Revista Iberoamericana de Automática e Informática Industrial

Revista Iberoamericana de Automáica e Informáica Industrial 19 (2022) 369-379

www.revista-riai.org

Planificación de sistemas de tiempo real crítico mediante técnicas no convencionales

Patricia Balbastre*, José María Aceituno, Ana Guasque, Juan Francisco Blanes, Alfons Crespo, José Luis Poza

Instituto de Automática e Informática Industrial, Universitat Politècnica de València, Camino de Vera, s/n, 46022, Valencia, España.

To cite this article: Balbastre, P., Aceituno, J.M., Guasque, A., Blanes, J.F., Crespo, A. 2022. Scheduling of hard real-time systems using non-conventional techniques. Revista Iberoamericana de Automática e Informática Industrial 19, 369-379. https://doi.org/10.4995/riai.2022.17148

Resumen

En la planificación de sistemas de tiempo real crítico es clave encontrar un plan temporal en el que las tareas pueden ejecutarse antes de que venza el plazo establecido. Para lograr este objetivo se pueden utilizar diferentes tipos de algoritmos de planificación. Además de encontrar un plan factible, muchas veces es beneficioso, de todos los planes factibles existentes, encontrar aquel que minimiza algún parámetro temporal del sistema. Existen muchos algoritmos de planificación que son capaces de encontrar una planificación correcta del conjunto de tareas, pero no es tan común encontrar algoritmos que optimicen otros parámetros para mejorar el funcionamiento del sistemas en términos de estabilidad, consumo de potencia, etc. Los heurísticos existentes puede mejorar el comportamiento pero sin asegurar un resultado óptimo. En este trabajo exploramos las técnicas de planificación no convencionales basadas en programación lineal entera para resolver la planificación en sistemas monoprocesador con el objetivo de minimizar el peor tiempo de respuesta y el cambio de contexto de las tareas del sistema y la asignación de tareas a procesadores en sistemas multiprocesador con el objetivo de minimizar la interferencia producida por el acceso a recursos hardware comunes.

Palabras clave: Sistemas de control de tiempo real, Planificación y planificabilidad de sistemas de control, Sistemas ciber-físicos en control, Sistemas de control embebidos o empotrados.

Scheduling of hard real-time systems using non-conventional techniques

Abstract

In the scheduling of hard real time systems it is key to find a temporal plan in which tasks can be executed before the deadline expires. Different types of scheduling algorithms can be used to achieve this goal. In addition to finding a feasible plan, it is often beneficial, out of all existing feasible plans, to find one that minimises some time parameter of the system. There are many scheduling algorithms that are able to find a correct scheduling of the set of tasks, but it is not so common to find algorithms that optimise other parameters to improve the performance of the system in terms of stability, power consumption, etc. Existing heuristics can improve the performance but without ensuring an optimal result. In this work we explore unconventional scheduling techniques based on integer linear programming to solve scheduling in mono processor systems with the objective of minimising the worst-case response time of the system tasks and the assignment of tasks to processors in multiprocessor systems with the objective of minimising the interference produced by the access to common hardware resources.

Keywords: Real-time control systems, Control system scheduling, cyber physical systems, Embedded control systems.

1. Introducción

En la planificación de sistemas de tiempo real crítico todas las tareas deben cumplir estrictamente sus plazos ya que, de lo contrario, podría producirse un fallo catastrófico. Para planificar correctamente este tipo de sistemas se pueden utilizar diferentes tipos de algoritmos de planificación, estáticos o dinámicos. Si el sistema es altamente crítico, es necesario utilizar planificadores estáticos, ya que suele ser un requisito de certificación.

La planificación estática requiere un conocimiento previo de las características de las tareas. El planificador estático genera

* Autor para correspondencia: patricia@ai2.upv.es

un plan fuera de línea, es decir, una secuencia de ejecuciones de tareas. Este esquema se conoce como ejecutivo cíclico (Baker and Shaw, 1988; Locke, 1992). Este plan estático se guarda en una tabla e indica cuándo debe ejecutarse cada tarea. El análisis de planificabilidad utilizando esta estrategia debe realizarse durante la construcción del plan. Debido a este determinismo en la ejecución, la planificación estática se utiliza con mucha frecuencia en los sistemas de tiempo real de alta integridad. La planificación estática presenta ventajas como el bajo coste en tiempo de ejecución, y desventajas, como la falta de flexibilidad.

Pero, además de encontrar un plan factible, muchas veces es beneficioso, de todos los planes factibles existentes, encontrar aquel que minimiza algún parámetro temporal del sistema. La optimización de ciertos parámetros temporales, como el tiempo de respuesta, puede ser interesante, especialmente cuando las restricciones temporales imponen límites inferiores a los tiempos de respuesta a los eventos. El *jitter* es el otro parámetro crítico que mide la variación del tiempo de respuesta. Especialmente en los sistemas de control, cualquier retraso puede causar la inestabilidad del sistema, ya que estos retrasos no se tienen en cuenta en la ley de control (Crespo et al., 1999).

Los métodos exactos de optimización, como la programación lineal entera (ILP), se han utilizado principalmente para la planificación de tareas en sistemas multiprocesadores más que en sistemas monoprocesadores, ya que existen heurísticos que pueden planificar de forma óptima las tareas en un sistema monoprocesador en tiempo polinomial. Al formular el problema como un ILP, podemos determinar teóricamente la planificación óptima en cualquier sistema considerando diferentes objetivos y restricciones. La posibilidad de personalizar una función objetivo que satisfaga las necesidades de los diseñadores es una propuesta interesante que puede aplicarse a diferentes campos con metas distintas. Sin embargo, los tiempos de solución no siempre son manejables. Los recientes avances en las aplica-ciones *software* de optimización disminuyen este problema, ya que pueden resolver eficazmente instancias de tamaño práctico.

En este trabajo exploramos las técnicas de planificación no convencionales basadas en programación lineal entera. Nuestro objetivo es abordar un sistema más simple (monoprocesador) a uno más complejo (multiprocesador). En concreto, se pretende:

- La planificación en sistemas monoprocesador con el objetivo de minimizar tanto el peor tiempo de respuesta de las tareas del sistema, como el cambio de contexto de las mismas.
- La asignación de tareas a procesadores en sistemas multiprocesador con el objetivo de minimizar la interferencia producida por el acceso a recursos hardware comunes.

El presente artículo esta organizado de la siguiente forma: en el apartado 2 se comentan brevemente los artículos relacionados con el tema de nuestro trabajo, en el apartado 3 se aborda el uso de técnicas no convencionales de planificación para sistemas monoprocesador y para sistemas multiprocesador en el apartado 4. En ambos casos, se presenta el modelo temporal utilizado y el modelo de programación lineal aplicado. Las técnicas presentadas se evalúan en el apartado 5, mientras que las conclusiones se presentan en el apartado 6.

2. Trabajos relacionados

La planificación de sistemas de tiempo real en monoprocesador se ha resuelto tradicionalmente con técnicas heurísticas conocidas, ya que son planificadores óptimos. Pero, desde ha-ce algunos años, muchos autores están convirtiendo el proble-ma clásico de planificación en un problema ILP para optimizar modelos más complejos.

Estos trabajos que obtienen planificaciones factibles con técnicas ILP en modelos complejos de tiempo real incluyen, entre otros, sistemas multiprocesador (Baruah, 2004), optimización del consumo energético (Hong et al., 1999), conside-ración de arquitecturas con cachés locales de instrucciones o datos (Nguyen et al., 2019), sistemas de tiempo real débilmen-te duros (Sun and Natale, 2017), criticidad mixta (Fleming and Burns, 2015), etc. Esto se debe a la ausencia de algoritmos óptimos heurísticos para modelos más complejos que el típico modelo de tareas periódicas.

