiperperíodo* (também chamado de ciclo maior), intervalo após o qual o padrão de execução reinicia integralmente. O hiperperíodo é subdividido em *frames* (também chamados de ciclos menores), que constituem as janelas de ativação nas quais as tarefas devem ser alocadas.

O problema de geração automatizada de ECs, insere-se na categoria de escalonamento cíclico [de Dinechin e Kordon, 2014], também conhecida como problema de escalonamento periódico (Periodic Scheduling Problem - PSP) [Minaeva e Hanzálek, 2021]. Esses problemas envolvem o escalonamento de tarefas que devem ser executadas repetidamente, de forma indefinida, e possuem aplicações em diversos domínios, tais como: manufatura [Glass, 1994] e manutenção de máquinas [Wei e Liu, 1983]. Em sistemas de manufatura, por exemplo, as tarefas frequentemente representam etapas de produção que devem ser repetidas para a fabricação de múltiplas unidades de um produto.

Os conceitos de ECs são particularmente valorizados em sistemas de tempo real críticos devido ao seu comportamento determinístico e ao seu baixo *overhead*. Por exemplo, no padrão ARINC 657, uma especificação importante para arquitetura de software em sistemas aviônicos, as tarefas são executadas ciclicamente em *frames* predefinidos, garantindo isolamento temporal e previsibilidade [Parkinson, 2017]. De forma semelhante, o padrão de certificação IEC 61508 recomenda o uso de ECs para garantir restrições temporais em aplicações críticas [Perez et al., 2014]. Devido ao seu baixo *overhead*, os ECs também são usados em sistemas embarcados de tempo real com restrições de recursos [Aerabi et al., 2022].

Historicamente, sistemas de tempo real altamente críticos dependem de processadores construídos sob medida para garantir o comportamento temporal previsível [Deutschbein et al., 2019]. Esses processadores são tipicamente *unicore*, facilitando a garantia da previsibilidade temporal, razão pela qual a maioria das técnicas de geração de ECs foca em sistemas *unicore*. No entanto, a crescente disponibilidade de processadores *multicore* comerciais *off-the-shelf* (COTS) (por exemplo, LPC4300¹, RP2040², P8X32A³) oferece novas oportunidades devido às vantagens que esses processadores oferecem em termos de redução de tamanho, peso e consumo de energia. No entanto, a implementação de ECs em plataformas *multicore* introduz desafios não presentes em sistemas *unicore*, tanto em termos de implementação quanto de certificação [Parkinson, 2017].

Um dos principais desafios em ECs *multicore* é gerenciar relações entre tarefas, como exclusão mútua e precedência. Em ECs *unicore*, onde geralmente não existe preempção, o acesso exclusivo aos recursos compartilhados é garantido de forma inerente. Em contraste, em ECs *multicore*, tarefas executando em diferentes *cores* podem acessar recursos compartilhados simultaneamente, levando a *race conditions*, a menos que medidas explícitas sejam tomadas para impor exclusão mútua. Além disso, o acesso simultâneo a recursos compartilhados (e.g., barramentos compartilhados) pode gerar contenção de recursos, dificultando a previsão do pior caso de tempo de execução (Worst-Case Execution Time - WCET).

A ausência de métodos automatizados que tratem adequadamente esses problemas em ECs *multicore* motivou nosso trabalho. Neste artigo, propomos uma solução para esses desafios

¹<https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/lpc4300-arm-cortex-m4-m0>

²<https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf>

³<https://www.parallax.com/propeller-1/>

introduzindo modelos de Programação Linear Inteira (Integer Linear Programming - ILP) para gerar ECs ótimos, preemptivos e não preemptivos, considerando estratégias de alocação particionada e global. Nossa abordagem considera especificamente relações de precedência e exclusão entre tarefas, frequentemente negligenciadas em trabalhos existentes. Além disso, o uso de esquemas de alocação particionada mitiga a contenção de recursos, outro problema normalmente ignorado por métodos atuais. Avaliamos nossos modelos utilizando tanto instâncias sintéticas (geradas artificialmente) quanto *benchmarks* da literatura, abrangendo milhares de tarefas e interdependências complexas.

Este artigo está estruturado da seguinte forma. A Seção 2 formula o problema, enquanto a Seção 3 revisa os trabalhos relacionados. A Seção 4 introduz os modelos propostos. A Seção 5 apresenta a avaliação experimental dos nossos métodos, e a Seção 6 conclui o artigo.

2. Formulação do problema

A formulação do problema baseia-se no trabalho de Deutschbein et al. [2019]. Considere um sistema composto por um conjunto de N tarefas periódicas, denotado por $\tau = \{\tau_i = (C_i, T_i)\}_{i=1}^N$, que devem ser escalonadas em uma plataforma *multicore* com m *cores* idênticos. Cada tarefa τ_i é caracterizada pelo WCET C_i , um período T_i e um *deadline* relativo igual ao seu período. Devido à sua natureza periódica, τ_i gera um *job* em cada instante $k \times T_i$, onde $k \in \mathbb{N}$, e esse *job* deve ser finalizado até $(k + 1) \times T_i$.

