Universidad Zaragoza

Trabajo Fin de Máster

Planificación Tiempo Real en multiprocesadores: reducción de cambios de contexto y migraciones en AlECS

Autor

Abel Chils Trabanco

Director

José Luis Briz Velasco

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2021

AGRADECIMIENTOS

En primer lugar, quiero dedicar este trabajo a mi familia, que sin su apoyo este no sería posible.

También al equipo de investigación en el que he estado desarrollado mi actividad investigadora en los últimos meses, formado por el profesor José Luis Briz del GaZ, el profesor Antonio Ramírez-Treviño del CINVESTAV y Laura Rubio-Anguiano del CINVESTAV. Agradecerles la oportunidad que me ofrecieron de trabajar junto a ellos, he aprendido mucho y ha sido un placer hacerlo. Un agradecimiento especial a José Luis, director de este TFM, por toda la ayuda prestada durante el mismo.

Otro agradecimiento al grupo de investigación GaZ, donde he aprendido mucho en los diferentes seminarios organizados, influenciando algunos de ellos en ideas incorporadas en este TFM.

Por último, agradecer al grupo de investigación I3A por la oportunidad que me ofreció para iniciarme en la investigación. Comentar que este trabajo ha sido financiado por MINECO/AEI/FEDER (UE) (proyecto PID2019-105660RB-C21 / AEI / 10.13039/501100011033), el Gobierno de Aragon (Grupo T58_20R) y FEDER 2014-2020 "Construyendo Europa desde Aragón".

RESUMEN

El diseño de algoritmos de planificación tiempo real sobre multiprocesadores es un campo en el que la industria tiene interés porque permiten reducir el peso, las dimensiones y el consumo de los sistemas. Uno de los problemas que introducen estos algoritmos es que o bien desperdician tiempo de procesador, o bien introducen sobrecostes al intentar aprovecharlo al máximo, al incrementar las migraciones y cambios de contexto de las tareas.

Este trabajo presenta el algoritmo de planificación CAlECS que trabaja con utilización máxima y genera un número de cambios de contextos y migraciones menor que RUN, algoritmo de referencia en este aspecto. La principal novedad en su implementación es el uso de una técnica de *clustering* (agrupamiento) capaz de obtener planificaciones con utilización máxima, a la vez que permite limitar los procesadores a los que puede migrar cada tarea. Además, genera de forma *off-line* un ejecutivo cíclico, situándose en la línea de los estándares de la industria de automoción y aeroespacial.

También se ha actualizado el entorno de simulación Tertimuss, usado como base experimental durante el trabajo. Se ha renovado completamente su arquitectura y modo de simulación, pasando de simular ciclos de procesador a simular eventos del sistema. Esto reduce de horas a minutos el tiempo de cómputo cuando se simulan grandes volúmenes de experimentos. Tertimuss se ha ampliado con herramientas que permiten el análisis automático de planificaciones obteniendo diferentes métricas (ej. número de cambios de contexto). Se han añadido nuevos métodos de representación de planificaciones, y nuevos algoritmos de planificación y generación de tareas. En la práctica, Tertimuss es más eficiente al usarse como una biblioteca, por lo que se ha modificado sustancialmente la interfaz de programación que ofrece para facilitar su uso en esa forma.

Por último, se ha planteado un flujo de trabajo que facilita la aplicación de algoritmos de planificado ejecutados *off-line* sobre sistemas reales. Este es un flujo de trabajo iterativo, que busca en cada iteración ajustar el máximo tiempo de ejecución de las tareas teniendo en cuenta la planificación de las mismas. Esto permite limitar los sobrecostes que se contabilizan en este. Además, se ha analizado los costes en los que incurre un cambio de contexto y una migración en una placa de desarrollo.

Acrónimos

ECU Electronic Control Unit

TFM Trabajo de Fin de Máster

TPP Tareas por procesador

AlECS Allocation and Execution Control Scheduler

CAlECS Clustered Allocation and Execution Control Scheduler

RUN Reduction to uniprocessor scheduler

BPP Bin packing problem

BFD Best fit descending

PPE Problema de Programación Entera

PPL Problema de Programación Lineal

TPP Tareas Por Procesador

TR Tiempo Real

TRB Tiempo Real Blando

TRD Tiempo Real Duro

TRF Tiempo Real Firme

TUM Totally Unimodular Matrix

EDF Earliest Deadline First

WCET Worst Case Execution Time

DVFS Dynamic voltage scaling

FIFO First in first out

CPU Central Processing Unit

MCD Máximo Común Divisor

MCM Mínimo Común Múltiplo

FTP Fixed Task Priority

FJP Fixed Job Priority

PD Dynamic Priority

TFG Trabajo de Fin de Grado

MPSoC Multiprocessor Sytems on Chip

TCPN Timed Continuous Petri Net

Índice

Capítulo 1

Introducción

1.1. Motivación

Lo sistemas multiprocesador en chip (MPSoC) están en el foco de interés de la industria aeroespacial y de automoción, ya que permiten reducir el coste, tamaño, peso y consumo de los sistemas empotrados Tiempo Real (TR) en relación a soluciones sobre monoprocesador. Sin embargo, su utilización supone varios retos. Uno de ellos es la planificación de tareas TR sobre los mismos, más compleja que en el caso de monoprocesadores. Uno de los objetivos de la planificación es conseguir tasas altas de utilización de los procesadores para poder disminuir bien el número de unidades de control (ECU), bien la potencia de los microprocesadores necesarios, evitando lo que se conoce como *sobreaprovisionamiento*, cumpliendo a su vez las restricciones TR o de energía y temperatura del sistema. Las soluciones más próximas a la realidad industrial conllevan tasas de utilización muy bajas, mientras que otras propuestas que alcanzan utilizaciones óptimas resultan poco viables.

1.2. Contexto

Este proyecto continúa el trabajo iniciado en el Trabajo de Fin de Grado (TFG) del mismo autor, que se sitúa en una línea de investigación conjunta participada por el Grupo de Arquitectura de Computadores de la Univ. de Zaragoza (GaZ) y el grupo de Ingeniería de Control de la Unidad Guadalajara del Centro de Investigaciones y Estudios Avanzados (CINVESTAV) de México. Dicha línea se centra en los algoritmos de planificación TR sensibles a temperatura y energía sobre multiprocesadores. Los responsables en sus grupos respectivos son el Prof. José Luis Briz (Univ. Zaragoza) y el Prof. Antonio Ramírez Treviño (CINVESTAV). De este esfuerzo conjunto han resultado diferentes propuestas publicadas [1, 2, 3, 4, 5, 6] y una tesis doctoral ya completada [6].

Parte de las nuevas contribuciones de este Trabajo de Fin de Máster (TFM) se han incorporado en las dos publicaciones más recientes de la línea [7, 8], relacionadas con la tesis doctoral en curso de la estudiante Laura E. Rubio-Anguiano (CINVESTAV - Univ. de Zaragoza). Esta memoria desarrolla dichas contribuciones (Cap. 4, 5), aportando también nuevo material y dejando planteada una vía de continuación (Cap. 6). Así mismo, durante el desarrollo del TFM se ha continuado trabajando y mejorando el entorno Tertimuss [9], base experimental de todas las contribuciones.

1.3. Objetivos

El objetivo general del TFM es la introducción del autor en la investigación sobre planificación TR sobre multiprocesadores.

Los objetivos específicos son:

- Reducir los sobrecostes del algoritmo AlECS (Allocation and Execution Control Scheduler) minimizando el número de migraciones en la planificación.
- Evaluar de forma experimental la reducción lograda, y comparar los sobrecostes de la nueva versión con los producidos por el algoritmo de referencia RUN (Reduction to uniprocessor scheduler).
- Evaluar posibles vías de implementación de la solución propuesta en un sistema real.
- Ampliar y optimizar el entorno de simulación *Tertimuss* permitiendo la evaluación automática de los planificadores en los aspectos requeridos por este trabajo.

1.4. Alcance

El proyecto ha generado un nuevo algoritmo de planificación conforme a los objetivos propuestos. Este nuevo algoritmo ha sido comparado tanto con el algoritmo de partida AlECS como con el algoritmo de referencia RUN. Esta contribución forma parte sustancial de una publicación en conferencia [7] y de un artículo en revista [8] (Q1 JCR 2019), ambos internacionales con revisión por pares.

En el primero de los trabajos se contribuyó mediante una comparativa preliminar entre el planificador AlECS y RUN. Esta comparativa permitió verificar el buen comportamiento de AlECS, diagnosticando situaciones en las que podían mejorarse sus resultados, y marcando el camino de la siguiente contribución. Este artículo además es donde se presenta el algoritmo AlECS.

La contribución al segundo trabajo se detalla en el Cap. 4 y es una versión extendida de lo publicado en el artículo [8]. De forma similar, el Cap. 5 también amplía la comparativa incluida en dicho artículo. La comparativa preliminar publicada en [7] fue clave en el establecimiento de las hipótesis que llevaron a la solución posterior. Sin embargo, no se incluye en esta memoria de TFM ya que la comparativa extendida en Cap. 5 es mucho más amplia y minuciosa. El esquema de planificador CAlECS (Clustered Allocation and Execution Control Scheduler) presentado en [8] incluye otros componentes además de las contribuciones propias de este TFM, como el modelado del sistema mediante TCPN (Timed Continuous Petri Net), la resolución de la planificación global fluida y los componentes de control que permiten que las planificaciones obtenidas respeten los límites térmicos, toleren perturbaciones y permitan la gestión de tareas aperiódicas Tiempo Real Blando (TRB). En todo caso, estos componentes se describen en el Cap. 4 para que se puedan situar con precisión las contribuciones de este TFM.

A lo largo del proyecto también se realizó un mantenimiento y mejora del entorno de simulación *Tertimuss* acorde a los objetivos planteado. El mismo puede encontrarse en [10].

1.5. Entorno de trabajo

La herramienta principal utilizada para la escritura de la memoria y de los artículos de investigación ha sido LaTeX sobre Overleaf [11].

Para la coordinación del equipo de investigación, así como para las reuniones de los mismos, se ha utilizado la suite de aplicaciones Google Workspace.

Para el mantenimiento y mejora de *Tertimuss* las herramientas principales usadas han sido el entorno de desarrollo PyCharm [12] y el editor de texto Visual Studio Code [13], siendo ambos editores de código abierto [14, 15]. *Tertimuss* se ha desarrollado en Python 3.8 haciendo uso de diferentes bibliotecas de código abierto como son OR-Tools [16], Matplotlib [17], Scipy [18] y Numpy [19]. Para el control de versiones del mismo se utiliza Git [20], y se ha alojado en GitHub [21].

1.6. Metodología general

El trabajo a estado sujeto al método científico en cuanto que se han realizado experimentos que han conducido a hipótesis (Sec. 4.1.1), se han implementado soluciones basadas en dichas hipótesis (Cap. 4), se ha comprobado finalmente su corrección y evaluado su rendimiento y viabilidad (Cap. 5) y se han establecido líneas

futuras en base a estos últimos resultados (Caps. 5 y 7).

Así mismo, los resultados han podido difundirse en foros internacionales de prestigio con revisión por pares [7], [8] al situarse en el contexto de las aportaciones del equipo que trabaja en la línea de investigación mencionada en la sección anterior.

El curso del trabajo ha requerido metodologías específicas para el mantenimiento y programación del entorno Tertimuss [10] y para la realización de los experimentos. En este último caso, la metodología específica se detalla en la Sec. 5.1.

Durante todo el trabajo se han realizado reuniones semanales con los otros tres componentes del equipo de investigación, además de reuniones individuales específicas para abordar cuestiones concretas. Las reuniones han sido no presenciales, debido tanto a la situación de pandemia como a la dispersión geográfica (España, México), utilizándose un amplio conjunto de herramientas colaborativas para ello.

1.7. Planificación del proyecto

Las fases y horas dedicadas a este TFM se encuentran en la Tab. B.1, y el diagrama de Gantt del mismo en la Fig. B.1, ambas en el Anexo. B.1.

1.8. Descripción del resto del documento

Esta memoria se estructura como sigue. El Cap. 2 introduce los fundamentos sobre los que se sostiene el trabajo, conceptos sobre sistemas TR (Sec. 2.1) y conceptos sobre planificación de tareas con restricciones TR (Sec. 2.2).

El Cap. 3 realiza un resumen de los planificadores y herramientas de simulación más cercanos al trabajo propuestos en la literatura hasta el momento de redactar esta memoria.

El Cap. 4 describe el algoritmo de planificación propuesto nombrado como CAlECS (Clustered Allocation and Execution Control Scheduler). Esta descripción comienza con un resumen del funcionamiento del algoritmo (Sec. 4.1.3), tras este, se explica el algoritmo en detalle (Secs. 4.2, 4.3, 4.4, 4.5, 4.6), y se finaliza con un análisis de la complejidad del mismo (Sec. 4.7).

En el Anexo. C se resuelve un ejemplo que pretende ayudar al lector a comprender completamente el algoritmo CAlECS.

El Cap. 5 realiza una comparativa entre los planificadores RUN, AlECS y CAlECS, donde se analizan los cambios de contexto (Sec. 5.2) y las migraciones (Sec. 5.3) producidas por ellos.

El Cap. 6 comienza mostrando las dificultades que presenta la implementación de un

algoritmo de planificación sobre una plataforma real (Sec. 6.1), se prosigue analizando los costes de migración y cambio de contexto en una plataforma Linux TR (Sec. 6.2), y se finaliza proponiendo un flujo de trabajo que permitiría implementar (Sec. 6.3) el planificador dato en una plataforma real.

En el Anexo. D se muestra un ejemplo del uso del flujo de trabajo presentado.

El Cap. 7 recapitula el trabajo, aportando unas breves conclusiones y planteando vías de continuación.

El Anexo. A describe los cambios más importantes realizados sobre el entorno de simulación *Tertimuss*, a la vez que muestra la arquitectura final del mismo (Sec. A.1).

Por último, el Anexo. B contiene las horas dedicadas al proyecto, así como un diagrama de Gantt del mismo.

Capítulo 2

Fundamentos

El objetivo de este capítulo es dotar de carácter autocontenido a esta memoria, de manera que un lector ajeno al campo de planificación de sistemas TR no tenga que recurrir a fuentes externas para seguir el desarrollo de este trabajo. Buena parte de este material proviene de la Sec. 2.3 del Trabajo de Fin de Grado del mismo autor [22], incorporando mejoras y aclaraciones convenientes para este TFM. La Sec. 2.2 está ampliada.

A lo largo de esta memoria, se utilizan indistintamente los términos *procesador*, *núcleo de procesamiento* o simplemente *núcleo*, y *CPU*, en función del contexto.

2.1. Sistemas TR

Los conceptos más básicos de sistemas TR están convenientemente definidos en [23, 24, 6, 22].

Sistema tiempo real

Un sistema TR es un sistema de cómputo cuyo comportamiento correcto depende tanto del resultado del cómputo como del momento en el que este se produce.

Planificación

Una planificación es una secuencia de ejecución particular del conjunto de tareas que componen la aplicación en una plataforma de cómputo dada.

Planificación factible

Una planificación factible es aquella en la que todas las tareas se ejecutan cumpliendo las restricciones impuestas.

Tarea

Una tarea (*task*) es una secuencia de instrucciones, que en ausencia de otras actividades se ejecuta de forma continua en un núcleo de procesamiento hasta su terminación. Una tarea se activa normalmente varias veces con diferentes datos de entrada, generando una secuencia de instancias llamadas trabajos (*jobs*).

Según el patrón temporal de activación de los diferentes trabajos, las tareas pueden ser periódicas, aperiódicas o esporádicas.

Una *tarea periódica* es aquella cuyos trabajos se activan en intervalos regulares de tiempo, por lo que estos están separados por un intervalo fijo de tiempo llamado *periodo*.

Una *tarea aperiódica* es aquella cuyos trabajos pueden ser activados en intervalos de tiempo arbitrario.

Una *tarea esporádica* es aquella cuyas activaciones de trabajos están separadas por un intervalo mínimo de tiempo. El hecho de que las tareas periódicas sean muy complejas de analizar, por un lado, y que las esporádicas sean las más utilizadas, por otro, hace que sea habitual referirse a las tareas esporádicas simplemente como *tareas periódicas*, considerando el periodo como el tiempo mínimo entre dos trabajos consecutivos [24]. Nos referiremos a ellas de esta forma en el resto de este trabajo.

Por otra parte, las tareas TR pueden clasificarse como de TR *duro* (TRD), *firme* (TRF) o *blando* (TRB) según su nivel crítico. Así, fallar el cumplimiento de un plazo de una tarea TRD coloca al sistema en estado de fallo irrecuperable. Los resultados retrasados de una tarea TRB pueden aún recuperarse aunque perjudiquen el rendimiento o funcionamiento óptimo del sistema. En el caso de tareas TRF, los retrasos producen resultados inútiles y en general no perjudiciales para el sistema. Las tareas consideradas en el contexto de este trabajo son en general TRD, excepto cuando se tratan aperiódicas, que se consideran TRB.

Conjunto de tareas factible

Un conjunto de tareas factible es un conjunto de tareas para el que existe una planificación factible en una plataforma de cómputo particular.

Conjunto de tareas planificable

Un conjunto de tareas se dice planificable para un algoritmo A si este algoritmo es capaz de generar una planificación factible para ese conjunto de tareas.

Trabajo

Instancia de una tarea ejecutada con unos datos de entrada específicos (*job*).

Periodo

Separación temporal entre activaciones consecutivas de una tarea (p, *period*).

Hiperperiodo

El hiperperiodo (H) se define como el Mínimo Común Múltiplo (MCM) de todos los periodos de un conjunto de tareas.

Dividiendo el tiempo total de ejecución de todas las tareas del sistema en porciones del tamaño del hiperperiodo, ocurren los mismos eventos (fin de plazo, inicio de periodo) y en el mismo instante temporal relativo al inicio del hiperperiodo.

Plazo

El *plazo* (d, *deadline*) de una tarea es el tiempo máximo disponible para completar su ejecución. Puede ser *relativo* (al inicio del periodo) o *absoluto* (relativo al comienzo de la ejecución de todo el sistema de tareas TR). Con plazos relativos, el plazo absoluto para el trabajo j de una tarea con plazo d igual al periodo p es $j * p$ (el trabajo habrá llegado en el instante $(j-1) * p$). El plazo es *implícito* cuando coincide con el periodo ($p = d$), *limitado* (*constrained*) si $p <= d$, o *arbitrario* si la relación entre p y d varía.

Tiempo de ejecución del peor caso

El tiempo de ejecución máximo o del peor caso (WCET) de una tarea es el máximo tiempo de ejecución que puede necesitar un procesador para completar sin interrupción una tarea para cualquier conjunto de datos de entrada.

Laxitud

La laxitud (*laxity*) de un trabajo en un instante temporal t, se define cómo el tiempo máximo que puede permanecer sin ser ejecutado, a partir de ese instante temporal t, de forma que aún pueda comenzar o reanudar su ejecución sin violar el cumplimiento de su plazo.

Utilización

La *utilización* de una tarea τ_i con WCET c_i y periodo p_i se define como $u_i = c_i/p_i$. La utilización de un conjunto de n tareas se define como $U = \sum_{i=1}^{n} c_i/p_i$.

