Universidad Zaragoza

Trabajo Fin de Grado

Evaluación de dos métodos de cálculo de ejecutivos cíclicos para tiempo real duro sobre chips multinúcleo

Autor

Adrián Martín Marcos

Directores

José Luis Briz Velasco

Laura Elena Rubio Anguiano

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2022

AGRADECIMIENTOS

En primer lugar, quiero comenzar dando las gracias a mi director de TFG, Jose Luis Briz, por darme la oportunidad de realizar este trabajo que tanto he disfrutado. También a mi codirectora, Laura Elena Rubio, por su dedicación y ayuda durante todos estos meses, gracias a los cuales la realización de este trabajo ha sido más fácil.

Quiero aprovechar para recordar a todos los compañeros con los que he compartido tantas horas en clases y laboratorios, y también a esos profesores que sin duda han hecho más sencillo el paso por el grado.

Finalmente, quiero dar especialmente las gracias a mi madre, quien pese a no entender algunas de las cosas de las que le hablaba, siempre me ha seguido escuchando. También quiero agradecerle a mi novia, Almudena, el haber estado conmigo estos años de carrera, haciéndome más llevaderos los días que hemos coincidido en el CPS. Por último, quiero dar las gracias a mi abuelo Lorenzo, a quien pude llegar a decirle que haría esta carrera, pues le he tenido presente durante todo este tiempo.

Este trabajo ha sido parcialmente financiado por MINECO/AEI/FEDER (UE) (proyecto PID2019-105660RB-C21 / AEI / 10.13039/501100011033), y por ayudas a grupos de investigación de referencia (T5820R, Dpto. de Ciencia, Universidad y Sociedad del Conocimiento, Gobierno de Aragón).

RESUMEN

Este Trabajo de Fin de Grado (TFG) se centra en la implementación de los algoritmos de planificación estática descritos en [1] y en su comparación con el algoritmo CAlECS [2] y el planificador de referencia RUN [3]. Estos algoritmos persiguen la obtención de un Ejecutivo Cíclico (EC) que maximiza la utilización de los procesadores o núcleos disponibles mediante métodos de optimización matemática basados en el principio de *planificación fluida*. Esta comparación no se había realizado previamente y pone de relieve las debilidades de los algoritmos no expulsivos, y las limitaciones de las aproximaciones expuestas en [1] respecto a CAlECS. En la comparación se incluyen también otros métodos que, aunque ya habían sido comparados entre sí en [2], permiten comprobar la coherencia de los resultados y proporcionan una mejor perspectiva de los problemas subyacentes. El trabajo experimental se ha realizado sobre el entorno de simulación *Tertimuss* [4], al que se ha contribuido con nuevos componentes, mejorando también la estabilidad del mismo.

Índice

1. Introducción **3**

1.1. Motivación y contexto 3

1.2. Objetivos 4

1.2.1. Objetivos generales 4

1.2.2. Objetivos específicos 4

1.3. Contribuciones 4

1.4. Metodología y entorno de trabajo 5

1.5. Planificación 6

1.6. Estructura del documento 6

2. Fundamentos **7**

2.1. Conceptos básicos 7

2.1.1. Arquitectura subyacente 7

2.1.2. Unidades de cómputo (ciclos) 7

2.1.3. Sistema tiempo real 8

2.1.4. Modelo de tarea 8

2.1.5. Métricas básicas 9

2.2. Planificación 10

2.2.1. Factibilidad 10

2.2.2. Planificabilidad 10

2.2.3. Punto de planificación 11

2.2.4. Expulsión 11

2.2.5. Migración 11

2.2.6. Afinidad 11

2.2.7. Planificación expulsiva 11

2.2.8. Planificación no expulsiva 12

2.2.9. Planificación fuera de línea 12

2.2.10. Planificación en línea 12

2.2.11. Planificación particionada 12

2.2.12. Planificación global 13
2.3. Ejecutivo cíclico 13
2.4. Algoritmos de planificación comparados 13
2.4.1. Allocation and Execution Control Scheduler (AlECS) 13
2.4.2. Clustered Allocation and Execution Control Scheduler (CAlECS) 14
2.4.3. Reduction to Uniprocessor (RUN) 14
2.4.4. Deutschbein Cyclic Executive (DCE) 14
2.5. Tertimuss 15
2.6. Programación lineal 15
2.6.1. Algoritmo SIMPLEX 16
2.6.2. Método de punto interior (*inner-point method*) 16

3. Algoritmo Deutschbein Cyclic Executive (DCE) 17
3.1. Consideraciones previas 17
3.1.1. Notación 18
3.2. Definición del problema de programación lineal 19
3.2.1. Variables 19
3.2.2. Función objetivo 20
3.2.3. Restricciones 20
3.2.4. Cambios en la definición del PPL en el caso no expulsivo 25
3.2.5. Solucionadores de problemas de optimización 27
3.3. Discretización de los resultados del PPL 28
3.3.1. Necesidad de la discretización de los resultados 29
3.3.2. Mejora basada en el registro de los ciclos por asignar 30
3.3.3. Mejora basada en la ordenación con respecto al error cometido . 31
3.3.4. Suboptimalidad inherente a la solución discretizada 32
3.4. Cálculo de la frecuencia mínima requerida 32
3.5. Obtención del ejecutivo cíclico a partir de las asignaciones 33
3.5.1. Ejecutivo cíclico expulsivo 34
3.5.2. Ejecutivo cíclico no-expulsivo usando directamente la solución del PPL 35
3.5.3. Ejecutivo cíclico no-expulsivo usando la regla de *McNaughton* . 35
3.6. Ejemplo completo 37
3.7. Pruebas de corrección de la implementación 37
3.7.1. Corrección formal 37
3.7.2. Pruebas de integración continua 37

4. Comparativa entre ejecutivos cíclicos **39**
4.1. Metodología y entorno experimental . . . 39
4.2. Resultados experimentales . . . 40
4.2.1. Tiempo de ejecución . . . 40
4.2.2. Frecuencia mínima requerida . . . 40
4.2.3. Uso de los procesadores . . . 42
4.2.4. Migraciones y expulsiones . . . 43
4.3. Valoración . . . 47

5. Conclusiones **49**
5.1. Recapitulación . . . 49
5.1.1. Métodos implementados y su relevancia . . . 49
5.1.2. Métodos comparados y su relevancia . . . 49
5.1.3. Resultados de la comparación experimental . . . 49
5.1.4. Planificación basada en optimización matemática . . . 50
5.1.5. Otras contribuciones . . . 50
5.2. Trabajo futuro . . . 51
5.2.1. Rediseño de DCE para evitar la suboptimalidad . . . 51
5.2.2. Añadir herramientas de análisis y mejorar la visualización de los resultados . . . 52
5.2.3. Mejora en la documentación de *Tertimuss* . . . 52
5.2.4. Distribución de *Tertimuss* a través de *PyPi* . . . 52
5.2.5. Implementación de utilidad para la conversión de medidas de tiempo . . . 52
5.2.6. Importación y exportación de objetos de *Tertimuss* . . . 52
5.2.7. Formalización de los casos de prueba . . . 53
5.2.8. Jerarquía de clases con las que modelar los casos de error . . . 53
5.2.9. Ampliaciones mayores en cuanto a las funcionalidades de la herramienta . . . 53

Bibliografía **55**

Siglas **59**

Lista de Figuras **61**

Lista de Tablas **63**

Anexos **64**

A. Ejemplo de discretización de la solución del PPL **67**
A.1. Aplicación de la primera mejora 67
A.2. Aplicación de la primera y la segunda mejora 70

B. Ejemplo de aplicación de la regla de McNaughton **73**
B.1. Asignación usando la regla de *McNaughton* 76
B.1.1. Acumulación de los ciclos asignados a los trabajos durante el marco 76
B.1.2. Aplicación de la regla de *McNaughton* para obtener la asignación 77

C. Ejemplo de funcionamiento de Deutschbein CE **81**
C.1. Especificación de la plataforma y el conjunto de tareas 81
C.2. Definición del problema de programación lineal 82
C.2.1. Definición de las variables 82
C.2.2. Definición de las restricciones 83
C.3. Discretización 84
C.4. Obtención de la frecuencia 85
C.4.1. Caso expulsivo 85
C.4.2. Caso no expulsivo 85
C.5. Obtención del ejecutivo 86

D. Implementación y simulación de Deutschbein CE en Tertimuss **89**

E. Contribuciones a la herramienta Tertimuss **93**
E.1. Mejoras en la documentación 93
E.2. Planificador DCE 93
E.2.1. Añadir función para calcular el valor del ciclo menor del ejecutivo 94
E.2.2. Implementar una opción para la detección de la paralelización de trabajos durante la simulación 94
E.2.3. Actualizar la versión del componente *ortools* del paquete para solucionar el error con *Gurobi* 94

F. Instalación y configuración de Gurobi **95**
F.0.1. Problemas con la versión de *ortools* 95

G. Planificación del proyecto **97**
G.1. Horas dedicadas 97
G.2. Diagrama de Gantt 97

Capítulo 1

Introducción

1.1. Motivación y contexto

Los microprocesadores multinúcleo y especialmente los Multiprocessor System on a Chip (MPSoC)s han abierto nuevas posibilidades en el diseño de sistemas empotrados en Tiempo Real (TR). A la vez, también han supuesto un desafío en el diseño de nuevos algoritmos de planificación TR capaces de sacar el mejor partido posible a este tipo de hardware. Esta es una de las líneas de investigación del gaZ (Grupo de Arquitectura de Computadores de Zaragoza) de la Universidad de Zaragoza, desarrollada en colaboración con el CINVESTAV-IPN de Guadalajara, México. Esta colaboración ha producido contribuciones significativas en el campo durante los últimos años, traducidas en publicaciones en revistas de prestigio, dos tesis doctorales y varios trabajos de fin de grado y máster en las dos instituciones colaboradoras.

Se han desarrollado algoritmos de planificación TR muy eficientes en el uso de los procesadores disponibles, basados en principios de *planificación fluida*. Sin embargo, estos algoritmos son difícilmente adaptables a la realidad debido a la complejidad y la sobrecarga que pueden llegar a suponer en tiempo de ejecución. Es uno de los motivos por los que la industria sigue utilizando fundamentalmente ejecutivos cíclicos, con técnicas conservadoras que conducen al sobredimensionado innecesario de los sistemas, un efecto que la propia industria intenta evitar.

Recientemente se ha demostrado que pueden generarse ejecutivos cíclicos muy eficientes basados en los principios de *planificación fluida* [1, 5]. Es una línea muy novedosa y prometedora por el atractivo que puede tener para la industria [6]. Este TFG se centra en la implementación y comparación experimental de algunos de ellos, en el contexto de la línea de investigación antes referida.

1.2. Objetivos

1.2.1. Objetivos generales

Mediante la realización de este TFG se persigue, por una parte, la ampliación de conceptos relacionados con sistemas TR, explotando de forma integrada conocimiento y habilidades adquiridas durante la titulación, y por otra, un acercamiento a los problemas y métodos propios de la investigación en este campo.

1.2.2. Objetivos específicos

- Estudio de los problemas y métodos relacionados con la planificación tiempo real duro sobre multiprocesadores.
- Análisis del algoritmo Deutschbein Cyclic Executive (DCE) y su implementación en la herramienta *Tertimuss*.
- Comparación experimental de los algoritmos de planificación sobre multiprocesadores presentes en la herramienta *Tertimuss* con DCE.
- Integración de los componentes implementados en la herramienta *Tertimuss*.

1.3. Contribuciones

La consecución de los objetivos anteriores ha generado las siguientes contribuciones:

- Nuevo componente en *Tertimuss* correspondiente al método de planificación que se ha implementado.
- Ampliación de *Tertimuss* con nuevos métodos y funcionalidades, entre los que destacan:
    - La detección de la paralelización de *jobs* durante la simulación con base a las decisiones de planificación. Para comprobar el correcto funcionamiento de esta aportación, se diseñó e implementó un segundo planificador, de complejidad mucho menor, que se utilizó para comprobar el correcto funcionamiento de la herramienta ante la posibilidad y el impedimento de paralelizar los trabajos en la planificación.
    - Método para el cálculo del ciclo menor (duración del *frame* del Ejecutivo Cíclico (EC)), análogo al ya existente para el cálculo del hiperperiodo.
    - Edición de la documentación correspondiente a tutoriales de la herramienta, así como del código perteneciente a los mismos.

– Identificación y subsanación de errores en *Tertimuss*:

  - Actualización de la versión de *ortools* en el paquete de Python para corregir un fallo [7] que impedía la integración de herramientas de terceros (concretamente, Gurobi) para la resolución de problemas de programación lineal.
  - Identificación de problemas en el paquete con Python 3.10 (parcialmente subsanados al actualizar la versión de *ortools*).
  - Identificación de un fallo en el simulador, por el cual ciertos *jobs* no se activaban cuando corresponde al introducir como frecuencias disponibles ciertas combinaciones de valores. A fecha de la redacción de esta memoria, se sigue depurando la causa y las posibles formas de solventar este problema.

– La reformulación del Problema de Programación Lineal para solventar la suboptimalidad de la solución para DCE tras la discretización.

– Esta memoria de TFG, que incluye:

  - Diagramas UML de la implementación realizada.
  - Ejemplos de elaboración propia para ilustrar los contenidos desarrollados.

1.4. Metodología y entorno de trabajo

La realización de este trabajo estuvo inicialmente guiada por la implementación de las diferentes etapas del planificador descrito en el artículo de referencia [1]. A medida que se avanzó, surgieron problemas que requirieron del estudio de diferentes soluciones que se discutirán más adelante, así como de la toma de decisiones con respecto a cuáles de ellas aplicar en la implementación realizada. Una vez se tuvo una versión estable del planificador, se procedió a evaluarlo en una comparativa con el resto de planificadores ya implementados en *Tertimuss*, dando como resultado las diferentes métricas analizadas en el Cap. 4. Durante todo ese proceso, fue necesario profundizar en las técnicas planificación sobre multiprocesadores, así como en el marco teórico sobre el cual éstas se fundamentan. Gracias a la documentación de cada uno de los pasos se ha construido esta memoria, como una recopilación del trabajo realizado durante estos meses.

La principal herramienta de trabajo ha sido *Visual Studio Code* [8], tanto por su integración del sistema de control de versiones *Git* [9] (y la conectividad con la plataforma *GitHub* [10], en la cuál estaba alojado el código) como por las facilidades

que ofrece para el desarrollo en Python a través de la extensión para dicho lenguaje [11], entre las que destacan el depurador y el reconocimiento de las pruebas de integración continua.

En cuanto a la realización de esta memoria, se ha utilizado *Overleaf* [12] como editor colaborativo de *LaTeX*. Adicionalmente, para transcribir las notas manuscritas, se ha utilizado la tecnología *OCR* integrada en *Google Docs* [13]. Además, se ha utilizado la herramienta *Draw.io* [14] para la realización de los diagramas, exportando solo el recorte en formato *PDF* con el fin de mantener la calidad de la imagen vectorial.

La metodología y entorno específico de la comparativa se describe en la 4.1.

1.5. Planificación

Como ya se ha mencionado, la realización de este trabajo ha estado marcada por la consecución de los hitos que suponían la implementación de las etapas del planificador descrito en el artículo de referencia [1]. La planificación de las diferentes etapas está documentada en el Anexo G, donde se incluye una tabla con la dedicación aproximada en horas a cada una de ellas acompañada de un diagrama de Gantt.

1.6. Estructura del documento

Esta memoria expone en el Cap. 2 los conceptos básicos necesarios para comprender el trabajo. En el Cap. 3 se describe el planificador estudiado, explicando tanto sus bases teóricas como los detalles ligados a su implementación en *Tertimuss*. Tras ello, en el Cap. 4 se realiza una comparativa del mismo con los otros planificadores ya presentes en la herramienta, analizando los resultados obtenidos en la simulaciones con conjuntos de tareas generados sintéticamente. Por último, en el Cap. 5 se evalúa la satisfacción de los objetivos presentados y las líneas abiertas como consecuencia de la realización de este trabajo.

Capítulo 2

Fundamentos

En este capítulo se definen los términos que van a ser utilizados en el resto del documento, con el objetivo de que éste sea autocontenido. Así, si bien existe cierta heterogeneidad con respecto a la nomenclatura con la que los diferentes algoritmos hacen uso de estos términos, el fundamento teórico tras todos ellos es el mismo, y es precisamente lo que se busca transmitir con las diferentes definiciones.

La definición de los términos que hay a continuación es de elaboración propia, pero se basa en algunas de las principales referencias que existen al respecto [15, 16, 17].

Los términos *procesador*, *núcleo* y *CPU* serán usados de manera indistinta en lo que sigue.

2.1. Conceptos básicos

2.1.1. Arquitectura subyacente

En este trabajo se asume una arquitectura multiprocesador en el que cada procesador es capaz de ejecutar un único flujo de instrucciones y carece de mecanismos especulativos (tales como predicción de saltos o ejecución fuera de orden) y de ocultación de latencia de acceso a memoria (tales como memorias cache y mecanismos relacionados). En lo que sigue, se consideran equivalentes –y se utilizan indistintamente– los términos *procesador*, *núcleo* y CPU.

2.1.2. Unidades de cómputo (ciclos)

La frecuencia de cómputo de un procesador es la cantidad de ciclos (o unidad de cómputo) que puede procesar por unidad de tiempo, y se expresa en hercios (Hz). Análogamente, se define el ciclo o unidad de cómputo (uc) a la cantidad cómputo realizable por una CPU funcionando 1 Hz de frecuencia durante un 1 segundo. Del análisis dimensional, se obtiene $[uc] = [Hz] \cdot [s] = [s^{-1}] \cdot [s] = [\frac{1}{s}] \cdot [s] = 1$, es decir que los ciclos son adimensionales.

La frecuencia de cómputo es al ciclo lo que la velocidad a la distancia; representa la variación en esa magnitud con respecto al tiempo.

Sea $W(t)$ una carga de trabajo medida en ciclos y variable con respecto tiempo, durante el intervalo $[t_0, t_f]$ con duración $\Delta t = tf - t_0$, entonces la frecuencia de cómputo mínima o la «velocidad de cómputo» requerida para completarla se obtiene como,

$$f_{\min} = \frac{W(t_0 + \Delta t) - W(t_0)}{\Delta t} \tag{2.1}$$

cuando el diferencial de tiempo tiende a cero, esta expresión se reecribe

$$f_{\min} = \lim_{\Delta t \to 0} \frac{W(t_0 + \Delta t) - W(t_0)}{\Delta t} = \frac{\mathrm{d}}{\mathrm{d}x} W(t) \tag{2.2}$$

es decir, la frecuencia mínima para realizar un cómputo puede obtenerse como la primera derivada del cómputo requerido medido en ciclos con respeto al tiempo en el que éste ha de llevarse a cabo.

2.1.3. Sistema tiempo real

Un sistema Tiempo Real (TR) es aquel cuya corrección depende tanto del resultado computacional que produce como del momento en el que éste es obtenido.

2.1.4. Modelo de tarea

Una tarea (*task*) es una secuencia de instrucciones que supone una actividad de un sistema Tiempo Real, y que se activa varias veces a lo largo de la vida del sistema con diferentes datos de entrada y por tanto, puede implicar diferente tiempo de cómputo asociado. Entre sus atributos cabe destacar:

- El plazo de respuesta (*deadline*), que es el tiempo máximo que puede tardar en completarse la ejecución de la tarea desde su activación.
- El tiempo de cómputo en el peor caso o Worst Case Execution Time (WCET), medido en ciclos, que es el cómputo asociado a la mayor carga de trabajo que puede suponer la activación de la tarea.
- La criticidad, que determina las consecuencias que tiene sobre el sistema el incumplimiento del plazo de respuesta. Según ésta, puede ser:
  - Crítica (*hard*), si el incumplimiento del plazo establecido para cualquiera de sus resultados supone el fallo total del sistema.
  - Firme (*firm*), si la utilidad del resultado es nula en caso de que éste se produzca fuera del plazo establecido.

    - Blanda (*soft*), si la utilidad del resultado se degrada a medida que pasa el tiempo una vez ha pasado el plazo.

- El patrón de activación, según la regularidad temporal entre activaciones consecutivas de la tarea. Puede ser:

    - Periódica, si sus activaciones se producen en intervalos de tiempo regulares. Al intervalo de tiempo entre activaciones consecutivas se le llama periodo de la tarea.
    - Esporádica, si su activación se produce en intervalos de tiempo irregulares sujetos a la ocurrencia de ciertos eventos en el sistema.

Un trabajo (*job*) es la instancia de una tarea, y se corresponde con una activación concreta de ésta con unos datos de entrada determinados (y por lo tanto, también con un tiempo de cómputo asociado concreto). Sus atributos, por lo tanto, derivan de los atributos de la tarea.

El modelo de tres parámetros para la especificación de tareas periódicas consiste en la definición de las tareas periódicas del sistema a través de sus tres atributos principales: el periodo, el plazo de respuesta y el tiempo de cómputo en el peor caso.

El modelo de plazo de respuesta implícito para la especificación de tareas periódicas es una simplificación del modelo de tres parámetros que se tiene cuando el plazo de respuesta de la tarea es igual a su periodo, y por lo tanto, para definir a las tareas periódicas basta con especificar su periodo y su tiempo de cómputo en el peor caso.

2.1.5. Métricas básicas

Se define el trabajo solicitado al procesador en un instante t, $W(t)$, como la suma del tiempo de cómputo (en ciclos) requerido al procesador por todos los trabajos cuya activación se ha producido en el intervalo de tiempo $[0, t]$.

$$W(t) = \sum_{i=1}^{n} \lceil \frac{t}{T_i} \rceil \cdot C_i \tag{2.3}$$

Siendo $\tau = \{\tau_i = (C_i, T_i)\}_{i=1}^{n}$ el conjunto de tareas del sistema, donde C_i corresponde al WCET y T_i al plazo. Además, $W_i(t)$ expresa trabajo solicitado al procesador por una tarea τ_i en el instante t y se obtiene como,

$$W_i(t) = \lceil \frac{t}{T_i} \rceil \cdot C_i \tag{2.4}$$

es decir, es igual al número de activaciones hasta ese instante por el tiempo de cómputo que ha requerido cada una de ellas.

Suponiendo que el procesador ha ejecutando código ininterrumpidamente desde el instante $t = 0$, el trabajo efectivo realizado por el procesador, $W_e(t)$, es la cantidad de cómputo (en ciclos) que el procesador ha ejecutado y se representa con una recta de la forma,

$$W_e(t) = \alpha \cdot t, \qquad 0 \leq t \leq \infty \tag{2.5}$$

donde α es la velocidad a la que el procesador ha realizado ese cómputo, lo que es equivalente a la frecuencia de operación del procesador.

Cuando $W(t) = W_e(t)$, el procesador ha completado todo el trabajo que se le ha solicitado, y por lo tanto, quedará ocioso. Si el valor de t que hace cierta la igualdad es menor que los plazos de respuesta de las tareas, se puede concluir que éstas cumplen con las restricciones tiempo real establecidas.

El factor de utilización del procesador, U, para un conjunto de tareas τ, se define como:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i \alpha} \tag{2.6}$$

donde α representa la frecuencia a la que operan los procesadores. Si $U \leq 1$, entonces el conjunto de tareas podría ser planificado en un único procesador (con un algoritmo adecuado), por el contrario, si $U = m$, con $m > 1$, serán necesarios al menos m procesadores.

2.2. Planificación

Un problema de planificación consiste en la especificación de un conjunto de tareas y una plataforma de cómputo sobre la cual éstas deben ejecutarse.

Una planificación es una forma particular de ejecutar el conjunto de tareas especificado sobre la plataforma de cómputo dada.

2.2.1. Factibilidad

Un problema de planificación es factible cuando existe algún algoritmo capaz de encontrar una planificación de esas tareas sobre esa plataforma de cómputo en la que todas ellas cumplen sus plazos de respuesta.

2.2.2. Planificabilidad

Un problema de planificación es planificable por un algoritmo en particular si éste es capaz de encontrar una planificación de esas tareas sobre esa plataforma de cómputo en la que todas ellas cumplen sus plazos de respuesta.

2.2.3. Punto de planificación

Un punto de planificación es un instante en la vida del sistema en el que el algoritmo de planificación es sensible de cambiar las tareas que se están ejecutando sobre la plataforma de cómputo.

Esta intervención del algoritmo de planificación es la consecuencia de un evento interno o externo al sistema. Aunque suele desembocar en ello, la ocurrencia de un punto de planificación no tiene por qué conllevar el cambio de las tareas en ejecución.

2.2.4. Expulsión

Una expulsión es la intervención del algoritmo de planificación para que la tarea que actualmente ocupa el procesador deje de ejecutarse en él.

2.2.5. Migración

Una migración es la reanudación de una tarea en un procesador distinto a aquel del cual fue expulsada, e implica necesariamente la ocurrencia previa de una expulsión, porque la tarea ha tenido que dejarse de ejecutar en un procesador para ser reanudada en otro. Sin embargo, una expulsión no siempre conlleva una migración, ya que una tarea puede reanudarse más adelante en el mismo procesador al que fue expulsada.

En el esquema de planificación global de este trabajo se considera como migración únicamente la expulsión-reanudación de un mismo trabajo en más de un procesador. Es decir, si distintos trabajos de la misma tarea se ejecutan en procesadores diferentes no se cuenta como migración.

2.2.6. Afinidad

La ejecución de un trabajo es afín a un procesador cuando dicho trabajo se ejecuta únicamente en él. El trabajo puede ser expulsado pero mantendrá su afinidad siempre y cuando se reanude siempre en el mismo procesador. La afinidad, por lo tanto, es la ausencia de migraciones en la ejecución de un trabajo.

2.2.7. Planificación expulsiva

Una planificación expulsiva es aquella en la que el algoritmo de planificación puede expulsar tareas de los procesadores durante su ejecución.

2.2.8. Planificación no expulsiva

Una planificación no expulsiva es aquella en la que el algoritmo de planificación no puede expulsar las tareas en ejecución de los procesadores. Es decir, en una planificación no expulsiva los trabajos generados por las tareas se ejecutan ininterrumpidamente en el procesador al que fueron asignados, sin posibilidad de ser expulsados, ni por lo tanto migrados a otro procesador.

La consideración de esta restricción convierte al problema de planificación en NP-completo. Es decir, existen reducciones en tiempo polinomial de problemas que se sabe que son intratables a este, siendo la de Bin-packing problem (BPP) una de las más sencillas.

2.2.9. Planificación fuera de línea

Un algoritmo de planificación fuera de línea es aquel que dispone de todas las decisiones de planificación de las tareas sobre la plataforma de cómputo antes de la ejecución del sistema.