Definimos P como o mínimo múltiplo comum (MMC) dos períodos de todas as tarefas em τ , e F como o máximo divisor comum (MDC) desses períodos. Os valores de P e F determinam, respectivamente, a duração do ciclo maior (hiperperíodo) e o tamanho do *frame*. Em cada par (*frame*, *core*), pode haver zero, um ou mais *jobs* programados para execução. Adicionalmente, assume-se que os *cores* da plataforma *multicore* operam de forma sincronizada, garantindo que os *frames* em todos os *cores* possuam a mesma duração e alternem de forma simultânea, conforme discutido por Burns et al. [2015].

As relações entre tarefas são representadas por dois conjuntos: $prec_i$ (tarefas que precedem τ_i) e $excl_i$ (tarefas com exclusão mútua em relação a τ_i). Se $\tau_j \in prec_i$, então $T_j = T_i$, e o k -ésimo *job* de τ_i só inicia após a conclusão do k -ésimo *job* de τ_j , modelando dependências temporais. Já $excl_i$ garante que *jobs* de tarefas conflitantes não executem simultaneamente, evitando, por exemplo, acesso concorrente a recursos críticos.

O objetivo central é gerar um escalonamento viável no intervalo $[0, P)$, alocando todos os *jobs* em *frames* respeitando restrições de tempo, recursos e interdependências. É possível facilmente mostrar que este problema é $\mathcal{NP}$ -difícil por sua redução ao clássico problema do empacotamento.

Exemplo 1 Considere τ com cinco tarefas: $\tau_1(C_1 = 2, T_1 = 4)$, $\tau_2(C_2 = 1, T_2 = 6)$, $\tau_3(C_3 = 2, T_3 = 12)$, $\tau_4(C_4 = 2, T_4 = 6)$ e $\tau_5(C_5 = 0.75, T_5 = 6)$. O MMC dos períodos é $\text{mmc}(4, 6, 12) = 12$ e o MDC é $\text{mdc}(4, 6, 12) = 2$, resultando em $P = 12$ e $F = 2$. Definem-se as relações $\tau_2 \in prec_4$, $\tau_1 \in excl_2$ e $\tau_5 \in excl_2$.

A Figura 1(a) ilustra um escalonamento não preemptivo viável para este exemplo. Suponha agora que τ_3 seja alterada para $\tau_3(C_3 = 8, T_3 = 6)$. Como $C_3 > F$, um EC não preemptivo torna-se inviável, pois τ_3 não cabe em apenas um *frame*. Uma solução prática é construir um EC preemptivo dividindo τ_3 em quatro subtarefas $\tau_{3,i}(C_i = 2, T_3 = 12)$, $i = 1, 2, 3, 4$. A Figura 1(b) mostra um EC preemptivo viável. Neste trabalho, investigamos a geração automática de ECs preemptivos e não preemptivos.

Dado um escalonamento, seja f o máximo de tempo de computação alocado a qualquer *frame* (no Exemplo 1, $f = 2$). Seguimos Deutschbein et al. [2019] e tomamos a minimização de f como função objetivo, pois isso reduz a probabilidade de estouro do *frame* em caso de estimativas de

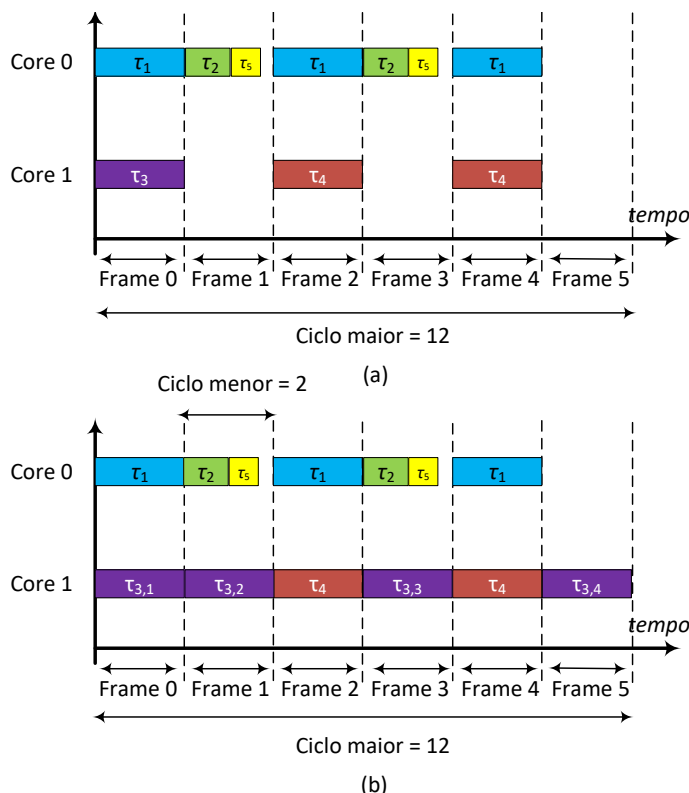


Figura 1: Exemplos de executivos cíclicos: (a) não preemptivo, e (b) preemptivo.

WCET erradas (subestimadas). Note que, se $f > F$, o escalonamento é inviável; então, a razão f/F indica o fator de aumento de velocidade do processador necessário para torná-lo viável.

Por fim, definimos a seguinte terminologia adicional: um EC é dito *particionado* se tarefas não migram entre os *cores* do processador durante a execução, e *global* se migram entre eles. Para simplificar o suporte em tempo de execução do escalonamento global, assumimos que, se um *job* iniciar em um frame, ele só poderá migrar para um *cores* distinto no *frame* seguinte.