En (Mangeruca et al., 2007) se aborda la planificación con relaciones de precedencia entre tareas. En este trabajo se utilizan técnicas de ILP relajadas para obtener una asignación óptima de prioridad/finalidad para la planificación expulsiva con prioridades dinámicas y relaciones de precedencia entre tareas. El problema de cálculo del tiempo de respuesta y asignación de prioridades con un ILP se describe en (Lisper and Mellgren, 2001), donde el techo de la ecuación del tiempo de respuesta se reformula como un problema ILP. En (Di Natale and Zeng, 2013) se presenta un test de planificabilidad que permite una definición exacta y eficiente de la región planificable con menos variables binarias. Nuestro objetivo no es sólo encontrar una planificación factible, sino que buscamos encontrar la planificación óptima considerando diferentes objetivos, como el tiempo de respuesta de las tareas, el cambio de contexto, etc.

Con respecto a sistemas multiprocesador, se ha popularizado su uso en sistemas embebidos debido a su alto rendimiento. Este artículo se centra en sistemas de tiempo real crítico, en los que el incumplimiento de las restricciones temporales podría tener consecuencias catastróficas. Por lo tanto, en estos sistemas se considera la planificación particionada, ya que no se permite la migración de tareas entre núcleos. Así, el problema de la planificación en un sistema multiprocesador de tiempo real implica una primera fase de asignación de tareas a los procesadores y una segunda fase de programación independiente de cada procesador. El *survey* de Davis and Burns (2011) es de obligada lectura sobre planificación en multiprocesadores. Recientemente se ha estudiado la planificación con técnicas ILP en Baruah (2022) para modelos de tareas que siguen un grafo acíclico directo (DAG).

Con respecto al estudio de las interferencias producidas por el acceso a recursos *hardware* compartidos, el *survey* de Maiza et al. (2019) describe los trabajos más importantes hasta 2018. La mayoría de trabajos que tratan de reducir esta interferencia, lo hacen para algún tipo específico de recurso hardware: bus de memoria (Dasari et al. (2011), Lampka et al. (2014)), memoria DRAM (Kim et al. (2014)).En nuestro caso, proponemos un modelo de interferencia general que es válido para cualquier tipo de recurso.Un trabajo similar el nuestro es el presentado por Davis et al. (2021). Los autores proponen un modelo que caracteriza el estrés que cada tarea supone para cada recurso y su sensibilidad a dicho estrés. En su modelo se asumen prioridades fijas.

3. Sistemas de tiempo real crítico monoprocesador

Este apartado se va a centrar en el uso de técnicas de planificación no convencionales para sistemas monoprocesador. en primer lugar se va a definir el modelo de tareas utilizado para luego pasar a describir el modelo de programación lineal utilizado para establecer la planificación.

3.1. Modelo temporal

Nuestro modelo de tareas está compuesto por n tareas periódicas de tiempo real que suponemos que son independientes (no comparten recursos ni relación de precedencia).

$$\tau = [\tau_1, ..., \tau_n] \tag{1}$$

Una tarea τ_i es una tupla con los siguientes parámetros:

$$\tau_i = (C_i, D_i, T_i) \tag{2}$$

donde C_i es el tiempo de cómputo de cada activación en el peor de los casos (WCET), D_i es el plazo relativo y T_i es el periodo de la tarea.

El plazo absoluto de una activación a es $d_{ia} = a \cdot T_i + D_i$. Podemos suponer, sin pérdida de generalidad, que todos los parámetros anteriores son valores enteros. La utilización de la tarea U_i es la relación entre el tiempo de cómputo y el periodo, $U_i = \frac{C_i}{T_i}$. Dado que nos centramos en los sistemas de tiempo real crítico, asumimos que lo plazos no pueden ser mayores que los periodos, por tanto: $D_i \leq T_i$.

Sin pérdida de generalidad, asumimos que las tareas son ordenadas por los plazos de modo creciente. Si los plazos relativos de dos tareas coinciden, se asume un orden creciente de los períodos.

Definimos el hiperperiodo (H) como el momento en que todas las tareas vuelven a activarse al mismo tiempo. Se corresponde con el mínimo común múltiplo de los periodos de las tareas.

Otro parámetro importante es el tiempo que necesita una tarea τ_i para completar la ejecución de una activación determinada a. Este tiempo se representa como w_{ia}.

Además, es importante cuantificar el valor del tiempo de respuesta más grande de todas las activaciones a lo largo del hiperperiodo. A este tiempo de finalización máximo se le llama tiempo de respuesta en el peor de los casos (Worst Case Response Time, WCRT en inglés) definido y calculado en (Harter, 1987; Joseph and Pandya, 1986).

Puede ser calculado como:

$$WCRT_i = \text{máx}\{w_{ia}\} \tag{3}$$

Y si el $WCRT_i \leq D_i$, entonces el conjunto de tareas será planificable.

La Figura 1 muestra los parámetros que definen una tarea periódica de tiempo real.

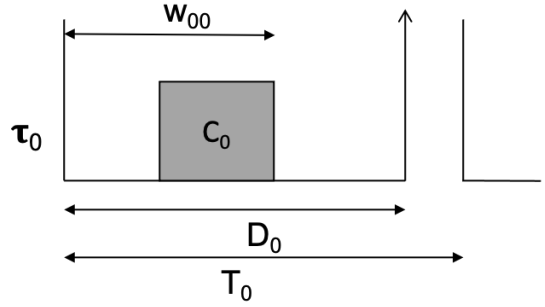

Figura 1: Definición de tarea de tiempo real

Suponemos que este trabajo se aplica a sistemas de tiempo real crítico, por tanto son planificados de forma estática, es decir, el plan se genera de forma offline. El problema a resolver es la obtención de una planificación de un conjunto de tareas, es decir, los instantes de tiempo en los cuales una tarea debe ser ejecutada respetando sus plazos y optimizando algún parámetro adicional.

3.2. Ejemplo

Consideremos el conjunto de tareas periódicas de la Tabla 1. La Figura 2 muestra la ejecución del cronograma cuando el conjunto de tareas es planificado bajo un algoritmo de prioridad fija. En este caso, las prioridades son inversamente proporcionales a los plazos (Deadline Monotonic algorithm, DM (Leung and Whitehead, 1982)). Cuando usamos el algoritmo DM, el WCRT de una tarea coincide con el tiempo de respuesta de la primera activación (w_{i0}). Como podemos observar en la Figura 2, el w_{i0} de las tareas y, por tanto el WCRT son $WCRT_0 = 1$, $WCRT_1 = 3$ y $WCRT_2 = 8$.

Tabla 1: Ejemplo de conjunto de tareas

	C	D	T
τ_0	1	4	4
τ_1	2	5	5
τ_2	2	8	8

Sin embargo, podemos observar en la Figura 3 como un ligero cambio en la planificación puede mejorar el WCRT de algunas tareas. Tal y como se puede observar, si la tarea 1, en su segunda activación, se retrasa un instante de tiempo, el WCRT de la tarea 2 pasará de 8 a 6. De este modo, la segunda planificación obtendría como valores de WCRT: $WCRT_0 = 1$, $WCRT_1 = 3$ y $WCRT_2 = 6$, los cuales no coinciden con los de la primera activación dado que se ha realizado una mejora en el WCRT de la tarea τ_2.