Dado que los cálculos mediante los cuales se toman las decisiones de planificación son realizados antes de la ejecución del sistema. La complejidad computacional que supone la obtención de la planificación no es tan crítica como cuando dichas decisiones se toman con el sistema en marcha. Los algoritmos de planificación fuera de línea suponen un sobrecoste muy bajo con respecto al tiempo de ejecución del sistema.

2.2.10. Planificación en línea

Un algoritmo de planificación en línea es aquel que realiza las decisiones de planificación de las tareas sobre la plataforma de cómputo durante la ejecución del sistema.

Dado que los cálculos de toma de decisión para la planificación son realizados durante la ejecución del sistema, la complejidad computacional de los algoritmos es muy relevante. Los algoritmos de planificación en línea pueden suponer un sobrecoste en el tiempo de ejecución del sistema que debe ser tomado en cuenta.

2.2.11. Planificación particionada

La planificación particionada asigna un subconjunto disjunto de tareas a cada núcleo del multiprocesador.

La principal ventaja que presenta este esquema es la garantía de afinidad de los trabajos de cada tarea con respecto al núcleo que se le ha asignado.

Su principal inconveniente es la disminución de la utilización total del multiprocesador como consecuencia de las condiciones de planificabilidad de los algoritmos de planificación sobre monoprocesadores.

2.2.12. Planificación global

La planificación global permite que cualquiera de las tareas pueda ser ejecutada en cualquiera de los núcleos del multiprocesador.

La principal ventaja que presenta este esquema es la consideración de todos los núcleos a la hora de realizar la planificación, lo cual posibilita que los algoritmos de planificación puedan conseguir la utilización total del multiprocesador.

El principal inconveniente consiste en la degradación de las prestaciones y la no predecibilidad de los multiprocesadores debido a la interferencia existente entre los núcleos cuando se permite la migración de tareas entre ellos.

2.3. Ejecutivo cíclico

Un Ejecutivo Cíclico (EC) es un modelo de planificación que entrelaza explícitamente la ejecución de las tareas periódicas sobre los procesadores disponibles de manera periódica.

Ciclo mayor

El ciclo mayor o hiperperiodo es la separación temporal entre repeticiones consecutivas de la planificación cíclica. Se define como el mínimo común múltiplo de los periodos de los tareas.

Ciclo menor

El ciclo menor o marco (*frame*) es cada uno de los fragmentos de tiempo equidistantes con duración máxima en las que puede dividirse de manera exacta el hiperperiodo. En cada uno de ellos se lleva a cabo una planificación secundaria, y la composición de todas ellas da como resultado la planificación cíclica principal. Se define como el máximo común divisor de los periodos de las tareas.

2.4. Algoritmos de planificación comparados

2.4.1. Allocation and Execution Control Scheduler (AlECS)

Allocation and Execution Control Scheduler (AlECS) es un modelo de planificación sobre multiprocesadores para la construcción de ejecutivos cíclicos. Utiliza un Problema

de Programación Lineal Entera (PPLE) para formular las restricciones con las que se garantiza el cumplimiento de los plazos de respuesta de las tareas y optimizar la utilización del multiprocesador. Gracias a que la matriz de restricciones del PPLE es totalmente unimodular, el problema se resuelve como un Problema de Programación Lineal. Después, aplica un algoritmo de *Zero-laxity* para obtener una asignación de las tareas a los procesadores a partir de la solución obtenida del PPL. Con estos dos pasos es posible obtener un EC con ciclo menor variable.

2.4.2. Clustered Allocation and Execution Control Scheduler (CAlECS)

Clustered Allocation and Execution Control Scheduler (CAlECS) es una mejora de AlECS que realiza un agrupamiento de las tareas con respecto a los procesadores previo al planteamiento del Problema de Programación Lineal. Con ello consigue reducir el número de migraciones y el propio espacio de soluciones del problema de optimización.

2.4.3. Reduction to Uniprocessor (RUN)

Reduction to Uniprocessor (RUN) [3] es un algoritmo de planificación sobre multiprocesadores que transforma el problema inicial a una jerarquía arborescente de problemas de planificación sobre monoprocesadores, cuya solución se compone para obtener la asignación de las tareas sobre la plataforma de cómputo original.

2.4.4. Deutschbein Cyclic Executive (DCE)

Deutschbein Cyclic Executive (DCE) es el término que utilizamos en este TFG para denominar el modelo de planificación sobre multiprocesadores para la construcción de ejecutivos cíclicos introducido en [1]. Utiliza un Problema de Programación Lineal para garantizar el cumplimiento de los plazos de respuesta. Dependiendo de la especificación del dominio de las variables del PPL es posible obtener tanto planificaciones expulsivas como no expulsivas. El artículo de referencia aplica la regla de envoltura *McNaughton* a la solución del PPL en el caso expulsivo, para obtener una asignación de las tareas en los procesadores que no paralelice la ejecución de los trabajos.

Tras implementar los métodos de [1] y estudiar los resultados, en este TFG se consideró la aplicación de la regla de envoltura *McNaughton* a la solución del PPL también en el caso no expulsivo, mejorando los resultados.

Todos los detalles del algoritmo se tratan en el Cap. 3.

2.5. Tertimuss

Tertimuss es un entorno de simulación tiempo real sobre multiprocesadores que permite considerar también las restricciones térmicas presentes en las plataformas de cómputo.

Integra un buen número de algoritmos de planificación TR ya disponibles para ser utilizados en las simulaciones, entre ellos *AlECS*, *CAlECS* y *RUN*. En este TFG se ha implementado e integrado también DCE, con las características que se describen en esta memoria.

Además, Tertimuss proporciona herramientas para el análisis de los resultados obtenidos.

2.6. Programación lineal

La especificación de un PPL consiste en:

- Un conjunto de v variables, cuyo valor se busca determinar.

- Una función objetivo, que es una evaluación sobre las variables planteada como una expresión lineal, cuyo valor se busca optimizar (maximizar o minimizar, según la naturaleza del problema).

- Una colección de restricciones lineales, expresadas como inecuaciones lineales sobre el conjunto de las v variables. Adicionalmente, pueden expresarse restricciones sobre el dominio de las variables del problema, que son llamadas restricciones de dominio.

Cada restricción lineal (incluyendo las de dominio) define un hiperplano en el espacio v-dimensional de posibles asignaciones de las variables. El subespacio delimitado por todos los hiperplanos se llama región factible, y sus puntos suponen las asignaciones a las variables que satisfacen todas las restricciones del Problema de Programación Lineal.

La región factible, por cómo está definida, es un poliedro convexo en $\mathbb{R}^v$. Así, debido a sus propiedades geométricas, se tiene que el valor óptimo de la función objetivo se da en uno de los vértices de esa región convexa.

La solución al problema de programación lineal es la asignación de valores de las variables que supone la optimización de la función objetivo.

2.6.1. Algoritmo SIMPLEX

SIMPLEX es un algoritmo voraz de optimización que recorre vértices adyacentes del poliedro convexo que define la región factible hasta que ningún vecino del vértice considerado lo supera en la evaluación de la función objetivo. Su complejidad temporal en el caso peor es exponencial, pero polinomial en el caso promedio.

Los solucionadores *GLOP* y *Gurobi* disponen de implementaciones de este algoritmo de optimización.

2.6.2. Método de punto interior (*inner-point method*)

Se conoce como métodos de punto-interior o de barrera (*interior-point methods* o *barrier methods*) a una familia de algoritmos de optimización que utilizan una función lineal logarítmica de barrera, obtenida a partir de la definición del PPL, para explorar la región factible mediante técnicas de descenso de gradiente. Su complejidad temporal en el caso peor es cúbica.

El solucionador *Gurobi* dispone de una implementación de esta técnica de optimización.

Capítulo 3

Algoritmo Deutschbein Cyclic Executive (DCE)

En este capítulo se introduce el planificador descrito en [1], en adelante denominado Deutschbein Cyclic Executive (DCE). En él se propone una planificación global basada en ejecutivo cíclico para multiprocesadores, considerando tareas periódicas con plazo de respuesta implícito. El método utiliza un problema de programación lineal para resolver la asignación de tareas a procesadores, y es capaz de obtener ejecutivos cíclicos expulsivos y no expulsivos.

La planificación global que se propone en el artículo busca erigirse como alternativa a las soluciones con esquemas particionados que actualmente se aplican en la industria.

Entre los objetivos de los autores cabe destacar la predecibilidad y bajo sobrecoste en tiempo de ejecución de la planificación, con el fin de obtener una solución sobre multiprocesadores capaz de ajustarse a los requisitos de las certificaciones en la industria. Con el método de planificación propuesto ambos son satisfechos, pues se trata de una solución determinista y estática en la que todo el trabajo se realiza en la fase fuera de línea.

Cabe destacar que se considera el número de trabajos (*jobs*) para caracterizar el tamaño del ejecutivo cíclico a construir, y por ende, el orden de complejidad computacional de los algoritmos.

En el Anexo D se incluyen los diagramas UML de implementación del planificador que se describe.

3.1. Consideraciones previas

En esta primera sección se van a comentar algunas consideraciones que deben ser tenidas en cuenta para poder entender el resto del capítulo y las posibles diferencias en sus explicaciones con respecto a lo que se describe en el artículo de referencia [1].

Unidades

Uno de los principales problemas detectados durante la implementación se ha debido a la discordancia entre las unidades manejadas en *Tertimuss* y las especificadas en el artículo de referencia.

Por una parte, se tiene que en *Tertimuss* están bien definidas las unidades en las que se mide cada uno de los parámetros del sistema: el tiempo en segundos, el cómputo en ciclos y la frecuencia en hercios. No obstante, en la descripción del algoritmo del artículo, no se concreta en ningún momento las unidades en las que se miden los valores utilizados. Interpretando literalmente las descripciones del artículo se puede llegar a incongruencias, resolubles mediante una definición estricta de las unidades de medida de cada una de las variables.

Por ello, de ahora en adelante se van a utilizar las unidades especificadas en *Tertimuss* para abordar el método descrito en el artículo para la obtención de un ejecutivo cíclico.

3.1.1. Notación

Se parte de un sistema de N tareas periódicas con plazo de respuesta implícito, $\tau = \{\tau_i = (C_i, T_i)\}_{i=1}^{N}$, que deben ser planificadas en un multiprocesador con m núcleos idénticos entre sí. Así, se tiene que:

- τ es el conjunto de las N tareas periódicas a planificar.
- τ_i es la tarea i, $i \in [1, N]$, cuyo periodo (y su deadline), es T_i, medido en segundos, y su $WCET$ es C_i, medido en ciclos.
- m el número de núcleos del multiprocesador sobre el cual deben planificarse las tareas.

P es el mínimo común múltiplo de los periodos de los tareas, y por ende, el valor del hiperperiodo del ejecutivo cíclico medido en segundos.

F es el máximo común divisor de los periodos de las tareas, y por lo tanto, la duración de cada uno de los marcos del ejecutivo cíclico medido en segundos.

Se tiene n como el número total de trabajos (activaciones de las N tareas) que se dan en un intervalo de tiempo igual al hiperperiodo, P. Esto es $n = \sum_{n=1}^{N} \frac{P}{T_i}$.

Se define como $J = \{j_i\}_{i=1}^{n}$ al conjunto de trabajos (*trabajos*) generados por el conjunto de tareas τ, que tienen su tiempo de llegada y su plazo de respuesta en el intervalo de tiempo $[0, P]$. Tanto el tiempo de llegada como el plazo de respuesta que se consideran son en tiempo absoluto y monotónico.

Cada trabajo $j_i, j_i \in J$, se representa con una 3-tupla, (a_i, c_i, d_i), de tal forma que:

- a_i es el tiempo de llegada del trabajo j_i, o lo que es lo mismo, el instante de activación (en segundos) de la tarea que lo genera.
- c_i es el WCET de la tarea que genera el trabajo, medido en ciclos.
- d_i es el plazo de respuesta del trabajo, medido en segundos.
- El intervalo temporal $[a_i, d_i)$ se denota como ventana de planificación del trabajo j_i, y para que una planificación sea correcta éste deberá recibir al menos c_i unidades temporales de ejecución durante ella (pues es el tiempo de ejecución del trabajo y en caso de no llegar a hacerse perdería su plazo).

Adicionalmente, $\phi_k, k \in [1, \frac{P}{F}]$ representa el número de marco del ejecutivo cíclico que se considera. Así, se tiene que:

- P es divisible por F, y $\frac{P}{F}$ es el número de marcos en los que se divide el hiperperiodo.
- Cada marco $\phi_k, k \in [1, \frac{P}{F}]$, abarca el intervalo de tiempo $[F \cdot (k-1), F \cdot k]$.

3.2. Definición del problema de programación lineal

En esta sección de va a abordar el PPL utilizado por DCE para modelar su solución basada en ejecutivo cíclico.

3.2.1. Variables

El problema planteado en el artículo define $N \cdot m \cdot \frac{P}{F}$ variables x_{ijk}. Cada una de ellas representa la fracción del trabajo i (es decir, el trabajo j_i) que se planifica sobre el *j-ésimo* procesador durante el *k-ésimo* marco del ejecutivo. Recuérdese que:

- El índice i toma valores en el intervalo $[1, n]$, siendo n el número total de trabajos generados por las N tareas periódicas durante un hiperperiodo.
- El índice j toma valores en el intervalo $[1, m]$, siendo m el número de procesadores del sistema.
- El índice k solo toma valores en el rango de números de marco correspondientes al intervalo temporal $[a_i, d_i)$. Esto se debe a que el trabajo i solo puede ejecutarse en los marcos que están dentro de su ventana de planificación, es decir, los marcos en el intervalo temporal $[a_i, d_i)$.

Cada variable x_{ijk} está sujeta a una restricción de no-negatividad en su dominio, es decir, $x_{ijk} \geq 0$. No obstante, al representar cada variable una fracción de un trabajo, su valor, además de ser no-negativo, deberá ser menor o igual que 1, ya que la máxima fracción de cualquier trabajo que debe poder ejecutarse es el trabajo entero. Por lo tanto, se tendrá que $0 \leq x_{ijk} \leq 1$, aunque esta restricción adicional sobre el dominio de las variables x_{ijk} se modelará mediante la restricción 1 del PPL.

Por otra parte, se define una variable f como la cantidad de cómputo, en ciclos, que debe ejecutar cada procesador durante un intervalo de tiempo de duración F. En otras palabras, f representa el trabajo efectivo que debe poder realizar cada procesador (ec. 2.5), $W_e(t)$, evaluado en $t = F$.

3.2.2. Función objetivo

En el Problema de Programación Lineal que se plantea se busca minimizar la capacidad de cómputo que los procesadores deben tener para que sobre ellos pueda construirse un ejecutivo cíclico factible. Esto se modela mediante la minimización del valor de la variable f:

$$\text{mín } f \tag{3.1}$$

Equivale a determinar la velocidad mínima que necesitan tener los procesadores para garantizar que su trabajo efectivo en cada marco del ejecutivo es suficiente como para sacar adelante el trabajo que se les solicita en ese mismo intervalo de tiempo.

3.2.3. Restricciones

Además de las $N \cdot m \cdot \frac{P}{F}$ restricciones de no-negatividad definidas para cada una de las variables, el planteamiento del PPL requiere que se formulen $n+(m+N)\cdot\frac{P}{F}$ restricciones lineales adicionales para modelar aquellos aspectos necesarios para garantizar la factibilidad del ejecutivo cíclico obtenido a partir de su solución.

Primera restricción

Esta restricción asegura que cada trabajo complete su tiempo de ejecución, comprobando que la suma de las fracciones del mismo x_{ijk} suman la unidad.

Hay n restricciones de este tipo, una por cada trabajo. Se especifican mediante la ecuación 3.2.

$$\sum_{j=1}^{m} \sum_{k=\frac{a_i}{F}+1}^{\frac{d_i}{F}} x_{ijk} = 1, \text{ para cada i, } 1 \leq i \leq n \tag{3.2}$$

La variable k toma su valor en el intervalo $[\frac{a_i}{F} + 1, \frac{d_i}{F}]$, por lo que se restringe a aquellos marcos correspondientes a la ventana de planificación del trabajo i en cuestión.

Esta restricción sirve para acotar superiormente el valor de las variables x_{ijk}, pues de esta forma a la restricción de no negatividad se le suma la semántica de *fracción del trabajo a ejecutar* que limita su valor a $0 \leq x_{ijk} \leq 1$.

Segunda restricción

Representa el requisito de que la carga de trabajo de cada procesador no supere las f unidades de cómputo durante cada ciclo menor. Es decir, esta trata de asegurar que en cada intervalo de tiempo $[0, F)$ a cada procesador j no se le han asignado más unidades de cómputo de las f que puede procesar.

Cada instancia de esta restricción se entiende como la relación entre el trabajo asignado a un procesador durante un marco y el trabajo efectivo que cada uno de ellos debe poder realizar.

Se define $W_{jk}(F)$ como el trabajo asignado a un procesador j durante un marco k. Así, en el contexto del ejecutivo, se trata de un valor constante, pues no guarda relación con las activaciones de las tareas, sino que viene determinado directamente por las asignaciones x_{ijk} que se hayan hecho como planificación para ese marco k y ese procesador j. Este valor se obtiene con la ecuación 3.3, que acumula el número de ciclos asignados a los trabajos activos durante el marco k en el procesador j con la asignación que supone el valor de las variables x_{ijk}.

$$W_{jk}(t) = \sum_{i=1}^{n} x_{ijk} \cdot c_i \tag{3.3}$$

Habrá que asegurar que cada uno de los procesadores, en cada ciclo menor k de duración F, haya sido capaz de ejecutar el trabajo que se le ha asignado. La forma de hacerlo es garantizar que el trabajo realizado por cada procesador en ese intervalo de tiempo, $W_e(F)$, haya sido suficiente como para igualar o superar la carga de trabajo que se le ha asignado, $W_{jk}(F)$.

Eso lleva a que la ecuación $W(t) = W_e(t)$ para la garantía del cumplimiento de plazos se formule en este caso particular como $W_{jk}(F) = W_e(F)$. Sustituyendo $W_{jk}(F)$ por la definición que se ha dado en la ec. 3.3 y $W_e(F)$ por la expresión de trabajo efectivo realizado por el procesador en el instante F, se obtiene la ecuación 3.4.

$$\sum_{i=1}^{n} x_{ijk} \cdot c_i = \alpha \cdot F \tag{3.4}$$

Si se evalúa el cumplimiento del cómputo requerido en función de los posibles valores de α (Fig. 3.1) se tiene que el procesador es capaz de realizar todo el trabajo que se le ha solicitado en ese intervalo de tiempo solo cuando $W_{jk}(F) \leq W_e(F)$.

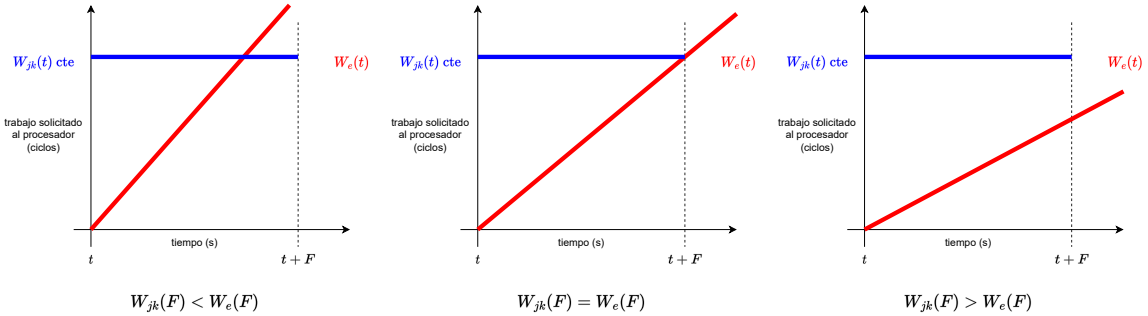


Figura 3.1: Posibles situaciones en función del valor de α en la ecuación descrita 3.4. En (a) y (b) se tiene que $W_{jk}(F) \leq W_e(F)$ y por lo tanto el trabajo solicitado es atendido por el procesador en un intervalo de tiempo de duración F. En (c) se tiene que $W_{jk}(F) > W_e(F)$, y por lo tanto el procesador no es capaz de sacar a delante la carga de trabajo que se le ha solicitado

Con base en los casos descritos en la Fig. 3.1, se concluye que el procesador solo puede efectuar su asignación en un marco cuando $W_{jk}(F) \leq W_e(F)$, o lo que es lo mismo, que debe satisfacerse la inecuación $\sum_{i=1}^{n} x_{ijk} \cdot c_i \leq \alpha \cdot F$ para cada procesador en cada marco del ejecutivo cíclico. Y se escribe la segunda restricción de la siguiente forma,

$$\sum_{i \in trabajos\_en\_marco_k} x_{ijk} \cdot c_i \leq f, \text{ para cada j, } 1 \leq j \leq m, \text{ y cada k, } 1 \leq k \leq \frac{P}{F} \qquad (3.5)$$

donde $f = \alpha \cdot F$. Además, la frecuencia mínima poder ejecutar el ejecutivo cíclico será α. Hay $m \cdot (\frac{P}{F})$ restricciones de este tipo. El valor de f obtenido de la resolución del Problema de Programación Lineal garantizará que se cumplan los plazos de respuesta al satisfacer el resto de restricciones que se plantean.

La variable f se relaciona con una desigualdad ($\leq$) porque habrá procesadores en los que el valor de α establecido estará sobredimensionado durante algún marco, resultando en un desaprovechamiento de recursos. El valor de f debe asegurar que todos los procesadores pueden sacar adelante la carga de trabajo que se les ha asignado durante ese mismo intervalo de tiempo, por lo que será tal que permita realizar la carga de trabajo correspondiente al procesador más ocupado de entre todos los marcos del ejecutivo. Esto resulta en que, al minimizar el valor de la variable f, se busquen aquellas fracciones x_{ijk} que repartan de manera homogénea la carga de trabajo entre los procesadores durante los marcos.

En cuanto al planteamiento de la restricción, cabe destacar que $trabajos\_en\_marco_k$ es el conjunto de trabajos en cuya ventana de planificación está contenido el marco k, y se define como $trabajos\_en\_marco_k = \{i | a_i \leq F \cdot (k-1) \wedge F \cdot k \leq d_i\}$, teniéndose que $\phi_k \in [a_i, d_i) \Rightarrow [F \cdot (k-1), F \cdot k) \subseteq [a_i, d_i)$. Puede verse también como el conjunto

de trabajos cuya ventana de planificación se solapa con el intervalo de tiempo que supone el marco k. En cuanto a su implementación, conviene precalcular ese vector de conjuntos dado que iterar los n identificadores de los trabajos y comprobar para cada uno de ellos si se cumple o no la condición es muy costoso.

Tercera restricción

Representa el requisito de que cada trabajo no tenga asignadas más de f unidades de cómputo durante cada ciclo menor. Es decir, se trata de asegurar que cada procesador es capaz de ejecutar todo el tiempo de cómputo asignado a ese trabajo en un solo marco. Visto de otro modo, impide que la planificación paralelice la ejecución del trabajo en ese marco.

Hay $N \cdot (\frac{P}{F})$ restricciones de este tipo sobre cada procesador (i), una por cada posible combinación de número de tarea (j) y número de marco (k).

$$\sum_{j=1}^{m} x_{ijk} \cdot c_i \leq f, \text{ para cada i, } 1 \leq i \leq n, \text{ y cada k, } \frac{a_i}{F} + 1 \leq k \leq \frac{d_i}{F} \qquad (3.6)$$

El motivo por el cual se considera la asignación entre todos los procesadores es que f, en unidades de cómputo, es la máxima cantidad de tiempo de cómputo que una misma tarea puede ser procesada de manera secuencial durante un marco del ejecutivo (de duración F).

Considérese el *principio del palomar* [18] (Fig. 3.2). Se dispone de f almacenes en cada marco para ejecutar una misma tarea de manera secuencial entre todos los procesadores. La tarea podrá ser ejecutada de forma secuencial siempre que se le asignen $z \leq f$ unidades de cómputo ese marco. Sin embargo, si se le asignan $z > f$ unidades de cómputo, habrá $z - f$ unidades de cómputo que deberán ir a parar a huecos ya ocupados (solaparse con algunas de las f unidades de cómputo ya asignadas) al tratar de acomodar los z ciclos en los f almacenes. En el contexto que nos atañe, significa que al menos $z - f$ unidades de cómputo deberían ejecutarse paralelamente con algunas de las f que que sí son ejecutables secuencialmente.

Además, si a un trabajo i se le asignara un tiempo de cómputo superior a las f unidades de cómputo que cada procesador puede ejecutar en un marco (esto es, que $x_{ijk} \cdot c_i > f$), entonces esa tarea en ese marco tendría que ser asignada a un mínimo de dos procesadores distintos siguiendo la regla de *McNaughton*(Fig. 3.3), a fin de satisfacer la carga de trabajo que supone dicha asignación (es decir, $x_{ijk} \cdot c_i$). Un solo procesador no tendría la potencia suficiente para ello, al ser su capacidad de f unidades de cómputo cada F unidades de tiempo.

Por lo tanto, como se muestra en la Fig. 3.3, existirían momentos en ese marco en

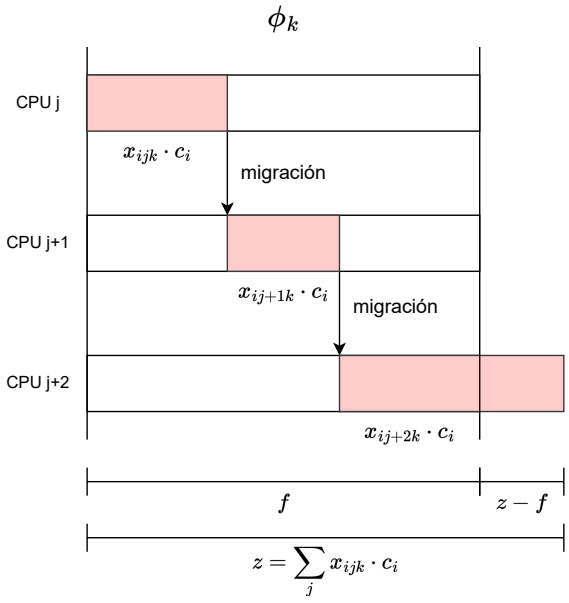


Figura 3.2: Secuenciación en la ejecución de un trabajo a lo largo de un marco del ejecutivo

los que esa misma tarea estaría siendo ejecutada en al menos dos procesadores distintos, o lo que es lo mismo, estaría siendo paralelizada la ejecución de esa tarea, y eso no se permite.