Modelo de tareas esporádicas de tres parámetros

En este modelo cada tarea se define mediante su WCET, su periodo y su plazo. Es el modelo utilizado en este trabajo de acuerdo al contexto del mismo (Sec. 1.2).

2.2. Planificación TR

Bases de la planificación

Los planificadores intervienen en *puntos de planificación* determinados por intervalos fijos (quantum) o variables (obedeciendo a eventos). Eventos típicos son la finalización de un trabajo o la llegada de una tarea aperiódica. En cada punto de planificación, se activa el planificador para seleccionar la tarea de mayor prioridad y ejecutarla. Las prioridades pueden asignarse de forma estática o dinámica, dando lugar a diferentes opciones según se aplique a tareas o a trabajos. Describimos las más comunes:

- Tareas de prioridad fija (Fixed Task Priority (FTP)): La prioridad asignada a las tareas forma parte de su definición, y sus instancias (trabajos) la heredan. Dos ejemplos conocidos son *Rate Monotonic* y *Deadline Monotonic*.

- Trabajos de prioridad fija (Fixed Job Priority (FJP)): Las instancias sucesivas de una misma tarea pueden tener diferente prioridad. Sin embargo, una vez asignada, la prioridad de un trabajo dado no cambia durante su ejecución. El ejemplo más conocido es EDF (Earliest Deadline First).

- Prioridad Dinámica (Dynamic Priority (PD)): la prioridad puede variar sin restricciones, dando lugar a diferentes posibilidades.

Se entiende por planificación *off-line* cuando puede calcularse para todo el hiperperiodo antes de lanzar a ejecución el conjunto de tareas. Al conocerse de antemano el número y localización de los cambios de contexto (y en su caso, migraciones), puede resolverse mediante una tabla de planificación, generando una sobrecarga muy pequeña y calculable con mucha precisión. El caso más simple y utilizado en sistemas sencillos es el ejecutivo cíclico.

Alternativamente, la planificación puede realizarse *on-line* (en tiempo de ejecución), tomando dinámicamente las decisiones de planificación en cada punto de planificación, a costa de una sobrecarga mayor que en el caso *off-line*. También es posible realizar una etapa de preparación *off-line* que simplifique y disminuya la sobrecarga de planificación en tiempo de ejecución.

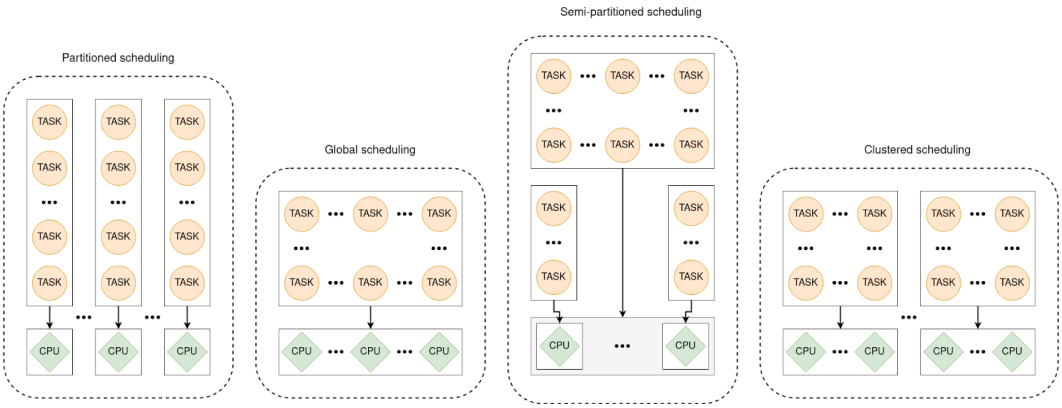


Figura 2.1: Esquemas comunes de planificación.

Planificación particionada

En este modelo cada núcleo de procesamiento ejecuta un subconjunto disjunto del total de tareas, manteniendo siempre una utilización en estos conjuntos menor o igual a 1 (Fig. 2.1). Estos conjuntos pueden ser planificados mediante cualquier algoritmo de planificación sobre monoprocesadores.

El particionado evita la migración de las tareas entre núcleos de procesamiento. Su implementación es sencilla, basada en algoritmos muy conocidos. Su inconveniente reside en que solo garantiza formalmente utilizaciones iguales o inferiores al 50 % [25]. El uso de heurísticas en casos concretos puede permitir utilizaciones mayores [26].

Planificación global

Modelo de planificación en el que cada núcleo de procesamiento puede ejecutar cualquier trabajo generado por el conjunto total de tareas. Los únicos algoritmos que se han demostrado capaces de cumplir utilizaciones máximas pertenecen a esta familia (Sec. 2.2). Son complejos de implementar, y tienden a generar muchos cambios de contexto y migraciones en su esfuerzo por garantizar una utilización máxima.

Planificación semi-particionada

Modelo de planificación híbrido entre los dos modelos anteriores. Los sistemas de este modelo dividen el conjunto global de tareas en dos subconjuntos disjuntos. Uno de ellos se planifica mediante planificación global, el otro mediante planificación particionada, y ambas planificaciones comparten los mismos recursos (procesadores). Es decir, pueden existir tareas de los dos subconjuntos ejecutándose sobre una misma CPU (Central Processing Unit).

De este modo, sólo las tareas gestionadas mediante planificación global pueden migrar. Este esquema permite planificar conjuntos de tareas con utilización máxima,

a cambio de una implementación generalmente compleja.

Planificación basada en agrupamientos (*clusters*)

En esta ocasión, CPU y tareas se dividen en el mismo número de conjuntos disjuntos. Cada conjunto de tareas se planifica en un único conjunto de procesadores mediante alguno de los modelos vistos anteriormente. El conjunto de tareas debe tener una utilización menor o igual al número de CPU del conjunto que se le asigna, a fin de asegurar una planificación factible,

Este modelo ha sido utilizado por ejemplo agrupando los núcleos según características físicas del microchip tales como la memoria cache compartida [27].

Cambio de contexto

Un cambio de contexto se produce cuando una tarea finaliza o bien cuando es expulsada (*preempted*) antes de completarse para reanudarse más adelante.

Migración

Una migración se produce cuando tras expulsar un trabajo que no ha completado su ejecución, se reanuda su ejecución en una CPU diferente a la última en la que se estuvo ejecutando.

Quantum

En este trabajo consideramos el *quantum* (Q) como un intervalo de tiempo fijo entre *puntos de planificación* (intervenciones consecutivas de un planificador).

Particionado por fin de plazos

La planificación basada en quantum supone una sobrecarga excesiva. Se puede reducir notablemente esta sobrecarga utilizando la técnica de *particionado por fin de plazos* (*Deadline Partitioning* [28]). Se basa en planificar dividiendo el hiperperiodo en intervalos determinados por el conjunto de plazos de finalización de todas las tareas periódicas y de sus múltiplos.

Planificación fluida

La planificación fluida es un concepto abstracto consistente en el reparto instantáneo de las tareas a ejecutar entre las CPUs disponibles. Así, dos tareas se ejecutarían simultáneamente sobre una única CPU utilizando cada una un 50 % de la misma, pudiendo variar la proporción a fin de priorizar una tarea sobre la otra. Una

implementación tan directa como inviable consistiría en el reparto infinitesimal en el tiempo de las CPUs entre las dos tareas. Es posible sin embargo implementar planificadores óptimos en utilización sobre esta base haciendo que la diferencia entre el tiempo realmente ejecutado por un trabajo y el tiempo previsto por la planificación fluida (denominado *error fluido*) sea cero en los puntos de planificación. Estos puntos pueden venir dados por un quantum, o por un particionado por fin de plazos. Una implementación muy conocida de un planificador fluido ajena al contexto TR, basada en el concepto de tiempo virtual, es el *Complete Fair Scheduler* incorporado al núcleo de Linux desde la versión V2.6.23 en 2007.

Equidad proporcional

La equidad proporcional (*proportionate fairness*) es un concepto que consiste en el progreso proporcional de todas las tareas basándose en sus necesidades de cómputo sobre los recursos de cómputo disponibles. Los planificadores basados en esta idea constituyen una solución óptima para planificación multiprocesador en TRD y TRB para conjuntos de tareas con plazos implícitos [29]

Capítulo 3

Estado del arte

Este trabajo se centra en dos aspectos específicos dentro del campo de planificación TR sobre multiprocesadores: Algoritmos de planificación con utilización máxima (capítulos 4, 5 y 6) y Herramientas de simulación de planificadores TR (anexo A). Esta sección se limita por tanto a estos dos aspectos. Para una visión sobre el estado del arte relativo al contexto general del trabajo (planificación TR sobre multiprocesadores con restricciones de temperatura y energía) remitimos a [5, 7, 8].

3.1. Algoritmos de planificación TR con utilización máxima

En esta sección se discuten propuestas de planificación directamente relacionadas con el algoritmo presentado, en particular las basadas en el concepto de *equidad proporcional* (Sec. 2.2) y con el concepto de *clustering* o agrupamiento de tareas.

3.1.1. Pfair

La idea de equidad proporcional aparece originalmente en los planificadores Pfair [30], PD [31], PD^2 [32]. El principal inconveniente de estos algoritmos es que tienden a generar un alto número de migraciones y cambios de contexto. Para reducir dicho inconveniente, se propuso la técnica de particionado por plazos (Deadline partitioning), originalmente mediante el algoritmo *DP-Fair* [28].

3.1.2. Algoritmos basados en EDF

Las soluciones basadas en EDF, como es el caso de G-EDF, son fáciles de implementar pero solamente proveen optimalidad TRB [33]. También existen soluciones simples basadas en EDF que proveen optimalidad en cuanto a TRD, aunque sacrificando la optimalidad en cuanto a utilización, como puede ser U-EDF [34].

3.1.3. RUN

Otra propuesta óptima en TRD tanto en el aspecto temporal como de utilización es RUN [35, 36, 37]. Este es un planificador global no fluido, que consigue un número muy bajo de cambios de contexto y migraciones con una utilización máxima del sistema [35, 36, 37]. Se considera una referencia en cuanto al número de cambios de contexto y migraciones.

El planificador trata conjuntos de tareas factibles e independientes con plazo implícito en un multiprocesador homogéneo. Para realizar la planificación, transforma el problema de planificación sobre multiprocesadores a una jerarquía de problemas de planificación sobre monoprocesadores. Esto se consigue mediante *servidores* que se tratan como tareas durante el cálculo de la planificación, aunque no son tareas físicas, sino una representación de un conjunto de tareas.

En su etapa *off-line* RUN aplica una operación de *empaquetado* para formar grupos de tareas con una utilización menor o igual a 1, que pueden ser asignadas a un monoprocesador. Cada uno de estos grupos de tareas forma un *servidor EDF*, que posee un plazo relativo igual al MCD (Máximo Común Divisor) de los plazos de todas la tareas que lo componen, y una utilización igual a la suma de la utilización de todas ellas.

Tras esto, se realiza la operación *dual*. Esta consiste en crear un *servidor dual* a partir de un *servidor EDF*. Este *servidor dual* poseerá el mismo plazo que el *servidor EDF*, aunque su utilización será igual a 1 menos la utilización del *servidor EDF*.

Estas operaciones se repiten de forma iterativa utilizando como tareas los *servidores duales* generados en la iteración anterior, hasta que se obtiene un conjunto de tareas con utilización igual a 1. El algoritmo asegura la convergencia. Una vez alcanzada, se habrá generado un árbol donde todos los nodos hoja tienen el mismo número de nodos intermedios hasta el nodo raíz.

Siempre que tras una operación de *empaquetado* el servidor resultante posea utilización igual a 1, se independiza tanto este servidor como sus descendientes del árbol principal. La utilización de este nuevo árbol es siempre un número entero, por lo que se le puede asignar un número de núcleos de procesamiento en exclusividad.

Posteriormente en la fase *on-line*, la planificación se realiza de forma independiente en cada uno de los árboles formados. Esta comienza planificando el nodo raíz, que siempre es un *servidor EDF*, y termina planificando las hojas, que son las tareas del sistema. Esta planificación utiliza dos reglas, por un lado, si un *servidor EDF* está siendo ejecutado este ejecuta también su nodo hijo con el plazo mas cercano entre los hijos que aún les resten trabajo. Por otro lado, si un *servidor dual* no está siendo

ejecutado ejecutará su único nodo hijo.

3.1.4. QPS

En cuanto a planificación semi-particionada, el algoritmo QPS (*Quasi-Partitioned Scheduling*) [38] incluye la planificación de tareas aperiódicas. Cuando solamente planifica tareas periódicas tiene el mismo comportamiento que RUN [37, 39].

3.1.5. AlECS

El planificador AlECS [5, 7] se basa en el concepto planificación fluida con equidad proporcional. Es óptimo en TRD y en utilización. Se estructura en una fase *off-line* y otra *on-line*. Soporta la gestión de tareas periódicas TRD, tareas aperiódicas TRB y perturbaciones, y minimiza la energía consumida por las tareas en una plataforma dada. Para ello, asume que la energía consumida por los trabajos solamente depende de su tiempo de ejecución y de la frecuencia de las CPUs.

Durante la fase *off-line* se calcula la frecuencia mínima de ejecución del conjunto de tareas TRD para asegurar el cumplimiento de plazos y maximizar la utilización, y la frecuencia máxima que el sistema soporta sin violar la restricción térmica impuesta. A continuación, se resuelve un Problema de Programación Entera (PPE) basado en particionado por plazos, que calcula los ciclos de cada tarea a ejecutar por cada intervalo de planificación. En [5] se demuestra que la matriz de restricciones de este PPE es totalmente unimodular (TUM), por lo que puede resolverse como un Problema de Programación Lineal (PPL). En [7] se modifica la formulación del PPE mejorando la planificabilidad pero manteniendo una matriz de restricciones TUM de modo que el problema sigue siendo resoluble como PPL. La solución al problema fluido se obtiene en ciclos de CPU, por lo que no requiere discretización posterior.

La asignación de tareas a CPUs se obtiene a continuación también en la fase *off-line* mediante una planificación basada en el algoritmo *Zero-Laxity* [40, 7], aplicado por cada uno de los intervalos definidos por particionado por plazos. Este algoritmo tiene la particularidad de que las porciones de trabajos a planificar en el mismo intervalo comparten activación y plazo.

El algoritmo *Zero-laxity* aplicado se guía por los siguientes marcadores:

- Inicio/fin de un intervalo
- Una porción de trabajo pendiente alcanza laxitud 0
- Finalización de una porción de trabajo

En cada marcador, se asigna la máxima prioridad a las tareas cuya porción de trabajo restante tiene laxitud 0. Las tareas actualmente en ejecución y con laxitud mayor que 0 obtienen una prioridad intermedia, y el resto una prioridad baja. La asignación de tareas a CPUs minimiza los cambios de contexto impidiendo la migración de las tareas en ejecución que posean la suficiente prioridad para seguir ejecutándose.

La fase *on-line* utiliza la planificación *off-line* para extraer las referencias de ejecución calculadas. Estas referencias indican cuándo debe comenzar o finalizar cada porción de tarea y en qué procesador, a fin de asegurar su cumplimiento aunque lleguen tareas aperiódicas TRB o existan perturbaciones. Estas perturbaciones podrían deberse por ejemplo a variaciones paramétricas del modelo sobre el que se ha calculado la planificación *off-line* del conjunto de tareas TRD [3, 5, 7].

Al detectar la llegada de una tarea aperiódica (TRB) en tiempo de ejecución, se calcula la nueva frecuencia mínima a la que deben trabajar las CPUs. Si la nueva frecuencia supera la máxima calculada en la fase *off-line*, la aperiódica se rechaza. Además se calcula el número de ciclos que debe ejecutarse esta tarea en cada intervalo para cumplir las restricciones temporales.

La comparación preliminar entre AlECS y RUN realizada en el contexto de este TFM y publicada en [7] mostró el bajo número de cambios de contextos y migraciones de AlECS. De aquí surgió la idea de realizar un agrupamiento (*clustering*) de tareas previo a la planificación global: en el mejor de los casos se lograría eliminar las migraciones, y en el peor (en el que RUN tampoco lograría particionar) se estaría en un escenario favorable para un planificador global como AlECS.

Por otra parte, como se ha explicado anteriormente, la fase *off-line* de AlECS consigue una planificación y asignación del conjunto de tareas TRD durante el hiperperiodo previas a la ejecución. Es decir, en un contexto estrictamente TRD, AlECS permite obtener fácilmente un ejecutivo cíclico, cuyo coste de implementación es muy bajo. Esto permitiría situarlo en el contexto de los procedimientos de AUTOSAR [41] y ARINC [42] de automoción y aeronáutica respectivamente.

De estas dos ideas surge el esquema de planificación CAlECS [8]. En CAlECS, la fase de agrupamiento previo de tareas, la articulación de las fases *off-line* para obtener una planificación de las tareas TRD mediante ejecutivo cíclico son aportaciones de este TFM. En [8] se incluyó también una comparación entre los planificadores AlECS, CAlECS y RUN que forma parte de este TFG, y que se recoge en el Cap. 5 de esta memoria en versión extendida. Esta comparativa confirma los resultados de la preliminar publicada en [7]. Así mismo, muestra el acierto de añadir la fase previa de agrupamiento, que unida al uso de AlECS cuando el agrupamiento no es total, proporciona los mejores resultados.

3.1.6. Algoritmos de agrupamiento

La aproximación más parecida a la adoptada en este trabajo para el agrupamiento de tareas es [43]. En ella se resuelve un Bin packing problem (BPP) para distribuir tareas entre grupos (*clusters*) de un número dado de CPUs. En nuestro caso, es el propio algoritmo quien selecciona el tamaño del *cluster*, con el objetivo de minimizar el número de migraciones a la vez que se busca conseguir una utilización máxima.

3.2. Herramientas para la simulación de planificadores TR

Existe una amplia variedad de herramientas de soporte a diferentes etapas del proceso de diseño de un planificador. Su principal inconveniente es que se centran normalmente en aspectos particulares aislados (ej. influencia de la memoria cache).

La propuesta más cercana a nuestro entorno *Tertimuss* es *SimSo* [44]. Tertimuss ofrece mucho más control sobre la definición de tareas y del sistema. Entre las características de *Tertimuss* que se encuentran ausentes en *SimSo* están el modelado y simulación del comportamiento térmico, el control dinámico de la frecuencia de los procesadores, y el análisis automático de planificaciones. Esto último permite la ejecución automática de un gran número de pruebas para comparar el comportamiento de dos o más planificadores.

Características presentes en *SimSo* y ausentes en *Tertimuss* son la integración de los efectos de la memoria cache dentro de la planificación, según un modelo propuesto por los autores. *SimSo* también posee una mayor cantidad de planificadores implementados, así como generadores automáticos de tareas.

Por otra parte, *Tertimuss* trabaja con el *cc* (WCET expresado en ciclos de procesador) de las tareas, *SimSo* trabaja con el WCET (peor coste de ejecución de las tareas en segundos).

En cuanto a la simulación, en ambos entornos la simulación va marcada por eventos. Sin embargo, la definición de los planificadores en *Tertimuss* utiliza el paradigma de orientación a objetos, mientras que en *SimSo* lo combina con el paso de mensajes. *SimSo* procesa los eventos invocando bibliotecas externas introduciendo una sobrecarga ausente en *Tertimuss*.