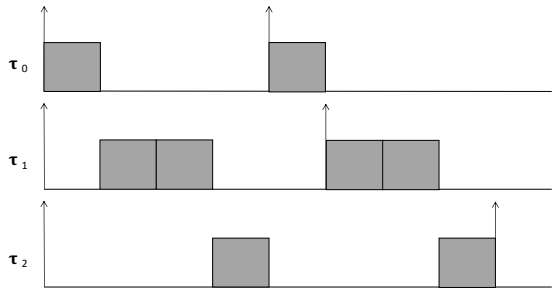

Figura 2: Cronograma del ejemplo bajo DM

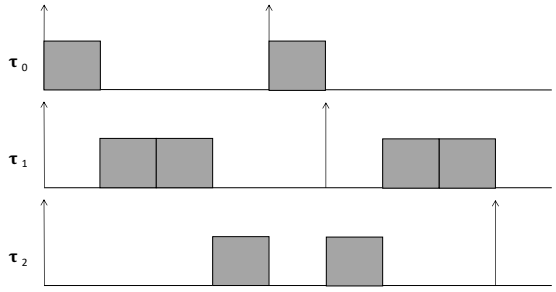

Figura 3: Cronograma del ejemplo modificado

En el siguiente apartado, abordaremos el problema de la planificación de sistemas en tiempo real como un problema de optimización con el objetivo de mejorar la planificación obtenida con los algoritmos heurísticos tradicionales.

3.3. Modelo MILP

En este apartado, se detallará el problema de optimización para obtener la planificación de un modelo de tareas como el presentado en el apartado 3.1. Un objetivo inicial podría ser minimizar el WCRT de todas las tareas. Los tiempos de respuesta bajos tienen varias ventajas, por ejemplo, en aplicaciones de control, donde la estabilidad del sistema está asegurada si la variación del tiempo de respuesta es baja. El problema de optimización presentado es una mejora del presentado en Guasque et al. (2020). En el trabajo mencionado se utiliza una matriz tridimensional (tarea, activación, tiempo) mientras que en este artículo se elimina la dimensión de la activación. En la Tabla 2 se definen las variables utilizadas en el modelo.

Queremos obtener la matriz de ejecución de tareas (x_{it}) que es una matriz binaria que representa si una tarea τ_i se ejecuta o no en el tiempo t. Con el contenido de esta matriz podemos conocer los tiempos de respuesta de las tareas, que son las variables a minimizar. La planificación tiene que cumplir unas reglas, que se expresan mediante las siguientes restricciones.

Tabla 2: Notación del modelo MILP

Conjuntos e índices	
i	Tareas $\tau_i \in \{0, 1, 2, ..., n\}$
a_i	Activaciones de $\tau_i \in \{0, 1, 2, ..., N_i\}$
Parámetros	
C_i	WCRT de τ_i
D_i	Plazo relativo de τ_i
T_i	Periodo de τ_i
H	Hiperperiodo del conjunto de tareas
N_i	Número de activaciones de τ_i (H/T_i)
d_{ia_i}	Plazo de τ_i en la activación a_i
R_{ia_i}	$[a_i \cdot T_i, (a_i + 1) \cdot T_i]$ Intervalo de ejecución de la tarea i en la activación a_i
Variables de decisión	
x_{it}	Matriz de ejecución de las tareas. 1 si τ_i se ejecuta en el instante t y 0 en caso contrario.
w_{ia_i}	Matriz de tiempos de respuesta. Tiempo de respuesta de τ_i en la activación a_i

$$\text{mín} \quad \text{Obj} = \sum_{\forall(i,a_i)} \frac{w_{ia_i}}{D_i} \tag{4}$$

s.t:

$$\sum_{t \in R_{ia_i}} x_{it} = C_i \quad \forall i, a_i \tag{5}$$

$$t \cdot x_{it} \leq d_{ia_i} - 1 \quad \forall t \in R_{ia_i} \tag{6}$$

$$\sum_{i} x_{it} \leq 1 \quad \forall t \in \{0, 1, ..., H\} \tag{7}$$

$$w_{ia_i} \geq t \cdot x_{it} - a_i \cdot T_i + 1 \quad \forall t, i, a_i \tag{8}$$

$$x_{it} \in \{0, 1\} \tag{9}$$

$$w_{ia_i} \geq 0 \tag{10}$$

Las restricciones 5, 6 y 7 están relacionadas con las condiciones de planificabilidad que debe cumplir el plan resultante. La restricción 5 asegura que la tarea ejecuta C_i unidades de tiempo dentro de cada activación del periodo T_i. La restricción 6 asegura que la tarea es planificable, es decir, que la tarea termina su ejecución en todas las activaciones antes del tiempo previsto D_i. La restricción 7 está relacionada con la planificación en sistemas monoprocesadores ya que, en cada instante de tiempo, no se puede ejecutar más de una tarea. La restricción 8 calcula el tiempo de respuesta de cada tarea en todas las activaciones. Las ecuaciones 9 y 10 definen los dominios de las variables de decisión.

Ya que el modelo contiene tanto variables enteras como binarias, es un modelo de optimización basado en programación lineal entera mixta (MILP).

Este modelo es demasiado costoso en términos computacionales para ser utilizado para grandes conjuntos de tareas con alta utilización y largos hiperperiodos. Por ese motivo, en Guasque et al. (2020) se propone una formulación MILP alternativa basada en horizonte rodante.

4. Sistemas de tiempo real crítico multiprocesador

En este caso, para obtener la planificación debemos seguir los siguientes pasos:

- Asignación tareas a procesadores.
- Planificación de las tareas asignadas a cada procesador.

4.1. Modelo temporal

En un sistema multiprocesador, las diferentes tareas pueden ser ejecutadas simultáneamente en un conjunto de m procesadores homogéneos M_1,..., M_m. N_k tareas se asignan en cada procesador. En este trabajo se asume que no se permite migración de tareas entre procesadores, es decir, se sigue un enfoque particionado. En un sistema de tiempo real crítico, hay un conjunto de n tareas de tiempo real independientes, $\tau = [\tau_1, ..., \tau_n]$, donde cada tarea genera un conjunto de activaciones ($\tau_{ij}, 1 \leq i \leq n, j \geq 0$) que deben completarse antes de un cierto plazo temporal. este trabajo considera que todas las tareas son periódicas y se caracterizan por $\tau_i = (C_i, D_i, T_i, I_i)$, donde C_i es el tiempo de ejecución en el peor caso (WCET), D_i es el plazo, T_i es el periodo e I_i es el tiempo de interferencia en el peor caso (explicado en detalle en 4.1.1). Al igual que sucede en los sistemas monoprocesador, asumimos un modelo de tareas de plazos restringidos ($D_i \leq T_i$) y tareas periódicas o esporádicas.

Cada tarea τ_i tiene A_i activaciones durante el hiperperiodo H. Por tanto, $A_i = \frac{H}{T_i}$.

La utilización de un procesador M_k es la suma de las utilizaciones de todas las tareas que pertenecen a ese procesador: $U_{M_k} = \sum_{\tau_i \in M_k} U_i$. La utilización total del sistema es la suma de las utilizaciones de todos los procesadores: $U_\tau = \sum_m U_{M_k}$.