Recapitulación

Algunas consideraciones relevantes en lo que respecta a la implementación de las restricciones son las siguientes:

- Con respecto la primera y la segunda restricción: A partir del identificador de trabajo i se pueden saber los números de marco k en los cuáles está su ventana de planificación (caso de la primera restricción). Por el contrario, a partir de un número de marco k no se pueden saber los identificadores de trabajo i cuya ventana de planificación se solapa con él (caso de la segunda restricción), salvo recorriendo todos los trabajos para cada marco. De ahí el motivo de recomendar en la segunda restricción el precalcálculo de ese vector de conjuntos, para evitar realizar esos recorridos de manera repetida.

- Con respecto a la segunda y la tercera restricción: Cualquier valor de f mayor al obtenido del PPL satisfará también ambas restricciones:

  - En el caso de la segunda restricción, porque supone asignar más unidades de cómputo por marco a la capacidad que se les exige a los procesadores, lo cual

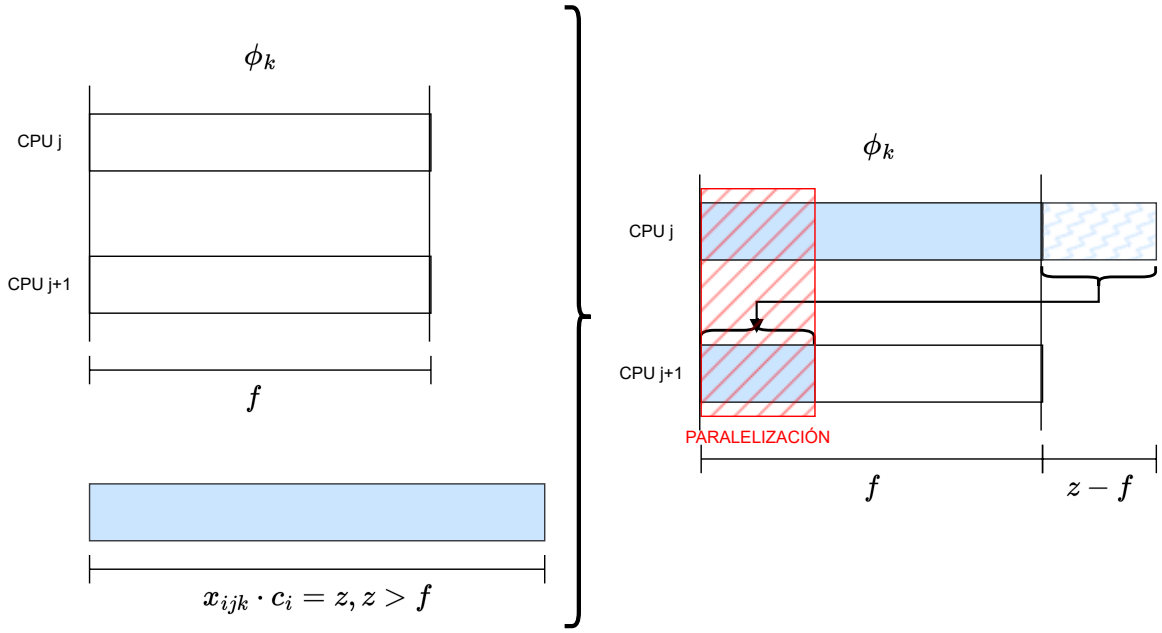


Figura 3.3: Aplicación de la regla de *McNaughton* que da como resultado la paralelización de la ejecución de un trabajo al no cumplirse la tercera restricción

aumenta el requisito de frecuencia, pero asegura igualmente la realización del trabajo solicitado por los trabajos asignados durante ese marco.

- En el caso de la tercera restricción, si se parte de que cada trabajo debe ejecutarse un tiempo menor o igual que f unidades de cómputo, entonces también se mantendrá esa misma relación al condiderar un valor mayor que f.

3.2.4. Cambios en la definición del PPL en el caso no expulsivo

En esta sección se detallan las modificaciones necesarias en la definición del PPL para obtener una asignación de las variables que permita la construcción de ejecutivos cíclicos no expulsivos.

Partiendo del PPL que se ha construido en esta sección, si se consideran soluciones enteras 0/1 en lugar de reales para las variables x_{ijk}, el hecho de que su valor solo pueda ser 0 o 1, atendiendo a la semántica con la que éstas se han definido, supone que cada trabajo i solo pueda ejecutarse entero o no ejecutarse en un procesador j dentro de un marco k. Por lo tanto, simplemente con esa modificación, se consigue que la planificación resultante sea no expulsiva.

Al introducir esa restricción al dominio de las variables, se pasa de un Problema de Programación Lineal a un Problema de Programación Lineal Mixta. El motivo de que sea un Problema de Programación Lineal Mixta y no un Problema de Programación Lineal Entera reside en el hecho de que la variable f sigue siendo real, aunque por la

semántica que se sigue a continuación se verá como siempre acaba tomando valores exactos.

Esta transformación conlleva que la restricción 1 hace que la restricción 3 sea redundante. Para ver por qué, considérese un trabajo (por ejemplo, j_{i_o}) y un marco (por ejemplo, ϕ_{k_o}) cualesquiera. Por la restricción 1 y el hecho de que cada variable x_{ijk} tiene asignado un valor 0 o 1, se sigue que para cualquier solución entera al problema de programación lineal se tendrá que $(\sum_j x_{i_o jk_o} = 0) \vee (\sum_j x_{i_o jk_o} = 1)$, dependiendo de si el trabajo j_{i_o} está planificado (en cualquier procesador) dentro del marco ϕ_{k_o} o no. Por lo tanto, vemos que como máximo uno de los $x_{i_o jk_o}$ puede ser igual a 1, de lo cual se sigue que la restricción 3 necesariamente se cumple para el trabajo j_{i_o} dentro del marco ϕ_{k_o}. Es por ello que la restricción 3 puede omitirse del problema de PPLM. Así, para planificadores no expulsivos, se tiene un problema de optimización más simple a resolver, constituido por las mismas $(N \cdot m \cdot \frac{P}{F})$ variables pero con solo $(n + m \cdot \frac{P}{F})$ restricciones.

Por otra parte, la transformación conlleva que, para que exista un ejecutivo cíclico no expulsivo, sea necesario que cada trabajo quepa dentro de un marco individual, es decir, se debe cumplir la Ec. 3.7 y cualquier sistema de tareas para el cual no se cumpla esa condición no podrá ser ser planificado de forma no expulsiva.

$$max_{i=1}^{N}(C_i) \leq f \tag{3.7}$$

Ese requisito, si bien es muy duro, está motivado porque las comprobaciones se limitan a un solo marco en el planteamiento del PPL. Es decir, tal y como está planteado el problema de optimización, no se puede asegurar una planificación no expulsiva cuando el valor C_i de una tarea (esto es, su WCET) tiene una duración mayor que las f u.c. que dura el ciclo menor. Es así porque no tienen forma de asignar el mismo trabajo dos marcos seguidos en el mismo procesador para conseguir que la planificación sea no expulsiva. Esto supone que asignaciones no expulsivas como la que se ilustra en la Fig. 3.4 no sean consideradas para la planificación.

Por ello el valor de α (frecuencia mínima requerida) se dispara con este tipo de soluciones: la planificación se reduce a asignar todo el tiempo de cómputo de un trabajo en un único procesador durante un mismo marco. Es decir, es no expulsivo a costa de que el valor de f sea mayor o igual al máximo valor de WCET del conjunto de tareas, provocando que $\alpha = \frac{f}{F}$ se dispare.

Conviene resaltar la relación existente entre el requisito planteado mediante la Ec. 3.7 y la restricción dos del PPL 3.5, pues de hecho, se genera como consecuencia de ésta. Debido al mantenimiento de la restricción uno 3.2 del PPL y al hecho de que cada variable x_{ijk} solo pueda tomar valores 0 o 1, se tiene que existirá una instancia

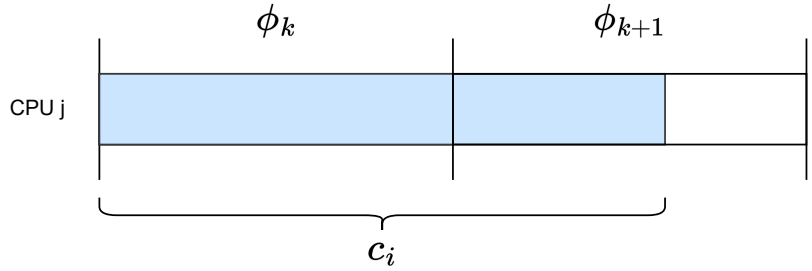


Figura 3.4: Asignación no expulsiva de un mismo trabajo a lo largo de dos marcos del ejecutivo cíclico

de la restricción dos tal que $x_{ij_ok_o} \cdot c_i \leq f$ en la que $x_{ijk} = 1$, y por lo tanto se tendrá especificado que $c_i \leq f$. Además, j_i puede no ser el único trabajo asignado al procesador j_o durante el marco k_o por la forma en la que está definida la restricción dos, lo que lleva a que el trabajo asignado a ese procesador en ese ciclo menor conlleve la ejecución de más de un trabajo. En la Sec. 3.5.2 se tratará la forma de proceder para obtener una planificación en esos casos.

Recapitulación

- La restricción uno hace la tres redundante, pero la tres no hace redundante la uno. Por eso se mantienen los restricciones uno y dos en el PPLM, pero se quita la tres.

- Dado que las comprobaciones se limitan a un solo marco, la forma de asegurar que la planificación es no expulsiva consiste en hacer que cada trabajo se ejecute entero en un mismo procesador durante un solo marco. Eso lleva a un aumento considerable en el valor de la variable f y a un desaprovechamiento de los recursos de cómputo.

- El requisito $max_{i=1}^{N}(C_i) \leq f$ se desprende de la restricción dos del PPL cuando las variables x_{ijk} solo pueden tomar valores en el conjunto $\{0, 1\}$. No obstante, la restricción dos además permite que más de un trabajo pueda ejecutarse entero en un mismo procesador durante el mismo marco, lo cual puede llevar a una sobredimensión todavía mayor del valor de f.

3.2.5. Solucionadores de problemas de optimización

En el código implementado se ha buscado la máxima transparencia posible en cuanto al solucionador de problemas de optimización que se utiliza en el planificador para resolver los PPL. Esto, en parte, ha sido posible gracias a la interfaz de modelado

de problemas de optimización que proporciona *ortools* [19], que permite resolver PPL con diferentes solucionadores realizando la definición de la misma manera para todos ellos [20].

Tertimuss utiliza *GLOP* [21] para resolver los PPL, que es solucionador de problemas de optimización de código abierto implementado por *Google* como parte del entorno de trabajo *ortools* [22].

No obstante, en el artículo de referencia se utiliza *Gurobi* [23] para resolver los Problema de Programación Lineal. A diferencia de *GLOP*, *Gurobi* es un solucionador de código cerrado sujeto a una estricta licencia de uso [24] que impide su libre distribución. Existe un paquete de *Python* que permite utilizar una versión limitada de *Gurobi* [25], pero éste fuerza a que se utilice su interfaz de modelado para definir los problemas de optimización.

Debido al complejo proceso de instalación y validación de la licencia (descrito en el Anexo F), se ha decidido utilizar *GLOP* por defecto sin posibilidad de especificar al crear el planificador qué solucionador debe utilizar. Es necesario modificar un atributo en el código de la clase para forzar a *Tertimuss* a que busque a través de *ortools* una instalación de *Gurobi* y la utilice para resolver los PPL.

3.3. Discretización de los resultados del PPL

En esta sección se parte de haber resuelto el PPL, y por lo tanto de haber obtenido una asignación de valores no negativos para las variables x_{ijk} que satisface todas las restricciones.

Para cada trabajo j_{i_o} se tiene que $x_{i_o} \stackrel{def}{=} \sum_{j=1}^{m} x_{i_o j k_o}$, siendo $x_{i_o j k_o}$ el valor obtenido al resolver el PPL para la variable x_{ijk}, fijando i como i_o y k como k_o.

x_{i_o} representa la cantidad total de ejecución asignada al trabajo j_{i_o} durante el marco ϕ_{k_o} en la solución del PPL. Esto es, x_{i_o} representa la fracción total del trabajo j_{i_o} que debe ejecutarse durante el marco ϕ_{k_o} según la asignación entre todos los procesadores.

Además, para los valores x_{i_o} se cumple que:

– Por la tercera restricción del PPL (la que asegura que la ejecución de un trabajo no se paraleliza), se sigue la ecuación 3.8, habiendo n inecuaciones como esta (una por cada trabajo) en cada marco. Por lo tanto, ningún trabajo tiene asignadas más de f unidades de cómputo en un mismo marco (entre todos los procesadores).

$$(c_{i_o} \cdot x_{i_o}) \leq f, \text{ para cada trabajo } j_{i_o} \tag{3.8}$$

Literalmente se trata de la tercera restricción fijando el valor del marco a k_o (esto es, $k = k_o$ en la inecuación de la restricción tres 3.6). Con esto asegura que ningún

trabajo tiene asignado durante el marco más que la capacidad de cómputo de un solo procesador durante ese intervalo de tiempo.

– Por la segunda restricción del PPL (la que asegura que cada procesador pueda sacar adelante el trabajo asignado en cada marco), sobre los m procesadores, es decir, para todos los valores que toma la variable j en la restricción dos, se tiene la ecuación 3.9, habiendo una inecuación como esta por cada marco.

$$(\sum_{i_o \in trabajos\_en\_marco_{k_o}} x_{i_o} \cdot c_{i_o}) \leq m \cdot f \tag{3.9}$$

Literalmente se trata de la segunda restricción fijando el valor del marco a k_o (esto es, $k = k_o$ en la inecuación de la restricción dos 3.5) y considerando en cada instancia la suma para todos los procesadores en vez de cada procesador en particular, es decir, lo que se tiene es la suma cada m instancias de la restricción dos que comparten el mismo valor de k. Con esto asegura que la cantidad total de ejecución planificada durante ese intervalo de tiempo no excede la cantidad acumulativa de cómputo durante el marco de los m procesadores,

3.3.1. Necesidad de la discretización de los resultados

A partir de la solución planteada, se tiene que tanto f como $x_{i_o} \cdot c_{i_o}$ deben ser valores naturales al representar unidades de cómputo, pero al ser f y los x_{ijk} definidos en el PPL como variables reales, los valores que se obtienen como solución al PPL son decimales,y por lo tanto no es posible aplicarlos directamente para generar una planificación.

Dado que tanto el valor decimal de f como el de cada $x_{i_o} \cdot c_{i_o}$ garantizan la satisfacción de las restricciones, es importante asegurar que, tras los cambios que se apliquen, los valores lo naturales resultantes lo sigan haciendo.

En el caso de $x_{i_o} \cdot c_{i_o}$, como el valor decimal supone la garantía de la ejecución total del trabajo, el resultado de la discretización deberá asegurar una asignación mayor o igual que ese valor decimal, pues de otra forma, el trabajo no se ejecutará entero. Por ello, se propone tomar para cada trabajo i_o en cada marco ϕ_k el valor entero $\lceil x_{i_o} \cdot c_{i_o} \rceil$, es decir, el número entero más pequeño que es mayor o igual que $x_{i_o} \cdot c_{i_o}$.

En el caso de *f*, el valor decimal supone las unidades de cómputo que cada procesador debe poder ejecutar en F unidades de tiempo, y asegura tanto la no paralelización de los trabajos como que los procesadores puedan sacar adelante la carga de trabajo asignada durante el marco. Por motivos análogos a los antes expuestos, deberá asignarse como discretización de f un natural que sea mayor o igual que el valor decimal de *f*, pero en este caso, al haber sobredimensionado los ciclos asignados a los trabajos, es posible que

ese valor de f no sirva para satisfacer las restricciones. Por ello, por cada restricción no satisfecha, el valor de f discretizado deberá ser incrementado en tantas unidades como sea necesario para que sí lo haga. Dado que en las restricciones el valor de f siempre aparece a la derecha de un "$\leq$", es seguro que si un valor de f aseguró el cumplimiento de una restricción, entonces un valor mayor también lo hará, por lo que aunque se incremente el valor de f para hacer que se satisfaga una restricción en particular, es seguro que también satisfará todas las antes comprobadas. Por ello, se propone inicializar la discretización de f como $\lceil f \rceil$, es decir, la parte entera por encima del valor decimal de *f*, e incrementarla hasta asegurar que satisface todas las restricciones del PPL usando los valores resultantes de la discretización de $x_{i_o} \cdot c_{i_o}$.

A continuación se describen dos mejoras que se han implementado para reducir el error cometido en la discretización con respecto a los valores decimales originales: la primera consiste en registrar el número de ciclos que quedan por asignar para cada trabajo, de tal forma que no se le dé más tiempo de cómputo que el especificado como su WCET (Sec. 3.3.2), mientras que la segunda aprovecha la aplicación de la primera para tratar de tomar la parte entera por encima como discretización solo en aquellos marcos en los que esto suponga un error decimal pequeño, con el objetivo de que en aquellos en los que el error decimal sea mayor se tome como discretización la parte entera por abajo (Sec. 3.3.3). Ambas se utilizan conjuntamente en el planificador que se ha desarrollado, y un ejemplo de su funcionamiento puede encontrarse en el Anexo A.

3.3.2. Mejora basada en el registro de los ciclos por asignar

La discretización antes propuesta supone un sobreaprovisionamiento de recursos, debido al uso de la operación *parte entera por arriba*. Se ha comprobado que cuanto mayor sea el tiempo de ejecución de las tareas (C_{i_o}, medido en unidades de cómputo), menor es el error entre la discretización presentada y la solución decimal del PPL.

Existe una mejora posible que consiste en utilizar como discretización las unidades de cómputo que quedan por asignar a un trabajo en lugar de $\lceil x_{i_o} \cdot c_{i_o} \rceil$, cuando la parte entera por encima del valor decimal sea mayor que la cantidad de trabajo que queda por ejecutar 3.10.

$$x_{i_o} \cdot c_{i_o} \approx \lceil x_{i_o} \cdot c_{i_o} \rceil \text{ if } \lceil x_{i_o} \cdot c_{i_o} \rceil < not\_assigned_{i_o} \text{ else } not\_assigned_{i_o} \tag{3.10}$$

De esta forma, se consigue reducir a cero el error debido a la discretización de $x_{i_o} \cdot c_{i_o}$, pero sigue siendo necesario aplicar la misma técnica para la discretización de *f*, con el sobreaprovisionamiento correspondiente. Esto último es irremediable, pues se debe a la forma en la que el PPL ha repartido la ejecución de los trabajos en los marcos.

Al aplicar esta mejora deben tenerse en cuenta las siguientes consideraciones:

– Hay que almacenar y actualizar una variable $not\_assigned_{i_o}$ por cada trabajo.

– Para discretizar f, deben usarse los valores obtenidos al discretizar $x_{i_o} \cdot c_{i_o}$ de esta manera.

La anterior mejora debe de implementarse necesariamente al usar *Tertimuss*, pues los trabajos dejan de estar activos una vez se les ha asignado un número de ciclos de procesador igual a su WCET, por lo que no es posible asignarles ciclos adicionales como se proponía inicialmente.

3.3.3. Mejora basada en la ordenación con respecto al error cometido

Otra posible mejora consiste en aprovechar la mejora descrita en la sección anterior (Sec. 3.3.2) para propiciar la aplicación de la misma en aquellos marcos en los que tomar la parte entera por encima de la asignación incrementa el error.

Ahora, al discretizar cada $x_{i_ojk} \cdot c_{i_o}$ en el primer marco, la cantidad de ciclos que quedan por asignar de un trabajo es siempre mayor o igual que $\lceil x_{i_ojk} \cdot c_{i_o} \rceil$. Al tomarse la parte entera por encima como la discretización para todos los trabajos en ese marco, la asignación de ciclos que se hace en dicho marco queda sobrecargada, haciendo que tenga que aumentar el valor de f hasta el menor de los superiores disponibles en el procesador, de forma que quede holgura para la asignación que tienen el resto de marcos.

Por ello, en vez de empezar a discretizar siempre por el primer marco, sin mirar nada más, se debería:

1. Evaluar, para cada marco, la suma del error decimal que supone la discretización los trabajos asignados en él tomando la parte entera por encima.

2. Ordenar los marcos según el error decimal en la discretización.

3. Discretizar los marcos en orden de menor a mayor error cometido en la discretización.

Al usar la regla establecida en la Ec. 3.10 para discretizar, no se toma la parte entera por encima como discretización del trabajo en todos los marcos, sino que solo se hace cuando ese valor es menor que el número de ciclos del WCET del trabajo que quedan por asignar. Realizar la discretización de los marcos en el orden comentado supone que se toma la parte entera por encima en aquellos marcos en los que eso supone un

menor error. Alternativamente, se tenderá a tomar como discretización de los trabajos el número de ciclos que queden por asignar en aquellos marcos en los que discretizar usando la parte entera por encima supone un mayor error.

No obstante, lo ideal sería una discretización híbrida en la que en algunos marcos, para el mismo trabajo, se tome la parte entera por encima mientras que en otros se toma la parte entera por abajo, a fin de buscar la minimización del error cometido en la discretización.

3.3.4. Suboptimalidad inherente a la solución discretizada

Por último, ha de tenerse en cuenta que cualquier asignación que se obtenga de modificar los valores decimales dados como solución al PPL será una asignación subóptima, pues solo ese resultado para el PPL garantiza minimizar el valor de f y satisfacer todas las restricciones.

Por ello, la única forma de garantizar la optimalidad de la solución discretizada sería plantear el PPL discretizado desde un primer momento. Esta opción se deja planteada más adelante como trabajo futuro.

3.4. Cálculo de la frecuencia mínima requerida

La frecuencia mínima se calcula como la primera derivada del cómputo realizable por cada uno de los procesadores (valor discretizado de f) con respecto al tiempo en el que lo debe realizar (duración del marco del ejecutivo, que se corresponde con el valor de F):

$$\alpha = \frac{\partial W(t)}{\partial t} = \frac{f}{F} \tag{3.11}$$

La frecuencia elegida finalmente será la más pequeña entre las disponibles que satisface ser mayor o igual que $\alpha = \frac{f}{F}$, siendo f el resultado de discretizar el valor decimal de la variable f obtenido en la solución al PPL. Así, si se tiene como Λ el conjunto de frecuencias disponibles para la plataforma de cómputo sobre la cual debe realizarse la planificación, se tendrá β, la frecuencia elegida, como:

$$\beta = \text{mín}(\{\lambda | \lambda \in \Lambda \wedge \lambda \geq \alpha\}) \tag{3.12}$$

Es decir, β es la mínima frecuencia disponible entre las que cumplen ser mayores o iguales que α, la frecuencia mínima necesaria tras la discretización.

Utilizando el valor de frecuencia β, que es el valor de frecuencia finalmente elegida, deberá recalcularse la duración que se tiene para cada marco en ciclos, pues ya no se corresponderá con el valor discretizado de la variable *f*. Así, despejando el número de

ciclos de la ecuación Ec. 3.11 y usando el valor de β como frecuencia, se tiene como $f' = \beta \cdot F$ a la duración en ciclos de cada marco F considerando β como la frecuencia de cómputo real a la que van a funcionar los procesadores.

Dado que $\alpha \leq \beta$, también se tiene que $f \leq f'$. Por lo tanto, por el mismo motivo que se exponía para la discretización (Sec. 3.3), se tendrá que si las restricciones eran satisfechas con el valor resultante de discretizar *f*, también lo serán con el valor *f'*.

3.5. Obtención del ejecutivo cíclico a partir de las asignaciones

De ahora en adelante, con el fin de poder discernir entre el valor decimal obtenido como solución al PPL y el valor discretizado, se denota como e_{i_o} al resultado de discretizar la asignación de ejecución $x_{i_o} \cdot c_{i_o}$ para el trabajo j_{i_o} con cualquiera de las técnicas que se han planteado en la Sec. 3.3. De igual forma, al hablar de la duración de cada marco en ciclos, se estaŕa haciendo referencia a *f'*, que es el número de ciclos de cada marco con el valor de frecuencia β (Sec. 3.4).

La obtención de un ejecutivo se realiza atendiendo al modelo de expulsión especificado para las tareas, que debe haber sido tenido en cuenta a la hora de definir el PPL, y que ahora acaba resultando en lo siguiente:

- Dada una asignación de valores reales no negativos a las variables x_{i_o} que satisfaga las restricciones del PPL, y considerando las asignaciones en ciclos e_{i_o} ya obtenidas para los trabajos en cada ciclo menor, se puede construir un ejecutivo cíclico global expulsivo planificando, en cada marco ϕ_k, para cada x_{i_o} que tenga asignado un valor distinto de cero, la ejecución de cada trabajo j_{i_o} un número de ciclos igual a e_{i_o} durante el k-ésimo marco del hiperperiodo. No obstante, para asegurar que ningún trabajo se ejecuta concurrentemente sobre dos procesadores diferentes, será necesario aplicar la regla de envoltura de *McNaughton* de la forma en la que se describe en la Sec. 3.5.1.

- Dada una asignación de valores enteros cero o uno a cada una de las variables x_{ijk} que satisfaga las restricciones del PPLM, se puede construir un ejecutivo cíclico no expulsivo:

  - Planificando la ejecución de cada trabajo j_i en el j-ésimo procesador durante el marco k-ésimo, para cada x_{ijk} que tenga asignado el valor 1. Si bien consiste en una asignación directa, han de tenerse en cuenta algunas consideraciones que aparecen descritas en la Sec. 3.5.2.

- Planificando la ejecución de cada trabajo j_i durante el marco k-ésimo, para cada x_{ijk} que tenga asignado el valor 1, asignando los procesadores utilizando la regla de envoltura de *McNaughton*, pero sin permitir que la ejecución de los trabajos se parta en caso de que alguno de ellos no quepa en el espacio que queda por asignar en el núcleo que se considera. Esto supone ciertas modificaciones en la forma en que se aplica la regla con respecto a como se describe en el artículo para el caso expulsivo, y se resumen en la Sec. 3.5.3.

3.5.1. Ejecutivo cíclico expulsivo

En esta sección se describe la manera de construir un ejecutivo cíclico basado en un planificador expulsivo para el k-ésimo marco (ϕ_{k_o}). El ejecutivo cíclico completo se obtiene de repetir este procedimiento para cada k_o, con $1 \leq k_o \leq \frac{P}{F}$.

Se puede construir una planificación para el marco en cuestión usando la regla de envoltura de *McNaughton* [26] de la siguiente manera:

1. Ordenar los trabajos que reciben alguna cantidad de ejecución durante el marco ϕ_{k_o} de manera arbitraria. Esto es, aquellos trabajos j_{i_o} tales que $x_{i_o} \neq 0$.