Capítulo 4

Algoritmo CAlECS (Clustered AlECS)

En este capítulo se expone el algoritmo de planificación CAlECS (Clustered Allocation and Execution Control Scheduler). Este algoritmo parte de un algoritmo previo denominado AlECS destinado a la planificación de conjuntos de tareas TRD sobre multiprocesadores con restricciones térmicas (Sec. 3.1.5).

4.1. Introducción

4.1.1. Motivación

En la investigación que condujo a la propuesta de AlECS y como parte de este TFM se realizó una comparativa preliminar con el planificador RUN, referencia en términos de utilización de procesadores, para estimar el rendimiento de AlECS y detectar posibles puntos de mejora [7].

Los resultados nos llevaron a realizar experimentos más exhaustivos en los que se comprobó que AlECS siempre obtiene en media menos cambios de contexto que RUN, aunque no consigue batir a RUN en migraciones cuando el ratio $\frac{número\ de\ tareas}{número\ de\ procesadores}$ es elevado. Se concluyó que esto se debe a que RUN es capaz de crear grupos (*clusters*) de tareas con utilización máxima, limitando el número de migraciones, en ciertos conjuntos de tareas que AlECS, por su carácter fluido, tiende a distribuir generando más migraciones. También se apreció que el número de migraciones de AlECS depende del resultado del PPL inicial que resuelve y no del algoritmo de laxitud cero utilizado para la asignación de tareas a procesadores sobre la solución del PPL.

Estos resultados motivaron la decisión de incluir una etapa de agrupamiento *(clustering)* en el algoritmo AlECS como posible solución, a fin de reducir el número de migraciones. Tras el estudio de literatura sobre particionado y agrupamiento, ninguno de ellos aseguraba su funcionamiento sobre un sistema con utilización máxima, lo que

nos llevó a proponer una solución propia que lo garantice.

4.1.2. Novedad

La novedad de CAlECS se basa en tres puntos:

- Agrupamiento (*clustering*) previo de las tareas TRD
- Planificación independiente de cada uno de los grupos obtenidos.
- Obtención completamente fuera de línea de un ejecutivo cíclico.

El primer punto persigue reducir el número de migraciones, limitándolas a un *cluster*. El mejor caso es un particionado perfecto (sin migraciones). El peor caso es la obtención de un único cluster, dejando el problema de planificación global en su dimensión inicial. El segundo punto reduce el tamaño del problema que AlECS (o cualquier otro planificador global) tiene que resolver, llegando a eliminar el problema de planificación global si se da el mejor caso que acabamos de citar.

La reducción del tamaño del problema de planificación global reduce el espacio de soluciones. Esto disminuye el coste computacional, un factor no crítico al tratarse de cálculos fuera de línea.

Otro efecto derivado de nuestra solución basada en agrupamiento previo, como se muestra posteriormente en el Cap. 5, es que el número de cambios de contexto en algunos casos se reduce y en el resto se mantiene similar respecto al algoritmo de partida AlECS.

La obtención completamente fuera de línea de un ejecutivo cíclico se traduce en una planificación de mínima sobrecarga en tiempo de ejecución. Este ejecutivo permite por una parte alcanzar la máxima utilización teórica del núcleo de procesamiento, y por otra reducir el número de migraciones que se producen respecto a AlECS o el planificador de referencia RUN (Cap. 5).

Estos tres puntos permiten que CAlECS explote la principal ventaja de la planificación global (maximizar la utilización) reduciendo su principal inconveniente (el incremento de la sobrecarga derivado de un habitual exceso de cambios de contexto y migraciones). Visto desde otro punto de vista, explota la principal ventaja de la planificación particionada (ausencia de migraciones) evitando su principal limitación (baja utilización de los núcleos de procesamiento) (Cap. 3).

4.1.3. Visión general

En esta sección proporciona una visión general de CAlECS que facilitará la identificación de los diferentes componentes que intervienen en el algoritmo y de sus

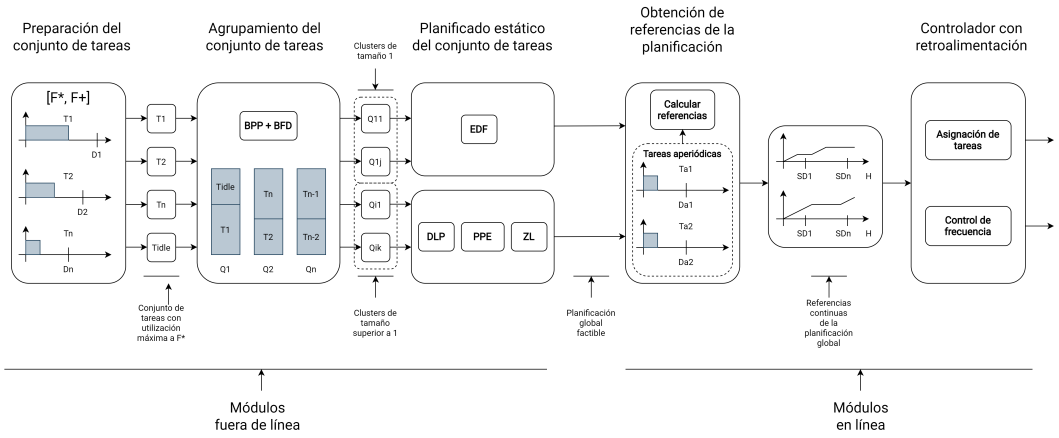


Figura 4.1: Visión general de CAlECS.

funciones cuando se desarrollen en las siguientes secciones.

La arquitectura de CAlECS se muestra en la Fig. 4.1. Los módulos *preparación del conjunto de tareas*, *agrupación del conjunto de tareas*, y *planificación estática del conjunto de tareas* se ejecutan fuera de línea, y generan un ejecutivo cíclico que ejecuta las tareas respetando las restricciones del problema y utilizando los CPUs con la menor frecuencia posible.

Nuestro trabajo se centra solamente en la obtención del ejecutivo cíclico a partir de tareas periódicas. En caso de que sea necesario gestionar tareas aperiódicas TRB, o tolerar perturbaciones en la ejecución, se utilizan los módulos *obtención de referencias de la planificación* y *controlador con retroalimentación*, cuyos detalles pueden consultarse en [8]. Estos módulos se ejecutan en tiempo de ejecución, y el primero de ellos utiliza el ejecutivo cíclico como entrada.

El módulo de *preparación del conjunto de tareas* determina si dicho conjunto es planificable en los CPUs disponibles cumpliendo las restricciones TRD. En caso de que lo sea, selecciona la frecuencia mínima a la que pueden operar los CPUs cumpliendo además los plazos de las tareas, para reducir de esta manera el consumo energético.

El módulo de *agrupación del conjunto de tareas* divide el conjunto de tareas en k conjuntos (*clusters*) tal que la utilización s_j en cada conjuntos Q_j sea un número entero y que $\Sigma_j^k s_j = m$. Para realizarlo, el módulo resuelve varios *Bin Packing Problem* usando la heurística *Best Fit Descending*. El objetivo es minimizar el número de migraciones de las tareas, restringiendo las migraciones que puede realizar una tarea a los CPUs asignados al conjunto al que pertenece. En el peor caso, este módulo devuelve solamente un conjunto Q_1 y $s_1 = m$, lo que obliga a resolver un problema de planificación global en los siguientes módulos.

El objetivo del *planificador estático del conjunto de tareas* es encontrar una planificación factible para cada uno de los conjuntos calculados en el modulo anterior.

Para ello, a cada conjunto Q_j se le asigna s_j núcleos de procesamiento, y se aplica un algoritmo de planificación sobre ellos. Cuando $s_j = 1$, Q_j se asigna a un único núcleo de procesamiento y se utiliza una política EDF para encontrar una planificación factible. Si $s_j > 1$, entonces Q_j se asigna a s_j núcleos de procesamiento y se realiza una planificación del conjunto de tareas en el hiperperíodo usando un algoritmo de planificación global. En la solución propuesta se utiliza la etapa de planificado del algoritmo *AlECS* para estos últimos, aunque puede utilizarse cualquier otro algoritmo de planificación global. Esta etapa de planificación realiza un particionado por plazos *(deadline partitioning*, Cap. 2). Mediante un problema de programación entera (que puede resolverse como un problema de programación lineal) calcula la cantidad de trabajo que debe ejecutar cada tarea en cada intervalo. La planificación en cada intervalo se resuelve mediante un algoritmo de laxitud cero (*Zero-Laxity*).

La unión de las planificaciones de todos los *clusters* proporciona una planificación única, factible para el conjunto de tareas TRD. Dicha planificación puede ser implementada de una forma sencilla como un simple ejecutivo cíclico, con un mínimo sobre coste en tiempo de ejecución en el caso de que no se tengan en cuenta tareas aperiódicas o perturbaciones.

En tiempo de ejecución, el módulo *obtención de referencias de la planificación* utiliza esta planificación factible fuera de línea para generar referencias de asignación de tareas para el siguiente módulo. Estas referencias consisten en el número de ciclos que cada tarea debe ser asignada a cada núcleo de procesamiento durante cada intervalo. El submódulo *manejo de aperiódicas* es el encargado de aceptar o rechazar tareas aperiódicas tiempo real blando y de generar sus referencias en línea.

El *controlador con retroalimentación* se centra en la asignación de tareas y en su ejecución. Este controlador varía la frecuencia de los núcleos de procesamiento para asegurar que las tareas se ejecutan según las referencias calculadas previamente.

En el Anexo. C se encuentra un ejemplo del funcionamiento del algoritmo.

4.2. Preparación del conjunto de tareas

El modulo de preparación del conjunto de tareas comprueba que el conjunto de tareas TRD $\mathcal{T}$ es planificable en los núcleos disponibles y proporciona un conjunto de frecuencias $\mathcal{F} = \{F^*, \ldots, F_{max}\}$, tal que F^* minimiza el consumo de energía, y F_{max} representa la frecuencia máxima permitida por el núcleo de procesamiento.

Se asume que el núcleo de procesamiento soporta la técnica de escalado dinámico de energía (Dynamic voltage scaling (DVFS)), permitiendo variar la frecuencia del mismo entre un conjunto de valores preestablecidos $\{F_1, ..., F_{max}\}$. Esta técnica modela la

energía consumida por el núcleo de procesamiento como

$$E_j = \int_{\zeta_1}^{\zeta_2} P_{CPU_j}(F) d_\zeta \tag{4.1}$$

En esta expresión $P_{CPU_j}(F)$ representa la energía consumida por el CPU_j, calculándose esta como $P_{CPU_j}(F) = P_{dyn_j}(F) + P_{leak_j}$. Esta energía consumida depende de la potencia dinámica requerida por la actividad computacional de las tareas, $P_{dyn}(F)$, y de la potencia debida a fugas P_{leak}. La potencia dinámica puede ser modelada como $P_{dyn}(F) = kF^3$, siendo k una constante de modelado, mientras que la potencia debida a fugas se puede modelar como una función lineal de la temperatura [45], $P_{leak} = \delta T + \rho$, donde T es la temperatura del núcleo de procesamiento y δ y ρ son constantes de modelado.

Por lo tanto, la frecuencia del núcleo de procesamiento que minimiza el consumo de energía mientras que satisface los requisitos temporales es:

$$F^{**} = \text{máx}\{\frac{1}{m}\sum_{i=1}^{n}\frac{cc_i}{\omega_i},\ \ \max_{\forall i}\{\frac{cc_i}{\omega_i}\}\}. \tag{4.2}$$

A la frecuencia F^{**} la utilización del sistema es $U = \sum_{i=1}^{n} \frac{cc_i}{\omega_i F^*} \leq m$, y $\forall i\ u_i \leq 1$.

Si $F_{max} < F^{**}$ implica que el conjunto de tareas no puede planificarse en la plataforma.

Dada la naturaleza discreta del conjunto de frecuencias de los núcleos de procesamiento, la frecuencia de trabajo que se selecciona es:

$$F^* = \text{mín}\{F \in \mathcal{F} | F \geq F^{**}\} \tag{4.3}$$

Cuando se calcula F^* en la Ec. 4.3, se pretende obtener un sistema con la máxima utilización posible. En las configuraciones en las que no se obtenga esta utilización máxima, debido a la naturaleza de las tareas, se introduce una o varias tareas de relleno $\tau_*(cc_*, H)$ para obtener un sistema con máxima utilización:

$$cc_* = m \times F^* - \sum_{i=1}^{n}\frac{cc_i}{\omega_i} \tag{4.4}$$

Con la frecuencia mínima y la máxima se define el conjunto de frecuencias operacionales como:

$$\mathcal{F}^s = \{F \in \mathcal{F} | F^* \leq F \leq F_{max}\}, \tag{4.5}$$

4.3. Agrupamiento del conjunto de tareas

El problema de *clustering* puede ser formalmente definido de la siguiente forma. Dado un conjunto de tareas $\mathcal{T}$ con n tareas TRD planificables en m núcleos de procesamiento, se debe encontrar una partición $Q = \{Q_1, ..., Q_k\}$ de $\mathcal{T}$ en $Q_1, ..., Q_k$ subconjuntos (*clusters*) tal que estos sean disjuntos $\bigcup_{j=1}^{k} Q_j = \mathcal{T}$, la utilización s_j de tareas en el *cluster* Q_j sea un entero, y $\Sigma_j s_j = m$.

Este módulo solamente trata con sistemas con utilización máxima. Las alternativas para transformar un sistema sin utilización máxima en un sistema con utilización máxima serán debatidas posteriormente.

Un conjunto de tareas TRD que es factible antes del *clustering* lo seguirá siendo después debido a que el algoritmo utilizado asegura que la utilización en cada *cluster* es igual al número de núcleos de procesamiento asignados al mismo (i.e.la máxima utilización del *cluster*).

Para encontrar una combinación de *clusters* Q de capacidad s que asegure la máxima utilización en cada caso, se proponen las siguientes restricciones:

$$\left(\sum_{i=1}^{n} u_i * x_{i,s,j} \right) - x_{s,j} * s = 0 \tag{4.6}$$

donde $x_{i,s,j}$ es una variable de decisión que es 1 si la tarea τ_i es asignada al *j-esimo cluster* de tamaño s, y $x_{s,j}$ es otra variable booleana, con valor 1 cuando el *j-esimo cluster* de tamaño s contiene alguna tarea. Esta ecuación considera todos los escenarios posibles, existiendo un máximo de $\lfloor \frac{m}{s} \rfloor$ *clusters* de tamaño s para $s = 1, \ldots, m$. La constante n representa el número total de tareas, y la constante m el número de núcleos de procesamiento en el sistema. La constante u_i representa la utilización de cada una de las tareas.

La siguiente restricción garantiza que cada tarea pertenece solamente a un *cluster*:

$$\sum_{s=1}^{m} \sum_{j=1}^{\lfloor \frac{m}{s} \rfloor} x_{i,s,j} = 1 \tag{4.7}$$

Por último, la suma de las utilizaciones de todos los *clusters* debe ser igual al número de núcleos de procesamiento inicial,

$$\sum_{s=1}^{m} \sum_{j=1}^{\lfloor \frac{m}{s} \rfloor} x_{s,j} * s = m \tag{4.8}$$

Las ecuaciones anteriores 4.6, 4.7 y 4.8 definen un espacio de posibles soluciones que engloba todos los *clusters* posibles a partir de la definición del problema. Este espacio de soluciones puede ser usado para encontrar un agrupamiento con el menor

número posible de migraciones. Encontrar este punto óptimo absoluto implica explorar todo el espacio de estados, aplicando el algoritmo de planificado sobre cada una de las soluciones analizando su número de migraciones, convirtiendo la solución en inabordable debido a su complejidad computacional.

Por este motivo, se propone relajar el problema mediante una heurística utilizada en la búsqueda. Se parte de la intuición de que el número de migraciones decrece con el tamaño del *cluster*; más adelante en la sección 5 se mostrará la veracidad de esta intuición. Esto resulta en la formulación del siguiente PPE:

$$\text{mín} \quad \sum_{s=1}^{m} \sum_{j=1}^{\lfloor \frac{m}{s} \rfloor} x_{s,j} * m^{s-1} \tag{4.9}$$

$$\text{s.t.} \tag{4.10}$$

$$\forall s \in \{1..m\} \; \forall j \in \{1..\lfloor \frac{m}{s} \rfloor\} \tag{4.11}$$

$$\left(\sum_{i=1}^{n} u_i * x_{i,s,j} \right) - x_{s,j} * s = 0 \tag{4.12}$$

$$\forall i \in \{1..n\} \sum_{s=1}^{m} \sum_{j=1}^{\lfloor \frac{m}{s} \rfloor} x_{i,s,j} = 1 \tag{4.13}$$

$$\sum_{s=1}^{m} \sum_{j=1}^{\lfloor \frac{m}{s} \rfloor} x_{s,j} * s = m \tag{4.14}$$

$$x_{i,s,j} \in \{0,1\}, \; x_{s,j} \in \{0,1\} \tag{4.15}$$

La función objetivo de la ec. 4.15 maximiza el número de *clusters* de tamaño pequeño (empezando con tamaño 1) sujeto a las restricciones mostradas previamente.

Al tratarse de un PPE, su solución tiene un coste exponencial. Sin embargo, se puede aproximar una solución formulando dicho problema como otro problema NP-completo, como el *Bin Packing Problem* (BPP) [46], resoluble mediante heurísticas. Concretamente, el problema original se transforma en varios BPP que se van resolviendo de forma iterativa.

Para ello, primero se resuelve el problema con contenedores de tamaño 1. Todos los contenedores completos (utilización 1) son transformados directamente en *clusters*. Si existen contenedores con utilización menor a 1, se resuelve nuevamente otro BPP con contenedores de tamaño 2 (*clusters* con tamaño $k = 2$) a partir solamente de las tareas que se habían asignado a estos contenedores con utilización menor a 1. Después se prosigue con contenedores de tamaño 3 y superiores hasta que todas las tareas se encuentren en algún *cluster*.

En la solución propuesta se utiliza la heurística *Best-Fit Descending* (BFD) para

resolver el BPP, pero otras soluciones son también posibles. De hecho el problema puede ser transformado en un *Knapsack Problem* o un *Cutting Stock problem*.