3. Trabalhos relacionados

Conforme discutido por Deutschbein et al. [2019], a prática atual de desenvolvimento de ECs frequentemente se baseia em uma abordagem manual e informal, que depende fortemente da experiência dos projetistas. No entanto, esse método torna-se impraticável à medida que as instâncias do problema aumentam, frequentemente exigindo a imposição de restrições adicionais na carga de trabalho, como a necessidade de períodos harmônicos.

Como alternativa, a comunidade de pesquisa desenvolveu diversos métodos para a geração automatizada de ECs, que podem ser, de maneira geral, categorizados da seguinte forma (ou como combinações destes): (i) *model-checking*, como abordagens baseadas em redes de Petri [Tavares et al., 2010; Rubio-Anguiano et al., 2021] e autômatos temporizados [Ravn e Schoeberl, 2012]; (ii) métodos baseados em heurísticas, como busca tabu [Tamas-Selicean e Pop, 2011], *simulated annealing* [Burns et al., 1995] e heurísticas gulosas [Burns e Baruah, 2017; Aerabi et al., 2022]; (iii) programação por restrições [Yépez et al., 2006; Schild e Würtz, 1998]; (iv) métodos baseados em *Satisfiability Modulo Theories (SMT)* [Minaeva e Hanzálek, 2021; Kyriakakis et al., 2021]; e (v) métodos baseados em ILP [Deutschbein et al., 2019; Nogueira et al., 2019; Aerabi et al., 2022].

Os métodos nas categorias (i), (iii), (iv) e (v) podem garantir resultados ótimos: se existir um escalonamento viável, esses métodos serão capazes de encontrá-lo. No entanto, ferramentas de

model-checking enfrentam limitações devido ao problema de explosão de estados, o que restringe a escalabilidade à medida que o tamanho do problema aumenta. A principal vantagem dos métodos baseados em SMT e ILP sobre a programação por restrições é que eles permitem a definição de uma função objetivo, facilitando a otimização em múltiplas métricas de projeto durante o processo de escalonamento. Embora métodos heurísticos não garantam otimalidade, eles geralmente oferecem um bom equilíbrio entre custo e benefício. Uma limitação chave dos métodos heurísticos existentes, no entanto, é a falta de comparações com métodos exatos, dificultando a avaliação de quão próximas as soluções heurísticas estão do ótimo verdadeiro.

A estrutura dos escalonamentos usados nos métodos de geração automatizada de ECs varia. Muitas abordagens adotam uma estrutura de escalonamento mais flexível [Rubio-Anguiano et al., 2022; Kyriakakis et al., 2021; Nogueira et al., 2019; Ravn e Schoeberl, 2012], onde o ciclo principal é definido sem o conceito de ciclos menores. Nesses casos, as decisões de escalonamento podem ser tomadas em qualquer ponto dentro do ciclo principal. Para uma revisão recente sobre este tipo de escalonamento, indica-se o trabalho de Minaeva e Hanzálek [2021].

Em contraste, o presente trabalho adota a definição clássica de um EC, em que as decisões de escalonamento são tomadas apenas no início de cada *frame* [Erciyes e Erciyes, 2019]. Essa estrutura clássica é mais fácil de implementar e permite monitoramento sistemático e ações de correção no início de cada *frame*, aumentando a confiabilidade do sistema [Erciyes e Erciyes, 2019; Liu, 2000]. Por exemplo, no início de cada *frame*, o escalonador pode verificar se cada *job* programado para aquele *frame* foi liberado e está pronto para execução, ou checar se o tempo de execução dos *jobs* ultrapassou o tempo total do *frame*, permitindo um tratamento rápido de erros se problemas forem detectados.

Considerando a estrutura clássica dos ECs, alguns métodos para geração automatizada de escalonamentos foram propostos para arquiteturas *unicore*, como em [Burns et al., 1995; Aerabi et al., 2022; Yépez et al., 2006; Bril, 2022]. Para arquiteturas *multicore*, os métodos para ECs têm se concentrado principalmente em uma forma restrita de escalonamento, voltada para sistemas de criticidade mista [Trüb et al., 2017; Zhang, 2020; Burns et al., 2015]. Nessa forma restrita, apenas tarefas do mesmo nível de criticidade podem utilizar os recursos da plataforma simultaneamente, eliminando interferências entre aplicações com diferentes níveis de criticidade.

No melhor do nosso conhecimento, o trabalho de Deutschbein et al. [2019] é o único que aborda a geração de ECs para múltiplos *cores* em um contexto geral. Eles propõem modelos ILP e um algoritmo aproximativo para gerar escalonamentos globais para ECs preemptivos e não preemptivos. No entanto, uma limitação chave do trabalho é a falta de suporte para relações entre tarefas, como exclusão mútua e restrições de precedência. Em contraste, nosso trabalho estende essa abordagem em dois aspectos importantes: (i) incorporamos relações entre tarefas, e (ii) consideramos escalonamentos particionados, que ajudam a reduzir a contenção de recursos e melhoram a confiabilidade geral do sistema.

4. Modelos Propostos

Nesta seção apresentamos os modelos propostos para a geração de ECs, considerando as relações entre tarefas. O Quadro 1 fornece acesso rápido aos parâmetros e variáveis utilizados nestes modelos. Ela serve como referência, enquanto suas definições e funções são detalhadas a seguir.