2. Empezar acomodando los trabajos sobre los procesadores en ese orden, llenando el procesador j-ésimo antes de empezar con el *(j+1)*-ésimo.

   Por lo tanto, la ejecución de un trabajo j_{i_o} puede ser partida entre procesadores, asignando las últimas t unidades de tiempo del marco al procesador j-ésímo y las $(e_{i_o} - t)$ unidades de tiempo del frame en el procesador *(j+1)*-ésimo, y como $e_{i_o} \leq f'$, esas asignaciones no se solaparán en el tiempo, por lo que la ejecución del trabajo no será paralelizada.

   Nótese que esta es la única situación posible en la que puede producirse una migración de un trabajo durante el marco.

Es evidente que esta planificación puede lograrse eficientemente, en tiempo polinomial con la representación propuesta para el sistema de tareas.

El objetivo de agrupar las asignaciones en todos los procesadores durante el marco y usar la regla de *McNaughton* es minimizar las migraciones entre procesadores de trabajos durante el ciclo menor. Al sumar todas las asignaciones del mismo trabajo en el marco, se consigue planificarlas todas seguidas y, en principio, en un único procesador. En caso de quedar parte por ejecutar y haber llenado el procesador, se asigna lo que quede de trabajo al principio del siguiente procesador, haciendo que el trabajo empiece ejecutándose en este último y tenga que migrar en algún momento del marco al procesador anterior para completar la cantidad de ejecución asignada, siendo

ese es el único caso posible de migración de un trabajo durante un marco, y dado que la asignación de valores a las variables x_{ijk} satisface la tercera restricción, se sabe que la planificación resultante no paralelizará la ejecución del trabajo.

Para demostrarlo, en el artículo se plantea lo siguiente: si el trabajo tiene asignado un tiempo de cómputo e_{i_o} durante el marco y t u.c. se asignan seguidas al final del marco en el procesador j y las $e_{i_o} - t$ restantes al principio del marco en el procesador *(j+1)* también seguidas, es imposible que se solapen en el tiempo en las f' u.c. ejecutables durante un ciclo menor, porque $\not{t} + (e_{i_o} - \not{t}) \leq f'$

Para ver un ejemplo completo de la aplicación de la regla de *McNaughton* para obtener una planificación expulsiva, consúltese el Anexo B.

3.5.2. Ejecutivo cíclico no-expulsivo usando directamente la solución del PPL

La planificación en este caso consiste en utilizar la asignación de procesadores dada en la solución al PPLM (es decir, si $x_{ijk} = 1$, entonces el trabajo i deberá ejecutarse en el procesador j durante el marco k del ejecutivo), y no acumularse todos los trabajos que han sido asignados durante el mismo marco de manera consecutiva hasta llenar las procesadores, como sí se hace en el caso expulsivo al aplicar la regla de *McNaughton*.

Lo que si está permitido es que, dados dos o más trabajos asignados a un mismo procesador durante un mismo marco, que el orden en que éstos se ejecuten, con tal de que lo hagan enteros de principio a fin (sin expulsiones), sea indistinto (Fig. 3.5).

3.5.3. Ejecutivo cíclico no-expulsivo usando la regla de *McNaughton*

Alternativamente a la planificación no expulsiva que se hace en el artículo, se puede plantear el uso de la regla de *McNaughton* también en este caso, solo que cambiando ligeramente la forma en que se aplica dado un marco ϕ_{k_o} en particular:

1. Ordenar los trabajos tales que $x_{ijk_o} = 1$ de manera arbitraria. Esto es, aquellos trabajos que han sido asignados a ese frame en alguno de los m procesadores.

2. Empezar acomodando los trabajos sobre los procesadores en ese orden, llenando el procesador j-ésimo antes de empezar con el *(j+1)*-ésimo.

   No obstante, ahora **la ejecución de los trabajos no puede ser partida**, porque en tal caso se estaría llevando a cabo una planificación expulsiva. Por lo tanto, cuando las e_{i_o} unidades de cómputo no caben en el procesador j que se está considerando, la ejecución completa del trabajo j_{i_o} tiene que ser llevada al *(j+1)*-ésimo procesador (Fig. 3.6).

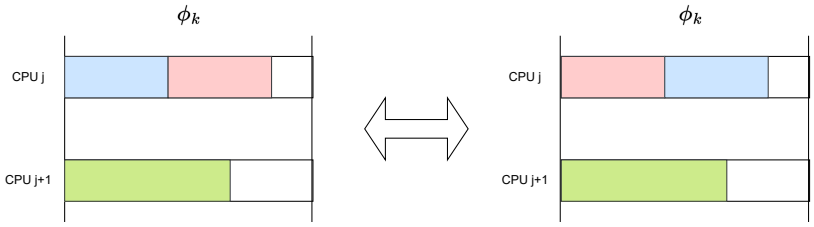


Figura 3.5: Asignaciones igualmente válidas al aplicar directamente la solución del PPL

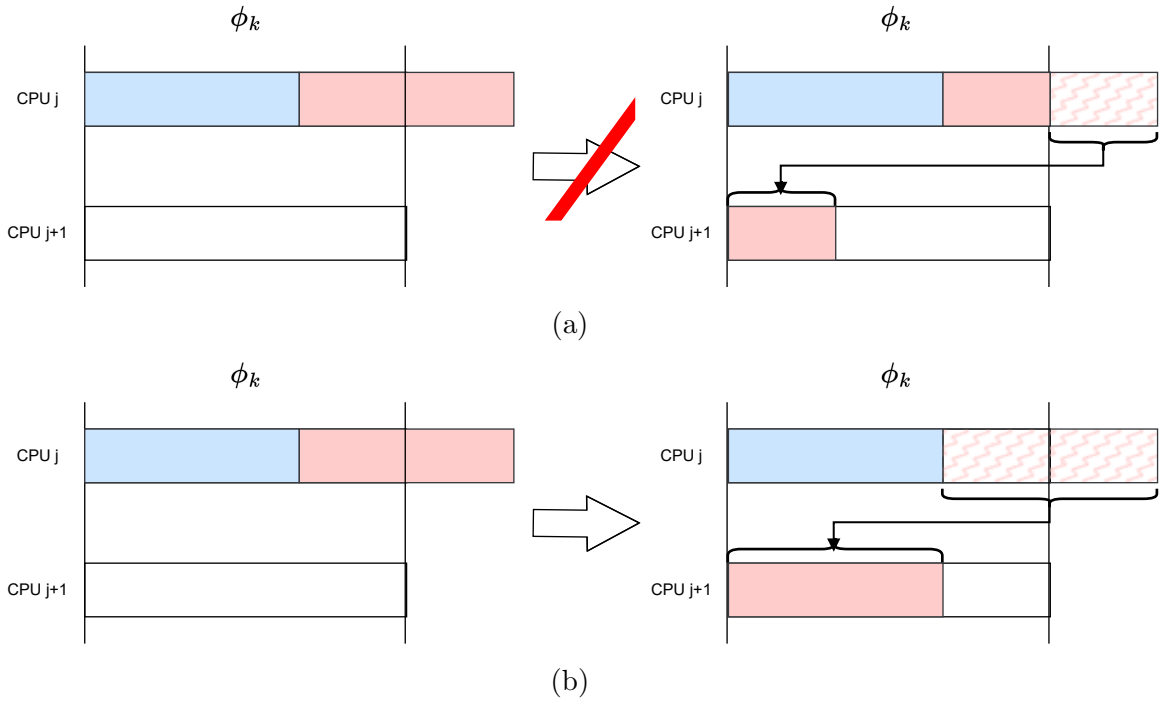


Figura 3.6: Aplicación errónea (a) y acertada (b) de la regla de *McNaughton* en el caso no expulsivo para una misma asignación

Dado el sobreaprovisionamiento de recursos que el planteamiento no expulsivo provoca, la aplicación de la regla de *McNaughton* tiene un interés práctico importante, pues puede llevar a resultados en los que que solo se necesiten x procesadores, con $x \ll m$, siendo m el número de procesadores de la plataforma. En tal caso, podrían apagarse los $m - x$ procesadores no necesarios o dejarlos inactivos para que no consuman. De hecho, si $x == 1$ y realmente se quisiese aplicar la solución no expulsiva, podría replantearse el uso del multiprocesador para llevar a cabo el despliegue en una plataforma monoprocesador.

Además, el Problema de Programación Lineal Mixta que se ha planteado tiene espacio de búsqueda muy grande, por lo que tiene sentido tratar de encontrar soluciones igual de óptimas en lo que respecta al problema de optimización planteado pero en las cuales se consideren otras propiedades a la hora de llevar a cabo la asignación, como es el caso que se tiene con respecto al número de procesadores utilizados.

3.6. Ejemplo completo

El anexo C presenta la ejecución de todos los pasos descritos para la obtención de un ejecutivo cíclico tanto expulsivo como no expulsivo a partir de un sistema de tareas periódicas con plazo de respuesta implícito.

3.7. Pruebas de corrección de la implementación

3.7.1. Corrección formal

Los valores presentados como ecuaciones e inecuaciones invariantes en la parte teórica son utilizados en aserciones para comprobar la corrección formal de la implementación realizada. Entre las propiedades comprobadas están:

- El número de variables x_{ijk} definidas en el problema de programación lineal es igual a $N \cdot m \cdot \frac{P}{F}$.
- El número de restricciones definidas en el problema de programación lineal es igual a $n+(m+N)\cdot\frac{P}{F}$ en el caso expulsivo y a $(n+m\cdot\frac{P}{F})$ en el caso no expulsivo, siendo:
  - n correspondientes a la primera restricción.
  - $m \cdot \frac{P}{F}$ correspondientes a la segunda restricción.
  - $N \cdot \frac{P}{F}$ restricciones correspondientes a la tercera restricción.
- La satisfacción de las restricciones con los valores obtenidos tras la discretización, usando las ecuaciones 3.9 y 3.8 al hablar de dicho proceso.

3.7.2. Pruebas de integración continua

Además de las comprobaciones de corrección formal, se ha desarrollado una batería de pruebas con la que se busca cubrir diferentes caminos en el flujo de ejecución que se pueden dar en el algoritmo según los valores de entrada. Con ese fin, se proporcionan parámetros de entrada con distintas características que buscan forzar la ejecución de todos los caminos posibles en el código. Entre las variaciones consideradas, están:

- La obtención de resultados expulsivos y no expulsivos, siendo el caso de estos últimos probados tanto para la asignación directa a partir de la solución del PPL como utilizando la regla de *McNaughton*.
- El número de procesadores de la plataforma.

- Las frecuencias proporcionadas, propiciando el uso de la frecuencia mínima requerida, el uso de frecuencias mayores que ella y la no viabilidad en la construcción del ejecutivo por no disponer de la frecuencia suficiente.

- Nivel de ocupación de los procesadores, en función de la carga de trabajo que suponen las tareas especificadas y los valores de frecuencia disponibles. Con esto se busca que, al aplicar la regla de *McNaughton*, se den todos los casos posibles de asignación para las últimas unidades de cómputo de los marcos, tanto en el caso expulsivo como en el no expulsivo.

- Los periodos de las tareas, especificando periodos con valores primos para detectar fallos que podrían no surgir debido a la divisibilidad entre los valores utilizados.

- Conjuntos de tareas utilizados en otros tests de integración continua presentes en *Tertimuss*.

- La especificación de conjuntos de tareas no permitidos, como tareas periódicas cuyo plazo de respuesta es distinto de su periodo, o que se especifican con valores de tiempo decimales.

Nótese que, al haber especificado las condiciones de corrección formal mediante aserciones del lenguaje, éstas se comprueban para cada uno de los casos antes presentados.

Adicionalmente, se ha implementado el soporte durante la simulación para la detección de la paralelización de trabajos en las asignaciones que hace el planificador. Si bien es una opción que está desactivada por defecto (pues supone una funcionalidad adicional que podría alterar el funcionamiento de otros componentes ya existentes), en todos los casos de prueba se especifica para poder asegurar el cumplimiento de la restricción tres del PPL (inecuación 3.6) también en la fase en línea de planificador.

Capítulo 4

Comparativa entre ejecutivos cíclicos

4.1. Metodología y entorno experimental

Los conjuntos de tareas para la comparación se generaron utilizando el algoritmo UUniFast-discard [27]. La utilización total de cada conjunto de tareas es igual al número de procesadores del experimento. Los periodos de las tareas se seleccionaron aleatoriamente entre los divisores de 60, para obtener un ciclo mayor de 60s como máximo, y una frecuencia base de 1000Hz, de tal forma que el tiempo de cómputo de cada tarea se puede calcular fácilmente. Los conjuntos de tareas se ejecutan en sistemas con 2 y 4 núcleos. El número de tareas se eligió de acuerdo con tasas de 4, 8, 12, 16 y 20 tareas por núcleo. Es decir, en un sistema con dos núcleos se pueden encontrar experimentos con 8, 16, 24, 32 y 40 tareas. Por combinación se generaron 200 conjuntos de tareas, es decir, un total de 2000 experimentos.

Cada conjunto de tareas generado se planificó con los algoritmos AlECS, CAlECS, RUN y DCE. Para este último planificador se presenta su variante expulsiva y no expulsiva, cada uno a su vez usando dos solucionadores diferentes: *GLOP* y *Gurobi*. En el caso no expulsivo, se muestran además las planificaciones cuando se utiliza directamente la solución de su PPLM y la regla de *McNaughton*. En total se evualúan seis variantes del planificador DCE.

Se ha recurrido a utilidades estándar de UNIX [28], como *test*, *sed* y *grep*. También se ha hecho uso características propias del *shell bash* [29], tales como los vectores. Para comprimir los resultados de simulación se ha utilizado el algoritmo LZMA [30] tras comprobar que ofrecía resultados especialmente buenos con los ficheros en formato PICKLE [31], alcanzando ratios de compresión de hasta 0,1 (es decir, el fichero resultante de la compresión ocupaba diez veces menos que el original).

Para poder realizar la comparativa de una forma objetiva, las frecuencias disponibles

en los experimentos de DCE se han ajustado para que, siempre que sea posible (su valor sea un número natural), la frecuencia mínima requerida por DCE para cada experimento esté entre ellas, de tal forma que éste pueda escogerla y llevar a cabo con ella la planificación. El motivo es que el resto de planificadores disponen de la frecuencia mínima requerida por su algoritmo para la planifiación, por lo que para que los resultados sean representativos, DCE también debe.

Esos valores de frecuencia se obtienen ejecutando DCE primero, solo para obtener los mensajes de advertencia reportados, con el fin de extraer de ellos la frecuencia mínima requerida. Después, con esos valores ya conocidos, se proporcionan en una segunda ejecución a partir de la cual se realiza la comparativa que se presenta.

4.2. Resultados experimentales

4.2.1. Tiempo de ejecución

La Tab. 4.1 muestra los tiempos de ejecución obtenidos con los planificadores comparados. Al tratarse de ejecutivos cíclicos el cálculo de la planificación se genera fuera de línea, por lo que no compromete el rendimiento del sistema.

En todo caso es interesante observar que el tiempo de ejecución para DCE con Gurobi es mucho menor que con GLOP. Esto puede deberse al algoritmo de optimización que utiliza cada uno de ellos. GLOP usa SIMPLEX, que en el caso promedio tiene un coste polinomial en tiempo de ejecución pero en el peor es exponencial. Gurobi usa el método del punto-interno, que siempre tiene un coste polinomial.

4.2.2. Frecuencia mínima requerida

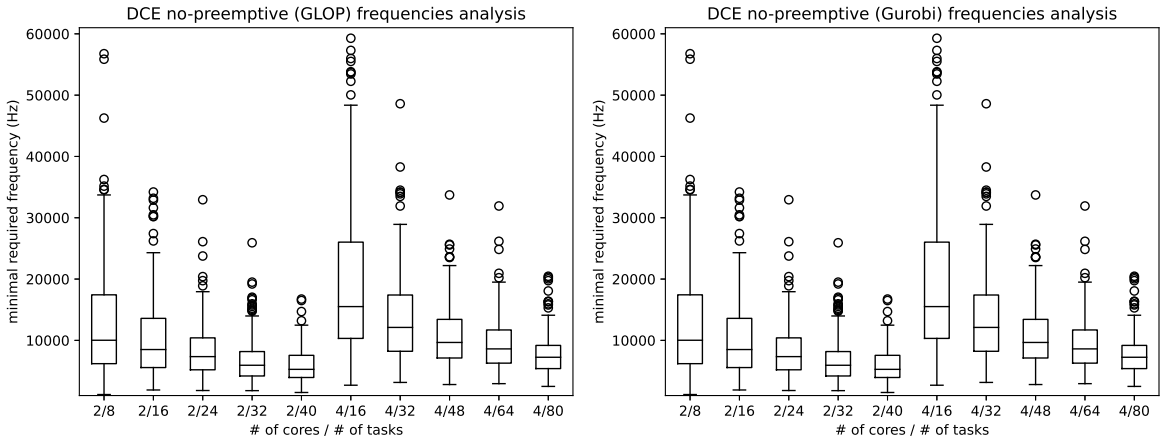
La frecuencia mínima requerida por el planificador DCE con expulsión permitida usando los solucionadores GLOP y Gurobi es igual a 1001 y no 1000 como se esperaba, salvo en casos espurios. Esto podría deberse al proceso de discretización que se ha tenido que aplicar a la solución obtenida en el PPL para poder llevar a cabo la implementación. No obstante, puede deberse también a un problema de precisión decimal derivado de la forma en la que se almacena el tipo de dato *float* en el lenguaje Python, que se ha que considera una unidad más al tomar la parte por encima.

En lo que respecta a dicho problema de precisión decimal (error numérico), es un aspecto que escapa del ámbito del problema que se está tratando de resolver, por lo que simplemente debe tomarse un criterio al respecto:

– Tratar de detectarlo y truncar el valor decimal en vez de tomar la parte entera

Planificador	**Tiempo de ejecución**
AlECS	201.28 s (3 m 21 s)
CAlECS	111.96 s (1 m 52 s)
RUN	936.35 s (15 m 36 s)
DCE (GLOP) expulsivo	245.72 s (4 m 6 s)
DCE (GLOP) no expulsivo (usando solución al PPLM)	5648.78 s (1 h 34 m 9 s)
DCE (GLOP) no expulsivo (usando regla de McNaughton)	5760.59 s (1 h 36 m 1 s)
DCE (Gurobi) expulsivo	158.66 s (2 m 39 s)
DCE (Gurobi) no expulsivo (usando solución al PPLM)	249.27 s (4 m 9 s)
DCE (Gurobi) no expulsivo (usando regla de McNaughton)	250.38 s (4 m 10 s)

Tabla 4.1: Tiempos de ejecución de los experimientos



(a) (b)

Figura 4.1: Frecuencia mínima requerida por el planificador DCE sin expulsión permitida usando los solucionadores GLOP (a) y Gurobi (b)

por arriba en caso de que se considere que hay error (lo cual puede resultar en una decisión arbitraria en lo que respecta a la precisión que se considera para tomarlo como error).

– Tomar siempre la parte entera por encima del valor decimal independientemente de que eso suponga o no el sobredimensionar el valor cuando se trate de un error numérico de precisión decimal (lo cual a priori es la decisión más conservadora al tratarse de sistemas tiempo real estricto, en los que es normal sobredimensionar ciertas cifras con tal de asegurar la corrección).

En este caso, se opta por el enfoque más conservador, y para la comparativa se optará por dar frecuencias mayores a los 1000 Hz iniciales, siendo incrementados de unidad en unidad (es decir; 1001, 1002, 1003,..., de uno en uno), con el fin de dar la posibilidad al planificador de elegir la mínima frecuencia requerida aunque esta sea superior a los 1000 Hz.

No hay diferencias significativas entre las soluciones obtenidas con los distintos solucionadores de PPL, ni en el caso expulsivo ni en el no expulsivo.

El aumento en 1Hz de la frecuencia mínima requerida en el caso expulsivo hace pensar que se deba a la aproximación de la variable de frecuencia f al valor de frecuencia discreto disponible en el procesador más próximo por arriba respecto a la frecuencia mínima calculada teóricamente.

Los resultados de la frecuencia requerida en el caso no expulsivo concuerdan con el nivel de carga de trabajo a la que se somete al multiprocesador, y parece haber una relación directa entre dichos valores.

4.2.3. Uso de los procesadores

Número de procesadores necesarios

El número de procesadores utilizados por el planificador DCE no-expulsivo asignando las tareas a partir de la solución del PPL es igual, con ambos solucionadores, al número de procesadores disponibles para cada experimento. Al arrojar cada experimento un valor único (con apenas dos casos espurios) la representación de resultados mediante un diagrama de *caja y bigotes* se omite.

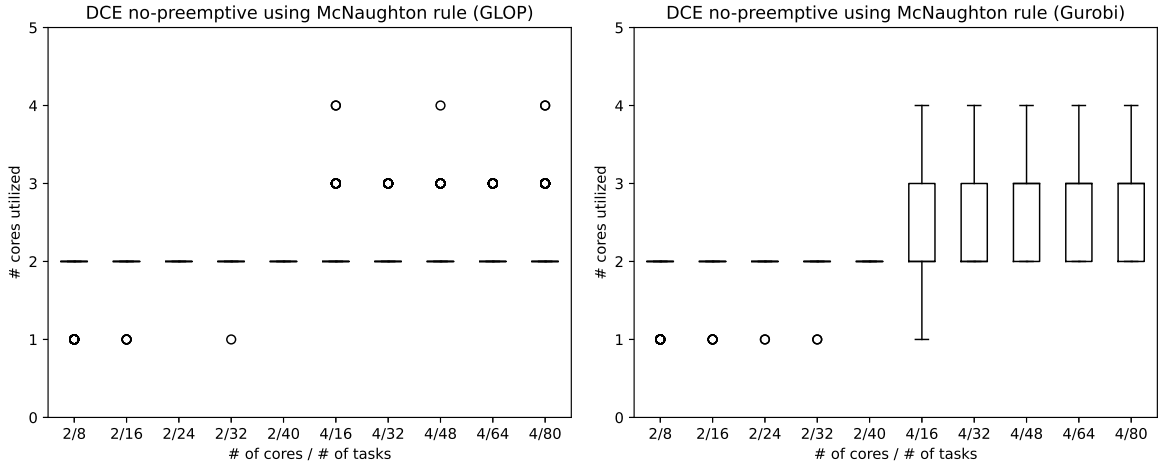
En la asignación obtenida al utilizar la regla de McNaughton se puede observar que la asignación obtenida con GLOP (Fig. 4.2.a) supone un uso de menos procesadores con respecto a la asignación obtenida con Gurobi (Fig. 4.2.b). Esto resulta más evidente cuando la plataforma de cómputo dispone de cuatro núcleos.

Utilización del multiprocesador

Los planificadores expulsivos (AlECS, CAlECS, RUN y DCE expulsivo) consiguen una utilización máxima de los procesadores disponibles. Sin embargo, las variantes de DCE sin expulsión (Fig. 4.3) no superan el 50 % de utilización en el caso promedio, de forma similar a un conocido resultado relativo a los métodos de particionado basados en RM [32].

Cada conjunto de tareas en los experimentos realizados resulta en una utilización igual al 100 % de los procesadores con la frecuencia base de 1000Hz. Dado que las variantes no expulsivas de DCE solo consiguen utilizar un 50 %, se ven obligadas a aumentar la frecuencia de los procesadores en más del doble para poder planificar los conjuntos de tareas respetando las restricciones temporales, tal como hemos observado en el análisis de frecuencia mínima de la Fig. 4.1.

No existen diferencias significativas entre planificaciones obtenidas a partir de GLP o Gurobi en cuanto a la utilización lograda. Respecto a la aplicación de McNaughton



(a) (b)

Figura 4.2: Número de procesadores utilizados por el planificador DCE sin expulsión asignando las tareas con la regla de McNaughton, usando los solucionadores GLOP (a) y Gurobi (b)

a la solución del PPL en el DCE no expulsivo, ésta no conlleva ninguna variación en la utilización respecto a la planificación directamente derivada del PPL, porque tanto las asignaciones como la frecuencia son iguales, y solo varía la ordenación sobre los procesadores.

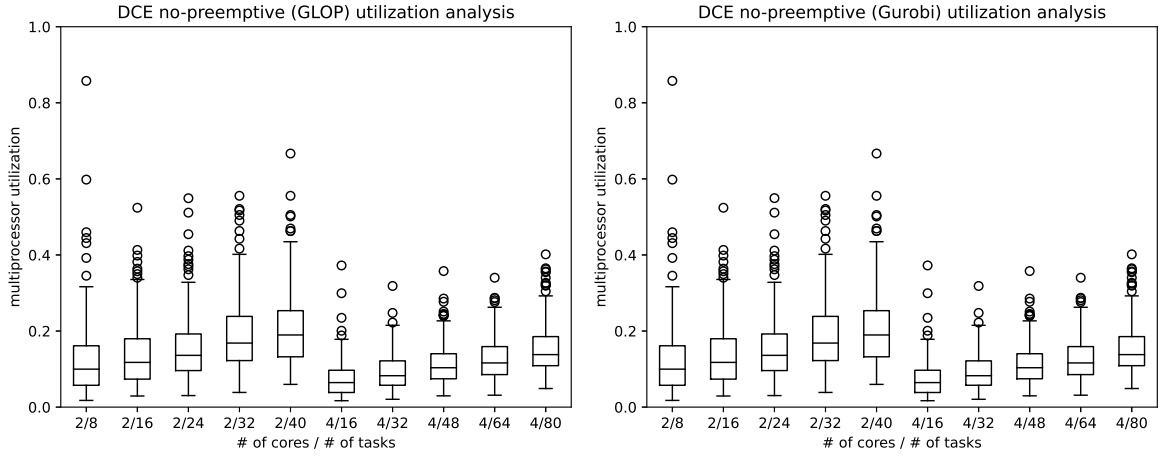
Finalmente, se puede apreciar cómo la evolución de la utilización de los procesadores es inversamente proporcional a la frecuencia mínima requerida (Fig. 4.1) con respecto a los niveles de carga de trabajo.

4.2.4. Migraciones y expulsiones

Las Figs. 4.4 y 4.5 muestran los resultados de expulsiones y migraciones de los planificadores comparados.

Los resultados de AlECS, CAlECS y RUN son coherentes con las comparaciones conocidas ya publicadas [2]. AlECS consigue menos expulsiones y migraciones que el algoritmo de referencia RUN en muchos casos, debido a que está basado en planificación fluida, pero RUN llega a conseguir un particionado perfecto (cero migraciones) en algunos casos (con 24, 32 y 40 tareas sobre 2 procesadores, y con 64 y 80 tareas sobre 4 procesadores). CAlECS aprovecha la optimalidad que proporciona AlECS en los casos en los que no es posible encontar un particionado perfecto, pero además es capaz de encontrar incluso más particionados perfectos que RUN, debido a la utilización de una fase previa de *clustering* mediante la resolución de un Bin-Packing Problem.