Algorithm 1 Algoritmo de clustering

1: **Input** $\mathcal{T}$: Conjunto de tareas; m: Número de núcleos de procesamiento;
2: **Salida** Un conjunto de clusters;
3: **Funciones auxiliares**
· ResuelveBPP(*conjunto de tareas, volumenContenedor*) – *Resuelve el BPP para contenedores con volumen volumenContenedor tal que cada contenedor posea utilización máxima*;
· utilización(*contenedor*) – *Devuelve la suma de las utilizaciones de las tareas en el contenedor*;
4: $Q = \emptyset$,
5: $volumenContenedor = 1$,
6: $procesadoresPorAsignar = m$,
7: $tareasNoClusterizadas = T$
8: $\mathcal{T} = \{\mathcal{T} \cup \tau_{\text{idle}} | u_{\tau_{\text{idle}}} = m - \sum u_i\}$
9: **while** $volumenContenedor <= procesadoresPorAsignar$ **do**
10: $ResuelveBPP(tareasNoClusterizadas, volumenContenedor)$
11: **for all** $contenedor \in contenedores$ **do**
12: **if** $utilization(contenedor) == volumenContenedor$ **then**
13: *// Cada cluster está formado por (número de núcleos de procesamiento en el cluster, tareas en el cluster)*
14: $Q = Q \cup (volumenContenedor, contenedor)$
15: $tareasNoClusterizadas = tareasNoClusterizadas \setminus contenedor$
16: $procesadoresPorAsignar = procesadoresPorAsignar - volumenContenedor$
17: **end if**
18: **end for**
19: $volumenContenedor = volumenContenedor + 1$
20: **end while**
21: *// Asegura que todas las tareas y núcleos de procesamiento están en algún cluster*
22: **if** $procesadoresPorAsignar! = 0$ **then**
23: $Q = Q \cup (procesadoresPorAsignar, tareasNoClusterizadas)$
24: **end if**

El algoritmo se encuentra en la Fig. 1. Comienza añadiendo una tarea de relleno al conjunto para obtener una utilización del sistema igual al número de núcleos de procesamiento. La tarea a añadir debe tener una utilización $u_{\tau_{\text{idle}}} = m - \sum u_i$ (línea 8). Esta tarea se establece con un plazo relativo igual al hiperperíodo para de esta forma conseguir que tenga la menor prioridad: solo es usada durante el cálculo, y nunca será planificada en tiempo de ejecución. El algoritmo comienza resolviendo un BPP de tamaño 1. Mientras queden tareas sin asignar, se va incrementando en cada paso, en una unidad, el tamaño del contenedor. En cada uno de los pasos todos los contenedores que hayan alcanzado el máximo de su volumen son transformados en *clusters*. Estos *clusters* contendrán todas las tareas que se encuentren en el contenedor, y un número de núcleos de procesamiento igual al tamaño del contenedor.

Las variables $procesadoresPorAsignar$ y $tareasNoClusterizadas$, representan respectivamente el número de núcleos de procesamiento y el de tareas no asignadas aún a ningún cluster.

Debido a que la heurística usada al resolver el BPP no garantiza encontrar la

solución óptima, la condición en la línea 22 asegura que el último *cluster* contiene cualquier posible grupo de tareas con utilización $y < m$, que no se agrupó en un contenedor cuando $volumenContenedor = y$.

4.3.1. Tarea de relleno: opciones

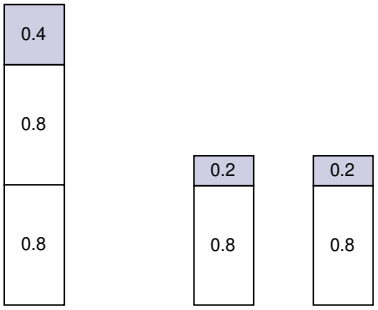
El añadido de una sola tarea de relleno τ_{idle} como en la línea 8 del Alg. 1, es una solución simple. Encaja correctamente en el sistema conjunto de planificación de CAlECS, que en etapas posteriores puede lidiar con tareas aperiódicas y perturbaciones. Sin embargo, este esquema puede imposibilitar la obtención de un *clustering* óptimo, entendiendo por óptimo el que minimiza el PPE de la ec. 4.15, y por tanto limitar la reducción de migraciones. Consideremos dos tareas tal que $u_1 = u_2 = 0{,}8$ que serán planificadas en dos núcleos de procesamiento. Añadiendo una tarea de relleno con utilización $u_{\text{idle}} = 0{,}4$, lleva a la solución de la Fig. 4.2 (a), con un único *cluster* conteniendo las tres tareas τ_1, τ_2 y τ_{idle}. Un planificador global podría producir migraciones en este esquema. Alternativamente, si se hubiesen añadido dos tareas de relleno con utilización 0,2, se habría obtenido la solución (b) en la Fig. 4.2, con dos *clusters* de tamaño 1. Esta solución no habría producido migraciones, al proporcionar un particionado del sistema.

4.4. Planificado de grupos monoprocesador

Para la planificación de los grupos de tamaño $s_j = 1$ recurrimos al algoritmo Earliest Deadline First (EDF) [47], óptimo en el caso de monoprocesadores en cuanto a utilización, y ampliamente conocido y utilizado. Este algoritmo asigna más prioridad a la instancia de tarea que tengan el plazo más cercano. Así, la prioridad de una instancia de tarea solamente cambia cuando se activa otra instancia con un plazo absoluto más corto.

En un esquema de tareas con plazos implícitos como el que se está tratando, la única instancia que puede sufrir un cambio de contexto involuntario (producido por el algoritmo de planificación) entre la finalización de dos plazos consecutivos de dos instancias es la que se estaba ejecutando al finalizar el primero de dichos plazos. Esta situación solamente sucede si la nueva instancia de alguna de las tareas que agota su plazo en el primero de los dos plazos tiene un plazo absoluto inferior al de la instancia que se estaba ejecutando. Esto supone que el número de cambios de contexto generados por el algoritmo sea previsiblemente bajo.

Este caso se ejemplifica en la figura 4.3, donde se aprecia como en el instante temporal t_3 tras la llegada del trabajo $T_{2,1}$, el único trabajo que puede ser expulsado



(a) 1 tarea relleno

(b) 2 tareas relleno

Figura 4.2: Diferencias en el clustering entre el uso de una tarea de relleno (fuerza un cluster) y dos tareas de relleno (las tareas de relleno están en color gris)

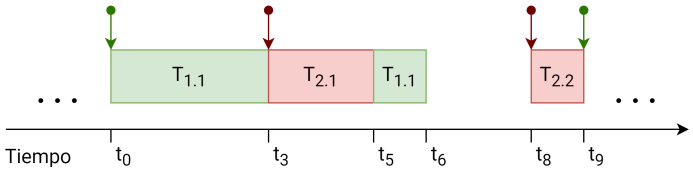


Figura 4.3: Cambio de contexto EDF.

involuntariamente es el trabajo $T_{1,1}$

4.5. Planificado de grupos multiprocesador

Veamos ahora la planificación de los *clusters* Q_j de tamaño $s_j > 1$. El algoritmo diseñado parte de la planificación proporcionada por AlECS, que se explica en las siguientes secciones.

4.5.1. Asignación de trabajo

Para resolver la asignación de trabajo para el conjunto de tareas contenida en el *cluster* Q_j, se formula un problema de programación entera (PPE) donde cada restricción contempla un comportamiento deseado del planificador. La solución a este PPE es un conjunto X de x_i^k que representa el trabajo que cada tarea τ_i en Q_j debe ejecutar en cada intervalo de planificación, I_{SD}^k, $k = 1, .., \alpha$, como sucede con las aproximaciones basadas en DP-fair (Cap. 3). Cada intervalo de planificación viene definido como el tiempo entre plazos de instancias de tareas consecutivas (sd), $I_{SD}^k = [sd_{k-1}, sd_k)$, donde los plazos de cada instancia sd_k está en el conjunto ordenado de plazos $SD = \{sd_0, ..., sd_\alpha\}$, donde $sd_0 = 0$ y sd_α corresponden al hiperperíodo.

Se define la *laxitud absoluta* (*laxity*) de τ_i en ciclos como $cc_i^* = (\omega_i F^*) - cc_i$, siendo este el número de ciclos que τ_i puede estar sin ejecutarse antes de su plazo

sin comprometerlo. La variable ω_i representa el tiempo restante hasta el plazo de τ_i.

La formulación del PPE viene dada por la ec. (4.16). Cada restricción se define por intervalo de planificación y por tarea, en el hiperperíodo. La restricción de *máxima utilización* asegura que la utilización del sistema por I_{SD}^k es 100 %. Las restricciones de *ejecución* y *laxitud* aseguran que las tareas individuales τ_i se completen antes de su plazo. La restricción de *secuencialidad* evita que una misma tarea se ejecute de forma paralela consigo misma.

$$
\begin{aligned}
&\max \quad \sum_{i=1}^{n} \sum_{k=1}^{\alpha} x_i^k \\
&s.t \\
&\quad \forall k \sum_{i=1}^{n} x_i^k = s_j \times |I_{SD}^k| \times F^* && \text{Máxima utilización} \\
&\qquad \text{si } r_i = 0 \ \sum_{j=\gamma}^{k} x_i^j = cc_i && \text{Ejecución} \\
&\qquad \text{si } r_i \neq 0 \ \sum_{j=\gamma}^{k} x_i^j \geq max\{0, r_i - cc_i^*\} && \text{Laxitud} \\
&\qquad \text{donde } \gamma \text{ es 1 o el último intervalo de plazo} \\
&\qquad \forall i \quad x_i^k \leq |I_{SD}^k| \times F^* && \text{Secuencialidad}
\end{aligned}
\tag{4.16}
$$

Dado un conjunto de tareas $\mathcal{T}$, donde la utilización a frecuencia F^* es igual al número de núcleos de procesamiento, se demuestra que la matriz de restricciones del PPE es unimodular y puede formularse como un Problema de programación lineal PPL (resoluble por ejemplo mediante el algoritmo Simplex), cuya solución siempre es entera [7]. Si cada tarea τ_i se ejecuta exactamente x_i^k ciclos durante el k-esimo intervalo, entonces se obtiene una solución óptima.

Política de laxitud cero

El PPE resuelto previamente proporciona el número de ciclos x_i^k de la tarea τ_i que deben ser ejecutados durante el intervalo I_{SD}^k con los núcleos de procesamiento a frecuencia F^* para cumplir con los plazos de todas las tareas. Aún así, no resuelve ni la planificación de las tareas ni su asignación a núcleos de procesamiento dentro de cada intervalo. Para resolver la planificación y asignación se recurre a un algoritmo de laxitud cero (ZL, *Zero Laxity*) [40], adaptado como indica el Alg. 2. La laxitud se define como el tiempo del que dispone la instancia de una tarea para poder completar su ejecución sin violar su plazo.

Según este algoritmo de planificación global, las tareas que más prioridad poseen son las que tienen laxitud cero: si no son ejecutadas inmediatamente perderán su plazo. Las siguientes en prioridad son las que están actualmente en ejecución (de esta forma se evita cambios de contexto innecesario). El resto de tareas tendrán una prioridad

Algorithm 2 Algoritmo ZL

Input I_{SD}^k – Intervalos de planificación; X^k – ciclos de *CPU* por intervalo de cada tarea; ex_i^k – Número de ciclos en el intervalo $[t_0, t_f]$
Output Una planificación factible;
$k = 0$,
for $t = t_0$ **to** t_f **do**
    Calcula la laxitud de cada tarea activa
    if *reach* I_{SD}^{k+1} **then**
        k=k+1;
        Calcula las prioridades de las tareas como: *Las tareas con laxitud cero obtienen la máxima prioridad, seguida por las tareas que están siendo ejecutadas*
        Ejecuta las m tareas con máxima prioridad
    else if *alcanza laxitud cero* **then**
        Calcula la prioridad de las tareas
        Ejecuta las m tareas con máxima prioridad
    end if
end for

menor a todas las anteriores, que será inversamente proporcional a su laxitud.

Para realizar la asignación de tareas a núcleos de procesamiento, todas las tareas que ya estaban en ejecución se mantienen en el mismo núcleo de procesamiento. Las que no lo estaban se asignan arbitrariamente entre los núcleos de procesamiento. Se experimentó la asignación del resto de tareas al último núcleo de procesamiento en el que se había ejecutado para de esta forma intentar reducir las migraciones, aunque los resultados no mostraron una reducción del número de migraciones, mientras que aumentaba el coste de su cálculo.

4.6. Obtención del ejecutivo cíclico

Una vez obtenida la planificación de los diferentes *clusters*, la generación del ejecutivo cíclico es una operación trivial que consiste en la agregación de las diferentes planificaciones. Solamente se debe tener en cuenta que la agregación de todos los *clusters* debe tener un hiperperíodo igual al hiperperíodo del conjunto de tareas inicial. En caso de que algún grupo posea un hiperperíodo inferior, deberá replicar su planificación tantas veces sea necesario hasta lograr tener una planificación en este hiperperíodo global.

4.7. Complejidad computacional del algoritmo

El módulo de preparación del conjunto de tareas calcula la mínima frecuencia de operación, y esto lo hace con un coste lineal en el número de tareas para calcular F^{**}, y luego lineal en el número de frecuencias para calcular F^*. Esto nos lleva a que su coste es $\mathcal{O}(max(n, |\mathcal{F}|))$, siendo $\mathcal{F}$ el número de frecuencias distintas disponibles en los

núcleos de procesamiento.

La complejidad computacional del algoritmo de agrupación de tareas (Alg. 1) es como sigue. El bucle externo `while` itera m veces. En cada iteración, primero ejecuta *ResuelveBPP* cuya complejidad es $n \times \log(n)$. Después, este ejecuta el bucle interno `for`. Este es ejecutado $\alpha \times m$ veces (la resolución de un BPP con utilización m produce a lo sumo $2 \times m$ contenedores), estableciendo una complejidad computacional de orden $\mathcal{O}(m \times n \times \log(n))$, ya que $m \leq n \times \log(n)$.

La matriz de resolución del PPE que se encuentra en la ec. (4.16) es unimodular [8]. Esto permite relajar la formulación y resolverlo como un PPL mediante el algoritmo Simplex, porque este devolverá siempre una solución entera en estos casos. La resolución del Simplex tiene un coste de orden $\mathcal{O}(n^2)$. Finalmente, tanto EDF como el Alg.2 se ejecutan en tiempo polinomial. Esto implica que la complejidad computacional de esta etapa es de orden $\mathcal{O}(n^2)$.

Por lo tanto, el algoritmo de planificado completo logra ejecutarse en un tiempo polinomial.

Capítulo 5

Comparativa entre RUN, AlECS y CAlECS

En este capítulo se realiza una comparativa entre los planificadores AlECS, CAlECS y RUN usando conjuntos de tareas periódicas con utilización máxima. Para ello, se analiza el número de cambios de contexto y migraciones producidos en cada caso. El planificador RUN es usado como una referencia óptima (Sec. 3), y las diferencias entre AlECS y CAlECS mostrarán si el nuevo planteamiento ha sido una forma efectiva de reducir el número de migraciones respecto al algoritmo de partida. Este trabajo es una versión ampliada de la comparativa realizada en [8], que estaba sujeta a restricciones de espacio. Se incluyen ahora nuevos experimentos y se realiza un análisis más pormenorizado.

5.1. Entorno de pruebas

La comparación se ha realizado utilizando el entorno de simulación Tertimuss (Anexo. A). Los cambios de contexto solamente se contabilizan cuando son producidos antes de la finalización de las tareas. Los producidos al inicio y final de las mismas se obvian, ya que se producen obligatoriamente, con independencia del algoritmo usado. Se considera una migración cuando una tarea retoma su ejecución en un núcleo de procesamiento diferente al que estaba previamente ejecutándose tras un cambio de contexto.

Los conjuntos de tareas son generados mediante el algoritmo UUniFast-Discard [48]. La generación se parametriza para que estos conjuntos se generen con una utilización igual al número de núcleos de procesamiento en el experimento, de modo que la utilización de los núcleos de procesamiento sea máxima. Para evitar que el hiperperíodo tome valores excesivamente grandes, los períodos de cada una de las tareas se seleccionan aleatoriamente entre los divisores de 60, para de esta forma obtener un hiperperíodo de 60 segundos como máximo. En los experimentos, se han usado sistemas

con 2 y 4 núcleos de procesamiento con una tasa de tareas por núcleo de procesamiento (TPP) de 4, 8, 12, 16, 20 y 24. La tasa TPP se define como *número de tareas / número de núcleos de procesamiento*. Por cada una de las combinaciones se han ejecutado 500 experimentos, resultando un total de 6000 experimentos diferentes.

El algoritmo de planificación RUN opera con el peor caso de ejecución de las tareas medido en segundos en lugar de ciclos. Esto puede producir que al obtener la equivalencia en ciclos, se solicite la actuación del planificador durante el transcurso de un ciclo y no al final, resultando en un comportamiento no aplicable sobre un núcleo de procesamiento. Para evitar esta situación, y realizar una comparación justa, se ajustó la generación de las tareas obligando a que el número de ciclos que tarda una tarea en ejecutarse en el peor caso sea múltiplo del período de la tarea.

Siguiendo la descripción original del algoritmo RUN [35], este se ha implementado usando la política *Worst-Fit* para la operación de empaquetado, la política EDF para la planificación de los servidores, y la política original de asignación de tareas a núcleos de procesamiento. Cuando se aplica EDF, los empates en los plazos de los servidores son resueltos seleccionando el último servidor que se ha ejecutado; en caso de un nuevo empate se selecciona uno aleatoriamente.

Dado que los tres planificadores son óptimos en cuanto a utilización, y todos los conjuntos de tareas son factibles, los tres planificadores logran calcular una planificación factible en todos los experimentos.

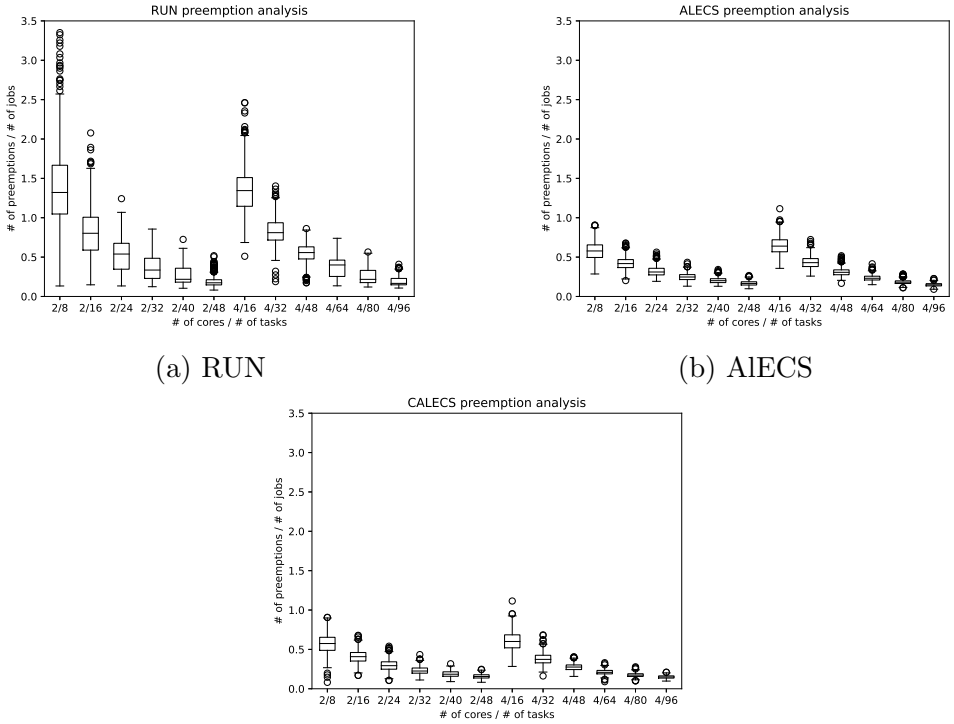
5.2. Análisis de cambios de contexto

Los boxplots de la figura 5.1 muestran el número medio de cambios de contexto por trabajo en cada experimento (eje Y) en cada una de las configuraciones de tareas y núcleos de procesamiento (eje X). El análisis de la media y la desviación típica de los mismos datos se encuentra en la tabla 5.1. La tabla 5.2 recoge los cuartiles, y la tabla 5.3 el valor mínimo y el máximo.