Sejam $\mu_1, \dots, \mu_n$ os *jobs* gerados pelas tarefas de um sistema τ no intervalo $[0, P)$, onde

$$n = \sum_{i=1}^N \frac{P}{T_i}.$$

Cada *job* μ_i é definido pela tupla $(a_i, c_i, d_i, e_i, p_i)$, em que a_i é o instante de chegada, c_i o WCET, d_i o *deadline* absoluto, e_i o conjunto de relações de exclusão e p_i o conjunto de relações de precedência.

Dado o tempo de chegada e o *deadline* de μ_i , definimos sua *janela de escalonamento*

$$\gamma_i = \{k \in \{1, \dots, q\} : a_i \leq (k-1)F < d_i\},$$

onde $q = P/F$ é o número total de *frames* no ciclo maior. Os modelos visam otimizar a alocação dos n *jobs* nos q *frames* de cada um dos m *cores*, respeitando tanto suas janelas γ_i quanto as restrições de precedência e exclusão.

Parâmetros	Descrição
N, m	Número de tarefas e de <i>cores</i>
$\tau_i = (C_i, T_i)$	WCET C_i e período/ <i>deadline</i> relativo T_i da tarefa τ_i
$excl_i, prec_i$	Conjuntos de tarefas que possuem relações, respectivamente, de exclusão mútua e precedência com τ_i
P, F, q	Ciclo maior $P = \text{lcm}(T_1, \dots, T_N)$, tamanho do <i>frame</i> $F = \text{gcd}(T_1, \dots, T_N)$, e número de <i>frames</i> $q = P/F$
Ω_i	Conjunto de <i>jobs</i> da tarefa τ_i
n	Total de <i>jobs</i> em um ciclo maior, $n = \sum_{i=1}^N (P/T_i)$
$\mu_i = (a_i, c_i, d_i, e_i, p_i)$	Para o i -ésimo <i>job</i> : tempo de chegada a_i , WCET c_i , prazo d_i , exclusões e_i e precedências p_i
γ_i	Janela de escalonamento de μ_i , i.e., $\gamma_i = \{k \in \{1, \dots, q\} : a_i \leq (k-1) \cdot F < d_i\}$
Variáveis	Descrição
x_{ijk}	Fração do <i>job</i> μ_i no <i>core</i> j , <i>frame</i> k
f	Carga máxima em qualquer <i>frame</i> . Se $f > F$, o escalonamento é inviável
z_{ik}	$z_{ik} = 1$ se μ_i está em algum <i>core</i> no <i>frame</i> k , e 0 caso contrário
o_{ilk}	$o_{ilk} = 1$ se μ_i e μ_l compartilham o mesmo <i>core</i> no <i>frame</i> k , e 0 caso contrário
end_{ik}	$end_{ik} = 1$ se μ_i termina no <i>frame</i> k , e 0 caso contrário
y_{ijk}	$y_{ijk} = 1$ se μ_i ocupa o <i>core</i> j no <i>frame</i> k , e 0 caso contrário
b_{il}	$b_{il} = 1$ se μ_i precede μ_l , e 0 caso contrário
s_{ij}	$s_{ij} = 1$ se tarefas de τ_i estão alocadas no <i>core</i> j , e 0 caso contrário

Quadro 1: Parâmetros e variáveis dos modelos propostos.

As principais variáveis de decisão do modelo são x_{ijk} , que indicam a fração do *job* μ_i executada no *core* j durante o *frame* k . Nos modelos não preemptivos, $x_{ijk} \in \{0, 1\}$, indicando se μ_i ocupa o *frame* k no *core* j . Nos modelos preemptivos, $x_{ijk} \in [0, 1]$, permitindo fracionar um *job* em vários *frames*.

4.1. Escalonamento Global

O primeiro modelo proposto, chamado de GlobalNonPreemp (Modelo não preemptivo com alocação global), é apresentado abaixo. Conforme indicado na Equação (1), o objetivo deste modelo é determinar os valores das variáveis x_{ijk} de modo que f , que representa a carga máxima de qualquer *frame*, seja minimizado.

$$\min f \tag{1}$$

$$\text{s.t. } \sum_{j=1}^m \sum_{k \in \gamma_i} x_{ijk} = 1, \quad \forall i \tag{2}$$

$$\sum_{i=1}^n c_i x_{ijk} \leq f, \quad \forall j, k \tag{3}$$

$$\sum_{j=1}^m x_{ijk} \leq z_{ik}, \quad \forall i, k \in \gamma_i \quad (4)$$

$$o_{ilk} \leq z_{ik}, \quad o_{ilk} \leq z_{lk}, \quad \forall i, l, k \in \gamma_i \cap \gamma_l \quad (5)$$

$$o_{ilk} \geq z_{ik} + z_{lk} - 1, \quad \forall i, l, k \in \gamma_i \cap \gamma_l \quad (6)$$

$$|x_{ijk} - x_{ljk}| \leq 1 - o_{ilk}, \quad \forall i, l, \mu_i \in e_l \cup p_l, j, k \in \gamma_i \cap \gamma_l \quad (7)$$

$$z_{ik} \leq \sum_{\substack{h \in \gamma_l \\ h \leq k}} z_{lh}, \quad \forall i, \mu_l \in p_i, k \in \gamma_i \quad (8)$$