4.1.1. Tiempo de interferencia en el peor caso

En los sistemas multiprocesador, a diferencia de los monoprocesador, las diferentes tareas comparten los recursos del sistema al mismo tiempo y los diferentes procesadores pueden necesitar disponer de algún recurso de manera simultanea, como los buses o la memoria, debido a la naturaleza del proceso que están ejecutando. Llegados a este punto, podemos afirmar que durante la ejecución de un sistema multiprocesador, existirá probablemente contención o interferencia entre diferentes procesadores, que podrán retrasar la ejecución esperada de los procesos. Tal y como se afirma en (Fernandez et al., 2014), a partir de la contención en el acceso al recurso compartido es cuando surge el efecto de la interferencia entre varias tareas que están asignadas en diferentes procesadores.

I_i es el tiempo de peor caso que τ_i utiliza para realizar operaciones de lectura y escritura en memoria. Tal y como se observa en la Figura 4, podemos observar que todo el tiempo que τ_0 emplea en las operaciones de lectura y escritura es parte del parámetro de interferencia I_0. Los tiempos de ejecución de las tareas se representan en rectángulos de color gris, mientras que la interferencia se representa en rectángulos con líneas diagonales. Desde el punto de vista de las otras tareas, I_i es el tiempo extra que τ_i provoca en otras tareas que están ejecutándose al mismo tiempo en otros procesadores, debido a la contención.

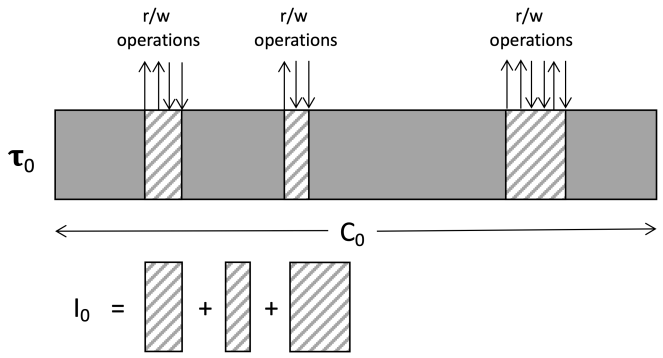

Figura 4: Ejemplo de interferencia entre tareas.

Por ejemplo, supongamos que $\tau_i = \{10, 50, 75, 3\}$, esto significa que el tiempo de cómputo de τ_i es de 10 unidades, de las cuales 3 están dedicadas a los accesos a memoria. En el peor caso, podemos asumir que todas las tareas que se ejecutan en otros procesadores al mismo tiempo que τ_i sufrirán un retraso de 3 unidades en sus ejecuciones.

La Figura 4 muestra el parámetro de interferencia desde el punto de vista de la tarea. Esta interferencia implica un retraso en la ejecución de tareas en otros procesadores. Nótese que I_0 se representa como un solo bloque cuando afecta a otras tareas, mientras que en la Figura 4 se presenta como bloques separados. De ahora en adelante, la interferencia siempre se representará como un único bloque, ya que vamos a investigar el efecto que tiene en otras tareas en otros procesadores. El siguiente ejemplo ilustra este efecto.

Ejemplo. Considérese un conjunto de tres tareas τ_0, τ_1 y τ_2 asignadas en una plataforma con tres procesadores tal y como se muestra en la Figura 5. Supongamos que $I_0 = I_2 = 1$ y $I_1 = 0$, es decir, τ_0 y τ_2 son las tareas que comparten los recursos, de forma que provocan y reciben interferencia. Sin embargo, τ_1 no utiliza los recursos y por tanto no sufre interferencia. cada vez que τ_0 se ejecuta al mismo tiempo que τ_2, ambas tareas aumentan su tiempo de cómputo: τ_0 se ve incrementada en I_2 unidades debido a la interferencia causada por τ_2 mientras que τ_2 aumenta I_0 unidades debido a la interferencia causada por τ_0.

De la Figura 5 se deduce que, si se considera la contención, la utilización total de una tarea depende de su propio tiempo de cómputo y periodo así como de la interferencia recibida de otras tareas. Además, esta interferencia no debe ser considerada en todas las activaciones, si no en solo aquellas en las que hay ejecución en ambos procesadores simultáneamente.

Por tanto, definimos los valores reales equivalentes para U_i, U_{M_k} y U_τ:

$$U'_i = U_i + U_i^{int} \tag{11}$$

donde U_i^{int} es la utilización debida a la interferencia causada por otras tareas a τ_i.

Expresando U'_i con respecto al hiperperiodo se obtiene::

$$U'_i = \frac{A_i C_i}{H} + \frac{I_i^T}{H} \tag{12}$$

donde I_i^T es la interferencia total que τ_i recibe debido a la ejecución de la contención en otros procesadores.

Del mismo modo, la utilización real de un procesador M_k es:

$$U'_{M_k} = \sum_{\tau_i \in M_k} U'_i \tag{13}$$

Y la utilización real del conjunto de tareas τ es:

$$U'_\tau = \sum_m U'_{M_k} \tag{14}$$

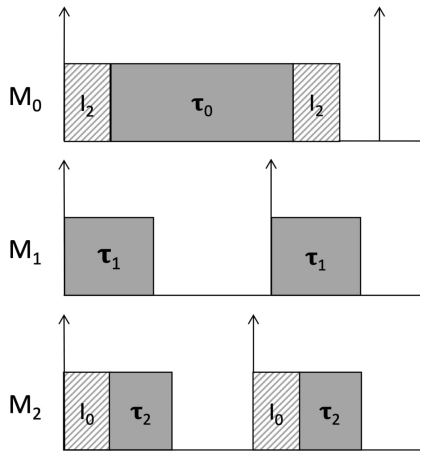

Figura 5: Ejemplo de la influencia de la interferencia en la planificación.

4.2. Planificación teniendo en cuenta la interferencia

Este apartado describe las reglas que un algoritmo de planificación debe seguir para considerar la interferencia exacta que una tarea sufre en cada activación. Nótese que este planificador no intenta reducir la interferencia. Esto se logrará en la fase de asignación de tareas a procesadores descrita en el apartado 4.3.

Inicialmente se necesitan las siguientes definiciones:

Definición 1. *Se define una tarea receptora como aquella que accede a un recurso compartido hardware y sufre un incremento de su tiempo de cómputo debido a la interferencia producida por otras tareas asignadas en otros procesadores.*

Definición 2. *Se define una tarea emisora como aquella que accede a un recurso compartido hardware y provoca un incremento en el tiempo de cómputo de otras tareas asignadas en otros procesadores debido a la contención.*

Si $I_i = 0$, τ_i no es una tarea emisora ni receptora. Si $I_i > 0$, τ_i será una tarea emisora y receptora si existe al menos una tarea τ_j en otro procesador cuyo $I_j > 0$.

La interferencia se produce siempre que dos tareas emisoras se ejecutan al mismo tiempo en diferentes procesadores. El instante en el que se puede producir una interferencia es cuando se da una de las dos situaciones siguientes:

- Se activa una tarea receptora τ_i. En este momento, las tareas emisoras activas en otros procesadores causan interferencia a la tarea τ_i.
- Se activa una tarea emisora τ_j. En este momento, τ_j provoca interferencia a las tareas receptoras asignadas en otros procesadores.

Además, como una tarea puede recibir interferencias de más de una tarea (si hay más de dos procesadores), y en diferentes instantes de tiempo, será necesario registrar las interferencias producidas por cada tarea en una matriz.

Definición 3. *Sea W una matriz binaria de $n \times n \times H$. En cada instante t, el valor de W_{ijt} indica si τ_i provoca interferencia a τ_j o no, de la siguiente manera:*

- $W_{ijt} = 1$*: Se produce interferencia.*
- $W_{ijt} = 0$*: No se produce interferencia.*

Esta matriz se utilizará para establecer cuándo una tarea τ_i debe aumentar su tiempo de cálculo debido a la interferencia causada por otras tareas que se ejecutan en diferentes procesadores.