Estos resultados muestran que la media, desviación típica y los cuartiles de AlECS son menores a los de RUN en todos los casos, aunque la diferencia entre ambos se reduce según el TPP aumenta. También se comprueba que la diferencia en la media se reduce según aumenta el número de núcleos de procesamiento o el TPP.

En AlECS, con TPP de 4 y 8, la media de cambios de contexto es menor en dos núcleos de procesamiento, mientras que en el resto de los casos es menor en 4 núcleos de procesamiento. Según estos resultados, AlECS se comporta mejor conforme aumenta el número de núcleos de procesamiento y tareas.

En el caso de dos núcleos de procesamiento, el valor mínimo es siempre menor en



(a) RUN

(b) AlECS

(c) CAlECS

Figura 5.1: Resultados de la experimentación, número de cambios de contexto / número de trabajos.

Número de CPUs	Número de tareas	RUN		AlECS		CAlECS	
		Media	Desviación típica	Media	Desviación típica	Media	Desviación típica
2	8	1.405	0.572	0.578	0.117	0.568	0.125
	16	0.809	0.322	0.420	0.082	0.407	0.086
	24	0.530	0.208	0.319	0.064	0.297	0.069
	32	0.362	0.151	0.248	0.046	0.230	0.048
	40	0.265	0.110	0.203	0.037	0.187	0.038
	48	0.199	0.077	0.168	0.029	0.158	0.029
4	16	1.348	0.300	0.648	0.117	0.606	0.120
	32	0.823	0.172	0.432	0.077	0.380	0.074
	48	0.547	0.131	0.311	0.051	0.275	0.046
	64	0.375	0.121	0.235	0.038	0.211	0.034
	80	0.255	0.093	0.183	0.028	0.174	0.026
	96	0.193	0.065	0.149	0.021	0.150	0.020

Tabla 5.1: Resultados de la experimentación, media y desviación típica del número de cambios de contexto / número de trabajos.

RUN. Con cuatro núcleos de procesamiento, solo en dos de las seis combinaciones dicho valor mínimo es mayor en RUN.

Al comparar los resultados anteriores de AlECS y RUN con los de CAlECS, se comprueba que este último obtiene menor número de cambios de contexto en media, y también cuartiles menores, salvo en el caso de cuatro núcleos de procesamiento y 96

Número de CPUs	Número de tareas	RUN			AlECS			CAlECS		
		Q1	Q2	Q3	Q1	Q2	Q3	Q1	Q2	Q3
2	8	1.046	1.321	1.667	0.495	0.578	0.655	0.488	0.575	0.654
	16	0.588	0.803	1.007	0.367	0.418	0.469	0.351	0.408	0.461
	24	0.344	0.539	0.676	0.274	0.311	0.358	0.248	0.293	0.343
	32	0.229	0.335	0.486	0.215	0.247	0.278	0.198	0.225	0.264
	40	0.182	0.218	0.359	0.176	0.201	0.226	0.159	0.185	0.215
	48	0.147	0.174	0.213	0.145	0.165	0.187	0.136	0.156	0.178
4	16	1.144	1.345	1.511	0.567	0.639	0.720	0.520	0.600	0.685
	32	0.716	0.810	0.937	0.380	0.430	0.481	0.329	0.373	0.426
	48	0.475	0.558	0.632	0.276	0.307	0.338	0.244	0.276	0.303
	64	0.254	0.400	0.464	0.209	0.231	0.258	0.189	0.209	0.233
	80	0.176	0.217	0.332	0.165	0.181	0.198	0.156	0.173	0.191
	96	0.148	0.166	0.230	0.134	0.148	0.163	0.135	0.149	0.164

Tabla 5.2: Resultados de la experimentación, cuartiles del número de cambios de contexto / número de trabajos.

Número de CPUs	Número de tareas	RUN		AlECS		CAlECS	
		Mínimo	Máximo	Mínimo	Máximo	Mínimo	Máximo
2	8	0.133	3.688	0.286	0.907	0.081	0.907
	16	0.148	2.077	0.202	0.679	0.170	0.679
	24	0.134	1.242	0.192	0.563	0.105	0.540
	32	0.124	0.856	0.130	0.434	0.111	0.434
	40	0.104	0.725	0.129	0.343	0.091	0.318
	48	0.081	0.518	0.098	0.266	0.082	0.247
4	16	0.510	2.462	0.357	1.115	0.284	1.115
	32	0.189	1.403	0.259	0.723	0.162	0.685
	48	0.173	0.862	0.167	0.517	0.157	0.405
	64	0.135	0.739	0.150	0.415	0.090	0.331
	80	0.120	0.565	0.109	0.285	0.099	0.279
	96	0.107	0.409	0.091	0.228	0.098	0.211

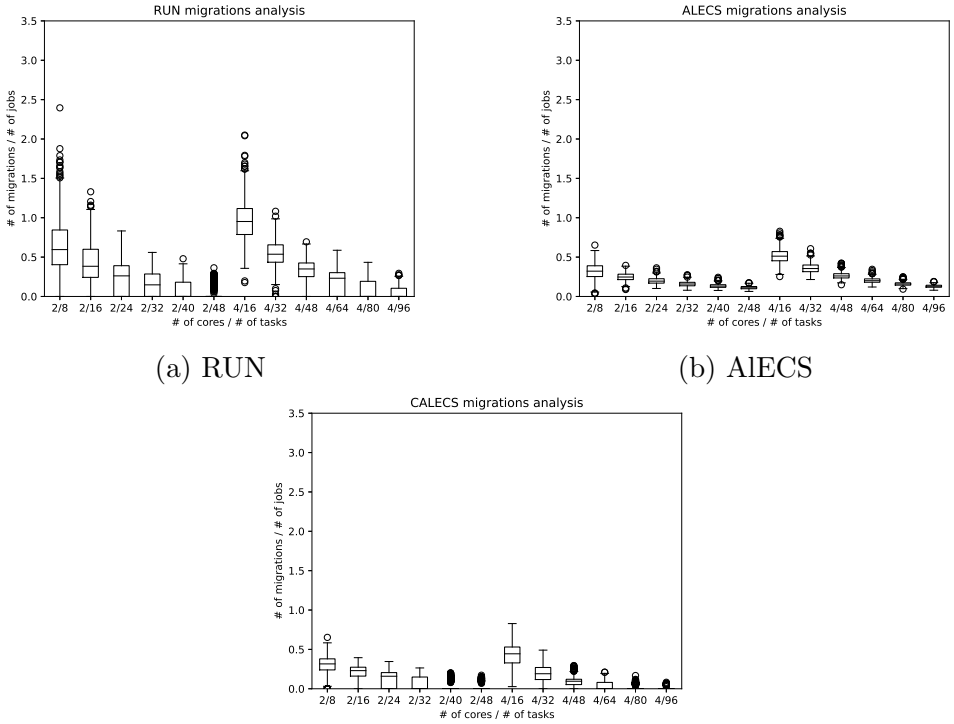
Tabla 5.3: Resultados de la experimentación, mínimo y máximo del número de cambios de contexto / número de trabajos.

tareas, en donde es muy ligeramente menor en AlECS, por una milésima. En todos los casos, CAlECS presenta una desviación típica ligeramente superior a AlECS.

Se observa también que la media de cambios de contexto, desviación típica y sus cuartiles, decrecen con los 3 planificadores según aumenta el tasa TPP o el número de núcleos de procesamiento.

5.3. Análisis de migraciones

Los boxplot que conforman la figura 5.2, muestran el número de migraciones medio por experimento (eje Y) conforme varían el número de tareas y núcleos de procesamiento (eje X) en cada uno de los planificadores. El desglose de la media y la



(a) RUN

(b) AlECS

(c) CAlECS

Figura 5.2: Resultados de la experimentación, número de migraciones / número de trabajos.

Número de CPUs	Número de tareas	RUN		AlECS		CAlECS	
		Media	Desviación típica	Media	Desviación típica	Media	Desviación típica
2	8	0.642	0.358	0.320	0.101	0.304	0.118
	16	0.409	0.273	0.249	0.055	0.195	0.114
	24	0.251	0.196	0.198	0.041	0.119	0.102
	32	0.146	0.156	0.159	0.031	0.066	0.083
	40	0.084	0.118	0.134	0.026	0.030	0.057
	48	0.041	0.082	0.112	0.021	0.015	0.039
4	16	0.964	0.266	0.515	0.094	0.429	0.143
	32	0.542	0.173	0.360	0.063	0.196	0.105
	48	0.329	0.149	0.265	0.043	0.093	0.067
	64	0.196	0.143	0.204	0.034	0.041	0.049
	80	0.095	0.111	0.159	0.025	0.015	0.029
	96	0.046	0.079	0.130	0.019	0.005	0.016

Tabla 5.4: Resultados de la experimentación, media y desviación típica del número de migraciones / número de trabajos.

desviación típica de los mismos datos se encuentra en la tabla 5.4. La tabla 5.5 muestra los cuartiles y la tabla 5.6 los valores mínimo y máximo.

En los tres planificadores, tanto la media como la desviación típica descienden con el número de núcleos de procesamiento o conforme el TPP se incrementa.

AlECS obtiene en media menos migraciones que RUN con TPP 4, 8 y 12. En cambio,

Número de CPUs	Número de tareas	RUN			AlECS			CAlECS		
		Q1	Q2	Q3	Q1	Q2	Q3	Q1	Q2	Q3
2	8	0.402	0.595	0.845	0.253	0.321	0.390	0.238	0.316	0.380
	16	0.242	0.383	0.600	0.214	0.247	0.283	0.159	0.230	0.274
	24	0.000	0.262	0.391	0.169	0.195	0.225	0.000	0.159	0.206
	32	0.000	0.148	0.286	0.136	0.158	0.180	0.000	0.000	0.149
	40	0.000	0.000	0.183	0.116	0.132	0.150	0.000	0.000	0.000
	48	0.000	0.000	0.000	0.098	0.110	0.126	0.000	0.000	0.000
4	16	0.786	0.952	1.117	0.453	0.512	0.570	0.328	0.444	0.531
	32	0.435	0.537	0.656	0.315	0.355	0.400	0.117	0.191	0.270
	48	0.251	0.349	0.426	0.235	0.261	0.289	0.052	0.095	0.121
	64	0.000	0.232	0.303	0.180	0.200	0.223	0.000	0.000	0.081
	80	0.000	0.000	0.192	0.144	0.158	0.174	0.000	0.000	0.000
	96	0.000	0.000	0.104	0.117	0.130	0.143	0.000	0.000	0.000

Tabla 5.5: Resultados de la experimentación, cuartiles del número de migraciones / número de trabajos.

Número de CPUs	Número de tareas	RUN		AlECS		CAlECS	
		Mínimo	Máximo	Mínimo	Máximo	Mínimo	Máximo
2	8	0.000	2.396	0.031	0.653	0.000	0.653
	16	0.000	1.330	0.089	0.395	0.000	0.395
	24	0.000	0.832	0.101	0.364	0.000	0.346
	32	0.000	0.559	0.079	0.275	0.000	0.264
	40	0.000	0.479	0.076	0.243	0.000	0.203
	48	0.000	0.364	0.064	0.174	0.000	0.172
4	16	0.176	2.050	0.253	0.828	0.027	0.828
	32	0.000	1.082	0.215	0.607	0.000	0.491
	48	0.000	0.694	0.149	0.424	0.000	0.295
	64	0.000	0.587	0.120	0.345	0.000	0.211
	80	0.000	0.433	0.093	0.250	0.000	0.168
	96	0.000	0.294	0.079	0.190	0.000	0.083

Tabla 5.6: Resultados de la experimentación, mínimo y máximo del número de migraciones / número de trabajos.

con TPP 16, 20 y 24 sucede lo contrario. La desviación típica es siempre menor en AlECS. En en casi todas las configuraciones RUN consigue particionar el conjunto de tareas de al menos un experimento, mientras que AlECS realiza migraciones en todos los experimentos.

Según estos resultados, CAlECS se comporta en general mejor que RUN y AlECS en todas las configuraciones, mostrando cuartiles siempre inferiores o iguales a sus dos competidores.

El motivo de estos resultados es atribuible a la capacidad de CAlECS para identificar clusters, como se puede comprobar en la tabla 5.7. En esta tabla se compara la capacidad para realizar clusters de RUN con la de CAlECS. En la primera columna, se muestra el número de núcleos de procesamiento, en la segunda el número de tareas,

Número de CPUs	Número de tareas	Distribución de las tareas en clusters	CAlECS	RUN
2	8	1,1 2	22 478	23 477
	16	1,1 2	110 390	84 416
	24	1,1 2	201 299	144 356
	32	1,1 2	297 203	241 259
	40	1,1 2	387 113	316 184
	48	1,1 2	435 65	388 112
4	16	1, 1, 1, 1 1, 1, 2 2, 2 1, 3 4	0 22 30 151 297	0 2 0 31 467
	32	1, 1, 1, 1 1, 1, 2 2, 2 1, 3 4	26 199 42 171 62	4 20 0 84 392
	48	1, 1, 1, 1 1, 1, 2 2, 2 1, 3 4	113 300 17 62 8	44 45 0 109 302
	64	1, 1, 1, 1 1, 1, 2 2, 2 1, 3 4	269 206 4 19 2	138 48 0 102 212
	80	1, 1, 1, 1 1, 1, 2 2, 2 1, 3 4	394 103 1 1 1	264 39 0 90 107
	96	1, 1, 1, 1 1, 1, 2 2, 2 1, 3 4	453 47 0 0 0	359 27 0 61 53

Tabla 5.7: Rendimiento en el clusterizado

en la tercera la configuración de cluster estudiada (ej. (1, 1) implica dos clusters de un núcleo de procesamiento cada uno), en la cuarta columna se muestra el número de

veces que CAlECS ha obtenido la configuración de cluster estudiada, y en la quinta se muestra el número de veces que lo ha conseguido RUN.

RUN no está específicamente diseñado para encontrar clusters, aunque su habilidad para encontrarlos, sobre todo en los casos en los que el TPP es alto, le permiten obtener cero migraciones en algunos experimentos. Así, RUN evita completamente las migraciones en más del 25 % de los experimentos realizados sobre dos núcleos de procesamiento con 24 y 32 tareas, así como con 64 tareas sobre cuatro núcleos de procesamiento. RUN también consigue más de un 50 % de los experimentos sin migraciones en el caso de dos núcleos de procesamiento y 40 tareas, y en los caso de cuatro núcleos de procesamiento y 80 o 96 tareas. RUN llega a reducir a cero las migraciones en más del 75 % de los experimentos configurados con dos núcleos de procesamiento y 48 tareas.

A diferencia de RUN, hemos diseñado CAlECS específicamente para generar clusters del menor tamaño posible, mediante una etapa específica para este fin Sec. 4.3. Según los resultados anteriores este diseño ha sido eficaz, ya que CAlECS consigue más particionados que RUN. Así, CAlECS no da lugar a ninguna migración en más del 25 % de los experimentos con 24 tareas en dos CPUs, en más del 50 % de los experimentos con TPP 16, y en más del 75 % de los experimentos con TPP 40 y 44 tampoco genera migraciones.

Capítulo 6

Optimización del ejecutivo cíclico

6.1. Motivación y fundamentos de la propuesta

La planificación de un conjunto de tareas TR parte de la determinación previa de su WCET. Es práctica común, especialmente en propuestas de investigación, incluir en el WCET los costes de cambios de contexto y migración, o incluso una cota pesimista de tiempos de bloqueo por acceso a recursos compartidos. Esto tiene dos consecuencias. Por una parte, sobre estima el WCET de forma excesivamente pesimista [49], llevando a un sobre-dimensionamiento (*overprovisioning*) innecesario del sistema. Por otra, hace que el WCET dependa del propio algoritmo de planificación. Las técnicas más usadas para el análisis del WCET son el análisis estático y la medida de la ejecución [50], y ninguno de los dos métodos está pensado para tener en cuenta estos sobre costes.

El método presentado en el Cap. 4 para el cálculo de un ejecutivo cíclico que maximiza la ocupación de los núcleos de procesamiento asegurando el cumplimiento de restricciones, puede utilizarse sobre el WCET estrictamente calculado por las herramientas al uso. En ese caso, el ejecutivo resultante no tendría en cuenta los costes de cambio de contexto y migración. Sin embargo, por su propia naturaleza el ejecutivo determina exactamente cuántos cambios de contexto y migraciones se van a producir. A partir de una estimación de esos costes, sería posible añadirlos a los WCET de las tareas y reajustar el cálculo del ejecutivo. A su vez, este nuevo cálculo variará la sobre carga, y requerirá de un nuevo ciclo de cálculo. Las pruebas preliminares que hemos realizado muestran que la convergencia se puede alcanzar en pocas iteraciones. La realización completa de semejante sistema de ajuste queda fuera de los objetivos iniciales de este TFM, pero se han realizado estimaciones experimentales para un sistema concreto, y se deja planteado el método a seguir. El flujo de trabajo está planteado para ser usado con CAlECS, debido a su bajo número de cambios de contexto y migraciones, pero es aplicable a cualquier algoritmo de planificación.

La siguiente sección presenta la determinación experimental de costes que se ha

realizado. Los valores obtenidos pueden ser usados como referencias para el estudio de la optimización del ejecutivo cíclico. La Sec. 6.3 detalla la propuesta de flujo de trabajo para optimizar el ejecutivo cíclico a partir de los mismos.

6.2. Determinación experimental del coste de cambio de contexto y migración en Linux RT

6.2.1. Metodología y entorno experimental

Para la estimación experimental de costes asociados a los cambios de contexto y migraciones sobre un sistema real se ha recurrido a una placa de desarrollo Raspberry Pi 3B cuyas características se recogen en el Anexo E.

Se ha utilizado una distribución Linux Debian 10 para ARM 64 bits (AArch64) con kernel 4.19 y el parche `PREEMPT_RT` aplicado. La elección de placa y sistema se debe al fácil acceso a este material, las características apropiadas del microprocesador (cuatro núcleos), y al buen conocimiento previo tanto del *hardware* como del kernel Linux y el parche `PREEMPT_RT`.

Se han diseñado tres programas [51] que miden respectivamente el tiempo de cambio de contexto, el de migración y el de rellenado (*refill*) del segundo nivel de memoria cache (L2). Los detalles metodológicos se describen en secciones posteriores. Los tres programas están optimizados para la plataforma elegida. Se ha evitado el uso de herramientas ya existentes como `rt-test` [52] o `ftrace` [53] debido a que introducen sobrecargas que distorsionan las medidas.

6.2.2. Coste de cambio de contexto

El programa desarrollado para la medición bloquea en primer lugar los contenidos en memoria física, a fin de impedir fallos de página. En segundo lugar, crea dos hilos. Se fuerza a que ambos hilos sean asignados al mismo núcleo de procesamiento controlando la afinidad del hilo desde el núcleo de Linux. Así mismo, se les asigna a ambos una prioridad TR y el planificador FIFO (First in first out). Cualquier prioridad TR es superior a la del resto de hilos o tareas del sistema que no sean TR. En cuanto al planificador FIFO, este asegura que una tarea se ejecuta en exclusividad en el CPU sin intervenciones del núcleo Linux mientras no exista una tarea de mayor prioridad, o mientras no se realice un cambio de contexto voluntario y exista una tarea de la misma prioridad esperando para ser ejecutada. Por otro lado, el núcleo se configura en modo *(CPU Isolation)* (aislamiento del CPU) para impedir que tanto el resto de procesos como las interrupciones sean asignadas al CPU al que se han asignado los dos hilos

anteriores.