Las aproximaciones propuestas en DCE [1] (Fig. 4.5) introducen en general un número mayor de expulsiones y migraciones que los algoritmos anteriores basados en



(a) (b)

Figura 4.3: Utilización del multiprocesador por el planificador DCE sin expulsión usando los solucionadores GLOP (a) y Gurobi (b)

planificación fluida (AlECS y CAlECS) y el algoritmo de referencia RUN. También presentan un mayor número de *outliers*, especialmente respecto a AlECS y CAlECS. Llama también la atención en DCE la mayor diferencia inter-cuartil con bajo número de tareas especialmente respecto a AlECS y CAlECS.

No se aprecia una diferencia significativa entre las soluciones DCE con GLOP (Fig. 4.5 a y b) y con Gurobi (Fig. 4.5 c y d).

También se puede ver que el número de migraciones es muy similar al de las expulsiones, lo cual significa que prácticamente todas las veces que un trabajo se expulsa de un procesador se acababa reanudando en otro distinto. Esto puede verse como una consecuencia de la utilización de la regla de *McNaughton* para acomodar las asignaciones de ejecución de los trabajos en los distintos marcos del Ejecutivo Cíclico. Estos resultados, a su vez, son parecidos a los que se tienen en AlECS.

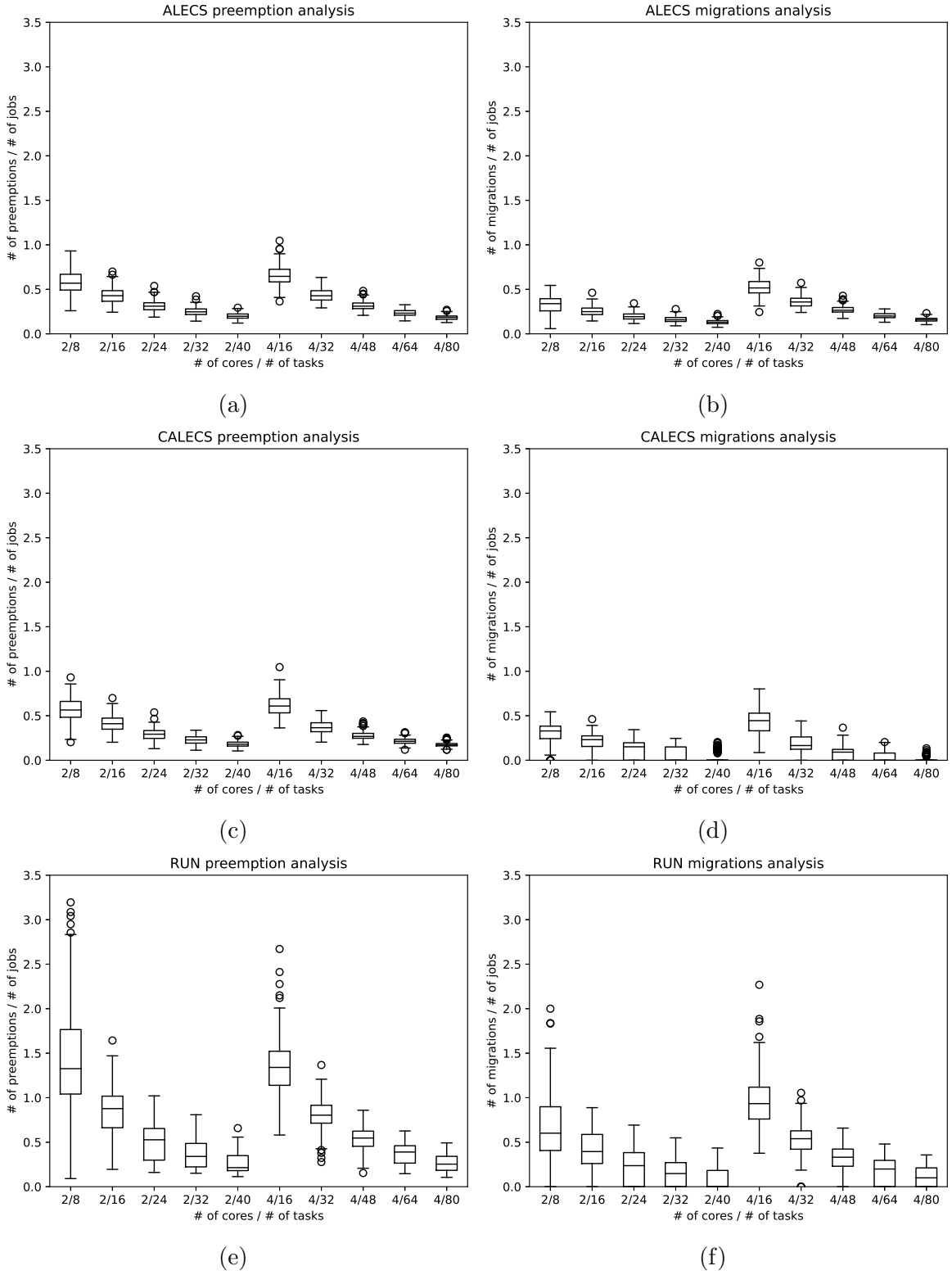
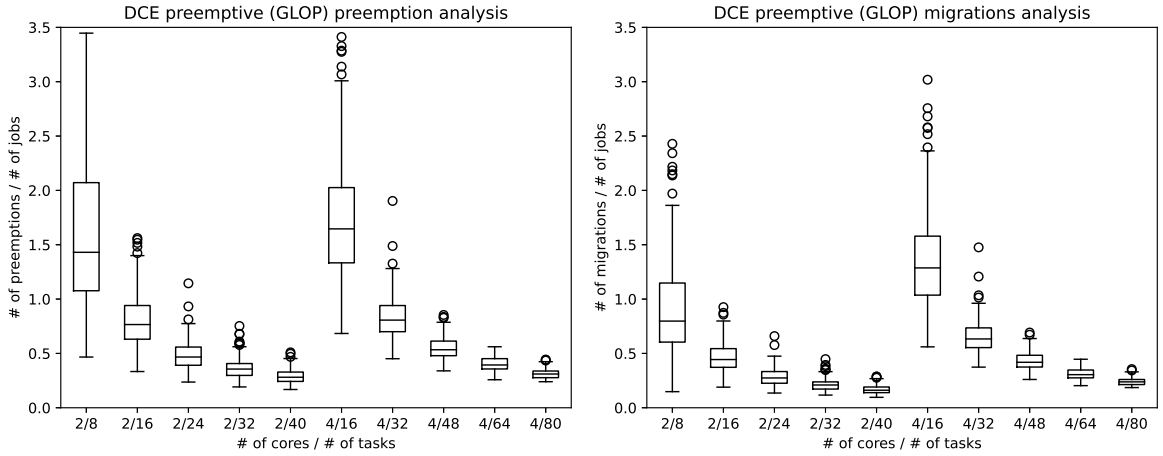
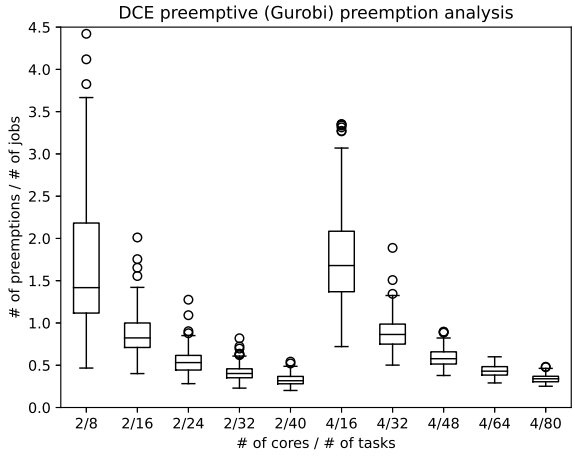


Figura 4.4: Expulsiones (izquierda) y migraciones (derecha) por trabajo al utilizar los planificadores (de arriba abajo) AlECS, CAlECS y RUN

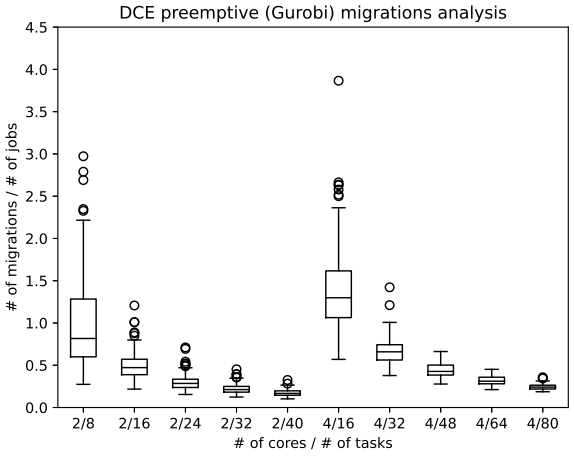


(a)

(b)



(c)



(d)

Figura 4.5: Expulsiones (izquierda) y migraciones (derecha) por trabajo al utilizar los planificadores DCE con expulsión permitida, usando los solucionadores GLOP (a,b) y Gurobi (c,d)

4.3. Valoración

Puede concluirse que DCE expulsivo no aporta ventajas en términos de migraciones respecto a RUN o CAlECS.

El DCE no expulsivo se sitúa por definición en el ámbito de la planificación particionada (cero migraciones), y como sucede con este tipo de planificadores, desperdicia potencia de cálculo (baja utilización) y además obliga al uso de frecuencias mayores (mayor gasto de energía), incluso no factibles. Sin embargo, se puede reducir el número de procesadores necesarios si se aplica *McNaughton* a la solución del PPL, en lugar de obtener la planificación directamente de dicha solución, tal como hacen los autores que proponen DCE.

Capítulo 5

Conclusiones

5.1. Recapitulación

5.1.1. Métodos implementados y su relevancia

En este trabajo se ha implementado variantes del algoritmo de planificación propuesto en [1] integrándolo en la plataforma de simulación *Tertimuss*. Se trata de un artículo relevante que confirma el interés, de cara a la industria, de generar ejecutivos cíclicos eficientes para sistemas TRC sobre procesadores multinúcleo. Ha sido necesario adaptar algunas especificaciones originales para solventar las discordancias en las unidades utilizadas.

5.1.2. Métodos comparados y su relevancia

Con la implementación realizada, se ha llevado a cabo una comparativa de las variantes propuestas en el artículo original con los métodos propuestos en AlECS [33], CAlECS [2, 5] y RUN [3], cuya implementación ya estaba disponible en *Tertimuss*. Los tres métodos permiten la generación eficiente de ejecutivos cíclicos sobre multiprocesador a partir del concepto de planificación global. Los dos primeros se basan en planificación fluida, e incorporan restricciones térmicas y de energía (que se obvian en este trabajo). RUN se basa en el concepto de planificación dual, y se ha venido considerando un algoritmo de referencia en cuanto a la minimización de cambios de contexto y migraciones.

5.1.3. Resultados de la comparación experimental

La comparación ha permitido determinar, en base a los resultados experimentales, que DCE expulsivo no supone ninguna ventaja particular con respecto a los otros algoritmos considerados para las métricas establecidas (expulsiones y migraciones), y ha puesto en manifiesto las limitaciones de las soluciones no expulsivas en la utilización

del multiprocesador y los valores de frecuencia requeridos. No obstante, una de las aportaciones de este TFG ha consistido en el uso de la regla de *McNaughton* al aplicar DCE para planificaciones no expulsivas, lo cual ha permitido determinar que existen soluciones a partir de las asignaciones obtenidas que, siendo igualmente factibles, permiten reducir el número de procesadores utilizados.

5.1.4. Planificación basada en optimización matemática

En la introducción de esta memoria (Sec. 1.1) nos referíamos a una paradoja a la que actualmente se enfrenta la industria. Por un lado, necesita explotar eficientemente los MPSoCs para evitar el sobrediseño. Por otro, continúa adherida a métodos muy conservadores de generación de ejecutivos cíclicos (particionado), que limitan por su propia naturaleza la eficiencia perseguida.

La experiencia de implementar DCE a partir de la solución de un PPL, y compararlo con métodos como CAlECS que combinan particionado y planificación fluida, recurriendo también a optimización matemática, permite realizar observaciones interesantes respecto a esta paradoja.

La síntesis de ejecutivos cíclicos en particular los no expulsivos es altamente intratable: es bien conocido que la planificación no expulsiva es NP-completo en el sentido fuerte, incluso para instancias altamente restringidas del problema, en las que todas las tareas tienen el mismo periodo. Aun así, motivados por los avances en la optimización de los solucionadores de Problema de Programación Lineal Entera, la disposición de computadores cada vez más potentes en los que ejecutarlos y el hecho de que todo eso se realiza como parte fuera de línea de la planificación, los autores del artículo de referencia de DCE [1] apuestan por la solución exacta.

Hemos visto las dificultades prácticas que esto supone al implementar estos métodos (ver especialmente la Sec. 3.3). Este aspecto pone en valor aún más si cabe las propuestasAlECS [33] y CAlECS [2, 5], basadas en el planteamiento de un PPLE resoluble como PPL, que en el caso de CAlECS consiguen una clara ventaja en los resultados (Sec. 4).

5.1.5. Otras contribuciones

Además de las conclusiones que se han podido extraer de los resultados empíricos, con este TFG se ha contribuido en el desarrollo de la herramienta *Tertimuss* al implementar el método de planificación DCE, nuevas funcionalidades para la simulación y solucionar errores que se han encontrado.

5.2. Trabajo futuro

Como consecuencia del trabajo realizado, han surgido diferentes ideas con las que continuar este TFG. Entre ellas cabe destacar las siguientes.

5.2.1. Rediseño de DCE para evitar la suboptimalidad

Los resultados de experimentales (Cap. 4) han mostrado que las soluciones no expulsivas (Secs. 3.5.2 y 3.5.3) presentan un bajo rendimiento, mientras que la expulsiva (Sec. 3.5.1) resulta subóptima debido a la necesidad de discretizar el resultado obtenido como solución al PPL (Sec. 3.3.4). Este último problema se solventa en AlECS y CAlECS) planteando un PPLE (resoluble como PPL). Por ello, proponemos una reformulación del problema de optimización descrito en [1] con el fin de simplificar su especificación, permitiendo además que los resultados sean directamente aplicables sin pérdida de optimalidad con respecto a los valores obtenidos.

Función objetivo

$minf$

Restricciones

1. $(\sum_{j=1}^{m} \sum_{k=\frac{a_i}{F}+1}^{\frac{d_i}{F}} x_{ik}) = c_i$, para cada i, $1 \leq i \leq n$
2. $(\sum_{i \in jobs\_in\_frame_k} x_{ik}) \leq m \cdot f$, para cada k, $1 \leq k \leq \frac{P}{F}$
3. $x_{ik} \leq f$, para cada i, $1 \leq i \leq n$, y cada k, $\frac{a_i}{F} + 1 \leq k \leq \frac{d_i}{F}$

Variables

x_{ik} el número de ciclos del trabajo i (esto es, $0 \leq x_{ik} \leq c_i$) que deben ejecutarse entre todos los procesadores disponibles durante el marco k, $x_{ijk} \in \mathbb{N}$

f el número de ciclos que cada procesador debe poder ejecutar en F unidades de tiempo, $f \in \mathbb{N}$

Obtención del ejecutivo cíclico

Para generar un ejecutivo cíclico a partir de la asignación para las variables obtenida como solución, se debe aplicar la regla de *McNaughton* de la misma forma que se ha explicado en la Sec. 3.5.1. La diferencia es que ahora ya no es necesario acumular la cantidad de computo asignada al mismo trabajo durante el marco, porque las variables del PPL ya tienen ese valor.

5.2.2. Añadir herramientas de análisis y mejorar la visualización de los resultados

Incluir gráficas de la utilización de los procesadores, el trabajo solicitado por unidad de tiempo y el trabajo efectivo realizado por unidad de tiempo.

5.2.3. Mejora en la documentación de *Tertimuss*

Añadir páginas de documentación autogenerada a partir de los comentarios que ya hay en el código, que sirva como documentación de los API de manera análoga a como se tiene en los proyectos de Java usando herremientas como *javadoc*.

5.2.4. Distribución de *Tertimuss* a través de *PyPi*

Dado que *Tertimuss* ya está empaquetado y se trata de una herramienta relevante para diversos grupos de investigación, convendría que estuviese disponible directamente en los repositorios de *PyPi* [34] en lugar de tener que clonar el repositorio de *GitHub* para su instalación.

5.2.5. Implementación de utilidad para la conversión de medidas de tiempo

Dado que algunos planificadores como DCE o RUN necesitan que las unidades de tiempo sean números exactos, y debido a que la medida de tiempo utilizada en el API son los segundos, sería conveniente proveer de una utilidad para la conversión de los valores de tiempo a medidas exactas de forma automática, almacenando internamente el submúltiplo de los segundos que se ha aplicado, para permitir así la posterior transformación tanto de los valores de los periodos como de las frecuencias calculadas.

5.2.6. Importación y exportación de objetos de *Tertimuss*

Se trata de desarrollar un API con el que exportar e importar los objetos de uso más común, como podrían ser los objetos de la clase *TaskSet* (para los cuales sería interesante ofrecer métodos de importación y exportación en formato *JSON*) y los objetos de la clase *RawSimulationResult* (para los cuales convendría más implementar métodos para importarlos y exportarlos en formato binario, como puede ser *PICKLE*, pero además dando soporte a compresión y descompresión *LZMA* debido al tamaño de almacenamiento que éstos pueden llegar a tener).

5.2.7. Formalización de los casos de prueba

Con el fin de estandarizar los tests de los planificadores, podría facilitarse una utilidad para la generación de documentación de forma automática, en forma de comentarios de código, con respecto al conjunto de tareas y la plataforma en la que se están probando las clases. Así, sería más entendible y mantenible la parte de integración continua de la herramienta.

5.2.8. Jerarquía de clases con las que modelar los casos de error

Con el fin de facilitar la gestión de errores en las diferentes etapas de la simulación, convendría crear una jerarquía de clases que permitiesen modelar los errores que se pueden dar durante la misma, tanto en la fase en línea como en la fuera de línea.

5.2.9. Ampliaciones mayores en cuanto a las funcionalidades de la herramienta

A diferencia de las posibles líneas de trabajo futuro mencionadas antes, éstas requerirían de un fuerte impulso tanto teórico como práctico para su correcta ejecución:

Modelado de entrada/salida y secciones críticas

Por una parte, habría que considerar un modelo teórico sobre el cual basar la implementación que se propusiese, y por otra, integrar su definición en la implementación existente tanto de la simulación como de la definición y el análisis de tareas en *Tertimuss*, lo cual podría suponer un rediseño completo de la herramienta.

Generación automática del código del ejecutivo cíclico

Consistiría en la generación automática del código que implementa los cambios de contexto en la etapa en línea correspondientes al ejecutivo cíclico.

Para poder implementarlo, sería necesario proporcionar otros componentes del sistema operativo TR tales como la rutina principal, las rutinas de servicio a interrupción o la definición de las estructuras de datos para la gestión de tareas.

Además, tanto la ejecución del planificador en sí como la de esos otros componentes afectarían a los tiempos obtenidos en la planificación, y habría que introducir el tiempo de cambio de contexto y las rutinas de servicio a interrupción en los resultados, siendo éstos además dependientes de la plataforma en la cual se quiera desplegar el sistema TR. Esto dependerá de la arquitectura objetivo el número de ciclos que lleve cada operación, y habría que caracterizarlo.

Bibliografía

[1] Calvin Deutschbein, Tom Fleming, Alan Burns, and Sanjoy Baruah. Multi-core cyclic executives for safety-critical systems. *Science of Computer Programming*, 172:102–116, 2019.

[2] Laura Elena Rubio-Anguiano, Abel Chils Trabanco, José Luis Briz Velasco, and Antonio Ramírez-Treviño. Maximizing utilization and minimizing migration in thermal-aware energy-efficient real-time multiprocessor scheduling. *IEEE Access*, 9:83309–83328, 2021.

[3] Paul Regnier, George Lima, Ernesto Massa, Greg Levin, and Scott Brandt. Run: Optimal multiprocessor real-time scheduling via reduction to uniprocessor. In *Proceedings of the 2011 IEEE 32Nd Real-Time Systems Symposium*, RTSS '11, pages 104–115, Washington, DC, USA, 2011. IEEE Computer Society.

[4] Abel Chils Trabanco, Laura Rubio-Anguiano, Gaddiel Desirena, José Luis Briz Velasco, Antonio Ramírez-Treviño, and Adrián Martín. Tertimuss: Simulation environment for real-time multiprocessor schedulers, 2019-2022. `https://gaz.i3a.es/tertimuss-simulation-environment-for-thermal-aware-real-time-scheduling/`.

[5] Laura E. Rubio-Anguiano, José Luis Briz, and Antonio Ramírez-Treviño. Accounting for preemption and migration costs in the calculation of hard real-time cyclic executives for mpsocs. *IEEE Robotics and Automation Letters*, 7(3):7990–7997, 2022.

[6] Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I. Davis. A comprehensive survey of industry practice in real-time systems. *Real-Time Syst.*, 58(3):358–398, sep 2022.

[7] Laurent Perron Garrett R Dowdy. Incidencia relativa al fallo en ortools que impide utilizar gurobi, 2022. `https://github.com/google/or-tools/issues/3293`.

[8] Visual studio code. `https://code.visualstudio.com/`.

[9] Git: free and open source distributed version control system. https://git-scm.com.

[10] Github. https://github.com/.

[11] Python extension for visual studio code. https://marketplace.visualstudio.com/items?itemName=ms-python.python.

[12] Overleaf, the online latex editor. https://www.overleaf.com.

[13] Tecnología de reconocimiento óptico de caracteres (ocr) integrada en google docs. https://support.google.com/drive/answer/176692.

[14] dagrams.net: Security-first diagramming for teams. https://www.diagrams.net/.

[15] Ieee technical commitee on real-time systems: Terminology and notation. https://cmte.ieee.org/tcrts/education/terminology-and-notation/.

[16] S. Baruah, M. Bertogna, and G. Butazzo. *Multiprocessor Scheduling for Real-Time Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2015.

[17] Jane W. S. W. Liu. *Real-Time Systems*. Prentice Hall PTR, USA, 1st edition, 2000.

[18] Susanna S. Epp. *Discrete Mathematics with Applications*. Brooks/Cole Publishing Co., USA, 4th edition, 2010.

[19] Documentación del api de ortools para la definición de problemas de optimización. https://google.github.io/or-tools/python/ortools/linear_solver/pywraplp.html.

[20] Documentación sobre los backends soportados por ortools. https://developers.google.com/optimization/lp/lp_advanced.

[21] Glop optimizer. https://developers.google.com/optimization/lp/glop.

[22] Código fuente de glop optimizer. https://github.com/google/or-tools/tree/stable/ortools/glop.

[23] Gurobi optimizer. https://www.gurobi.com/products/gurobi-optimizer/.

[24] Licencias para uso académico de gurobi optimizer. https://www.gurobi.com/academia/academic-program-and-licenses/.

[25] Paquete de python con una versión limitada del solucionador gurobi. `https://pypi.org/project/gurobipy/`.

[26] Robert McNaughton. Scheduling with deadlines and loss functions. *Manage. Sci.*, 6(1):1–12, oct 1959.

[27] R. I. Davis and A. Burns. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In *2009 30th IEEE Real-Time Systems Symposium*, pages 398–409, 2009.

[28] Ieee standard for information technology - portable operating system interface (posix(r)). *IEEE Std 1003.1-2008 (Revision of IEEE Std 1003.1-2004)*, pages 1–3874, 2008.

[29] Bash is the gnu project's shell—the bourne again shell. `https://www.gnu.org/software/bash/`.

[30] Implementación del algoritmo lzma como parte de la herramienta xz. `https://tukaani.org/xz/`.

[31] Pickle es el formato estándar de serialización de objetos de python. `https://docs.python.org/3/library/pickle.html`.

[32] Dong-Ik Oh and T. P. Bakker. Utilization bounds for n-processor rate monotone scheduling with static processor assignments. *Real-Time Systems*, 15(2):183–192, 1998.

[33] L Rubio-Anguiano, G Desirena-López, A Ramírez-Treviño, and JL Briz. Energy-efficient thermal-aware multiprocessor scheduling for real-time tasks using TCPN. *Discrete Event Dynamic Systems*, pages 1–28, 2019.

[34] Repositorio oficial de paquetes de python. `https://pypi.org/`.

[35] Abel Chils Trabanco, Laura Rubio-Anguiano, Gaddiel Desirena, José Luis Briz Velasco, Antonio Ramírez-Treviño, and Adrián Martín. Tertimuss development guide, 2019-2022. `https://github.com/uz-gaz/Tertimuss/blob/master/docs/development/index.md`.

[36] F. Zhang. Improvement to semi-partitioned cyclic executives for mixed-criticality scheduling on multiprocessor platforms. *IEEE Access*, 8:223606–223617, 2020.

[37] Sitio con la información de las licencias de uso académico de gurobi optimizer de las que se dispone. https://www.gurobi.com/downloads/free-academic-license/.

[38] Sitio web con las guías de instalación de gurobi en todas las plataformas soportadas. https://www.gurobi.com/documentation/quickstart.html.