Propiedad 1. *Si dos tareas τ_i y τ_j se asignan al mismo procesador M_k entonces W_{ijt}=0 y W_{jit}=0 para todo t=0,..,H.*

Propiedad 2. *Si una tarea τ_i tiene $I_i = 0$ entonces W_{ijt}=0 y W_{jit}=0 para todo t=0,..,H y para todo j= 1, .., n.*

Vamos a ilustrar el comportamiento de W con un ejemplo. Consideremos el siguiente conjunto de tareas, $\tau = [\tau_0, \tau_1]$ con $\tau_0 = (1, 3, 3, 1)$ y $\tau_1 = (2, 5, 5, 1)$, asignadas en un sistema de dos procesadores, τ_0 se asigna en M_0 y τ_1, en M_1.

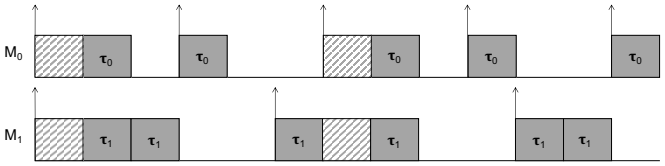

Figura 6: Cronograma de ejecución del ejemplo.

La Figura 6 muestra la planificación resultante del sistema aplicando el algoritmo RM (Rate Monotonic, (Liu and Layland, 1973)), teniendo en cuenta la interferencia producida por la contención en la memoria. RM es un algoritmo de planificación dinámido de prioridades fijas, donde la prioridad de una tarea es inversamente proporcional a su periodo. Como se puede ver en la Figura 6, hay un incremento en los tiempos de ejecución de las tareas τ_0 y τ_1 de I_1 y I_0 respectivamente en los instantes donde la otra tarea se activa. Si τ_0 o τ_1 se activa cuando la otra tarea no está ejecutándose, entonces la interferencia no se añade.

Los valores de W en cada instante de tiempo t se muestran en la Figura 7. Tal y como se indica en los comentarios, siempre que hay una activación de una tarea mientras la otra está activada, el correspondiente valor de la matriz cambia de 0 a 1. Cuando una tarea τ_i finaliza su activación, todos los elementos de W de la fila i pasan a ser 0. En cada instante de tiempo t, si W_{ijt} es 1, significa que τ_i está activa en el instante t y ha causado una interferencia en τ_j.

t	W	Comentario
0	$\begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}$	Activación de τ_0 y τ_1
1	$\begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}$	
2	$\begin{pmatrix}0 & 0\\1 & 0\end{pmatrix}$	τ_0 termina su primera activación
3	$\begin{pmatrix}0 & 0\\0 & 0\end{pmatrix}$	τ_1 termina su primera activación
4	$\begin{pmatrix}0 & 0\\0 & 0\end{pmatrix}$	
5	$\begin{pmatrix}0 & 0\\0 & 0\end{pmatrix}$	τ_1 se activa, pero τ_0 no está activa
6	$\begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}$	τ_0 se activa mientras τ_1 está activa
7	$\begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}$	
8	$\begin{pmatrix}0 & 0\\0 & 0\end{pmatrix}$	τ_0 y τ_1 terminan su activación
9->15	$\begin{pmatrix}0 & 0\\0 & 0\end{pmatrix}$	τ_0 y τ_1 no están activas al mismo tiempo

Figura 7: valores de W para el ejemplo

Observando la Figura 6, podemos ver que la interferencia total sufrida por τ_0 es 2, mientras que la interferencia sufrida por τ_1 durante H es 2 también. Por lo tanto, la utilización real de cada procesador es:

$$U'_{M_0} = \frac{C_0}{T_0} + \frac{I_0^T}{H} = \frac{1}{3} + \frac{2}{15} = 0{,}46$$

$$U'_{M_1} = \frac{C_1}{T_1} + \frac{I_1^T}{H} = \frac{2}{5} + \frac{2}{15} = 0{,}53$$

Por regla general, del estudio de W podemos ver que una interferencia aparace en el instante de tiempo t cuando W_{ijt} cambia de 0 a 1, es decir, cuando W_{ijt}-$W_{ij(t-1)}$= 1 para todas las tareas τ_j que no estén asignadas en el mismo procesador que τ_i.

De este modo, además de computar la interferencia total recibida I_i^T podemos definir y calcular este valor para cada activación, es decir, el tiempo de ejecución debido a la contención.

Teorema 1. *El tiempo de ejecución debido a la contención C'_{is} de τ_i en la activación s es la suma de C_i más las interferencias causadas por las tareas en ejecución en los otros procesadores y se calcula como:*

$$C'_{is} = C_i + \sum_{\tau_j \notin M_k} \left(W_{jisT_i} + \sum_{t=sT_i+1}^{t=(s+1)T_i-1} \text{máx}\left(W_{jit+1} - W_{jit}, 0\right)\right) \cdot I_j \tag{15}$$

Demostración 1. *En el intervalo de la s-ésima activación, $(sT_i, (s+1)T_i - 1)$, el número de interferencias causadas por una tarea emisora τ_j es igual al número de veces que W_{ji} cambia de 0 a 1 desde el instante t a $t+1$, multiplicado por I_j. Si el cambio es a la inversa, es decir, de 1 a 0, entonces no hay interferencia. De este modo, sólo debemos considerar las transiciones de 0 a 1 de la matriz W. Esto se expresa en el término $\text{máx}\left(W_{jit+1} - W_{jit}, 0\right)$. El término W_{jisT_i} da cuenta de la interferencia en el instante inicial en que τ_i se libera.*

Por tanto, la relación entre I_i^T y C'_{is} es:

$$I_i^T = \sum_s (C'_{is} - C_i) \quad \forall s = 0, .., A_i - 1 \tag{16}$$

La matriz de interferencia W se utiliza para establecer el tiempo de cómputo exacto que debe ejecutar una tarea en cada activación para considerar tanto su tiempo de cómputo en el peor caso C_i como el tiempo añadido por la contención C'_{is}. Esta matriz debe calcularse en cada instante t como parte del algoritmo de planificación *offline*.

El Algoritmo 1 muestra el pseudocódigo (tipo Python) para calcular la planificación teniendo en cuenta la interferencia.

Algoritmo 1: Planificación teniendo en cuenta la interferencia

```
1 #definicion de variables e inicializaciones
2 for t in range(H):
3     for k in range(m):
4         TareaEnEjecucion[k] = TareaPrio(M_k)
5     for k in range(m):
6         τ_i = TareaEnEjecucion[k]
7         if τ_i != TareaActual[k]
8             && t % T_i == 0:
9             for s in range(m):
10                if s != k and I_i > 0:
11                    τ_j = TareaEnEjecucion[s]
12                    W[j][i] = 1
13                    C'_i += I_j
14        else:
15            for s in range(m):
16                if s != k and I_i > 0:
17                    τ_j = TareaEnEjecucion[s]
18                    if W[j][i] == 0:
19                        W[j][i] = 1
20                        C'_i += I_j
21    for k in range(m):
22        TareaActual[k] = TareaEnEjecucion[k]
23        #contar ejecucion
24        acabada = Ejecuta(TareaActual[k], k)
25        if acabada:
26            C'_i = C_i
27            for j in range(n):
28                W[j][i] = 0
29                W[i][j] = 0
```

El algoritmo selecciona primero la tarea que se ejecutará en cada procesador según el algoritmo seleccionado. En este caso, se elige la tarea con mayor prioridad para cada procesador (línea 4). Para cada procesador, se evalúa entonces la condición para añadir la interferencia (líneas 7-8). Si la tarea seleccionada para ejecutarse (τ_i) es diferente de la tarea anterior (cambio de contexto) y τ_i se ha liberado en el tiempo t, entonces todas las tareas en ejecución en el resto de los procesadores cambian de 0 a 1 el valor en la columna i para indicar que estas tareas causan interferencia a τ_i (línea 12).