Con esta configuración se evitan cambios de contexto involuntarios (expulsiones). Así, desde el momento que alguno de los dos hilos creados toma el control del núcleo de procesamiento, permanece en exclusividad en el mismo hasta que invoca voluntariamente al planificador, realiza una llamada al sistema bloqueante o finaliza su ejecución. Además, mientras uno de estos dos hilos se ejecuta el otro hilo estará en la cola de espera para la ejecución. De este modo, si uno de los dos hilos realiza un cambio de contexto voluntario, el control pasa necesariamente al otro hilo.

Para realizar la medición se bloquean ambos hilos con una barrera (llamada al sistema `pthread_barrier_wait()`), se realiza una lectura de tiempo en cada hilo (llamada al sistema `clock_gettime()`), y se invoca el planificador para realizar un cambio de contexto voluntario (llamada al sistema `sched_yield`). Así se consigue tomar primero una lectura de tiempo en un hilo, realizar un cambio de contexto, y tomar a continuación otra lectura de tiempo en el otro hilo sin intervención de ninguna otra actividad. La diferencia entre ambas medidas es el tiempo de cambio de contexto que buscamos. El comportamiento conseguido se muestra en la Fig. 6.1 modelado como una Red de Petri. Cada experimiento de medida se lanza cien veces, resultando en unos valores mínimo, medio y máximo de 7760 ns, 8034 ns y 11250 ns respectivamente.

6.2.3. Coste de migración

La herramienta desarrollada para medir el coste de la migración hace uso de un único proceso, cuya migración entre dos núcleos se fuerza mediante el control de afinidad.

Esta herramienta comienza bloqueando en primer lugar la memoria usada para impedir fallos de página, y estableciendo prioridad TR y tipo de planificador FIFO al proceso. Además, el núcleo de Linux se configura para aislar los dos núcleos de procesamiento entre los que se forzará la migración del proceso. Para realizar la medición, primero se establece la afinidad del proceso a uno de los núcleos de procesamiento aislados (llamada al sistema `sched_setaffinity()`). Tras esto, se toma el tiempo del sistema, se establece la afinidad del proceso en el otro núcleo de procesamiento aislado, y se vuelve a tomar el tiempo del sistema. Al establecer por segunda vez la afinidad, el núcleo de Linux migra el proceso de un núcleo de procesamiento a otro. Al ser el proceso con más prioridad, reanuda inmediatamente su ejecución. La diferencia de tiempos medida de esta forma es el coste directo de la migración.

El experimento se repite cien veces, obteniendo unos costes de migración mínimo, medio y máximo de 31875 ns, 33058 ns y 56042 ns respectivamente.

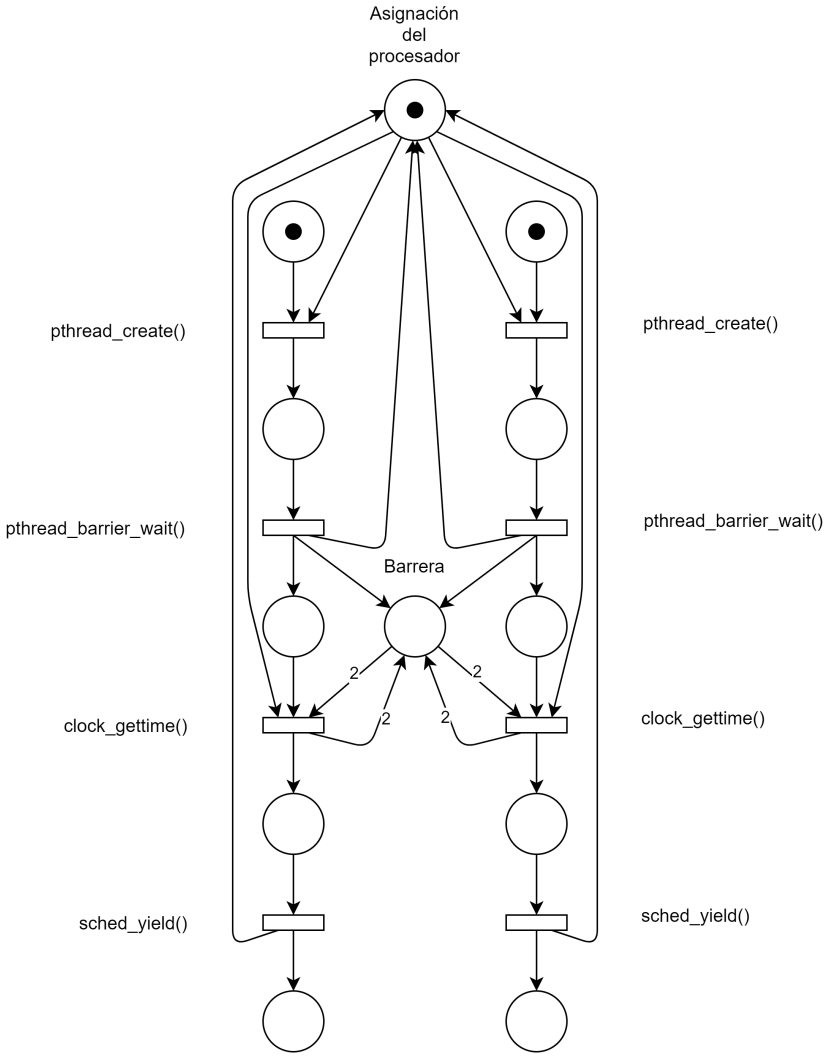


Figura 6.1: Comportamiento del programa de medición de coste de cambios de contexto modelado mediante una Red de Petri

6.2.4. Coste de rellenado de la cache de último nivel del microprocesador

Una migración real puede tener más costes indirectos que dependen del comportamiento de cada programa sobre una arquitectura o microarquitectura específica (fallos de cache, fallos de predicción). En un sistema TR estricto se intentan evitar estos costes desactivando cuando es posible todos los mecanismos especulativos o de ocultación de latencia. Sin embargo, es común utilizar microprocesadores con memoria cache debido a la dificultad de contar con microprocesadores específicos para TR. Por este motivo, hemos medido el coste de rellenado de la cache de último nivel en caso de que esta no sea compartida. Este valor estima el sobre-coste en el que se puede incurrir en caso de que una migración se realice entre dos núcleos que no comparten dicho nivel, o en el caso de una tarea con un *working set* secundario mayor que la L2,

que se ejecute entre la expulsión y la reanudación de otra tarea.

La herramienta desarrollada para este experimento mide el coste de llenado de la mitad de la cache de último nivel del microprocesador (L2 en el caso de la Raspberry Pi 3B) desde memoria principal. El procedimiento usado consiste en leer un espacio de memoria no cargado en cache L2, y después leer este mismo espacio de memoria ya cargado. La diferencia entre los tiempos representa el coste de transferir datos desde memoria principal hasta la memoria de último nivel. La decisión de rellenar solamente la mitad de la cache (256KB de los 512KB totales), permite dejar espacio en la cache para almacenar el resto de datos usados por el programa y el sistema operativo durante esta medición. En caso de usar la cache completamente, se pueden producir fallos en cache al querer acceder a elementos como la pila al realizar una llamada a una función.

El experimento se ha repetido cien veces obteniendo un coste mínimo, medio y máximo de 60209 ns, 81398 ns y 199999 ns respectivamente.

6.3. Propuesta de optimización del ejecutivo cíclico TRD

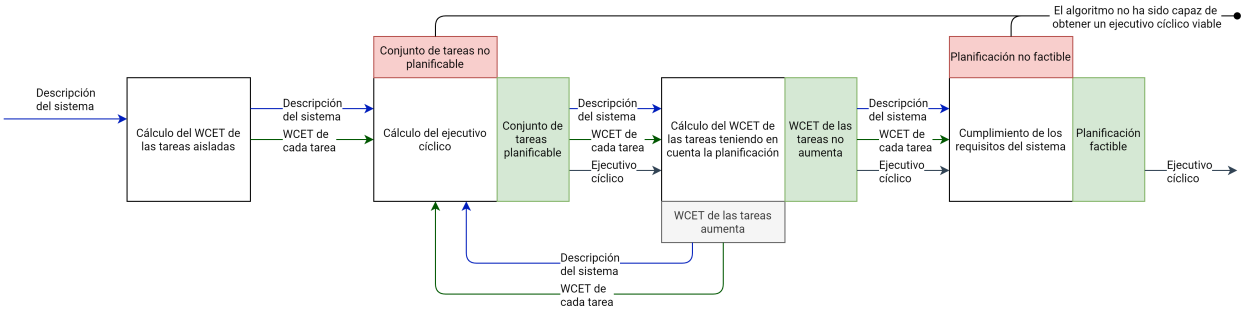


Figura 6.2: Flujo de trabajo para la optimización del ejecutivo cíclico TRD

En esta sección se presenta un flujo de trabajo que permite contabilizar sobrecostes en la planificación de conjuntos de tareas periódicas TRD.

El flujo de trabajo propuesto consta de cuatro componentes: *Cálculo del WCET de las tareas TRD aisladas*, *Cálculo del ejecutivo cíclico*, *Cálculo del WCET de las tareas teniendo en cuenta la planificación* y *Cumplimiento de los requisitos del sistema*. Estos componentes se retro-alimentan y ajustan de forma iterativa hasta que convergen en una planificación válida.

El proceso iterativo comienza con el cálculo del WCET de las tareas suponiendo que se ejecutan de forma aislada, todos los recursos de cómputo están a disposición de la tarea. Utilizando estos WCET se planifica el conjunto de tareas mediante CAlECS, y

se recalcula los WCET de las tareas teniendo en cuenta esta planificación. Si el WCET de las tareas aumenta tras la planificación, se vuelven a ejecutar de forma iterativa hasta que converjan tanto la planificación como el recálculo.

La Fig. 6.2 muestra el flujo de trabajo.

En el Anexo. D se encuentra en ejemplo de aplicación del flujo de trabajo.

6.3.1. Cálculo del WCET de las tareas aisladas

Este componente se encarga de calcular el WCET de las tareas asumiendo que estas se ejecutan de forma aislada. Esto significa que se ejecutan de principio a fin, sin cambios de contexto debidos a planificación y sin bloqueos debidos a interferencias con otras tareas. El módulo cuenta con información tanto de la plataforma objetivo como del conjunto de tareas.

Esta etapa puede ser realizada mediante técnicas bien de análisis estático o bien de estimación por ejecución [50]. Las primeras se integrarían mejor en un sistema de cálculo y optimización *off-line* como el que proponemos, al no requerir la ejecución preliminar para estimar el WCET.

6.3.2. Cálculo del ejecutivo cíclico

Este componente se encarga de calcular un ejecutivo cíclico a partir de los WCETs calculados en el componente anterior si se trata de la primera iteración, o bien utilizando los WCETs de iteraciones anteriores.

Para este cálculo se puede usar cualquier algoritmo de planificación, aunque el uso de propuestas como CAlECS donde los sobrecostes de cambio de contexto y migraciones son bajos facilitan que el algoritmo converja antes al aumentar poco el WCET entre iteraciones.

Herramientas como *Tertimuss* facilitan el cálculo de este ejecutivo cíclico.

Tras cada iteración, el WCET de las tareas cambia, y la utilización del sistema aumenta, por lo que podría suceder que el conjunto de tareas no fuese planificable para el planificador usado. Si estos sucede, el flujo terminará sin devolver un ejecutivo cíclico viable.

6.3.3. Cálculo del WCET de las tareas teniendo en cuenta la planificación

Este componente calcula los costes derivados de los cambios de contexto, migraciones y posibles interferencias entre trabajos que aparecen en el ejecutivo cíclico calculado en el paso anterior. Como en el primer componente de cálculo del WCET,

el análisis puede realizarse de nuevo bien con técnicas de análisis estático, bien con medición directa, aunque al tenerse que repetir en las múltiples iteraciones, encajaría mejor utilizar técnicas de análisis estático que pudiesen ser automatizadas.

Una vez recalculados los sobrecostes que se producen en cada instancia de la tarea (en el hiperperíodo), se toma el mayor de cada tarea, y en caso de ser superior al máximo contabilizado en iteraciones anteriores, se actualiza.

Tras cada iteración, el WCET de cada tarea es mayor o igual al de todas las iteraciones anteriores, ya que se contabilizan más sobrecostes, esto asegura la convergencia del algoritmo. En el peor caso se contabilizan todos los posibles sobrecostes en los que puede incurrir cada tarea.

Si tras el análisis el WCET de todos los trabajos se ha mantenido igual, implica que el algoritmo ha convergido, por lo que se debe avanzar a la siguiente etapa del algoritmo. En caso contrario, se debe realizar otra iteración utilizando este nuevo WCET calculado.

6.3.4. Cumplimiento de los requisitos del sistema

El último componente comprueba que el ejecutivo cíclico final cumple los requisitos del sistema. Este incumplimiento puede venir dado por diferentes motivos, como por ejemplo un intervalo entre dos cambios de contexto menor al coste de un cambio de contexto. Según el caso, el componente acepta la planificación o bien rechaza el ejecutivo cíclico obtenido.

En caso de rechazarse, se puede reintentar el proceso con un planificador diferente.

6.3.5. Optimizaciones sobre el flujo de trabajo: Contabilización de los sobrecostes por trabajo

Todos los planificadores que conocemos trabajan con el WCET de cada tarea, ninguno tiene en cuenta los sobrecostes individuales de cada trabajo para realizar la planificación. Conllevando que se contabilice en el WCET de cada tarea el máximo de los sobrecostes de sus trabajos.

Esto puede hacer inviables algunos conjuntos de tareas planificables como se demuestra en el siguiente ejemplo:

Asumiendo que contamos con las tareas periódicas TRD de la tabla 6.1, que se ejecutan sobre 2 procesadores, y que T_1 y T_2 cuentan con una región de exclusión mutua compartida de 1 segundo (no contabilizada en el WCET). El flujo de trabajo propuesto fallaría al encontraría una planificación factible, ya que al tenerse que ejecutar T_1 y T_2 en paralelo en cualquier planificación factible 6.3, ambas tendrían que ampliar su

Tarea	WCET (s)	Plazo (s)
T_1	4	10
T_2	4	5
T_3	6	10

Tabla 6.1: Conjunto de tareas TRD

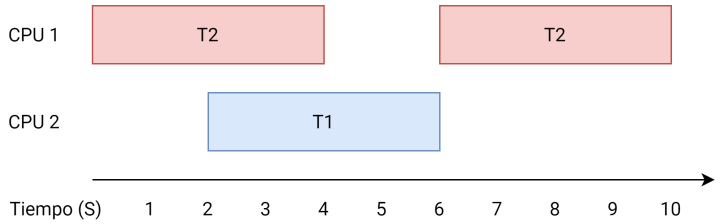


Figura 6.3: Ejecución paralela de las tareas T_1 y T_2 de la Tab. 6.1

WCET en 2 segundos, produciendo que la utilización del sistema sea mayor a 2.

En cambio, si solamente se aumenta el WCET de la instancia de T_2 que se ejecuta en paralelo con T_1, se podría encontrar planificaciones factibles como la que se muestra en la Fig. 6.4.

Adaptar el flujo de trabajo para tratar los sobrecostes por trabajo (durante la planificación) en lugar de por tarea es una tarea trivial, aunque requiere algoritmos de planificación que sean capaces de lidiar con esto también.

Con modificaciones triviales se puede conseguir este comportamiento en algoritmos como EDF global, aunque este no puede lidiar con algunos sistemas con utilización máxima.

AlECS también puede ser adaptado, para ello el problema de programación entero se ha de modificar para calcular el número de ciclos que el trabajo J_i ha de ejecutar en cada intervalo (inicialmente se calcula en base a la tarea T_i). Los intervalos se mantienen iguales, ya que los plazos no son modificados. El problema de programación entero resultante se encuentra en la eq. 6.1.

$$
\begin{aligned}
&\max \quad \sum_{i=1}^{n} \sum_{k=1}^{\alpha} x_i^k \\
&s.t \\
&\quad \forall k \sum_{i=1}^{n} x_i^k = s_j \times |I_{SD}^k| \times F^* && \text{Máxima utilización} \\
&\quad \forall i \sum_{j=\alpha}^{\beta} x_i^j = cc_i && \text{Ejecución} \\
&\qquad \text{donde } \alpha \text{ representa el intervalo de activación del trabajo i, mientras que } \beta \text{ el de su plazo} \\
&\quad \forall k \forall i \quad x_i^k \leq |I_{SD}^k| \times F^* && \text{Secuencialidad} \\
&\qquad\qquad x_i^k \geq 0
\end{aligned}
\tag{6.1}
$$

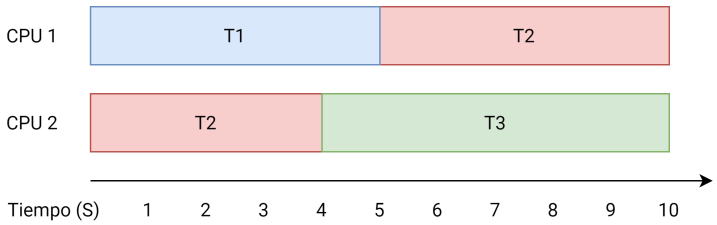


Figura 6.4: Planificación factible del conjunto de tareas TRD de la Tab. 6.1, tras tener en cuenta los sobrecostes por trabajo

Respecto a la ecuación original, se ha modificado las restricciones de ejecución y secuencialidad, mientras que se ha eliminado la de laxitud, ya que con el resto de restricciones se garantiza el cumplimiento de esta también. Con esta solución no se ha comprobado si la matriz resultante para resolver la relajación del PPE como un PPL es unimodular, por lo que no se puede garantizar que un PPL encuentre una solución entera de la misma.

Capítulo 7

Conclusiones y trabajo futuro

7.1. Recapitulación

En este trabajo se ha propuesto en nuevo algoritmo de planificación TRD sobre multiprocesadores que, basándose en la implementación de otro algoritmo previo llamado AlECS, ha conseguido obtener menos cambios de contextos y migraciones que RUN (algoritmo de referencia en este aspecto).

Se ha realizado una comparación experimental con otros dos planificadores, RUN y AlECS, y los resultados muestran que el algoritmo propuesto tiene mejor comportamiento en cuanto a las métricas mencionadas.

El esquema propuesto genera un ejecutivo cíclico fuera de línea usado como base para desarrollar un flujo de trabajo para la implementación de planificadores en una plataforma real, teniendo en cuenta restricciones no contempladas en el análisis de planificabilidad teórico, y acercando la solución a su uso de la industria.

Se ha mejorado el rendimiento de la plataforma experimental usada, *Tertimuss*, y se han incrementado sus características y mejorado su usabilidad.