Siglas

AlECS Allocation and Execution Control Scheduler. VI, 13, 14, 39, 42–45, 49–51, 61

API Application Programming Interface. 52, 96

BPP Bin-packing problem. 12

CAlECS Clustered Allocation and Execution Control Scheduler. III, VI, 14, 39, 42–45, 47, 49–51, 61

DCE Deutschbein Cyclic Executive. I, VI, 4, 5, 14, 15, 17, 19, 39–44, 46, 47, 49, 50, 52, 61, 62, 87, 89, 93, 94

EC Ejecutivo Cíclico. III, 4, 13, 14, 44

MPSoC Multiprocessor System on a Chip. 3, 50

NP Nondeterministic Polynomial. 12, 50

PPL Problema de Programación Lineal. I, VI, 5, 14–16, 19, 20, 22, 24–30, 32, 33, 35–38, 40, 42, 43, 47, 50, 51, 61–63, 67, 70, 73–76, 83, 87, 93, 94, 97, 98

PPLE Problema de Programación Lineal Entera. 13, 14, 25, 50, 51

PPLM Problema de Programación Lineal Mixta. 25–27, 33, 35, 36, 39, 86

RM Rate Monotonic. 42

RUN Reduction to Uniprocessor. III, VI, 14, 39, 42–45, 47, 49, 52, 61

TFG Trabajo de Fin de Grado. I, III, 3–5, 14, 15, 50, 51, 93, 97

TR Tiempo Real. 3, 4, 8, 15, 53, 97, 98

TRC Tiempo Real Crítico. 49

UML Unified Modeling Language. 5, 89

WCET Worst Case Execution Time. 8, 9, 19, 26, 30, 31, 81

Lista de Figuras

3.1. Posibles situaciones en función del valor de α en la ecuación descrita 3.4. En (a) y (b) se tiene que $W_{jk}(F) \leq W_e(F)$ y por lo tanto el trabajo solicitado es atendido por el procesador en un intervalo de tiempo de duración F. En (c) se tiene que $W_{jk}(F) > W_e(F)$, y por lo tanto el procesador no es capaz de sacar a delante la carga de trabajo que se le ha solicitado 22

3.2. Secuenciación en la ejecución de un trabajo a lo largo de un marco del ejecutivo 24

3.3. Aplicación de la regla de *McNaughton* que da como resultado la paralelización de la ejecución de un trabajo al no cumplirse la tercera restricción 25

3.4. Asignación no expulsiva de un mismo trabajo a lo largo de dos marcos del ejecutivo cíclico 27

3.5. Asignaciones igualmente válidas al aplicar directamente la solución del PPL 36

3.6. Aplicación errónea (a) y acertada (b) de la regla de *McNaughton* en el caso no expulsivo para una misma asignación 36

4.1. Frecuencia mínima requerida por el planificador DCE sin expulsión permitida usando los solucionadores GLOP (a) y Gurobi (b) 41

4.2. Número de procesadores utilizados por el planificador DCE sin expulsión asignando las tareas con la regla de McNaughton, usando los solucionadores GLOP (a) y Gurobi (b) 43

4.3. Utilización del multiprocesador por el planificador DCE sin expulsión usando los solucionadores GLOP (a) y Gurobi (b) 44

4.4. Expulsiones (izquierda) y migraciones (derecha) por trabajo al utilizar los planificadores (de arriba abajo) AlECS, CAlECS y RUN 45

4.5. Expulsiones (izquierda) y migraciones (derecha) por trabajo al utilizar los planificadores DCE con expulsión permitida, usando los solucionadores GLOP (a,b) y Gurobi (c,d) 46

B.1. Asignación de los trabajos a los marcos que supone la solución al PPL . 74
B.2. Uso directo de la solución al PPL como asignación de los trabajos a los procesadores 75
B.3. Reordenación de la asignación de trabajos a los procesadores dada por la solución al PPL que evita la paralelización de su ejecución 76
B.4. Ordenación de los trabajos a asignar por orden ascendente de su identificador 78
B.5. Se empieza acomodando j_1 entero en CPU 1, al caber sus 4 u.c. en él (a), después al comprobar que las 6 u.c. de j_2 no caben enteras en el espacio restante de CPU 1 (b), éstas se parten entre CPU 1 y CPU 2 (c), a continuación se asignan las 2 u.c. de j_3 en el espacio que quedaba en CPU 2 (d) y por último se acomodan las 5 u.c. de j_4 en CPU 3 (e) . 79
B.6. Asignación óptima en migraciones de los trabajos que deben ejecutarse durante el marco ϕ_{k_o} sobre los procesadores 80

C.1. Figura 2 del artículo de referencia [1] 82
C.2. Asignación de los trabajos a los procesadores en la planificación resultante de la aplicación de DCE en el caso expulsivo 87
C.3. Asignación de los trabajos a los procesadores en la planificación resultante de la aplicación de DCE en el caso no expulsivo usando la solución del PPL 87
C.4. Asignación de los trabajos a los procesadores en la planificación resultante de la aplicación de DCE en el caso no expulsivo usando la regla de *McNaughton* 87

D.1. Diagrama de actividades de la etapa fuera de línea del planificador *Deutschbein CE* 90
D.2. Diagrama de estados de la simulación del planificador *Deutschbein CE* en *Tertimuss* 91

G.1. Diagrama de Gantt del proyecto 98

Lista de Tablas

4.1. Tiempos de ejecución de los experimientos 41

A.1. Tabla con la ejecución asignada en cada uno de los marcos a partir de los valores de las variables en la solución al PPL 67
A.2. Error acumulado en cada marco al considerar la parte por encima de la asignación de los trabajos que se ejecutan en ellos como valor para la discretización 70

C.1. Inecuaciones correspondientes a la primera restricción del problema de programación lineal sobre el sistema de tareas dado como ejemplo en el artículo [1] 84
C.2. Inecuaciones correspondientes a la segunda restricción del problema de programación lineal sobre el sistema de tareas dado como ejemplo en el artículo [1] 84
C.3. Inecuaciones correspondientes a la tercera restricción del problema de programación lineal sobre el sistema de tareas dado como ejemplo en el artículo [1] 85

G.1. Tabla de horas que se estima que se han dedicado a cada actividad del proyecto 97

Anexos

Anexos A

Ejemplo de discretización de la solución del PPL

Supongamos un sistema de tareas que genera cuatro trabajos que deben ejecutarse sobre una plataforma de cómputo con dos procesadores, para el cual se ha obtenido como solución al PPL el valor $f = 12{,}5u.c.$ y el siguiente reparto en los tres marcos que supondrá el ejecutivo cíclico:

ϕ_1	ϕ_2	ϕ_3	Siendo:
$x_1 \cdot c_1 = 7{,}1u.c.$	$x_1 \cdot c_1 = 2{,}1u.c.$		$c_1 = 10u.c.$
$x_2 \cdot c_2 = 9{,}3u.c.$	$x_2 \cdot c_2 = 5{,}7u.c.$		$c_2 = 15u.c.$
	$x_3 \cdot c_3 = 7{,}8u.c.$	$x_3 \cdot c_3 = 12{,}2u.c.$	$c_3 = 20u.c.$
$x_4 \cdot c_4 = 8{,}6u.c.$	$x_4 \cdot c_4 = 8{,}6u.c.$	$x_4 \cdot c_4 = 12{,}8u.c.$	$c_4 = 30u.c.$
$\sum = 25u.c.$	$\sum = 25u.c.$	$\sum = 25u.c.$	

Tabla A.1: Tabla con la ejecución asignada en cada uno de los marcos a partir de los valores de las variables en la solución al PPL

A.1. Aplicación de la primera mejora

Se describen a continuación los pasos para la discretización de la solución del PPL utilizando la técnica descrita en la Sec. 3.3.2.

1. Valor inicial para la discretización de f

   $f \approx \lceil f \rceil = \lceil 12{,}5 \rceil = 13u.c.$

2. Inicialización del vector $not\_assigned$

   - $not\_assigned_1 = c_1 = 10u.c.$
   - $not\_assigned_2 = c_2 = 15u.c.$
   - $not\_assigned_3 = c_3 = 20u.c.$

- $not\_assigned_4 = c_4 = 30u.c.$

3. Discretización de la asignación de ciclos a cada trabajo usando la aproximación que supone la primera mejora

   - En ϕ_1

     - Para j_1, $x_1 \cdot c_1 \approx \lceil x_1 \cdot c_1 \rceil = 8u.c.$
       porque $\lceil x_1 \cdot c_1 \rceil = \lceil 7{,}1 \rceil = 8 < 10 = not\_assigned_1$
       y $not\_assigned_1 = 10 - 8 = 2u.c.$
       $8 \leq 13$✓, luego la tercera restricción se cumple
     - Para j_2, $x_2 \cdot c_2 \approx \lceil x_2 \cdot c_2 \rceil = 10u.c.$
       porque $\lceil x_2 \cdot c_2 \rceil = \lceil 9{,}3 \rceil = 10 < 15 = not\_assigned_2$
       y $not\_assigned_2 = 15 - 10 = 5u.c.$
       $10 \leq 13$✓, luego la tercera restricción se cumple
     - Para j_4, $x_4 \cdot c_4 \approx \lceil x_4 \cdot c_4 \rceil = 9u.c.$
       porque $\lceil x_4 \cdot c_4 \rceil = \lceil 8{,}6 \rceil = 9 < 30 = not\_assigned_4$
       y $not\_assigned_4 = 30 - 9 = 21u.c.$
       $9 \leq 13$✓, luego la tercera restricción se cumple
     - Ahora, al comprobar la segunda restricción, se tiene que:
       $\lceil x_1 \cdot c_1 \rceil + \lceil x_2 \cdot c_2 \rceil + \lceil x_4 \cdot c_4 \rceil \leq m \cdot \lceil f \rceil$
       Se evalúa como $8 + 10 + 9 \leq 2 \cdot 13$
       y $27 \not\leq 26$ y por lo tanto, f debe subir a 14 u.c. para poder acomodar la cantidad de trabajo asignada en el primer marco

   - En ϕ_2

     - Para j_1, $x_1 \cdot c_1 \approx not\_assigned_1 = 2u.c.$
       porque $\lceil x_1 \cdot c_1 \rceil = \lceil 2{,}9 \rceil = 3 \not< 2 = not\_assigned_1$
       y $not\_assigned_1 = 2 - 2 = 0u.c.$
       $2 \leq 14$✓, luego la tercera restricción se cumple
     - Para j_2, $x_2 \cdot c_2 \approx not\_assigned_2 = 5u.c.$
       porque $\lceil x_2 \cdot c_2 \rceil = \lceil 5{,}7 \rceil = 6 \not< 5 = not\_assigned_2$
       y $not\_assigned_2 = 5 - 5 = 0u.c.$
       $5 \leq 14$✓, luego la tercera restricción se cumple
     - Para j_3, $x_3 \cdot c_3 \approx \lceil x_3 \cdot c_3 \rceil = 8u.c.$
       porque $\lceil x_3 \cdot c_3 \rceil = \lceil 7{,}8 \rceil = 8 < 20 = not\_assigned_3$
       y $not\_assigned_3 = 20 - 8 = 12u.c.$
       $8 \leq 14$✓, luego la tercera restricción se cumple

- Para j_4, $x_4 \cdot c_4 \approx \lceil x_4 \cdot c_4 \rceil = 9u.c.$
  porque $\lceil x_4 \cdot c_4 \rceil = \lceil 8{,}6 \rceil = 9 < 21 = not\_assigned_4$
  y $not\_assigned_4 = 21 - 9 = 12u.c.$
  $9 \leq 14$✓, luego la tercera restricción se cumple
- Ahora, al comprobar la segunda restricción, se tiene que:
  $\lceil x_1 \cdot c_1 \rceil + \lceil x_2 \cdot c_2 \rceil + \lceil x_3 \cdot c_3 \rceil + \lceil x_4 \cdot c_4 \rceil \leq m \cdot \lceil f \rceil$
  Se evalúa como $2 + 5 + 8 + 9 \leq 2 \cdot 14$
  y $24 \leq 28$✓ y por lo tanto, con el valor fijado como discretización de *f*, la inecuación se cumple de sobras. De hecho, con el valor inicial asignado como discretización de f (esto es, $f = 13$) también lo habría hecho

– En ϕ_3

- Para j_3, $x_3 \cdot c_3 \approx not\_assigned_3 = 12u.c.$
  porque $\lceil x_3 \cdot c_3 \rceil = \lceil 12{,}2 \rceil = 13 \not< 12 = not\_assigned_3$
  y $not\_assigned_3 = 12 - 12 = 0u.c.$
  $12 \leq 14$✓, luego la tercera restricción se cumple
- Para j_4, $x_4 \cdot c_4 \approx not\_assigned_4 = 12u.c.$
  porque $\lceil x_4 \cdot c_4 \rceil = \lceil 12{,}8 \rceil = 13 \not< 12 = not\_assigned_4$
  y $not\_assigned_4 = 12 - 12 = 0u.c.$
  $12 \leq 14$✓, luego la tercera restricción se cumple
- Ahora, al comprobar la segunda restricción, se tiene que:
  $\lceil x_3 \cdot c_3 \rceil + \lceil x_4 \cdot c_4 \rceil \leq m \cdot \lceil f \rceil$
  Se evalúa como $12 + 12 \leq 2 \cdot 14$
  y $24 \leq 28$✓ y por lo tanto, con el valor fijado como discretización de f, la inecuación se cumple de sobras. De hecho, con el valor inicial asignado como discretización de f (esto es, f=13) también lo habría hecho

En este ejemplo se puede observar claramente como la sobreasignación de ciclos a los trabajos planificados en el primer marco supone sobredimensionar el valor de f hasta un punto que supone un derroche de recursos en el resto de marcos

Esto se debe a que en las primeras discretizaciones de la asignación de ciclos de un trabajo, la parte entera por encima del valor decimal siempre (o casi siempre) supone una cantidad menor que el número de ciclos que quedan por asignar para ese trabajo, y al empezar siempre por el primer marco sin estimar el error que esa aproximación supone, se corre el riesgo de sobredimensionar el valor de f hasta cotas excesivas para lo que es el requerimiento de cómputo real en el resto de marcos

A.2. Aplicación de la primera y la segunda mejora

Se describen a continuación los pasos para la discretización de la solución del PPL utilizando la técnica descrita en la Sec. 3.3.3.

1. Valor inicial para la discretización de f

   $f \approx \lceil f \rceil = \lceil 12{,}5 \rceil = 13 u.c.$

2. Inicialización del vector $not\_assigned$

   - $not\_assigned_1 = c_1 = 10 u.c.$
   - $not\_assigned_2 = c_2 = 15 u.c.$
   - $not\_assigned_3 = c_3 = 20 u.c.$
   - $not\_assigned_4 = c_4 = 30 u.c.$

3. Elección del orden en que debe realizarse la discretización de los marcos; para ello, se obtiene la suma del error que supone en cada marco el tomar como discretización de la asignación de sus trabajos la parte entera por arriba de los valores decimales:

ϕ_1	ϕ_2	ϕ_3
$\lceil x_1 \cdot c_1 \rceil - x_1 \cdot c_1 =$ $\lceil 7{,}1 \rceil - 7{,}1 =$ $8 - 7{,}1 = 0{,}9$	$\lceil x_1 \cdot c_1 \rceil - x_1 \cdot c_1 =$ $\lceil 2{,}9 \rceil - 2{,}9 =$ $3 - 2{,}9 = 0{,}1$	
$\lceil x_2 \cdot c_2 \rceil - x_2 \cdot c_2 =$ $\lceil 9{,}3 \rceil - 9{,}3 =$ $10 - 9{,}3 = 0{,}7$	$\lceil x_2 \cdot c_2 \rceil - x_2 \cdot c_2 =$ $\lceil 5{,}7 \rceil - 5{,}7 =$ $6 - 5{,}7 = 0{,}3$	
	$\lceil x_3 \cdot c_3 \rceil - x_3 \cdot c_3 =$ $\lceil 7{,}8 \rceil - 7{,}8 =$ $8 - 7{,}8 = 0{,}2$	$\lceil x_3 \cdot c_3 \rceil - x_3 \cdot c_3 =$ $\lceil 12{,}2 \rceil - 12{,}2 =$ $13 - 12{,}2 = 0{,}8$
$\lceil x_4 \cdot c_4 \rceil - x_4 \cdot c_4 =$ $\lceil 8{,}6 \rceil - 8{,}6 =$ $9 - 8{,}6 = 0{,}4$	$\lceil x_4 \cdot c_4 \rceil - x_4 \cdot c_4 =$ $\lceil 8{,}6 \rceil - 8{,}6 =$ $9 - 8{,}6 = 0{,}4$	$\lceil x_4 \cdot c_4 \rceil - x_4 \cdot c_4 =$ $\lceil 12{,}8 \rceil - 12{,}8 =$ $13 - 12{,}8 = 0{,}2$
$\sum_{\phi_1} = 2 u.c.$	$\sum_{\phi_2} = 1 u.c.$	$\sum_{\phi_3} = 1 u.c.$

Tabla A.2: Error acumulado en cada marco al considerar la parte por encima de la asignación de los trabajos que se ejecutan en ellos como valor para la discretización

Es decir, el mayor error se comete al empezar a discretizar por el marco 1 (se añaden 2 u.c. a la asignación real de los trabajos), mientras que el menor es cuando se comienza por el marco 2 o el 3 (solo se añade 1 u.c. a la asignación real de los trabajos)

Por lo tanto, el mejor orden para llevar a cabo la discretización de los marcos, es el siguiente:

a) ϕ_2

b) ϕ_3

c) ϕ_1

Nótese que sería indistinto empezar por ϕ_2 o por ϕ_3

4. Discretización de la asignación de ciclos a cada trabajo usando la asignación que supone la 1ª mejora y siguiendo el orden de marcos dado por la segunda mejora:

   – En ϕ_2

   - Para j_1, $x_1 \cdot c_1 \approx \lceil x_1 \cdot c_1 \rceil = 3u.c.$
     porque $\lceil x_1 \cdot c_1 \rceil = \lceil 2{,}9 \rceil = 3 < 10 = not\_assigned_1$
     y $not\_assigned_1 = 10 - 3 = 7u.c.$
     $3 \leq 13$✓, luego la tercera restricción se cumple
   - Para j_2, $x_2 \cdot c_2 \approx \lceil x_2 \cdot c_2 \rceil = 6u.c.$
     porque $\lceil x_2 \cdot c_2 \rceil = \lceil 5{,}7 \rceil = 6 < 15 = not\_assigned_2$
     y $not\_assigned_2 = 15 - 6 = 9u.c.$
     $6 \leq 13$✓, luego la tercera restricción se cumple
   - Para j_3, $x_3 \cdot c_3 \approx \lceil x_3 \cdot c_3 \rceil = 8u.c.$
     porque $\lceil x_3 \cdot c_3 \rceil = \lceil 7{,}8 \rceil = 8 < 20 = not\_assigned_3$
     y $not\_assigned_3 = 20 - 8 = 12u.c.$
     $8 \leq 13$✓, luego la tercera restricción se cumple
   - Para j_4, $x_4 \cdot c_4 \approx \lceil x_4 \cdot c_4 \rceil = 9u.c.$
     porque $\lceil x_4 \cdot c_4 \rceil = \lceil 8{,}6 \rceil = 9 < 30 = not\_assigned_4$
     y $not\_assigned_4 = 30 - 9 = 21u.c.$
     $9 \leq 13$✓, luego la tercera restricción se cumple
   - Ahora, al comprobar la segunda restricción, se tiene que:
     $\lceil x_1 \cdot c_1 \rceil + \lceil x_2 \cdot c_2 \rceil + \lceil x_3 \cdot c_3 \rceil + \lceil x_4 \cdot c_4 \rceil \leq m \cdot \lceil f \rceil$
     Se evalúa como $3 + 6 + 8 + 9 \leq 2 \cdot 13$
     y $26 \leq 26$✓ y por lo tanto, no es necesario aumentar el valor de f

   – En ϕ_3

   - Para j_3, $x_3 \cdot c_3 \approx not\_assigned_3 = 12u.c.$
     porque $\lceil x_3 \cdot c_3 \rceil = \lceil 12{,}2 \rceil = 13 \not< 12 = not\_assigned_3$
     y $not\_assigned_3 = 12 - 12 = 0u.c.$
     $12 \leq 13$✓, luego la tercera restricción se cumple

- Para j_4, $x_4 \cdot c_4 \approx \lceil x_4 \cdot c_4 \rceil = 13u.c.$
  porque $\lceil x_4 \cdot c_4 \rceil = \lceil 12{,}8 \rceil = 13 < 21 = not\_assigned_4$
  y $not\_assigned_4 = 21 - 13 = 8u.c.$
  $13 \leq 13$✓, luego la tercera restricción se cumple
- Ahora, al comprobar la segunda restricción, se tiene que:
  $\lceil x_3 \cdot c_3 \rceil + \lceil x_4 \cdot c_4 \rceil \leq m \cdot \lceil f \rceil$
  Se evalúa como $12 + 13 \leq 2 \cdot 13$
  y $25 \leq 26$✓ y por lo tanto, no es necesario aumentar el valor de f

– En ϕ_1

- Para j_1, $x_1 \cdot c_1 \approx not\_assigned_1 = 7u.c.$
  porque $\lceil x_1 \cdot c_1 \rceil = \lceil 7{,}1 \rceil = 8 \not< 7 = not\_assigned_1$
  y $not\_assigned_1 = 7 - 7 = 0u.c.$
  $7 \leq 13$✓, luego la tercera restricción se cumple
- Para j_2, $x_2 \cdot c_2 \approx not\_assigned_2 = 9u.c.$
  porque $\lceil x_2 \cdot c_2 \rceil = \lceil 9{,}3 \rceil = 10 \not< 9 = not\_assigned_2$
  y $not\_assigned_2 = 9 - 9 = 0u.c.$
  $9 \leq 13$✓, luego la tercera restricción se cumple
- Para j_4, $x_4 \cdot c_4 \approx not\_assigned_4 = 8u.c.$
  porque $\lceil x_4 \cdot c_4 \rceil = \lceil 8{,}6 \rceil = 9 \not< 8 = not\_assigned_4$
  y $not\_assigned_4 = 8 - 8 = 0u.c.$
  $8 \leq 13$✓, luego la tercera restricción se cumple
- Ahora, al comprobar la segunda restricción, se tiene que:
  $\lceil x_1 \cdot c_1 \rceil + \lceil x_2 \cdot c_2 + \lceil x_4 \cdot c_4 \rceil \leq m \cdot \lceil f \rceil$
  Se evalúa como $7 + 9 + 8 \leq 2 \cdot 13$
  y $24 \leq 26$✓ y por lo tanto, no es necesario aumentar el valor de f

Nótese que, al aplicar esta mejora, se ha tomado como discretización la parte entera por abajo en aquellos marcos en los que el tomar la parte entera por arriba suponía un mayor error, y así se ha logrado obtener una discretización en la que f baja un ciclo con respecto a no haber aplicado la mejora

Anexos B

Ejemplo de aplicación de la regla de McNaughton

La plataforma de cómputo consiste en un multiprocesador con 3 procesadors idénticos entre sí.

En el marco ϕ_{k_o}, que es el que se considera, hay 4 trabajos activos, que son:

- j_1, con $c_1 = 8$
- j_2, con $c_2 = 10$
- j_3, con $c_3 = 10$
- j_4, con $c_4 = 5$

La solución al PPL consiste en los siguientes valores para las variables:

- $f = 6$
- Para i = 1
  - $x_{11k_o} = 0{,}25$
  - $x_{12k_o} = 0{,}0625$
  - $x_{13k_o} = 0{,}1875$
- Para i = 2
  - $x_{21k_o} = 0{,}1$
  - $x_{22k_o} = 0{,}2$
  - $x_{23k_o} = 0{,}3$
- Para i = 3

- $x_{31k_o} = 0{,}1$
- $x_{32k_o} = 0{,}1$
- $x_{33k_o} = 0$

– Para i = 4

- $x_{41k_o} = 0{,}3$
- $x_{42k_o} = 0{,}4$
- $x_{43k_o} = 0{,}3$

.

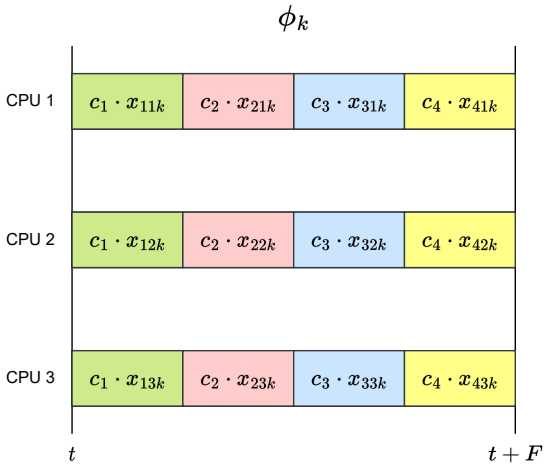


Figura B.1: Asignación de los trabajos a los marcos que supone la solución al PPL

La planificación resultante de la aplicación directa de esa solución sería la que se presenta en la Fig. B.2 (o una permutación en cada procesador de las asignaciones establecidas en ella).

Al ser:

– Para $i = 1$

- $x_{11k_o} \cdot c_1 = 0{,}25 \cdot 8 = 2$
- $x_{12k_o} \cdot c_1 = 0{,}0625 \cdot 8 = 0{,}5$
- $x_{13k_o} \cdot c_1 = 0{,}1875 \cdot 8 = 1{,}5$

– Para $i = 2$

- $x_{21k_o} \cdot c_2 = 0{,}1 \cdot 10 = 1$
- $x_{22k_o} \cdot c_2 = 0{,}2 \cdot 10 = 2$

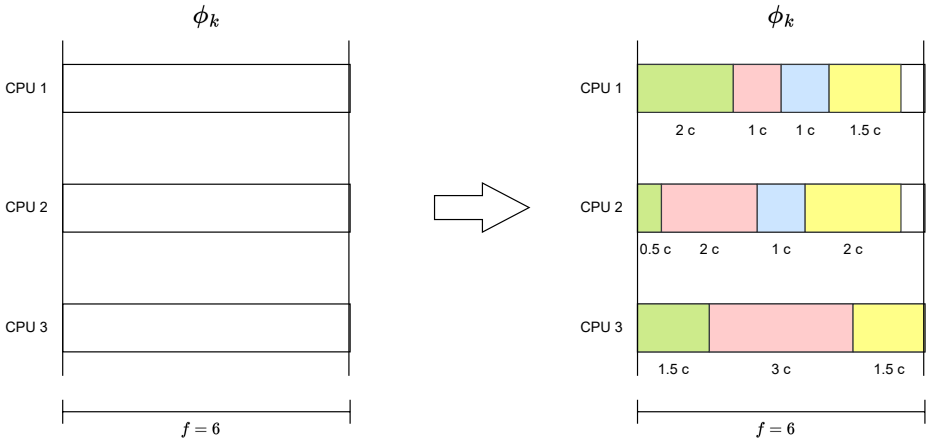


Figura B.2: Uso directo de la solución al PPL como asignación de los trabajos a los procesadores

- $x_{23k_o} \cdot c_2 = 0{,}3 \cdot 10 = 3$

– Para $i = 3$

- $x_{31k_o} \cdot c_3 = 0{,}1 \cdot 10 = 1$
- $x_{32k_o} \cdot c_3 = 0{,}1 \cdot 10 = 1$
- $x_{33k_o} \cdot c_3 = 0 \cdot 10 = 0$

– Para $i = 4$

- $x_{41k_o} \cdot c_4 = 0{,}3 \cdot 5 = 1{,}5$
- $x_{42k_o} \cdot c_4 = 0{,}4 \cdot 5 = 2$
- $x_{43k_o} \cdot c_4 = 0{,}3 \cdot 5 = 1{,}5$

No obstante, como es evidente, esa planificación no es válida tal y como se ha planteado, pues la ejecución de absolutamente todos los trabajos se están paralelizando en algún momento del marco, ya sea en dos o incluso en los tres procesadores; por lo tanto, es necesario permutar el orden en el que se ejecutan los trabajos en cada procesador para evitar ese tipo de situaciones. Esto no es trivial, y es posible que, por la forma en que estén repartidas las asignaciones, la ordenación de esos fragmentos para evitar la paralelización no sea posible. En el caso anterior, una permutación válida es la que se presenta en la Fig. B.3.