$$x_{ijk} \in \{0, 1\}, \quad \forall i \in \{1, \dots, n\}, j \in \{1, \dots, m\}, k \in \gamma_i \quad (9)$$

$$z_{ik} \in \{0, 1\}, \quad \forall i \in \{1, \dots, n\}, k \in \gamma_i \quad (10)$$

$$o_{ilk} \in \{0, 1\}, \quad \forall i, l \in \{1, \dots, n\}, k \in \gamma_i \cap \gamma_l \quad (11)$$

$$f \in \mathbb{R}_{\geq 0} \quad (12)$$

As Restrições (2) asseguram que todos os *jobs* sejam completamente executados em algum *frame* dentro de sua respectiva janela de escalonamento. A variável f , definida nas Restrições (3), representa a carga máxima de um *frame*. As Restrições (4)–(6) introduzem as variáveis binárias z_{ik} e o_{ilk} , utilizadas para modelar precedência e exclusão entre *jobs*. Especificamente, z_{ik} indica se o *job* μ_i está alocado em algum *core* durante o k -ésimo *frame*, enquanto o_{ilk} indica que os *jobs* μ_i e μ_l estão no mesmo *core* no *frame* k .

As Restrições (7) garantem que dois *jobs* com relações de precedência ou exclusão compartilhem um *frame* apenas se forem alocados no mesmo *core*. Vale destacar que, quando os *jobs* executam no mesmo *core*, as restrições de exclusão são automaticamente satisfeitas, devido à ausência de preempção. Além disso, as relações de precedência podem ser aplicadas ordenando a execução dos *jobs* dentro do *frame* por meio de um algoritmo de ordenação topológica.

As Restrições (8) formalizam as relações de precedência. Finalmente, as Restrições (9)–(12) definem os domínios das variáveis de decisão. Para fins de linearização, as Restrições (7) podem ser substituídas pelas seguintes restrições:

$$x_{ijk} - x_{ljk} \leq 1 - o_{ilk}, \quad x_{ljk} - x_{ijk} \leq 1 - o_{ilk}, \quad \forall i, l \in \{1, \dots, n\}, \mu_i \in e_l \cup p_l$$

O segundo modelo proposto, chamado de GlobalPreemp (Modelo preemptivo com alocação global), os valores de x_{ijk} são contínuos no intervalo $[0, 1]$, em vez de binários. Esse modelo é definido pela remoção das Restrições (7)–(9) do modelo GlobalNonPreemp e pela introdução das novas Restrições (13)–(19). Como resultado, o modelo GlobalPreemp é composto pelas Equações (1)–(6), (13)–(19).

$$\sum_{\substack{h \in \gamma_i \\ h \leq k}} \sum_{j=1}^m x_{ijk} \geq end_{ik}, \quad \forall i, k \in \gamma_i \quad (13)$$

$$z_{ik} \leq end_{lk}, \quad \forall i, \mu_l \in p_i, k \in \gamma_i \cap \gamma_l \quad (14)$$

$$z_{ik} \leq end_{lk} + b_{il}, \quad z_{lk} \leq end_{ik} + (1 - b_{il}), \quad \forall i, l, \mu_i \in e_l, k \in \gamma_i \cap \gamma_l \quad (15)$$

$$x_{ijk} \leq y_{ijk}, \quad \forall i, k \in \gamma_i \quad (16)$$

$$\sum_{j=1}^m y_{ijk} \leq 1, \quad \forall i, k \in \gamma_i \quad (17)$$

$$|y_{ijk} - y_{ljk}| \leq 1 - o_{ilk}, \quad \forall i, l, \mu_i \in e_l \cup p_l, j, k \in \gamma_i \cap \gamma_l \quad (18)$$

$$x_{ijk} \in [0, 1], y_{ijk}, end_{ik}, b_{il} \in \{0, 1\}. \quad (19)$$

As Restrições (13) definem as variáveis binárias end_{ik} , que indicam se o job μ_i já terminou no k -ésimo frame. As Restrições (14) impõem as relações de precedência. As Restrições (15) tratam as relações de exclusão, garantindo que, se dois *jobs* são mutuamente exclusivos, um só possa começar após o término do outro. Nessas restrições, $b_{il} = 1$ indica que o job μ_i precede μ_l , enquanto $b_{il} = 0$ indica o contrário.

As Restrições (16) definem as variáveis y_{ijk} , que indicam se o job μ_i utiliza o j -ésimo *core* no k -ésimo *frame*. As Restrições (17) evitam a migração de jobs entre *cores* em um mesmo *frame* ativo. As Restrições (18) têm o mesmo propósito das Restrições (7): em conjunto com as Restrições (14)–(16), elas asseguram que dois *jobs* com relação de precedência ou exclusão só possam compartilhar o mesmo *frame* se estiverem atribuídos ao mesmo *core* e um deles concluir a execução enquanto o outro se inicia. Por fim, as Restrições (19) definem o domínio das variáveis de decisão.