El autor ha logrado introducirse satisfactoriamente en el ámbito de la investigación, colaborando en un equipo de investigación multidisciplinar y contribuyendo a la publicación de dos trabajos científicos en una conferencia y en una revista internacional con revisión por pares [7], [8].

7.2. Conclusiones

El algoritmo CAlECS establece una nueva referencia en cuanto a reducción de cambios de contexto y migraciones en sistemas con utilización máxima, minimizando los sobrecostes producidos por el planificador y evitando el sobreaprovisionamiento al aprovechar mejor los recursos de cálculo.

La combinación en el equipo de trabajo de metodologías formales y experimentales

ha permitido abordar un problema especialmente complejo dado el gran número de elementos que influyen en la ejecución de las tareas. En particular, se demuestra que la aplicación del método científico en este tipo de problemas puede llevar a encontrar y proponer soluciones sencillas a problemas complejos en base a diagnósticos fundamentados, mejorando algoritmos previos.

7.3. Trabajo futuro

El trabajo realizado tiene diversas vías de continuidad, entre ellas:

- Caracterizar el cambio de contexto y migración en varias plataformas, permitiendo comparar con más precisión los sobrecostes producidos por un planificador
- Llevar a cabo como prueba de concepto del flujo de trabajo que se deja planteado en la Sec. 6.3 para la optimización del ejecutivo cíclico TRD sobre un sistema real
- Adaptar el algoritmo CAlECS teniendo en cuenta los sobrecostes de los trabajos.
- Implementar CAlECS sobre un sistema operativo TR
- Probar las capacidades de CAlECS sobre conjuntos de tareas utilizados por la industria
- Ampliar Tertimuss con nuevos algoritmos de planificación, generación de tareas, análisis de planificaciones y generación de visualizaciones

Bibliografía

[1] G. Desirena-Lopez, C. R. Vázquez, A. Ramírez-Treviño y D. Gómez-Gutiérrez. «Thermal modelling for Temperature Control in MPSoC's Using Fluid Petri Nets». En: *IEEE Conference on Control Applications part of Multi-conference on Systems and Control*. 2014.

[2] G. Desirena-Lopez, J. L. Briz, C. R. Vázquez, A. Ramírez-Treviño y D. Gómez-Gutiérrez. «On-line Scheduling in Multiprocessor Systems based on continuous control using Timed Continuous Petri Nets». En: *13th International Workshop on Discrete Event Systems*. 2016.

[3] L. Rubio-Anguiano, G. Desirena-López, A. Ramírez-Treviño y J.L. Briz. «Energy-Efficient Thermal-Aware Scheduling for RT Tasks Using TCPN». En: *IFAC-PapersOnLine* 51.7 (2018). 14th IFAC Workshop on Discrete Event Systems WODES 2018, págs. 236 -242. DOI: https://doi.org/10.1016/j.ifacol.2018.06.307.

[4] G. Desirena-López, A. Ramírez-Treviño, J. L. Briz, C. R. Vázquez y D. Gómez-Gutiérrez. «Thermal-aware Real-time Scheduling Using Timed Continuous Petri Nets». En: *ACM Trans. Embed. Comput. Syst.* 18.4 (jul. de 2019), 36:1-36:24. ISSN: 1539-9087. DOI: 10.1145/3322643. URL: http://doi.acm.org/10.1145/3322643.

[5] L. Rubio-Anguiano, G. Desirena-López, A. Ramírez-Treviño y J. L. Briz. «Energy-efficient thermal-aware multiprocessor scheduling for real-time tasks using TCPN». En: *Discrete Event Dynamic Systems* (2019). ISSN: 1573-7594. DOI: 10.1007/s10626-019-00285-x. URL: https://doi.org/10.1007/s10626-019-00285-x.

[6] Gaddiel Desirena-López. «Thermal-Aware Hard Real Time Task Scheduling in MPSoC's using Timed Continuous Petri Nets». Tesis doct. CINVESTAV - IPN Unidad Guadalajara, 2019.

[7] L. Rubio-Anguiano, G. Desirena-López, A. Ramírez-Treviño, J.L. Briz y A. Chils. «Real time scheduler for multiprocessor systems based on continuous control using Timed Continuous Petri Nets». En: *IFAC-PapersOnLine* 53.4 (2020). 15th IFAC Workshop on Discrete Event Systems WODES 2020 11-13 Nov. Rio de Janeiro, Brasil- To appear, págs. 371-377. URL: https://www.sciencedirect.com/journal/ifac-papersonline/vol/53/issue/4.

[8] Laura Elena Rubio-Anguiano, Abel Chils Trabanco, José Luis Briz Velasco y Antonio Ramírez-Treviño. «Maximizing Utilization and Minimizing Migration in Thermal-Aware Energy-Efficient Real-Time Multiprocessor Scheduling». En: *IEEE Access* 9 (2021), págs. 83309-83328. DOI: 10.1109/ACCESS.2021.3086698.

[9] G. Desirena, L. Rubio, A. Ramirez y J.L. Briz. *Tertimuss: A Simulation Environment for Thermal-aware Real Time Multiprocessor Scheduling*. 2019. URL: https://webdiis.unizar.es/gaz/repositories/tertimuss (visitado 09-06-2021).

[10] *Repositorio git de Tertimuss*. URL: https://github.com/AbelChT/Tertimuss/tree/TFM (visitado 02-06-2021).

[11] *Overleaf, the Online LaTeX Editor*. URL: https://www.overleaf.com (visitado 21-08-2019).

[12] *PyCharm: the Python IDE for Professional Developers by JetBrains*. URL: https://www.jetbrains.com/pycharm/ (visitado 20-08-2019).

[13] *Visual Studio Code - Code Editing. Redefined*. URL: https://code.visualstudio.com/ (visitado 20-08-2019).

[14] *PyCharm: Source Code*. URL: https://github.com/JetBrains/intellij-community/tree/master/python (visitado 20-08-2019).

[15] *Visual Studio Code: Source Code*. URL: https://github.com/microsoft/vscode (visitado 20-08-2019).

[16] Laurent Perron y Vincent Furnon. *OR-Tools*. Ver. 8.1. Google, 2020. URL: https://developers.google.com/optimization/.

[17] J. D. Hunter. «Matplotlib: A 2D graphics environment». En: *Computing in Science & Engineering* 9.3 (2007), págs. 90-95. DOI: 10.1109/MCSE.2007.55.

[18] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt y SciPy 1.0 Contributors. «SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python». En: *Nature Methods* 17 (2020), págs. 261-272. DOI: 10.1038/s41592-019-0686-2.

[19] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke y Travis E. Oliphant. «Array programming with NumPy». En: *Nature* 585 (2020), 357–362. DOI: 10.1038/s41586-020-2649-2.

[20] *Git*. URL: https://git-scm.com (visitado 20-08-2019).

[21] *GitHub*. URL: https://github.com/ (visitado 20-08-2019).

[22] Abel Chils Trabanco y José Luis Briz Velasco. «Entorno de simulación de planificadores tiempo real para multiprocesadores con restricciones térmicas y de energía». En: (2019).

[23] IEEE TCRTS. *IEEE Technical Commitee on Real-Time Systems: Terminology and Notation*. http://sites.ieee.org/tcrts/education/terminology-and-notation/. 2019. (Visitado 20-08-2019).

[24] S. Baruah, M. Bertogna y G. Butazzo. *Multiprocessor Scheduling for Real-Time Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2015. ISBN: 978-3-319-08695-8.

[25] Dong-Ik Oh y T. P. Bakker. «Utilization Bounds for N-Processor Rate Monotone Scheduling with Static Processor Assignments». En: *Real-Time Systems* 15.2 (1998), págs. 183-192.

[26] A. Mascitti, T. Cucinotta y L. Abeni. «Heuristic partitioning of real-time tasks on multi-processors». En: *2020 IEEE 23rd International Symposium on Real-Time Distributed Computing (ISORC)*. 2020, págs. 36-42. DOI: 10.1109/ISORC49007.2020.00015.

[27] Xuan Qi, Dakai Zhu y H. Aydin. «Cluster scheduling for real-time systems: utilization bounds and run-time overhead». En: *Real-Time Systems* 47 (2011), págs. 253-284.

[28] Shelby Funk, Greg Levin, Caitlin Sadowski, Ian Pye y Scott Brandt. «DP-Fair: a unifying theory for optimal hard real-time multiprocessor scheduling». En: *Real-Time Systems* 47.5 (2011), págs. 389-429. DOI: 10.1007/s11241-011-9130-0. URL: http://dx.doi.org/10.1007/s11241-011-9130-0.

[29] Nathan Fisher, Joël Goossens y Sanjoy Baruah. «Optimal Online Multiprocessor Scheduling of Sporadic Real-Time Tasks is Impossible». En: *Real-Time Syst.* 45.1-2 (jun. de 2010), págs. 26-71. ISSN: 0922-6443.

[30] Sanjoy K Baruah, Neil K Cohen, C Greg Plaxton y Donald A Varvel. «Proportionate progress: A notion of fairness in resource allocation». En: *Algorithmica* 15.6 (1996), págs. 600-625.

[31] Sanjoy K Baruah, Johannes E Gehrke y C Greg Plaxton. «Fast scheduling of periodic tasks on multiple resources». En: *IPPS 95*. IEEE. 1995, pág. 280.

[32] James H Anderson y Anand Srinivasan. «Mixed Pfair/ERfair scheduling of asynchronous periodic tasks». En: *Real-Time Systems, 13th Euromicro Conference on, 2001.* IEEE. 2001, págs. 76-85.

[33] Mauro Leoncini, Manuela Montangero y Paolo Valente. «A parallel branch-and-bound algorithm to compute a tighter tardiness bound for preemptive global EDF». En: *Real-Time Systems* 55.2 (2019), págs. 349-386. ISSN: 1573-1383.

[34] G. Nelissen, V. Berten, J. Goossens y D. Milojevic. «Reducing Preemptions and Migrations in Real-Time Multiprocessor Scheduling Algorithms by Releasing the Fairness». En: 1 (2011), págs. 15-24. ISSN: 2325-1271.

[35] Paul Regnier, George Lima, Ernesto Massa, Greg Levin y Scott Brandt. «RUN: Optimal Multiprocessor Real-Time Scheduling via Reduction to Uniprocessor». En: *Proceedings of the 2011 IEEE 32Nd Real-Time Systems Symposium*. RTSS '11. Washington, DC, USA: IEEE Computer Society, 2011, págs. 104-115. ISBN: 978-0-7695-4591-2.

[36] Paul Regnier, George Lima, Ernesto Massa, Greg Levin y Scott Brandt. «Multiprocessor scheduling by reduction to uniprocessor: an original optimal approach». En: *Real-Time Systems* 49.4 (2013), págs. 436-474.

[37] D. Compagnin, E. Mezzetti y T. Vardanega. «Putting RUN into Practice: Implementation and Evaluation». En: *2014 26th Euromicro Conference on Real-Time Systems*. 2014, págs. 75-84.

[38] Ernesto Massa, George Lima, Paul Regnier, Greg Levin y Scott Brandt. «Quasi-partitioned Scheduling: Optimality and Adaptation in Multiprocessor Real-time Systems». En: *Real-Time Syst.* 52.5 (sep. de 2016), págs. 566-597. ISSN: 0922-6443.

[39] E. Massa, G. Lima y P. Regnier. «Revealing the Secrets of RUN and QPS: New Trends for Optimal Real-Time Multiprocessor Scheduling». En: *2014 Brazilian Symposium on Computing Systems Engineering*. 2014, págs. 150-155.

[40] Robert I. Davis y Alan Burns. «A Survey of Hard Real-time Scheduling for Multiprocessor Systems». En: *ACM Comput. Surv.* 43.4 (oct. de 2011), 35:1-35:44. ISSN: 0360-0300. DOI: 10.1145/1978802.1978814. URL: http://doi.acm.org/10.1145/1978802.1978814.

[41] AUTOSAR. *Specification of RTE Software*. AUTOSAR CP Release 4.3.1. http://www.autosar.org. ESA Special Publication, 2017.

[42] ARINC. *Specification 651: Design Guide for Integrated Modular Avionics*. 1997.

[43] J. Mayank y A. Mondal. «Non-preemptive multiprocessor scheduling for periodic real-time tasks». En: *2017 7th International Symposium on Embedded Computing and System Design (ISED)*. 2017, págs. 1-6. DOI: 10.1109/ISED.2017.8303931.

[44] Maxime Chéramy, Pierre-Emmanuel Hladik y Anne-Marie Déplanche. «SimSo: A Simulation Tool to Evaluate Real-Time Multiprocessor Scheduling Algorithms». En: *Proc. of the 5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*. WATERS. 2014.

[45] Rehan Ahmed, Parameswaran Ramanathan y Kewal K Saluja. «Necessary and Sufficient Conditions for Thermal Schedulability of Periodic Real-Time Tasks Under Fluid Scheduling Model». En: *ACM Transactions on Embedded Computing Systems (TECS)* 15.3 (2016), pág. 49.

[46] David Johnson. «Fast Algorithms for Bin Packing». En: *Journal of Computer and System Sciences* 8 (jun. de 1974), págs. 272-314.

[47] M. L. Dertouzos. «Control Robotics: The Procedural Control of Physical Processes». En: *Proceedings of IFIP Congress (IFIP'74))*. 1974, págs. 807-813.

[48] R. I. Davis y A. Burns. «Priority Assignment for Global Fixed Priority Pre-Emptive Scheduling in Multiprocessor Real-Time Systems». En: *2009 30th IEEE Real-Time Systems Symposium*. 2009, págs. 398-409.

[49] Leonidas Kosmidis, Jaume Abella, Eduardo Quiñones y Francisco Cazorla. «Multi-Level Unified Caches for Probabilistically Time Analysable Real-Time Systems». En: dic. de 2013, págs. 360-371. DOI: 10.1109/RTSS.2013.43.

[50] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat y Per Stenström. «The Worst-Case Execution-Time Problem—Overview of Methods and Survey of Tools». En: *ACM Trans. Embed. Comput. Syst.* 7.3 (mayo de 2008). ISSN: 1539-9087. DOI: 10.1145/1347375.1347389. URL: https://doi.org/10.1145/1347375.1347389.

[51] *Medición de cambio de contexto, migración y rellenado de L2 en LinuxRT*. URL: https://github.com/AbelChT/TestRTPreemptionMigrationCost/tree/TFM (visitado 02-06-2021).

[52] The Linux Fundation. *RT-Tests test suite*. URL: https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/rt-tests (visitado 26-05-2021).

[53] The Linux Fundation. *Linux Kernel ftrace documentation*. URL: https://www.kernel.org/doc/Documentation/trace/ftrace.txt (visitado 26-05-2021).

[54] Gaddiel Desirena-Lopez, Laura Elena Rubio Anguiano, Antonio Ramírez-Treviño y José Luis Briz. «A Flexible Framework for Real-Time Thermal-Aware Schedulers using Timed Continuous Petri Nets». En: *Computación y Sistemas* 23.2 (2019), págs. 417-434. ISSN: 2007-9737. URL: http://www.cys.cic.ipn.mx/ojs/index.php/CyS/article/view/3204.

[55] Enrico Bini y Giorgio C. Buttazzo. «Measuring the Performance of Schedulability Tests». En: *Real-Time Syst.* 30.1-2 (mayo de 2005), págs. 129-154. ISSN: 0922-6443. DOI: 10.1007/s11241-005-0507-9. URL: http://dx.doi.org/10.1007/s11241-005-0507-9.

[56] *pip: package installer for Python*. URL: https://pip.pypa.io/en/stable/ (visitado 02-06-2021).

[57] *Poetry: Dependency Management for Python*. URL: https://python-poetry.org/ (visitado 02-06-2021).

[58] *PyPI: The Python Package Index*. URL: https://pypi.org/ (visitado 02-06-2021).

[59] ARM. *Arm Cortex-A53 MPCore Processor Technical Reference Manual*. URL: https://developer.arm.com/documentation/ddi0500/latest/ (visitado 26-05-2021).

[60] The Raspberry Pi Foundation. *BCM2837 Reference*. URL: https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2837/README.md (visitado 26-05-2021).

Lista de Figuras

Lista de Tablas

Anexos

Anexos A

Herramientas de simulación: Tertimuss

Este anexo describe el entorno de simulación de planificadores Tertimuss.

Tertimuss es un entorno de simulación de planificadores TR sobre multiprocesadores. Se ha desarrollado a partir de algoritmos y herramientas que venían siendo implementadas en Matlab [54] desde el inicio de la línea de investigación (Sec. 1.2) para cubrir las necesidades experimentales. Parte del trabajo realizado en [22] consistió en portar a Python los algoritmos de MatLab, añadir nuevos, estructurarlos de manera modular, y mejorar todos los componentes en términos de rendimiento, usabilidad, mantenibilidad y posibilidades. En esta forma se utilizó para la parte experimental de [7] y [8].

Durante el desarrollo de este último articulo se detectaron algunas características interesantes que podrían mejorar su uso, así como otras que no aportaban suficiente valor en su uso comparado con su coste de mantenimiento. Por este motivo, durante el desarrollo de este TFM Tertimuss se ha sometido a un rediseño casi total, añadiendo nuevas características y modificando el paradigma de simulación. Parte del esfuerzo se ha centrado en proporcionar distintos tipos de visualizaciones y mejores formas de interactuar con el entorno.

Durante la investigación culminada en [8], se hizo necesario incorporar al entorno herramientas automáticas de análisis, a fin de poder comparar planificadores sobre un conjunto grande de experimentos. Dado que en [8] no era necesario simular el componente térmico, los esfuerzos se han centrado en acelerar la ejecución de la simulación de la planificación solamente, reduciendo el tiempo inicial de horas a minutos.

A.1. Arquitectura

En su primera versión, Tertimuss fue diseñado para ser utilizado mediante ordenes en línea de comando, o mediante una interfaz gráfica. Estos métodos en la práctica resultaban poco útiles. En la práctica, Tertimuss se venía utilizando como un conjunto de bibliotecas dentro de scripts de Python en los que se definía la simulación a realizar, por lo que se ha re-estructurado como tal. Esta nueva orientación no excluye la posibilidad futura de añadir interfaces al entorno de simulación.

Organizado como un conjunto de bibliotecas, Tertimuss permite ahora definir las simulaciones más rápidamente. También se simplifica el desarrollo del entorno, ya que tanto la interfaz gráfica como la interfaz por línea de comandos requieren mantenimiento cada vez que se añaden nuevas características. Como inconveniente, requiere usuarios familiarizados con la sintaxis de Python.

A.1.1. Componentes principales

Tertimuss está formado por los componentes que se detallan en la figura A.1.

La *generación automática de tareas* genera un conjunto de tareas periódicas a partir de una especificación de la configuración del sistema. Permite optar por diferentes algoritmos para realizar esta generación.

El *simulador de planificadores* calcula una planificación a partir de una descripción del sistema y de un conjunto de tareas, generando también la evolución de la temperatura en el sistema. Utiliza para ello el *planificador* y el *simulador térmico*.

El *planificador* reacciona a diferentes eventos del simulador, y responde con la asignación de tareas a CPUs, el cambio de frecuencia en los núcleos de procesamiento, o la petición de generación de evento en un instante acordado del futuro.

El *simulador térmico* calcula la temperatura que alcanza un sistema A tras ejecutar un conjunto de tareas T durante un tiempo t.