El otro problema con esta forma de generar la planificación es el elevado número de migraciones de cada trabajo entre los procesadores. Salvo en el caso de j_3, que solo migra una vez, la planificación hace que los trabajos pasen por todos los núcleos durante el marco, lo cual se traduce en que deban realizar dos migraciones. Esto se

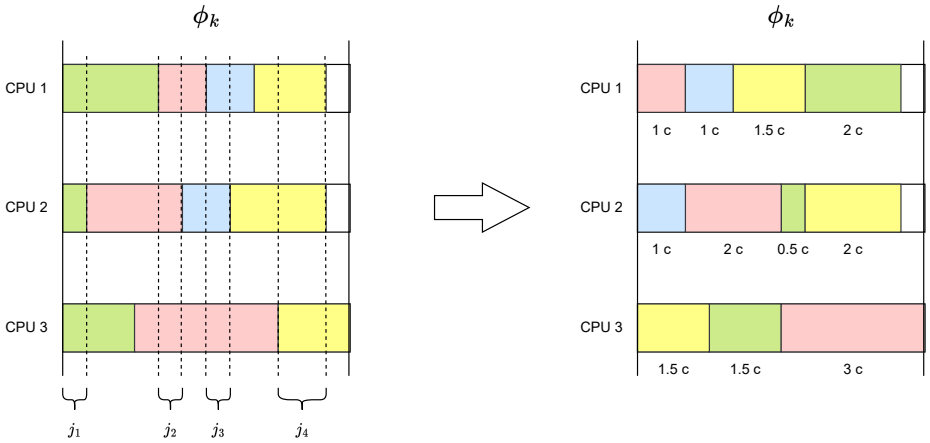


Figura B.3: Reordenación de la asignación de trabajos a los procesadores dada por la solución al PPL que evita la paralelización de su ejecución

debe a que se reparten los procesadores directamente con las fracciones asignadas por la solución del PPL. Si, por ejemplo, j_1 ejecutase las 0.5 u.c. que tiene asignadas en el procesador dos seguidas de las 2 u.c. que tiene asignadas en el procesador uno, el nº de migraciones de dicho trabajo quedaría reducido a uno. Esa idea, llevada al extremo, supone aplicar la regla de *McNaughton*.

B.1. Asignación usando la regla de *McNaughton*

B.1.1. Acumulación de los ciclos asignados a los trabajos durante el marco

El primer paso para la construcción de un ejecutivo cíclico de la manera descrita en el artículo es la obtención de los valores x_{i_o} del marco ϕ_{k_o}. Así, se suman los valores asignados como solución al PPL para las variables $x_{i_o j k_o}$, y se obtiene la fracción del tiempo de cómputo de cada trabajo que deberá ser ejecutada durante el marco ϕ_{k_o}:

- Para j_1 se tiene $\sum_{j=1}^{3} x_{1jk_o} = x_{11k_o} + x_{12k_o} + x_{13k_o} = 0{,}25 + 0{,}0625 + 0{,}1875 = 0{,}5$, por lo que $x_1 = 0{,}5$

- Para j_2 se tiene $\sum_{j=1}^{3} x_{2jk_o} = x_{21k_o} + x_{22k_o} + x_{23k_o} = 0{,}1 + 0{,}2 + 0{,}3 = 0{,}6$, por lo que $x_2 = 0{,}6$

- Para j_3 se tiene $\sum_{j=1}^{3} x_{3jk_o} = x_{31k_o} + x_{32k_o} + x_{33k_o} = 0{,}1 + 0{,}1 + 0 = 0{,}2$, por lo que $x_3 = 0{,}2$

- Para j_4 se tiene $\sum_{j=1}^{3} x_{4jk_o} = x_{41k_o} + x_{42k_o} + x_{43k_o} = 0{,}3 + 0{,}4 + 0{,}3 = 1$, por lo que $x_4 = 1$

Se comprueba además que con ellas se cumplen las dos inecuaciones planteadas en el artículo:

1. $(c_{i_o} \cdot x_{i_o}) \leq f$, para cada trabajo j_{i_o}
   - Con $i_o = 1$, se tiene que $(x_1 \cdot c_1) \leq f$ se evalúa como $(0{,}5 \cdot 8) \leq 6$, y por lo tanto se concluye que $4 \leq 6$✓
   - Con $i_o = 2$, se tiene que $(x_2 \cdot c_2) \leq f$ se evalúa como $(0{,}6 \cdot 10) \leq 6$, y por lo tanto se concluye que $6 \leq 6$✓
   - Con $i_o = 3$, se tiene que $(x_3 \cdot c_3) \leq f$ se evalúa como $(0{,}2 \cdot 10) \leq 6$, y por lo tanto se concluye que $2 \leq 6$✓
   - Con $i_o = 4$, se tiene que $(x_4 \cdot c_4) \leq f$ se evalúa como $(1 \cdot 5) \leq 6$, y por lo tanto se concluye que $5 \leq 6$✓
2. $(\sum_{i_o=1}^{4} x_{i_o} \cdot c_{i_o}) \leq m \cdot f$ para el marco ϕ_{k_o}.

   Desarrollando la expresión, se obtiene la inecuación

   $(x_1 \cdot c_1 + x_2 \cdot c_2 + x_3 \cdot c_3 + x_4 \cdot c_4) \leq 18$

   Que se evalúa como

   $(0{,}5 \cdot 8 + 0{,}6 \cdot 10 + 0{,}2 \cdot 20 + 1 \cdot 5) \leq 18$

   Y por lo tanto, al tenerse $(4 + 6 + 2 + 5) \leq 18$, se concluye que $17 \leq 18$✓

B.1.2. Aplicación de la regla de *McNaughton* para obtener la asignación

Una vez llegado este punto, se aplica la regla de *McNaughton*:

1. Ordenar los trabajos que reciben alguna cantidad de ejecución en el marco ϕ_{k_o}
   - Como todos los trabajos considerados, $\{j_{i_o}\}_{i_o=1}^{4}$, reciben una fracción de ejecución del trabajo distinta de cero (esto es, $x_{i_o} \neq 0, i \leq i_o \leq 4$) se consideran todos ellos.
   - La ordenación se hace directamente por valor ascendente de su identificador (ver Fig B.4).
2. Acomodar los trabajos sobre los m procesadores de la forma descrita en la Sec. 3.5.1.

   Así, se parte de los contenedores (procesadores en el marco) vacíos (sin ningún trabajo asignado) y se van llenando siguiendo el orden anterior, siendo este proceso ilustrado en la Fig. B.5.

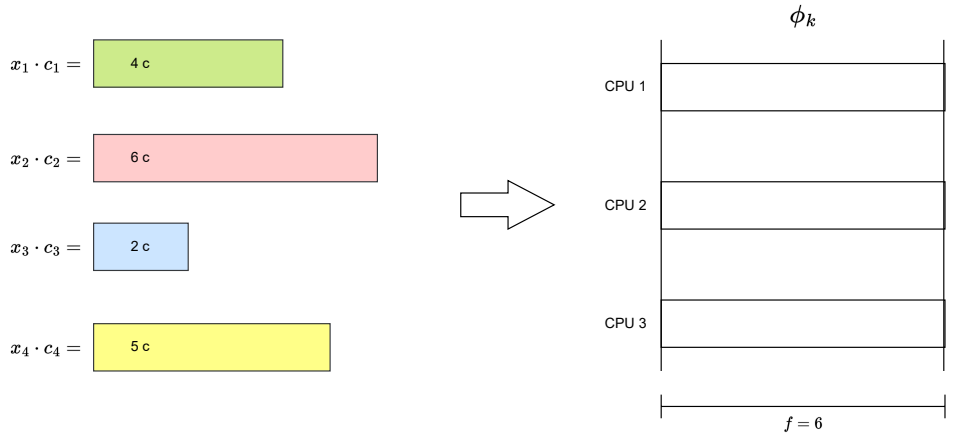


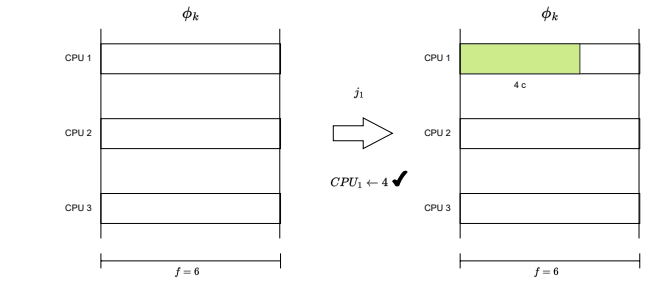
Figura B.4: Ordenación de los trabajos a asignar por orden ascendente de su identificador

Se puede observar como, aplicando (automáticamente) esta regla, se obtiene una planificación en la que no se paraleliza la ejecución de ningún trabajo y en la que, además, solo una tarea debe migrar entre procesadores durante el marco (j_2, del procesador dos al uno).

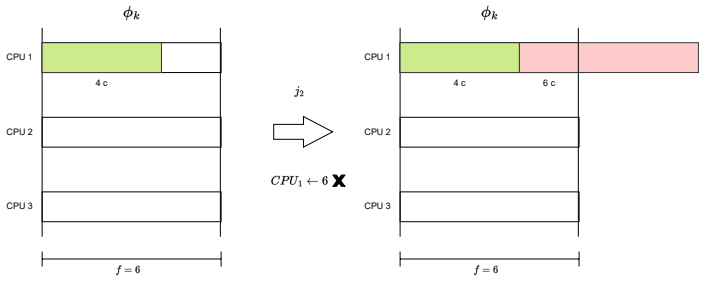
No obstante, esta planificación que se obtiene no es óptima en el número de migraciones, pues la que se muestra en la Fig. B.6 directamente no conlleva migraciones de trabajos entre procesadores.

Para obtenerla, simplemente, habría que haber considerado los trabajos en el orden (j_1, j_3, j_2, j_4) en lugar de en (j_1, j_2, j_3, j_4), es decir, primero j_3 y luego j_2 en lugar de al revés.

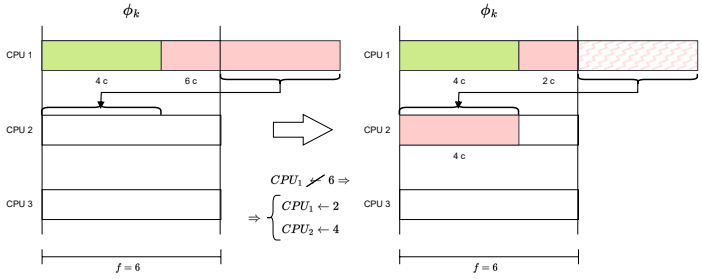
A pesar de la suboptimalidad, al aplicar esa regla se consigue que la ejecución de los trabajos no se paralelice y se disminuye considerablemente el número de migraciones. Hay que considerar que en el artículo de referencia [1] se prima la velocidad en la obtención de la planificación sobre la optimalidad en el número de migraciones en la misma.



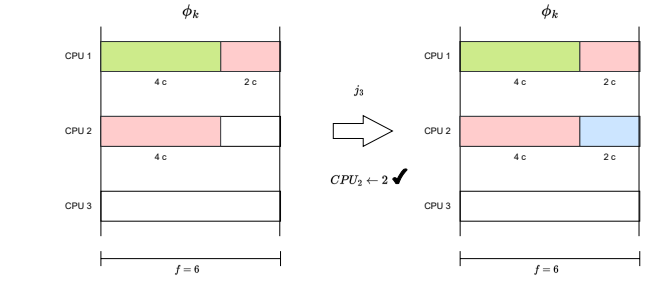
(a)



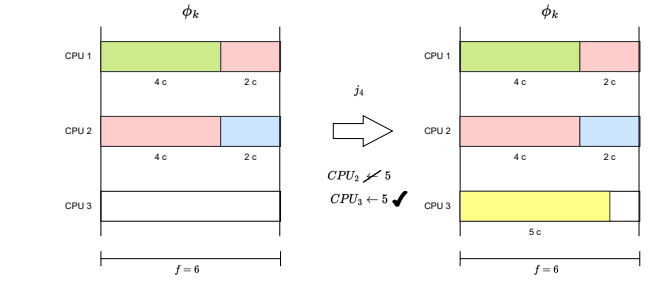
(b)



(c)



(d)



(e)

Figura B.5: Se empieza acomodando j_1 entero en CPU 1, al caber sus 4 u.c. en él (a), después al comprobar que las 6 u.c. de j_2 no caben enteras en el espacio restante de CPU 1 (b), éstas se parten entre CPU 1 y CPU 2 (c), a continuación se asignan las 2 u.c. de j_3 en el espacio que quedaba en CPU 2 (d) y por último se acomodan las 5 u.c. de j_4 en CPU 3 (e)

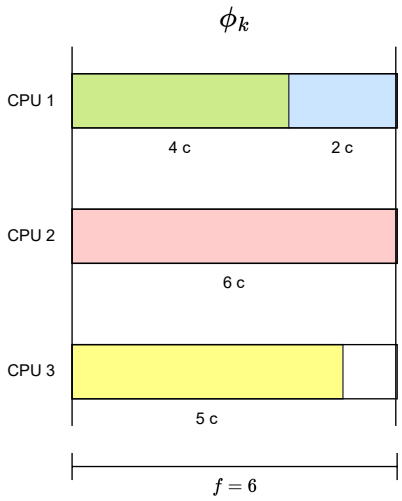


Figura B.6: Asignación óptima en migraciones de los trabajos que deben ejecutarse durante el marco ϕ_{k_o} sobre los procesadores

Anexos C

Ejemplo de funcionamiento de Deutschbein CE

La plataforma de cómputo y el sistema de tareas presentados en este ejemplo son los presentados como ejemplo vehicular en el artículo de referencia [1].

C.1. Especificación de la plataforma y el conjunto de tareas

En este ejemplo, se va a considerar una plataforma de cómputo que dispone de dos procesadores idénticos que pueden funcionar a 10 o a 40 Hz. Así, se tiene que $m = 2$.

En cuanto al conjunto de tareas, se tiene que τ es un sistema de $N = 3$ tareas tales que:

- $\tau_1 = (T_1 = 4, C_1 = 20)$
- $\tau_2 = (T_2 = 6, C_2 = 40)$
- $\tau_3 = (T_3 = 12, C_3 = 80)$

Se ha asignado 20, 40 y 80 como valores del WCET de las tareas porque en el artículo no estaban especificados, y dado que se quiere mostrar un ejemplo completo, incluyendo las diferentes planificaciones resultantes, era necesario hacerlo.

En cuanto al valor del ciclo mayor, se tiene que $P = lcm(4, 6, 12) = 12$, y para el ciclo menor, que $F = gcm(4, 6, 12) = 2$. Por lo tanto, en un hiperperiodo del ejecutivo cíclico (duración 12) habrá $\frac{P}{F} = \frac{12}{2} = 6$ marcos.

A partir de la especificación del conjunto de tareas, se puede saber que éstas generarán un número de trabajos $n = \sum_{i=1}^{N} \frac{P}{T_i} = \sum_{i=1}^{3} \frac{P}{T_i} = \frac{P}{T_1} + \frac{P}{T_2} + \frac{P}{T_3} = \frac{12}{4} + \frac{12}{6} + \frac{12}{12} = 3 + 2 + 1 = 6$ durante el intervalo $[0, P) = [0, 12]$.

Así, se tendrá que cada una de las tareas generará:

– En el caso de τ_1

- $j_1 = (a_1 = 0, c_1 = C_1 = 20, d_1 = a_1 + T_1 = 4)$
- $j_2 = (a_2 = 0, c_2 = C_1 = 20, d_2 = a_2 + T_1 = 8)$
- $j_3 = (a_3 = 0, c_3 = C_1 = 20, d_3 = a_3 + T_1 = 12)$

– En el caso de τ_2

- $j_4 = (a_4 = 0, c_4 = C_2 = 40, d_4 = a_4 + T_2 = 6)$
- $j_5 = (a_5 = 0, c_5 = C_2 = 40, d_5 = a_5 + T_2 = 12)$

– En el caso de τ_3

- $j_6 = (a_6 = 0, c_6 = C_3 = 80, d_6 = a_6 + T_3 = 12)$

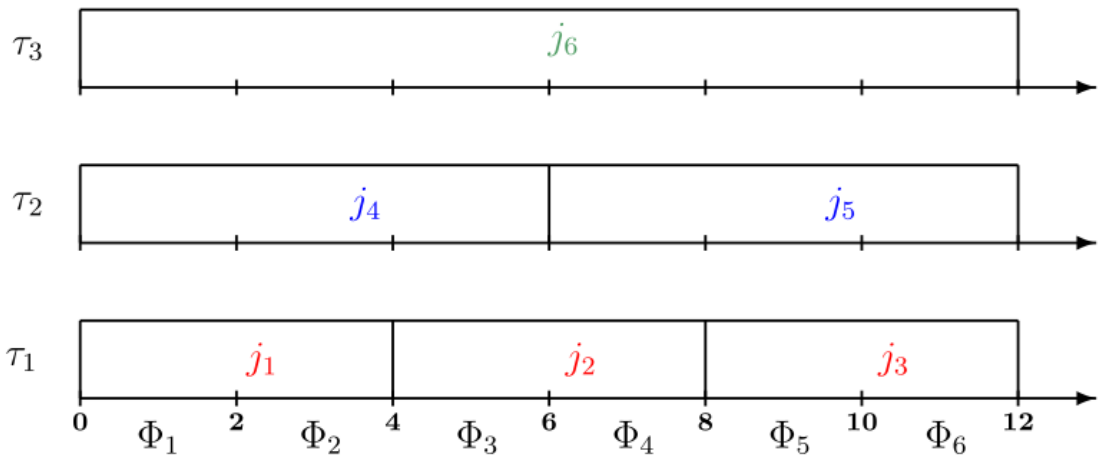


Figura C.1: Figura 2 del artículo de referencia [1]

C.2. Definición del problema de programación lineal

C.2.1. Definición de las variables

A la hora de generar las variables ligadas, por ejemplo, al trabajo 3, j_3, ha de tenerse en cuenta que éste solo se puede ejecutar en los marcos 5 y 6, ϕ_5 y ϕ_6, del ejecutivo.

Dado que la plataforma de ejecución tiene 2 núcleos ($m = 2$), las variables definidas por el trabajo 3, j_3, serán:

– x_{315}: fracción de j_3 ejecutada en el procesador 1 durante el marco ϕ_5.

– x_{316}: fracción de j_3 ejecutada en el procesador 1 durante el marco ϕ_6.

– x_{325}: fracción de j_3 ejecutada en el procesador 2 durante el marco ϕ_5.

– x_{326}: fracción de j_3 ejecutada en el procesador 2 durante el marco ϕ_6.

A continuación, se presenta el listado de todas las variables que deberán definirse en el Problema de Programación Lineal (PPL) a plantear para el sistema de tareas y la plataforma especificados:

– Para j_1, se tienen $x_{111}, x_{112}, x_{121}$ y x_{122}.

– Para j_2, se tienen x_{213}, x_{214}, x_{223} y x_{224}

– Para j_3, se tienen x_{315}, x_{316}, x_{325} y x_{326}

– Para j_4, se tienen x_{411}, x_{412}, x_{413}, x_{421}, x_{422} y x_{423}

– Para j_5, se tienen x_{514}, x_{515}, x_{516}, x_{524}, x_{525} y x_{526}

– Para j_6, se tienen x_{611}, x_{612}, x_{613}, x_{614}, x_{615}, x_{616}, x_{621}, x_{622}, x_{623}, x_{624}, x_{625} y x_{626}

Esto da un total de $N \cdot m \cdot \frac{P}{F} = 3 \cdot 2 \cdot \frac{12}{2} = 3 \cdot 2 \cdot 6 = 36$ variables.

C.2.2. Definición de las restricciones

A continuación, las treinta y seis restricciones aparecen agrupadas por las tres restricciones del PPL las que se generan, y para cada una de ellas se define:

– El identificador (absoluto) de la restricción.

– El valor de índice de la variable x_{ijk} que supone su instanciación.

– La inecuación de la restricción de la sustitución de los índices.

– La inecuación de la tras desarrollar el/los sumatorios.

Restricción 1

$n = 6$ restricciones obtenidas al instanciar
$(\sum_{j=1}^{m} \sum_{k=\frac{a_i}{F}+1}^{\frac{d_i}{F}} x_{ijk}) = 1$, para cada i, $1 \leq i \leq n$
que aparecen recogidas en la Tab. C.1.

Restricción 2

$m \cdot \frac{P}{F} =_{m=2,P=12,F=2} 2 \cdot \frac{12}{2} = 12$ restricciones obtenidas al instanciar
$(\sum_{i \in Trabajos\_in\_Marco_k} x_{ijk} \cdot c_i) \leq f$, para cada j, $1 \leq j \leq m$, y cada k, $1 \leq k \leq \frac{P}{F}$
que aparecen recogidas en la Tab. C.2.

id	índice(s) de x_{ijk} fijo(s)	inecuación resultante	inecuación desarrollada
1	i = 1	$\sum_{j=1}^{2}\sum_{k=1}^{2} x_{1jk} = 1$	$x_{111} + x_{112} + x_{121} + x_{122} = 1$
2	i = 2	$\sum_{j=1}^{2}\sum_{k=3}^{4} x_{2jk} = 1$	$x_{213} + x_{214} + x_{223} + x_{224} = 1$
3	i = 3	$\sum_{j=1}^{2}\sum_{k=5}^{6} x_{3jk} = 1$	$x_{315} + x_{316} + x_{325} + x_{326} = 1$
4	i = 4	$\sum_{j=1}^{2}\sum_{k=1}^{3} x_{4jk} = 1$	$x_{411} + x_{412} + x_{413} + x_{421} + x_{422} + x_{423} = 1$
5	i = 5	$\sum_{j=1}^{2}\sum_{k=4}^{6} x_{5jk} = 1$	$x_{514} + x_{515} + x_{516} + x_{524} + x_{525} + x_{526} = 1$
6	i = 6	$\sum_{j=1}^{2}\sum_{k=1}^{6} x_{6jk} = 1$	$x_{611} + x_{612} + x_{613} + x_{614} + x_{615} + x_{616} +$ $x_{621} + x_{622} + x_{623} + x_{624} + x_{625} + x_{626} = 1$

Tabla C.1: Inecuaciones correspondientes a la primera restricción del problema de programación lineal sobre el sistema de tareas dado como ejemplo en el artículo [1]

id	índice(s) de x_{ijk} fijo(s)	inecuación resultante	inecuación desarrollada
7	j = 1, k = 1	$\sum_{i\in\{1,4,6\}} x_{i11} \cdot c_i \leq f$	$20{,}0 \cdot x_{111} + 40{,}0 \cdot x_{411} + 80{,}0 \cdot x_{611} \leq f$
8	j = 1, k = 2	$\sum_{i\in\{1,4,6\}} x_{i12} \cdot c_i \leq f$	$20{,}0 \cdot x_{112} + 40{,}0 \cdot x_{412} + 80{,}0 \cdot x_{612} \leq f$
9	j = 1, k = 3	$\sum_{i\in\{2,4,6\}} x_{i13} \cdot c_i \leq f$	$20{,}0 \cdot x_{213} + 40{,}0 \cdot x_{413} + 80{,}0 \cdot x_{613} \leq f$
10	j = 1, k = 4	$\sum_{i\in\{2,5,6\}} x_{i14} \cdot c_i \leq f$	$20{,}0 \cdot x_{214} + 40{,}0 \cdot x_{514} + 80{,}0 \cdot x_{614} \leq f$
11	j = 1, k = 5	$\sum_{i\in\{3,5,6\}} x_{i15} \cdot c_i \leq f$	$20{,}0 \cdot x_{315} + 40{,}0 \cdot x_{515} + 80{,}0 \cdot x_{615} \leq f$
12	j = 1, k = 6	$\sum_{i\in\{3,5,6\}} x_{i16} \cdot c_i \leq f$	$20{,}0 \cdot x_{316} + 40{,}0 \cdot x_{516} + 80{,}0 \cdot x_{616} \leq f$
13	j = 2, k = 1	$\sum_{i\in\{1,4,6\}} x_{i21} \cdot c_i \leq f$	$20{,}0 \cdot x_{121} + 40{,}0 \cdot x_{421} + 80{,}0 \cdot x_{621} \leq f$
14	j = 2, k = 2	$\sum_{i\in\{1,4,6\}} x_{i22} \cdot c_i \leq f$	$20{,}0 \cdot x_{122} + 40{,}0 \cdot x_{422} + 80{,}0 \cdot x_{622} \leq f$
15	j = 2, k = 3	$\sum_{i\in\{2,4,6\}} x_{i23} \cdot c_i \leq f$	$20{,}0 \cdot x_{223} + 40{,}0 \cdot x_{423} + 80{,}0 \cdot x_{623} \leq f$
16	j = 2, k = 4	$\sum_{i\in\{2,5,6\}} x_{i24} \cdot c_i \leq f$	$20{,}0 \cdot x_{224} + 40{,}0 \cdot x_{524} + 80{,}0 \cdot x_{624} \leq f$
17	j = 2, k = 5	$\sum_{i\in\{3,5,6\}} x_{i25} \cdot c_i \leq f$	$20{,}0 \cdot x_{325} + 40{,}0 \cdot x_{525} + 80{,}0 \cdot x_{625} \leq f$
18	j = 2, k = 6	$\sum_{i\in\{3,5,6\}} x_{i26} \cdot c_i \leq f$	$20{,}0 \cdot x_{326} + 40{,}0 \cdot x_{526} + 80{,}0 \cdot x_{626} \leq f$

Tabla C.2: Inecuaciones correspondientes a la segunda restricción del problema de programación lineal sobre el sistema de tareas dado como ejemplo en el artículo [1]

Restricción 3

$N \cdot \frac{P}{F} =_{N=3,P=12,F=2} 3 \cdot \frac{12}{2} = 18$ restricciones obtenidas al instanciar $(\sum_{j=1}^{m} x_{ijk} \cdot c_i) \leq f$, para cada i, $1 \leq i \leq n$, y cada k, $\frac{a_i}{F} + 1 \leq k \leq \frac{d_i}{F}$ que aparecen recogidas en la Tab. C.3.