4.2. Escalonamento Particionado

Para garantir que todos os *jobs* de τ_u fiquem no mesmo *core*, conforme exigido pelo escalonamento particionado, adicionam-se a cada modelo as restrições:

$$\sum_{j=1}^m s_{uj} = 1, \quad \forall u = 1, \dots, N \quad (20)$$

$$\sum_{k \in \gamma_i} x_{ijk} \leq s_{uj}, \quad \forall \mu_i \in \Omega_u, u, j \quad (21)$$

$$s_{uj} \in \{0, 1\}. \quad (22)$$

Os modelos resultantes são chamados, respectivamente, de `PartitionedNonPreemp` e `PartitionedPreemp`.

5. Resultados Experimentais

Nesta seção, apresentamos os experimentos realizados para avaliar os modelos propostos. Todos os testes foram executados em um sistema equipado com CPU Intel Core i9-10900 (2,8 GHz) e 32 GB de memória. Avaliamos os quatro modelos, `GlobalNonPreemp`, `GlobalPreemp`, `PartitionedNonPreemp` e `PartitionedPreemp`, utilizando o *solver* Gurobi (versão 11.0.2) com parâmetros padrão, incluindo avaliação paralela *multi-threaded*.

5.1. Instâncias

Testamos os métodos em um conjunto de 15 instâncias, provenientes de três *benchmarks* distintos:

- **Conjunto genérico:** Três instâncias baseadas nos exemplos apresentados na Subseção 5.2 do trabalho de [Ravn e Schoeberl, 2012]. Para cada par de parâmetros n e T , cada instância contém:
 - $2n$ tarefas curtas com período T e WCET igual a 1, representando tarefas que controlam atuadores e sensores;
 - n tarefas médias com período $2T$ e WCET igual a 2, modelando tarefas de controle;
 - $\lceil n/3 \rceil$ tarefas longas com período $10T$ e WCET igual a 15, representando tarefas que fazem processamento inteligentes, registro remoto, etc.

As instâncias são rotuladas como *generic_X*, onde *X* é o valor de *n*.

- **Conjunto vehicle:** Três instâncias oriundas do cenário de controle em tempo real de um veículo autônomo descrito por Ravn e Schoeberl [2010].
- **Conjunto UUniFast:** Nove instâncias geradas conforme o procedimento da Subseção 6.2.1 do trabalho de Deutschbein et al. [2019], utilizando o algoritmo UUniFast. Cada instância segue o padrão $tXuYpZfH$, em que *X* é o número de tarefas, *Y* a utilização do sistema, *Z* o número de *cores* e *H* o número de *frames*.

Como as instâncias originais não incluíam relações entre tarefas, estas foram inseridas aleatoriamente. A Tabela 1 resume cada instância, listando ciclo maior (MMC), ciclo menor (MDC), utilização, número de tarefas, de *jobs*, de relações entre *jobs*, de *cores* e de *frames*.

Instância	Tarefas	Jobs	Relações	Cores	Frames	MDC	MMC	Utilização (%)
generic_30	100	760	150	6	10	20	200	5,25
generic_50	167	1267	275	10	10	20	200	8,775
generic_100	334	2534	375	20	10	20	200	17,55
vehicle_1	16	133	0	1	40	10	400	1,405
vehicle_2	16	133	4970	2	40	10	400	1,405
vehicle_3	16	133	4970	2	40	10	400	1,405
t50u2p5f60_1	50	872	8477	5	60	1	60	2,00
t50u2p5f60_2	50	749	4170	5	60	1	60	2,00
t50u2p5f60_3	50	822	5830	5	60	1	60	2,00
t50u2p5f660_1	50	8252	893632	5	660	1	660	2,00
t50u2p5f660_2	50	7503	409302	5	660	1	660	2,00
t50u2p5f660_3	50	8416	1003429	5	660	1	660	2,00
t50u2p5f3960_1	50	52267	32686560	5	3960	1	3960	2,00
t50u2p5f3960_2	50	50948	34626240	5	3960	1	3960	2,00
t50u2p5f3960_3	50	47881	20125620	5	3960	1	3960	2,00

Tabela 1: Resumo das características das instâncias.

5.2. Resultados dos modelos

Para avaliar os modelos, foi definido um tempo limite de 3000 segundos para o Gurobi. As Tabelas 2 e 3 mostram os resultados para os modelos sem preempção e com preempção, respectivamente. As colunas *best*, *GAP* e *t(s)* indicam: melhor resultado, *GAP* relativo à solução ótima (conforme retornado pelo Gurobi) e tempo até a melhor solução. Um traço “—” indica que não foi obtida solução viável. Quando o valor em *t(s)* é menor que o tempo limite (3000 segundos), isto indica que ou o Gurobi provou a solução ótima (*GAP* = 0.00%) ou ele apresentou falha antes da conclusão do tempo limite.

A comparação entre os modelos não preemptivos (Tabela 2) e preemptivos (Tabela 3) confirma que, embora sistemas preemptivos sejam mais desafiadores de implementar na prática, seus modelos tendem a apresentar resultados superiores quando soluções ótimas são encontradas. No entanto, essa vantagem vem acompanhada de um custo computacional mais elevado. Especificamente, o modelo global não preemptivo (GlobalNonPreemp) obteve soluções ótimas em todas as instâncias, exceto em *t50u2p5f3960_1* e *t50u2p5f3960_2*. Em contraste, o modelo global preemptivo (GlobalPreemp) não atingiu a otimalidade em oito instâncias e não encontrou soluções viáveis nas três de maior porte. Nos casos em que os modelos preemptivos não conseguiram determinar nenhuma solução viável, isso ocorreu devido à interrupção do solver por falta de memória.