A continuación, C'_i aumenta su valor debido a la interferencia de los otros procesadores que ejecutan tareas (línea 13). Si no hay un cambio de contexto o la tarea en ejecución τ_i se ha reanudado tras haber sido expulsada (línea 14), es posible que en ese intervalo de tiempo (desde la expulsión hasta la reanudación) algunas tareas en otros procesadores se hayan liberado,

por lo que su interferencia debe ser considerada. Esto puede verse si W_{ij} es 0, lo que significa que alguna tarea τ_j ha sido liberada en otro procesador mientras τ_i estaba expulsada. En este caso, se añade la interferencia (líneas 19 y 20) y se registra en W_{ij}. Todas las tareas se ejecutan una vez que se ha comprobado la interferencia de todos los procesadores. Las que terminan la ejecución de su activación, ponen a 0 su correspondiente fila (como tarea emisora) y columna (como tarea receptora) en W (líneas 28 y 29).

Nótese que no es necesario definir W como una matriz tridimensional (se elimina la dimensión t) ya que el cálculo de $C'_i s$ se realiza "al vuelo". Lo mismo ocurre con C'_{is}. Esto reduce notablemente la sobrecarga del algoritmo.

Tras la finalización de la planificación a lo largo de H, se calcula I_i^T para cada tarea y, así, se puede calcular la utilización real del sistema U'_τ.

4.3. Asignación de tareas a procesadores utilizando MILP

Antes de planificar, es necesario asignar las tareas a los procesadores. Como se comentó en el apartado 2, existen diferentes heurísticas de asignación para sistemas multiprocesadores particionados. Éstas tienen como objetivo maximizar el número de tareas a asignar asegurando la planificabilidad. Sin embargo, con nuestro modelo, la utilización real del procesador ya no es U_{M_k} sino U'_{M_k} porque la utilización real de las tareas U'_i se ve incrementada por las interferencias.

La interferencia total recibida por cada tarea I_i^T (y, equivalentemente, C'_{is} en cada activación) es difícil de estimar, ya que depende de la planificación de las tareas asignadas a cada procesador. Por esta razón, no es obvio estimar de antemano I_i^T o C'_{is} para calcular la utilización real por planificable, de forma que podamos saber antes de planificar si una asignación de tareas va a ser programable o no.

El objetivo de este apartado es estudiar W_{ijt} para derivar algoritmos de asignación que traten de tener en cuenta las posibles interferencias de forma que la utilización real (U'_{M_k}) disminuya asegurando la planificabilidad.

En cualquier instante t, la interferencia que una tarea τ_i asignada en el procesador M_k hace que todas las demás tareas en otros procesadores comprueben su correspondiente fila i de W en la que los valores son 1 y la multipliquen por I_i.

Para reducir las interferencias causadas, nos interesa tener el mayor número posible de ceros en la matriz W. Los elementos de la fila i que son siempre cero son:

- cuando $j = i$
- cuando para la columna j, τ_j se asigna al mismo procesador que τ_i
- cuando $I_i = 0$.

Por tanto, un límite para la máxima interferencia que puede recibir un procesador M_k es:

$$maxW_k = \sum_{\substack{\tau_i \in M_k \\ I_i \neq 0}} \sum_{\tau_j \notin M_k} I_j \tag{17}$$

Este valor no puede alcanzarse en un instante porque dos tareas en el mismo procesador no pueden estar activas al mismo tiempo. Pero este límite se puede alcanzar en un intervalo $[s \cdot T_i, (s+1) \cdot T_i]$.

Finalmente, suponiendo que todos los valores de W que pueden ser 1 son efectivamente 1, tenemos que el valor máximo de la suma de todos los elementos de W es:

$$maxW = \sum_{\forall k} maxW_k = \sum_{k=1}^{m} \sum_{\substack{\tau_i \in M_k \\ I_i \neq 0}} \sum_{\tau_j \notin M_k} I_j \tag{18}$$

Esto no es un límite de $\sum_j I_j^T$, ya que esta matriz cambia cuando t cambia. El valor total de la interferencia puede ser mayor si, por ejemplo, dos activaciones de τ_j interfieren con una activación de τ_i.

Del análisis anterior de la matriz W podemos intuir que un algoritmo de asignación de tareas que desequilibre la carga entre procesadores, o que minimice la ecuación 18, tenderá a tener una menor utilización real que un algoritmo de asignación que equilibre la carga. Esto es así porque si se agrupan el mayor número de tareas en un mismo procesador, estas tareas no producirán interferencia entre ellas.

Por tanto, un algoritmo de asignación que intente asignar el mayor número de tareas a un mismo procesador, generará menor interferencia total que si las tareas se reparten uniformemente entre los procesadores. Por el contrario, un algoritmo de asignación de tareas que equilibra la carga entre los procesadores tenderá a planificar más conjuntos de tareas que un algoritmo desequilibrado.

Para formular el algoritmo de asignación de tareas a procesadores Wmin mediante programación lineal entera, utilizaremos la notación de la Tabla 3.

Tabla 3: Notación del modelo

CONJUNTOS E ÍNDICES	
i	Tareas $\tau_i \in \{0, 1, 2, ..., n-1\}$
k	Procesadores $M_k \in \{0, 1, 2, ..., m-1\}$
PARÁMETROS	
C_i	WCRT de τ_i
T_i	Periodo de τ_i
U_i	Utilización teórica de τ_i
I_i	Factor de interferencia de τ_i sobre otras tareas
VARIABLES DE DECISIÓN	
O_{ik}	Matriz de asignación. 1 si τ_i se asigna en el procesador k y 0 en caso contrario.
U_{M_k}	Utilización teórica del procesador k.
$maxWk$	Valor máximo de la suma de todos los elementos de W para el procesador k.
$maxW$	Valor máximo de la suma de todos los elementos de W para todos los procesadores.

La función objetivo de este algoritmo es minimizar la ecuación (18).

$$\text{Minimizar } maxW \tag{19}$$

s.a:

$$\sum_{\forall k} O_{ik} = 1 \qquad \forall i \tag{20}$$

$$\sum_{i \in k} U_i \cdot O_{ik} = U_{M_k} \qquad \forall k \tag{21}$$

$$U_{M_k} \leq 1 \qquad \forall k \tag{22}$$

$$\sum_{\substack{\tau_i \in M_k \\ I_i \neq 0}} \sum_{\tau_j \notin M_k} I_j = maxW_k \qquad \forall k \tag{23}$$

$$O_{ik} \in \{0, 1\} \tag{24}$$

$$U_{M_k}, maxW_k \geq 0 \tag{25}$$

Las restricciones definidas en las ecuaciones (20), (21) y (22) definen la capacidad de cada procesador. La ecuación (23) calcula la máxima interferencia provocada por todos los procesadores, siguiendo la ecuación (18) definida anteriormente. Las ecuaciones (24) y (25) representan los dominios de las variables de decisión.

Como el modelo tiene variables binarias y enteras, es un problema de programación lineal entera mixta (MILP).