C.3. Discretización

Los resultados de la discretización para este ejemplo pueden verse aplicados directamente en la obtención del ejecutivo en la Sec. C.5.

id	índice(s) de x_{ijk} fijo(s)	inecuación resultante	inecuación desarrollada
19	i = 1, k = 1	$\sum_{j=1}^{2} x_{1j1} \cdot c_1 \leq f$	$20{,}0 \cdot x_{111} + 20{,}0 \cdot x_{121} \leq f$
20	i = 1, k = 2	$\sum_{j=1}^{2} x_{1j2} \cdot c_1 \leq f$	$20{,}0 \cdot x_{112} + 20{,}0 \cdot x_{122} \leq f$
21	i = 2, k = 3	$\sum_{j=1}^{2} x_{2j3} \cdot c_2 \leq f$	$20{,}0 \cdot x_{213} + 20{,}0 \cdot x_{223} \leq f$
22	i = 2, k = 4	$\sum_{j=1}^{2} x_{2j4} \cdot c_2 \leq f$	$20{,}0 \cdot x_{214} + 20{,}0 \cdot x_{224} \leq f$
23	i = 3, k = 5	$\sum_{j=1}^{2} x_{3j5} \cdot c_3 \leq f$	$20{,}0 \cdot x_{315} + 20{,}0 \cdot x_{325} \leq f$
24	i = 3, k = 6	$\sum_{j=1}^{2} x_{3j6} \cdot c_3 \leq f$	$20{,}0 \cdot x_{316} + 20{,}0 \cdot x_{326} \leq f$
25	i = 4, k = 1	$\sum_{j=1}^{2} x_{4j1} \cdot c_4 \leq f$	$40{,}0 \cdot x_{411} + 40{,}0 \cdot x_{421} \leq f$
26	i = 4, k = 2	$\sum_{j=1}^{2} x_{4j2} \cdot c_4 \leq f$	$40{,}0 \cdot x_{412} + 40{,}0 \cdot x_{422} \leq f$
27	i = 4, k = 3	$\sum_{j=1}^{2} x_{4j3} \cdot c_4 \leq f$	$40{,}0 \cdot x_{413} + 40{,}0 \cdot x_{423} \leq f$
28	i = 5, k = 4	$\sum_{j=1}^{2} x_{5j4} \cdot c_5 \leq f$	$40{,}0 \cdot x_{514} + 40{,}0 \cdot x_{524} \leq f$
29	i = 5, k = 5	$\sum_{j=1}^{2} x_{5j5} \cdot c_5 \leq f$	$40{,}0 \cdot x_{515} + 40{,}0 \cdot x_{525} \leq f$
30	i = 5, k = 6	$\sum_{j=1}^{2} x_{5j6} \cdot c_5 \leq f$	$40{,}0 \cdot x_{516} + 40{,}0 \cdot x_{526} \leq f$
31	i = 6, k = 1	$\sum_{j=1}^{2} x_{6j1} \cdot c_6 \leq f$	$80{,}0 \cdot x_{611} + 80{,}0 \cdot x_{621} \leq f$
32	i = 6, k = 2	$\sum_{j=1}^{2} x_{6j2} \cdot c_6 \leq f$	$80{,}0 \cdot x_{612} + 80{,}0 \cdot x_{622} \leq f$
33	i = 6, k = 3	$\sum_{j=1}^{2} x_{6j3} \cdot c_6 \leq f$	$80{,}0 \cdot x_{613} + 80{,}0 \cdot x_{623} \leq f$
34	i = 6, k = 4	$\sum_{j=1}^{2} x_{6j4} \cdot c_6 \leq f$	$80{,}0 \cdot x_{614} + 80{,}0 \cdot x_{624} \leq f$
35	i = 6, k = 5	$\sum_{j=1}^{2} x_{6j5} \cdot c_6 \leq f$	$80{,}0 \cdot x_{615} + 80{,}0 \cdot x_{625} \leq f$
36	i = 6, k = 6	$\sum_{j=1}^{2} x_{6j6} \cdot c_6 \leq f$	$80{,}0 \cdot x_{616} + 80{,}0 \cdot x_{626} \leq f$

Tabla C.3: Inecuaciones correspondientes a la tercera restricción del problema de programación lineal sobre el sistema de tareas dado como ejemplo en el artículo [1]

C.4. Obtención de la frecuencia

C.4.1. Caso expulsivo

Partiendo de $f = 19$ tras la discretización, se tiene que $\alpha = \frac{f}{F} = \frac{19}{2} = 9{,}5$ Hz. No obstante, por la especificación en *Tertimuss*, los valores de frecuencia no pueden ser decimales, por lo que la frecuencia disponible más pequeña que se le puede asignar es $\beta = 10$ Hz, lo cual resulta en un valor $f' = \beta \cdot F = 20$ ciclos como duración del cuadro del ejecutivo.

C.4.2. Caso no expulsivo

Partiendo de $f = 80$ tras la discretización, se tiene que $\alpha = \frac{f}{F} = \frac{80}{2} = 40$ Hz. Como ese mismo valor está entre las frecuencias disponibles, se tiene que $f' = \beta \cdot F = 40 \cdot 2 = 80$ ciclos como duración del cuadro del ejecutivo.

C.5. Obtención del ejecutivo

Como resultado final, se tiene el ejecutivo cíclico resultande de la asignación de los trabajos a los procesadores en los casos expulsivo (Fig. C.2), no expulsivo a partir de la solución del PPLM (Fig. C.3) y no expulsivo usando la regla de *McNaughton* (Fig. C.4).

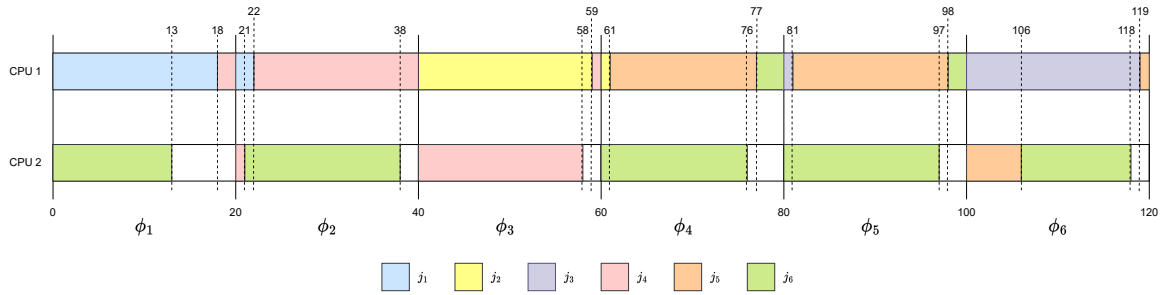


Figura C.2: Asignación de los trabajos a los procesadores en la planificación resultante de la aplicación de DCE en el caso expulsivo

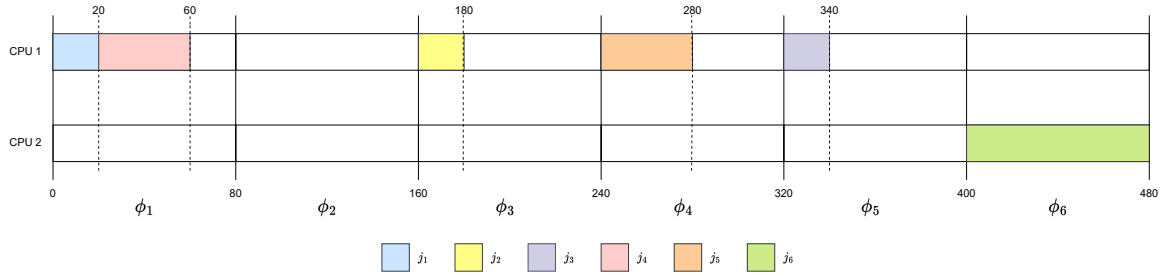


Figura C.3: Asignación de los trabajos a los procesadores en la planificación resultante de la aplicación de DCE en el caso no expulsivo usando la solución del PPL

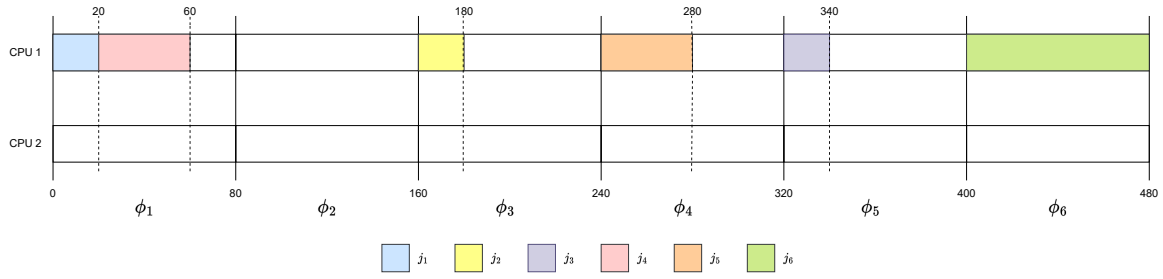


Figura C.4: Asignación de los trabajos a los procesadores en la planificación resultante de la aplicación de DCE en el caso no expulsivo usando la regla de *McNaughton*

Anexos D

Implementación y simulación de Deutschbein CE en Tertimuss

En este anexo se incluyen los diagramas UML de implementación del planificador DCE. Éstos consisten en un diagrama de actividades de la fase fuera de línea (Fig. D.1) y en un diagrama de estados de la simulación del planificador implementado (Fig. D.2).

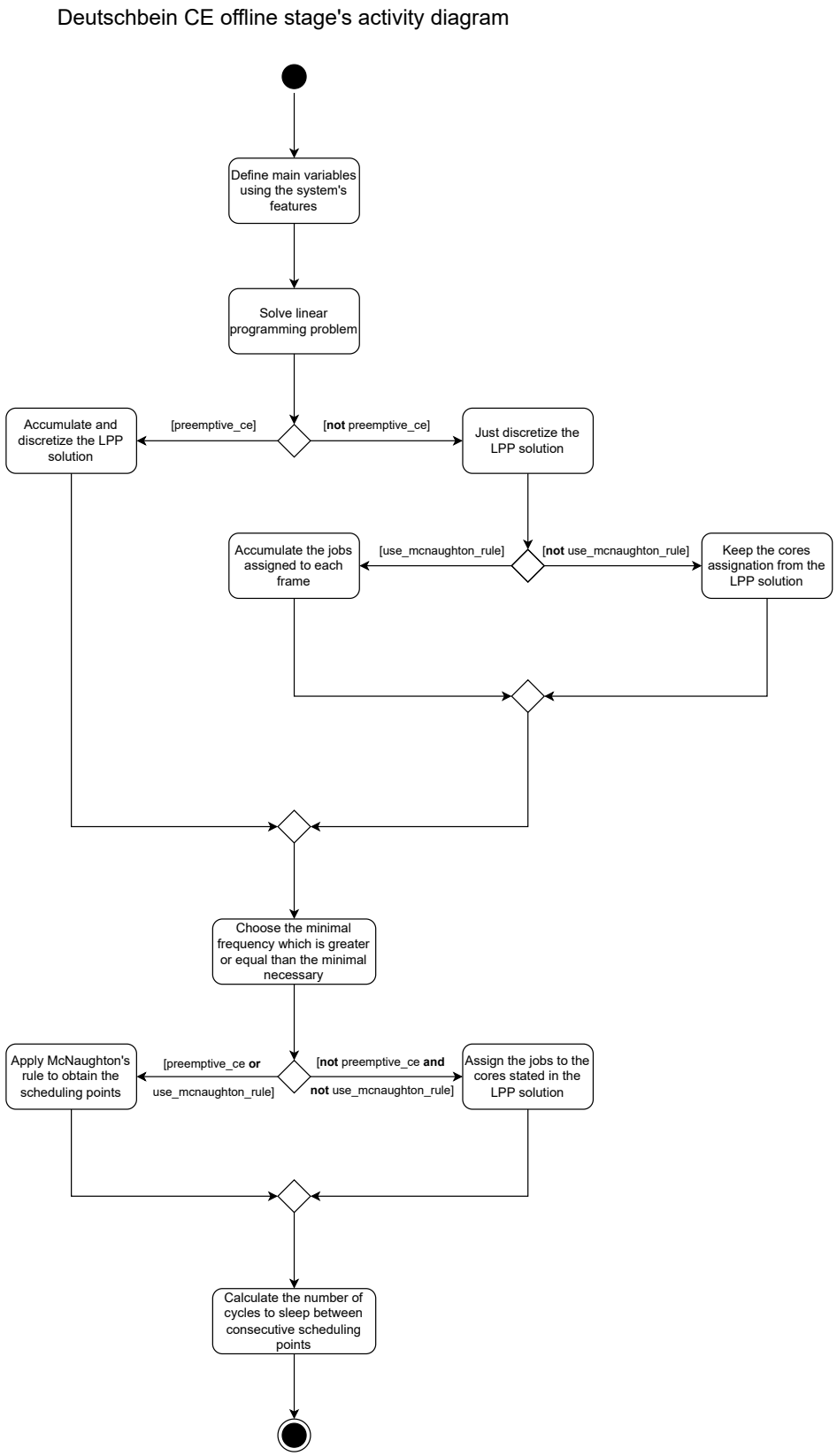


Figura D.1: Diagrama de actividades de la etapa fuera de línea del planificador *Deutschbein CE*

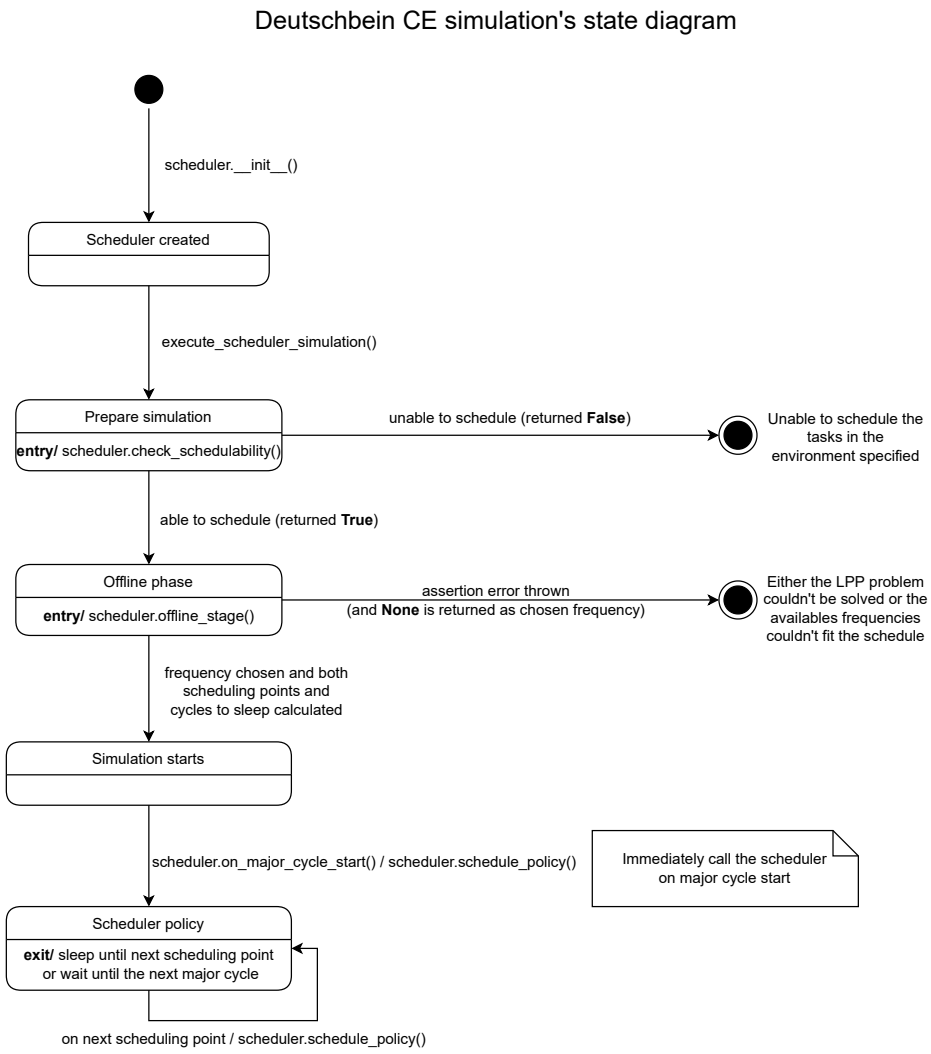


Figura D.2: Diagrama de estados de la simulación del planificador *Deutschbein CE* en *Tertimuss*

Anexos E

Contribuciones a la herramienta Tertimuss

En este anexo se van a listar brevemente las aportaciones realizadas a la herramienta *Tertimuss* como parte de este TFG.

E.1. Mejoras en la documentación

Completar el primero de los tutoriales de la documentación, corrigiendo errores en los ejemplos de uso y añadiendo la visualización de la asignación de las tareas y los trabajos a los procesadores.

E.2. Planificador DCE

Implementación de la clase *SDeutschbeinCE*, que se corresponde con el método de planificación presentado en el artículo de referencia [1] y en el cual se ha basado la realización de este TFG.

Por una parte, la implementación realizada integra en *Tertimuss* la especificación presentada en el artículo del planificador, adaptando las unidades e incorporando métodos para la discretización de la solución al PPL que se han discutido en la Sec 3.3.

Además de permitir la construcción tanto de ejecutivos cíclicos expulsivos como no expulsivos, como aportación de la realización de este TFG se tiene la posibilidad de especificar si utilizar la asignación original de la solución al PPL o si utilizar la regla de *McNaugthon* de la forma que se ha descrito en la Sec. 3.5.3.

Adicionalmente, existe una variable en el código que determina el solucionador de Problema de Programación Lineal que se utilizará en la simulación para obtener el ejecutivo cíclico, y requiere la reinstalación de la herramienta para hacer efectiva su modificación. El motivo que ha llevado a implementar esta configuración a través de una constante en el código es la complejidad que ha supuesto la instalación de *Gurobi* (ver

Anexo F) y al hecho de que *Tertimuss* utilice en el resto de planificadores que plantean Problema de Programación Lineal el solucionador *GLOP* para resolverlos, siendo además que éste se instala ya automáticamente como una dependencia obligatoria del paquete.

También se han implementado treinta y siete tests unitarios que se han incorporado a las pruebas existentes de integración continua de la herramienta.

E.2.1. Añadir función para calcular el valor del ciclo menor del ejecutivo

Se ha implementado una función que permite calcular el valor del ciclo menor del ejecutivo cíclico de manera análoga a la que ya existía para obtener el valor del ciclo mayor.

E.2.2. Implementar una opción para la detección de la paralelización de trabajos durante la simulación

Se ha añadido un atributo booleano a la clase que modela las opciones de la simulación que permite, en caso de indicarse explícitamente, comprobar si las decisiones del planificador han supuesto la paralelización de alguno de los trabajos en las asignaciones realizadas a los procesadores. Esto se ha utilizado en las pruebas de integración continua del planificador DCE, y ha servido para comprobar la satisfacción de la tercera restricción planteada en el PPL durante la fase en línea de la planificación.

También se ha implementado un test unitario que incluye ya definición de un planificador que paraleliza la ejecución de los trabajos asignándolos siguiendo la filosofía *FIFO* con respecto a su instante de activación, el cual se ha utilizado para comprobar el correcto funcionamiento de la simulación cuando se paraleliza la ejecución de los trabajos y la generación del error cuando se especifica la realización de la comprobación que se ha implementado.

E.2.3. Actualizar la versión del componente *ortools* del paquete para solucionar el error con *Gurobi*

Debido a la presencia de un error en la versión del paquete *ortools* especificado como dependencia de *Tertimuss* no era posible utilizar el solucionador *Gurobi* para resolver los PPL pese a haber completado correctamente su instalación. Por ello, a través de las incidencias del repositorio de código de *ortoools* [7], se pudo solucionar el problema actualizando la versión del componente en la especificación de las dependencias de la herramienta.

Anexos F

Instalación y configuración de Gurobi

Para poder utilizar el solucionador *Gurobi* desde *ortools*, es necesario instalarlo previamente en el sistema. Este proceso consiste en la instalación y la posterior validación de los binarios con la licencia obtenida:

1. Los pasos concretos de instalación se describen en la página [38] para cada sistema operativo soportado.

2. Tras ello, debe solicitarse una licencia académica en el sitio web de la herramienta [24]. Una vez obtenida, se podrá acceder a través de este enlace [37] a la información de la misma, en la que indicarán el argumento con el que el comando *grbgetkey* se debe ejecutar para llevar a cabo el proceso de validación.

   Nótese que, para que ese comando funcione, debe disponerse de una conexión a Internet y además ésta debe hacerse desde la red de la institución académica por la cual se solicita la licencia.

   Para evitar configuraciones adicionales con respecto a la ubicación del fichero con la licencia, conviene dejar la ruta que el comando propone por defecto. En caso de no utilizar esa ubicación, es necesario configurar la variable de entorno *GRB_LICENSE_FILE* asignándole la ruta que se ha elegido para el fichero *gurobi.lic*.

3. Para comprobar que la licencia se ha activado correctamente, se puede ejecutar el comando *gurobi.sh*. Si éste carga y aparece el intérprete de *Gurobi*, entonces la instalación se habrá realizado correctamente.

F.0.1. Problemas con la versión de *ortools*

Existe un fallo [7] en las versiones de *ortools* anteriores a la *9.4* que impide la utilización del solucionador *Gurobi* pese a haber completado correctamente su

instalación en el sistema.

Por lo tanto, si se desea utilizar este solucionador en *Tertimuss*, debe actualizarse la versión del paquete *ortools* que éste requiere en el fichero *pyproject.toml*, pasando de la *8.2.8487* a la *9.4.1874*, con el fin de evitar dicho problema. En caso de no hacerse, aparecerá el mensaje *Support for GUROBI not linked in, or the license was not found* cada vez que se intente crear un solucionador con el API de *ortools* usando Gurobi como la implementación.

Anexos G

Planificación del proyecto

Este Trabajo de Fin de Grado se ha realizado entre los meses de febrero y noviembre del año 2022, durante el curso académico 2021-2022.

G.1. Horas dedicadas

Actividad	*Horas dedicadas*
Repaso de conceptos previos	19 h
Estudio de TR sobre multiprocesadores	22 h
Estudio del artículo de referencia	107 h
Planteamiento del PPL	11 h
Discretización de la solución	39 h
Obtención del ejecutivo	52 h
Depuración y tests	91 h
Soporte a Gurobi	14 h
Refactorización de artifacts	55 h
Desarrollo de scripts	28 h
Análisis de resultados	13 h
Apuntes a mano	83 h
Ejemplos y pruebas	34 h
Transcripción de los apuntes	40 h
Redacción desde cero	11 h
Total	**629 h**

Tabla G.1: Tabla de horas que se estima que se han dedicado a cada actividad del proyecto

G.2. Diagrama de Gantt

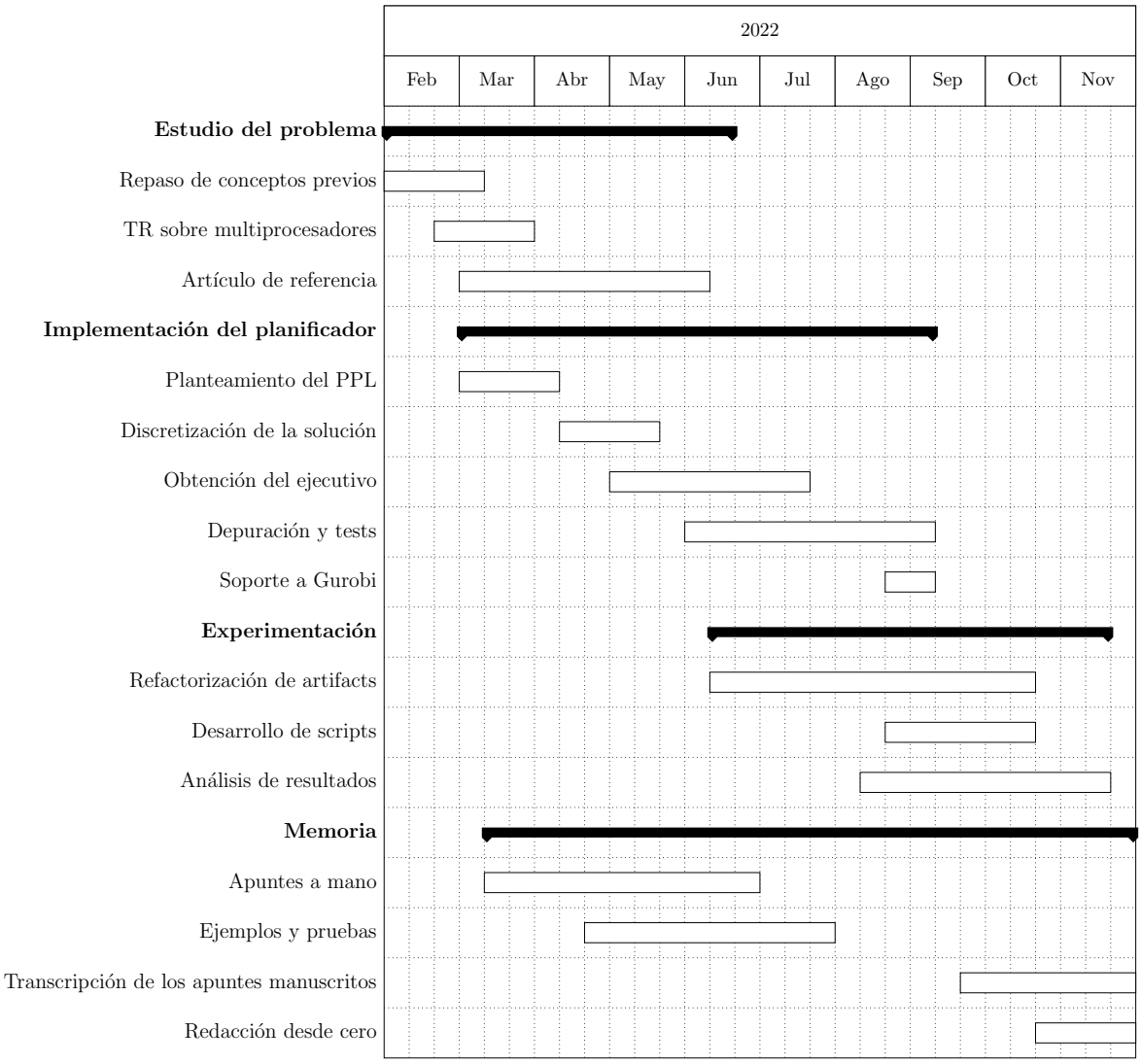


Figura G.1: Diagrama de Gantt del proyecto