É importante ressaltar que, embora a otimalidade não tenha sido alcançada em alguns casos, foi possível identificar soluções em que o tamanho máximo do *frame* ficou abaixo do MDC,

Instância	GlobalNonPreemp			PartitionedNonPreemp		
	best	GAP	t(s)	best	GAP	t(s)
generic_30	18,00	0,00%	0,48	18,00	0,00%	0,28
generic_50	18,00	0,00%	1,16	18,00	0,00%	24,23
generic_100	18,00	0,00%	5,42	18,00	0,00%	27,73
vehicle_1	15,00	0,00%	0,04	15,00	0,00%	0,04
vehicle_2	10,00	0,00%	2,64	10,00	0,00%	31,72
vehicle_3	10,00	0,00%	2,65	10,00	0,00%	31,75
t50u2p5f60_1	0,71	0,00%	1,47	0,71	0,00%	25,28
t50u2p5f60_2	0,84	0,00%	1,15	0,84	0,00%	7,10
t50u2p5f60_3	0,91	0,00%	1,14	0,91	0,00%	7,49
t50u2p5f660_1	0,73	0,00%	849,72	2,96	75,43%	3000,00
t50u2p5f660_2	0,88	0,00%	85,91	0,88	0,00%	2989,83
t50u2p5f660_3	0,93	0,00%	354,12	7,43	89,60%	1026,00
t50u2p5f3960_1	1,15	40,62%	3000,00	9,20	92,75%	3000,00
t50u2p5f3960_2	1,15	32,20%	1132,00	2,43	72,61%	3000,00
t50u2p5f3960_3	0,90	0,00%	2478,72	6,41	89,60%	3000,00

Tabela 2: Resultados dos modelos sem preempção.

Instância	GlobalPreemp			PartitionedPreemp		
	best	GAP	t(s)	best	GAP	t(s)
generic_30	17,50	0,00%	0,17	17,50	0,00%	1,64
generic_50	17,55	0,00%	1,69	18,00	2,50%	3000,00
generic_100	17,55	0,00%	3,71	18,00	2,50%	3000,00
vehicle_1	14,05	0,00%	<0,01	14,05	0,00%	<0,01
vehicle_2	9,50	26,05%	3000,00	9,50	0,00%	1076,84
vehicle_3	9,50	26,05%	3000,00	9,50	0,00%	1083,64
t50u2p5f60_1	0,40	0,00%	17,71	0,67	0,03%	3000,00
t50u2p5f60_2	0,40	0,00%	15,06	0,67	0,00%	432,91
t50u2p5f60_3	0,40	0,00%	5,05	0,67	0,01%	1377,45
t50u2p5f660_1	0,77	48,31%	3000,00	2,72	75,53%	3000,00
t50u2p5f660_2	0,42	6,31%	3000,00	2,83	76,41%	3000,00
t50u2p5f660_3	0,79	49,95%	3000,00	7,08	90,59%	3000,00
t50u2p5f3960_1	—	—	—	—	—	—
t50u2p5f3960_2	—	—	—	3,35	80,07%	3000,00
t50u2p5f3960_3	—	—	—	2,79	76,15%	3000,00

Tabela 3: Resultados dos modelos com preempção.

garantindo assim a escalonabilidade do sistema. Esse comportamento foi observado, por exemplo, no modelo *GlobalPreemp* para as instâncias *t50u2p5f660_X*.

A comparação entre os modelos particionados e os globais mostra que, embora sistemas particionados ofereçam melhor previsibilidade temporal, característica crítica em sistemas de tempo real, os modelos para estes sistemas são em geral mais difíceis de resolver de forma ótima. Os modelos globais determinaram a solução ótima em 20 instâncias, enquanto os particionados o fizeram em 15.

Os resultados indicam que os modelos propostos, principalmente os modelos não preemptivos, são eficientes em cenários de pequena e média escala (até 8500 jobs e milhares de relações). Para a grande maioria dos sistemas embarcados que adotam executivos cíclicos, tais cenários são suficientes, já que esses sistemas costumam ser implementados em arquitetura *bare-metal*, ou seja, sem a presença de um sistema operacional.

6. Conclusão

Este artigo apresentou novos modelos para a geração automática de executivos cíclicos *multicore*, considerando relações entre tarefas, assim como esquemas de escalonamentos globais e

particionados. Propusemos quatro modelos ILP, cada um adequado a uma configuração específica do sistema: (i) global não preemptivo, (ii) global preemptivo, (iii) particionado não preemptivo e (iv) particionado preemptivo.

Os resultados experimentais, abrangendo milhares de tarefas e dependências complexas, demonstraram que alguns modelos resolvem de forma ótima instâncias de até 8500 *jobs*. Estes resultados indicam que os modelos propostos são eficientes em cenários de pequena e média escala. Como trabalhos futuros, planejamos estender nossa metodologia para além do escalonamento por executivos cíclicos, incorporando tratamento de interrupções para lidar de modo eficiente com tarefas esporádicas. Além disso, planejamos desenvolver uma heurística para lidar melhor em instâncias de grande porte.