5. Evaluación

El *solver* elegido para ejecutar todos los problemas de optimización MILP presentados en este trabajo es Gurobi versión 9.0 (Gurobi, 2019), de Gurobi Optimization, Inc. Se trata de un potente optimizador diseñado desde cero para ejecutarse en multinúcleos y con capacidad para ejecutarse en modo paralelo. Ha conseguido mejorar el rendimiento con cada versión y ofrece una interfaz en Python. Todos los experimentos se han ejecutado en una CPU Intel Core i7 con 16 GB de RAM.

El escenario de simulación desarrollado para esta parte se divide en cinco pasos:

- Generación de la carga.La carga se genera utilizando un generador de tareas sintéticas. El número de tareas de cada conjunto y la utilización total del sistema dependen del número de procesadores en los que se asignan. Como estos experimentos se realizan en 2, 4, 8 y 10 procesadores, establecemos un número razonable de tareas y una carga factible para cada número de procesadores. Dado el valor de utilización del sistema y el número de tareas para cada conjunto, la utilización se reparte entre las tareas utilizando de nuevo el algoritmo UUniFast-discard (Davis and Burns, 2009). Los plazos se generan aleatoriamente en el rango [20,1000] y los tiempos de cómputo se deducen de la utilización del sistema.

  Sin pérdida de generalidad, los plazos se fijan para que sean iguales a los períodos, aunque podrían restringirse para que sean menores o iguales a los períodos.

- Asignación de tareas a procesadores: Se han probado los siguientes algoritmos de asignación:

  - FFDU (First Fit Decreased Utilization). Asigna una tarea al primer procesador donde es posible. El orden de elección de las tareas es por utilización decreciente.
  - WFDU (Worst Fit Decreased Utilization). Asigna una tarea al procesador que tiene el menor hueco de utilización disponible. El orden de elección de las tareas es por utilización decreciente.
  - UDMax, UDMin: Tienen como objetivo maximizar o minimizar la diferencia entre utilizaciones de los procesadores, respectivamente.
  - Wmin descrito en el apartado 4.3.

- Validación de la fase de asignación de tareas a procesadores. La primera fase de validación consiste en comprobar si se han asignado todas las tareas a los procesadores y en asegurarse de que no se supera la capacidad máxima por procesador. Si alguno de los algoritmos de asignación no puede asignar totalmente el conjunto de tareas, este conjunto de tareas se descarta y se genera uno nuevo.

- Planificación. Se utilizará el algoritmo descrito en el apartado 4.2.

- Validación de la fase de planificación. La validación de los planes generados implica dos pasos. En primer lugar, hay que comprobar la planificabilidad para garantizar que se cumplen todos los plazos dentro del hiperperiodo. En segundo lugar, se obtienen algunos parámetros de rendimiento para comparar los distintos métodos. En concreto, se obtiene la relación entre la utilización teórica del sistema y la utilización real del sistema para cada conjunto, medida después de la fase de planificación.

  Además, una vez terminada la ejecución de todos los conjuntos de tareas generados, los parámetros que deben evaluarse son:

  - Ratio de planificabilidad. El porcentaje de conjuntos de tareas con planificaciones factibles sobre el total de conjuntos de tareas con asignaciones factibles.
  - Utilización aumentada. El aumento de la utilización con respecto a la utilización teórica. Se mide como $1 - \sum_k \frac{U_{M_k}}{U'_{M_k}} = 1 - \frac{U_\tau}{U'_\tau}$.

Las figuras 8 y 9 muestran la ratio de planificabilidad y el aumento de la utilización en función de cada algoritmo de asignación. De estas figuras podemos concluir que los algoritmos FFDU, BFDU o UDmax consiguen hasta un 43 % de planificabilidad con un pequeño aumento de la utilización del sistema debido a las interferencias. Sin embargo, WFDU o UDmin consiguen casi un 100 % de planificabilidad a costa de un aumento del 2,3 % en la utilización. Los algoritmos anteriores funcionan de forma opuesta: los primeros consiguen una carga desequilibrada, lo que reduce la planificabilidad cuando aparecen interferencias, y los segundos garantizan la planificabilidad pero la utilización del sistema aumenta. Por último, el algoritmo Wmin proporciona una rato alta de planificabilidad (hasta el 89 %) con un aumento de la utilización de sólo el 0,266 %.

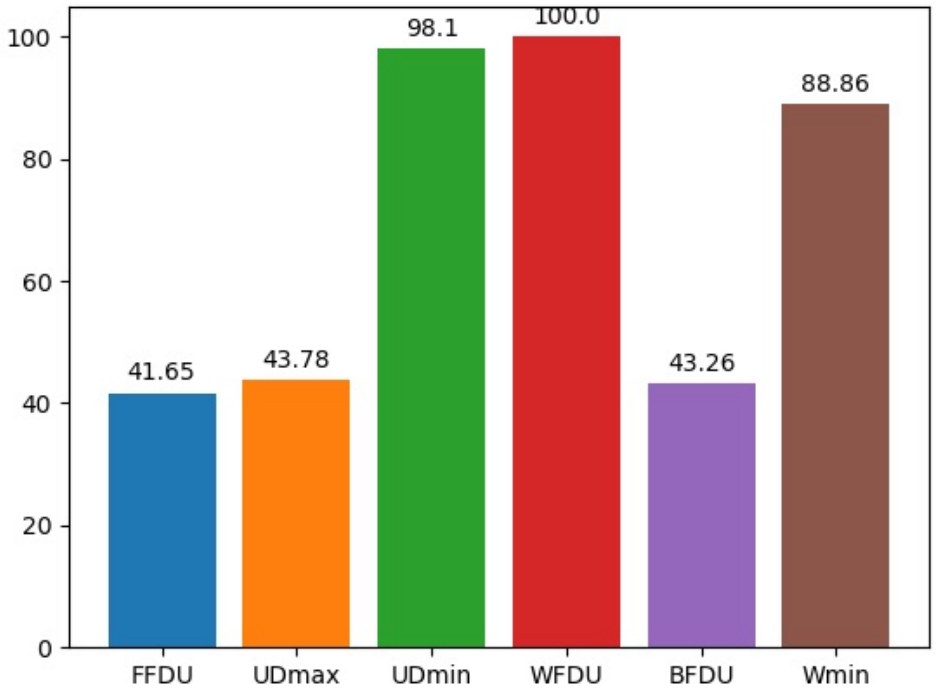

Figura 8: Porcentaje de conjuntos de tareas planificables en función del algoritmo de asignación.