Referências

- Aerabi, E., Fazeli, M., e Hély, D. (2022). Cyense: Cyclic energy-aware scheduling for energy-harvested embedded systems. *Microprocessors and Microsystems*, 89:104421.
- Baker, T. P. e Shaw, A. (1989). The cyclic executive model and ada. *Real-Time Systems*, 1(1):7–25.
- Bril, R. J. (2022). Optimal order assignment algorithms for single-rate time-driven afap cyclic executives. In *2022 IEEE 25th International Symposium On Real-Time Distributed Computing (ISORC)*, p. 1–8. IEEE.
- Burns, A. e Baruah, S. (2017). Migrating mixed criticality tasks within a cyclic executive framework. In *Reliable Software Technologies–Ada-Europe 2017: 22nd Ada-Europe International Conference on Reliable Software Technologies, Vienna, Austria, June 12-16, 2017, Proceedings 22*, p. 203–216. Springer.
- Burns, A., Fleming, T., e Baruah, S. (2015). Cyclic executives, multi-core platforms and mixed criticality applications. In *2015 27th Euromicro Conference on Real-Time Systems*, p. 3–12. IEEE.
- Burns, A., Hayes, N., e Richardson, M. (1995). Generating feasible cyclic schedules. *Control Engineering Practice*, 3(2):151–162.
- de Dinechin, B. D. e Kordon, A. M. (2014). Converging to periodic schedules for cyclic scheduling problems with resources and deadlines. *Computers & operations research*, 51:227–236.
- Deutschbein, C., Fleming, T., Burns, A., e Baruah, S. (2019). Multi-core cyclic executives for safety-critical systems. *Science of Computer Programming*, 172:102–116.
- Erciyes, K. e Erciyes, K. (2019). *Distributed real-time systems*. Springer.
- Glass, C. A. (1994). Feasibility of scheduling lot sizes of two frequencies on one machine. *European Journal of Operational Research*, 75(2):354–364.
- Kyriakakis, E., Sparsø, J., e Schoeberl, M. (2021). Evaluating a time-triggered runtime system by distributing a flight controller. In *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, p. 1–8. IEEE.
- Liu, J. W. S. (2000). *Real-Time Systems*. Prentice–Hall, Inc., Upper Saddle River, New Jersey. ISBN 07458.

- Minaeva, A. e Hanzálek, Z. (2021). Survey on periodic scheduling for time-triggered hard real-time systems. *ACM Computing Surveys (CSUR)*, 54(1):1–32.
- Nogueira, B., Andrade, E., e Tavares, E. (2019). Power-aware scheduling of real-time applications onto mp soc platforms with multi-bank shared memory. *Microprocessors and Microsystems*, 67: 93–102.
- Parkinson, P. (2017). Update on using multicore processors with a commercial arinc 653 implementation. *Aviation Electronics Europe*.
- Perez, J., Gonzalez, D., Nicolas, C. F., Trapman, T., e Garate, J. M. (2014). A safety certification strategy for iec-61508 compliant industrial mixed-criticality systems based on multicore partitioning. In *2014 17th Euromicro Conference on Digital System Design*, p. 394–400. IEEE.
- Ravn, A. P. e Schoeberl, M. (2010). Cyclic executive for safety-critical java on chip-multiprocessors. In *Proceedings of the 8th International Workshop on Java Technologies for Real-Time and Embedded Systems*, p. 63–69.
- Ravn, A. P. e Schoeberl, M. (2012). Safety-critical java with cyclic executives on chip-multiprocessors. *Concurrency and Computation: Practice and Experience*, 24(8):772–788.
- Rubio-Anguiano, L. E., Briz, J. L., e Ramírez-Treviño, A. (2022). Accounting for preemption and migration costs in the calculation of hard real-time cyclic executives for mp socs. *IEEE Robotics and Automation Letters*, 7(3):7990–7997.
- Rubio-Anguiano, L. E., Trabanco, A. C., Velasco, J. L. B., e Ramírez-Treviño, A. (2021). Maximizing utilization and minimizing migration in thermal-aware energy-efficient real-time multiprocessor scheduling. *IEEE Access*, 9:83309–83328.
- Schild, K. e Würtz, J. (1998). Off-line scheduling of a real-time system. In *Proceedings of the 1998 ACM symposium on Applied Computing*, p. 29–38.
- Tamas-Selicean, D. e Pop, P. (2011). Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures. In *2011 IEEE 32nd Real-Time Systems Symposium*, p. 24–33. IEEE.
- Tavares, E., Maciel, P., Dallegrave, P., Silva, B., Falcão, T., Nogueira, B., Callou, G., e Cunha, P. (2010). Model-driven software synthesis for hard real-time applications with energy constraints. *Design Automation for Embedded Systems*, 14:327–366.
- Trüb, R., Giannopoulou, G., Tretter, A., e Thiele, L. (2017). Implementation of partitioned mixed-criticality scheduling on a multi-core platform. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):1–21.
- Wei, W. e Liu, C. (1983). On a periodic maintenance problem. *Operations Research Letters*, 2(2): 90–93.
- Yépez, J., Guardia, J., Velasco, M., Ayza, J., Castañé, R., Martí, P., e Fuertes, J. M. (2006). Ciclic: A tool to generate feasible cyclic schedules. In *2006 IEEE International Workshop on Factory Communication Systems*, p. 399–404. IEEE.
- Zhang, F. (2020). Improvement to semi-partitioned cyclic executives for mixed-criticality scheduling on multiprocessor platforms. *IEEE Access*, 8:223606–223617.