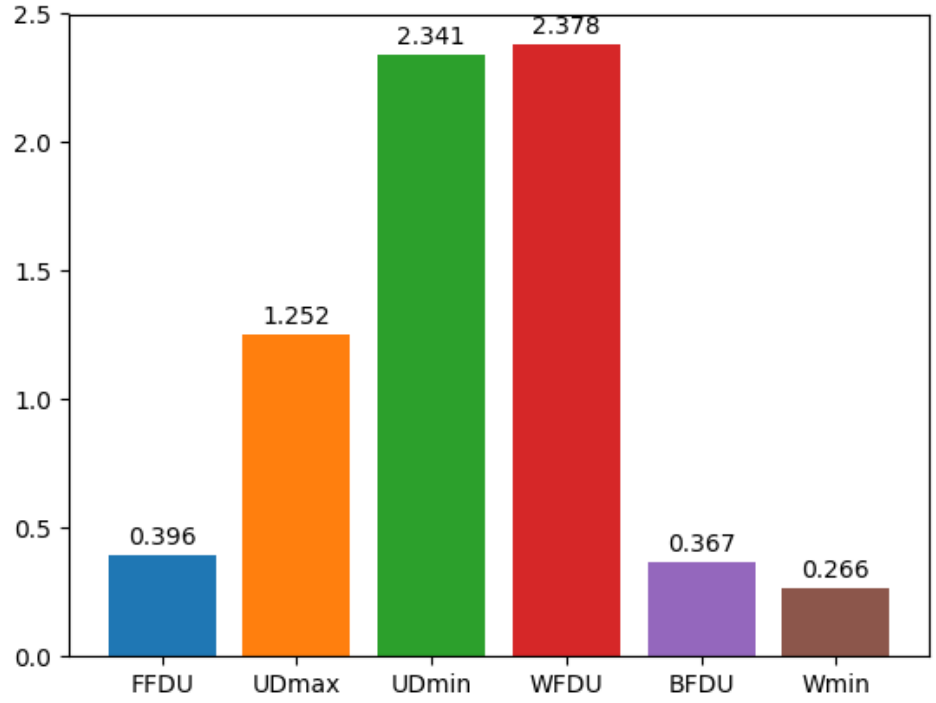

Figura 9: Aumento en la utilización con respecto al algoritmo de asignación

6. Conclusiones

En este trabajo se ha investigado sobre la utilización de técnicas de planificación no convencionales utilizando programación lineal entera. En primer lugar, se ha abordado el problema de la planificación en sistemas monoprocesador, proporcionando un modelo basado en programación lineal entera mixta. Con este tipo de planificación, se pueden proporcionar diferentes objetivos a minimizar, como puede ser el tiempo de respuesta propuesto en este artículo. Pero se pueden abordar otros objetivos, como reducir los cambios de contexto o el *jitter*. En segundo lugar, se ha abordado el mismo problema en multiprocesador. En este caso, es necesario tener en cuenta más factores: la interferencia generada por el acceso a recursos hardware comunes y la fase de asignación de tareas a procesadores. Respecto al primer punto, se ha propuesto un modelo general de interferencia entre procesadores y un algoritmo de planificación que tiene en cuenta esta interferencia. Respecto al segundo punto, se ha propuesto un modelo MILP que asigna tareas a procesadores con el objetivo de minimizar esta interferencia. La mayoría de trabajos existentes sirven para un tipo específico de recurso *hardware*. De los algoritmos de asignación de tareas a procesadores, podemos concluir que WMin serían una buena opción ya que mantiene una ratio de planificabilidad alta (un poco por debajo de WFDU que presenta la más alta) y una interferencia generada baja. Wmin, por tanto, sería capaz de planificar sistemas de tareas con carga más alta debido a que genera menos sobrecargas por interferencia entre procesadores.

Agradecimientos

Esta publicación es parte del proyecto de I+D+i PLEC2021-007609 financiado por MCIN/ AEI/ 10.13039/501100011033 y por "Unión Europea NextGenerationEU / PRTR".

Referencias

Baker, T. P., Shaw, A., Dec 1988. The cyclic executive model and ada. In: Proceedings. Real-Time Systems Symposium. pp. 120–129.

Baruah, S., 2004. Feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms. In: 25th IEEE International Real-Time Systems Symposium. pp. 37–46.

Baruah, S., Mar. 2022. An ILP representation of a DAG scheduling problem. Real-Time Systems 58 (1), 85–102.

Crespo, A., Ripoll, I., Albertos, P., 1999. Reducing delays in rt control: The control action interval. IFAC Proceedings Volumes 32 (2), 8527 – 8532, 14th IFAC World Congress 1999, Beijing, Chia, 5-9 July.

Dasari, D., Andersson, B., Nelis, V., Petters, S. M., Easwaran, A., Lee, J., 2011. Response time analysis of cots-based multicores considering the contention on the shared memory bus. In: 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications. pp. 1068–1075.

Davis, R. I., Burns, A., Dec 2009. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In: 2009 30th IEEE Real-Time Systems Symposium. pp. 398–409.

Davis, R. I., Burns, A., Oct. 2011. A survey of hard real-time scheduling for multiprocessor systems. ACM Comput. Surv. 43 (4).

Davis, R. I., Griffin, D., Bate, I., 2021. Schedulability analysis for multi-core systems accounting for resource stress and sensitivity. In: 33rd Euromicro Conference on Real-Time Systems, ECRTS 2021,.

Di Natale, M., Zeng, H., 04 2013. An efficient formulation of the real-time feasibility region for design optimization. IEEE Transactions on Computers 62, 644–661.

Fernandez, G., Abella, J., Quiñones, E., Rochange, C., Vardanega, T., Cazorla, F., 2014. Contention in multicore hardware shared resources: Understanding of the state of the art. In: WCET.

Fleming, T., Burns, A., 2015. Investigating mixed criticality cyclic executive schedule generation. In: Proc. Workshop on Mixed Criticality (WMC).

Guasque, A., Tohidi, H., Balbastre, P., Aceituno, J. M., Simó, J., Crespo, A., 2020. Integer programming techniques for static scheduling of hard real-time systems. IEEE Access 8, 170389–170403.

Gurobi, 2019. Gurobi optimizer reference manual. Inc. Gurobi Optimization.

Harter, Jr., P. K., Aug. 1987. Response times in level-structured systems. ACM Trans. Comput. Syst. 5 (3), 232–248.

Hong, I., Kirovski, D., Gang Qu, Potkonjak, M., Srivastava, M. B., 1999. Power optimization of variable-voltage core-based systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 18 (12), 1702–1714.

Joseph, M., Pandya, P., 1986. Finding response times in a real-time system. The Computer Journal 29 (5), 390–395.

Kim, H., de Niz, D., Andersson, B., Klein, M., Mutlu, O., Rajkumar, R., 2014. Bounding memory interference delay in cots-based multi-core systems. In: 2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS). pp. 145–154.

Lampka, K., Giannopoulou, G., Pellizzoni, R., Wu, Z., Stoimenov, N., 11 2014. A formal approach to the wcrt analysis of multicore systems with memory contention under phase-structured task sets. Real-Time Systems 50, 736–773.

Leung, J., Whitehead, J., 1982. On the complexity of fixed-priority schedulings of periodic, real-time tasks. Performance Evaluation 2 (4), 237–250.

Lisper, B., Mellgren, P., 10 2001. Response-time calculation and priority assignment with integer programming methods.

Liu, C. L., Layland, J. W., Jan. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM 20 (1), 46–61.

Locke, C. D., Mar. 1992. Software architecture for hard real-time applications: Cyclic executives vs. fixed priority executives. Real-Time Syst. 4 (1), 37–53.

Maiza, C., Rihani, H., Rivas, J. M., Goossens, J., Altmeyer, S., Davis, R. I., jun 2019. A survey of timing verification techniques for multi-core real-time systems. ACM Comput. Surv. 52 (3).

Mangeruca, L., Baleani, M., Ferrari, A., Sangiovanni-Vincentelli, A., Dec. 2007. Uniprocessor scheduling under precedence constraints for embedded systems design. ACM Trans. Embed. Comput. Syst. 7 (1).

Nguyen, V. A., Hardy, D., Puaut, I., 2019. Cache-conscious off-line real-time scheduling for multi-core platforms: algorithms and implementation. Real-Time Systems 55 (4), 810–849.

Sun, Y., Natale, M. D., Sep. 2017. Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks. ACM Trans. Embed. Comput. Syst. 16 (5s).