El *analizador de planificaciones* extrae información de una planificación, o genera visualizaciones de la misma. Este incluye distintas técnicas de análisis así como diferentes tipos de visualizaciones.

A.1.2. Modelo de simulación

En su primera versión, Tertimuss usaba un modelo por pasos constantes, simulando en cada paso un número fijo de ciclos establecido por el usuario. Disminuir este número de ciclos por paso permitía aumentar la precisión de la simulación, reduciendo el riesgo de aparición de eventos notificados tardíamente al planificador según este se acercaba a 1, mientras que aumentarlo reducía el coste de la misma.

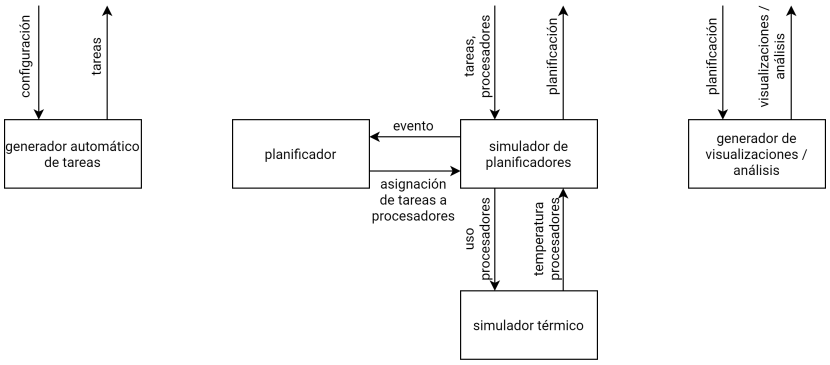


Figura A.1: Componentes principales de Tertimuss.

En algunos planificadores era sencillo determinar el número de ciclos máximo que no comprometía la precisión, mientras que en otros esta tarea era muy complicada y se terminaba simulando ciclo a ciclo.

Este problema llevó a proponer un nuevo paradigma, que basa la simulación en eventos notificados al planificador. Es decir, la interacción del planificador con el sistema se limita a la respuesta del primero a los eventos del segundo.

El simulador de planificadores es capaz de calcular el momento en el que ocurren estos eventos, por lo que su acción se limita a estos instantes temporales. Es decir, cada paso de simulación coincide con la activación de un evento. Esto acelera notablemente la simulación, ya que su costo es proporcional al número de eventos.

Como se verá posteriormente, el planificador puede solicitar la generación de un evento futuro. La frecuencia con la que este lo solicite, afectará al costo de la simulación. Es decir, en esta nueva aproximación, se traspasa la responsabilidad de la optimización desde el usuario al programador del planificador. Esto aumenta la dificultad de definición de un planificador optimizado, aunque si esto no se requiere, se puede obtener un comportamiento similar al previo haciendo que el planificador solicite un evento cada un número constante de ciclos.

Eventos disponibles

Los eventos que puede notificar el sistema son los siguientes:

- Inicio de un hiperperíodo: Permite conocer el inicio y el final de un hiperperíodo.
- Activación de un trabajo: Permite conocer la activación de un trabajo de una tarea periódica o aperiódica.
- Finalización de un trabajo: Permite conocer cuando un trabajo ha completado su ejecución, para no asignarlo nuevamente a un núcleo de procesamiento.

– Pérdida del plazo de un trabajo: Permite conocer cuando un trabajo ha perdido su plazo, para evitar continuar su ejecución.

Cada planificador configura de forma dinámica los eventos a atender. Tras atender un evento, el planificador puede solicitar la generación de un nuevo evento en el futuro.

A.1.3. Diagramas de clases

En la Fig. A.2 se encuentra el diagrama de paquetes del entorno tertimuss. Está compuesto por 7 paquetes agrupados en 3 conjuntos, *biblioteca de simulación de TCPNs*, *biblioteca de simulación térmica* y *biblioteca de simulación de planificadores*. Aunque estos conjuntos se suministren como una única biblioteca, no existen dependencias cíclicas entre ellos, por lo que se pueden desacoplar y distribuir en forma de bibliotecas.

A continuación se explican los paquetes que compone el entorno.

Simulador de TCPNs

El diagrama de clases del paquete se encuentra en la Fig. A.3.

Este se encarga de la simulación de TCPNs (Timed Continuous Petri Net). Existen diversas formas de simular este tipo de Redes de Petri. La clase *TCPNSimulator* es la base de la que deben heredar todas las implementaciones. Las clases *SVariableStep* y *SFixedStep* categorizan si el paso de simulación es constante o variable.

Las clases restantes representan implementaciones, en un caso mediante el método de Runge-Kutta, *SVSRungeKutta*, y en el otro mediante el método de Euler, *SVSEuler*.

Simulador térmico

El diagrama de clases del paquete se encuentra en la Fig. A.4. Este simulador se basa en el modelo de simulación térmica presentado en [1]. Aunque el modelo teórico no lo requiere, la implementación obliga a definir la malla a partir de cuboides.

Los cuboides se representan mediante la clase *Cuboid*, incluyendo el material del mismo, *SolidMaterial*, y su temperatura *CuboidTemperature*. Estos conforman el elemento de modelado base de los objetos físicos que serán simulados, *PhysicalCuboid*. Cada objeto físico se modela utilizando múltiples *PhysicalCuboid*s, según su composición y forma.

Se proporciona la definición de dos materiales, *SMCooper* y *SMSilicon*, que representan el cobre y el silicio respectivamente. El usuario puede definir nuevos materiales si lo requiere.

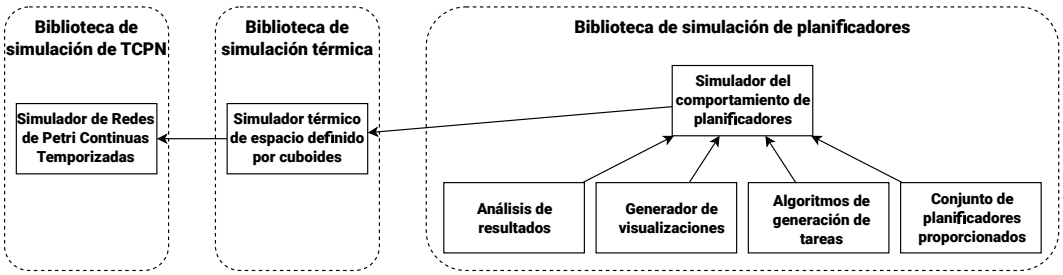


Figura A.2: Arquitectura del entorno de simulación tertimuss.

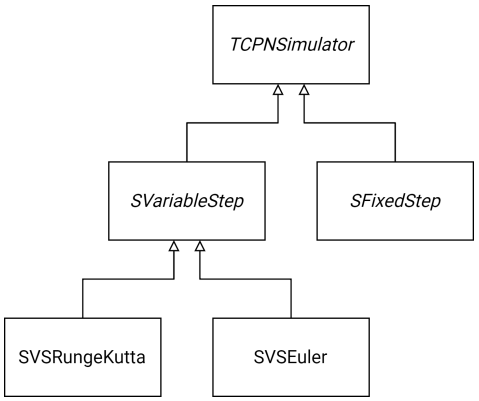


Figura A.3: Diagrama de clases del simulador de TCPNs.

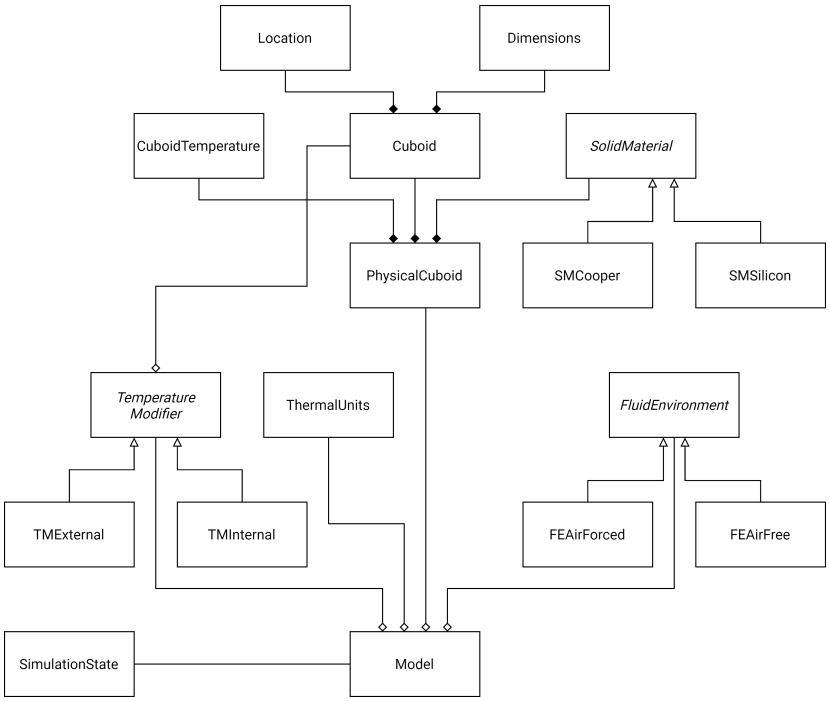


Figura A.4: Diagrama de clases del simulador térmico.

La clase *FluidEnvironment* representa el ambiente en el que se encuentra la malla. Se proporciona la definición de dos tipos, *FEForcedAir* y *FEAirFree*, representando aire forzado y aire sin movimiento respectivamente. El usuario puede definir nuevos tipos si lo requiere.

La clase *TemperatureModifier* y sus clases derivadas *TMExternal* y *TMInternal*, modelan los puntos de aplicación de energía sobre la malla. *TMExternal* representa un punto de aplicación no influenciado por la temperatura de la malla, y *TMInternal* representa un punto influenciado por la misma.

El modelo de simulación, *Model*, está conformado por la unión de todos los elementos anteriores. El estado de la temperatura durante la simulación se representa mediante *SimulationState*.

Simulador del comportamiento de planificadores

El diagrama de clases del paquete se encuentra en la Fig. A.5. Está formado por seis sub-paquetes, que se dividen según el elemento de la simulación que definen. Se explican a continuación.

- *Definición del microprocesador*: Aglutina las clases necesarias para definir físicamente el microprocesador, así como las frecuencias a las que este puede trabajar.
- *Definición del conjunto de tareas*: Contiene las clases necesarias para definir las tareas a planificar.
- *Definición del entorno*: Recoge la definición del entorno físico en el que se encuentra el microprocesador, utilizado durante la simulación térmica.
- *Definición del planificador*: Posee una clase base que deben implementar todos los planificadores, *Scheduler*, de la que extienden dos tipos de algoritmos de métodos de planificación. En primer lugar, los planificadores centralizados, *CentralizedScheduler*. En este tipo de planificadores, los eventos que aparecen son globales, de todo el sistema. Del mismo modo, el planificador actúa de forma global. Es decir, el planificador tras ser invocado tiene una visión total del sistema y puede asignar la ejecución de tareas a cualquier procesador. Alternativamente, los planificadores distribuidos, *DistributedScheduler* tienen una visión local del sistema, restringida a un único procesador. El planificador es invocado de forma individual en cada núcleo de procesamiento. Este segundo tipo de planificadores permiten la planificación en arquitecturas heterogéneas, o en arquitecturas donde los relojes de los núcleos de procesamiento funcionan a frecuencias distintas, no se ha implementado aún en Tertimuss pero se ha dejado el entorno preparado para poder hacerlo.
- *Definición del resultado de la planificación*: Contiene las clases necesarias para almacenar el resultado de la planificación.

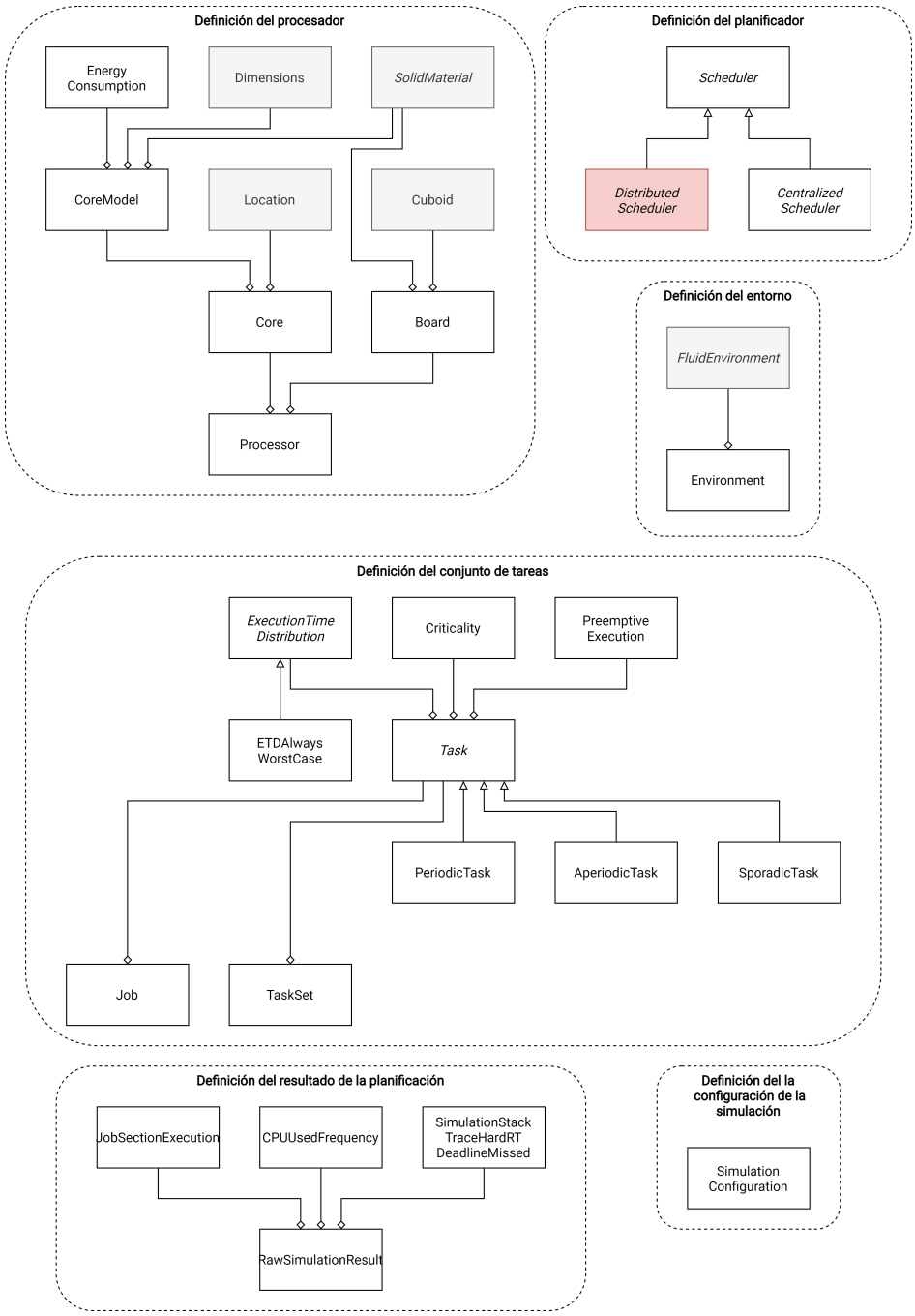


Figura A.5: Diagrama de clases de la librería de simulación del comportamiento de planificadores.

– *Definición de la configuración de la simulación*: Contiene una única clase, que se encarga de almacenar la configuración del simulador de planificadores.

Análisis de resultados

Este paquete ofrece diferentes herramientas para analizar una planificación realizada. Entre estas herramientas se encuentra un analizador de cambios de contexto y migraciones, y un analizador de cumplimiento de plazos. Ambos permiten categorizar

la información obtenida según características de las tareas o de los trabajos.

El diseño de este paquete se ha realizado para permitir su ampliación con nuevas herramientas.

Generador de visualizaciones

Este paquete permite incluir herramientas para la generación de visualizaciones. Las implementadas hasta el momento ofrecen varias visualizaciones relacionadas con la planificación, así como con la evolución de la temperatura de los procesadores. Como en los otros casos, el paquete está diseñado para poder ser ampliado.

Algoritmos de generación de tareas

El diagrama de clases del paquete se encuentra en la Fig. A.6. Aglutina diferentes generadores automáticos de tareas periódicas. Se define la clase base *PeriodicTaskGenerator* que debe implementar cualquier generador que se incorpore a este paquete. Se incluyen las implementaciones de los algoritmos UUniFast [55] y UUniFastDiscard [48].

El paquete también proporciona generadores automáticos de plazos, que son usados desde los generadores automáticos de tareas. Actualmente solo se proporciona una implementación, que genera plazos con una distribución uniforme en un intervalo dado.

Conjuntos de planificadores proporcionados

El diagrama de clases del paquete se encuentra en la figura A.6. Este contiene las diferentes implementaciones de planificadores que se proporcionan con el entorno de simulación.

Las implementaciones proporcionadas son: AlECS [7], CAlECS [8], EDF global, y OLDTFS [4].

A.1.4. Distribución e integración continua

El entorno de simulación Tertimuss está configurado para poder ser instalado mediante el gestor de dependencias de Python `pip` [56]. Esto se ha logrado haciendo uso del entorno `poetry` [57], de forma que puede ser distribuido desde los repositorios por defecto de paquetes oficiales de Python [58].

Por otra parte, el proyecto está públicamente disponible en GitHub [10]. Esto ha permitido automatizar la validación mediante pruebas automáticas del proyecto cada vez que se incluye un cambio en el mismo, explotando las herramientas de integración continua que ofrece esta plataforma.

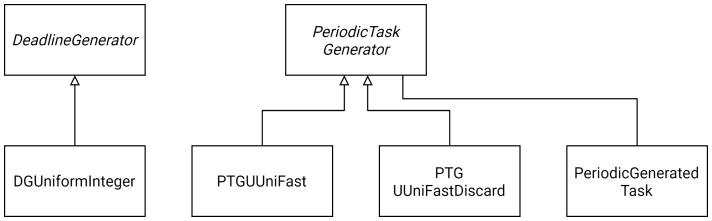


Figura A.6: Diagrama de clases de los generadores de tareas automáticos

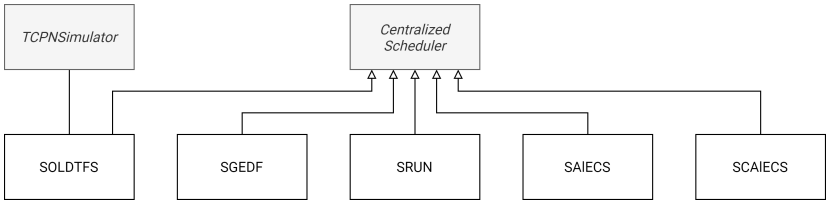


Figura A.7: Diagrama de generación de tareas

Anexos B

Planificación proyecto

B.1. Horas dedicadas

Tabla B.1: Dedicación de horas al TFM

Tarea	**Horas**
Rearquitectura de Tertimuss	200
Documentación de Tertimuss	20
Ampliación características de Tertimuss	40
Desarrollo de pruebas automáticas y corrección de errores en Tertimuss	50
Análisis de los costes asociados al RT en plataforma real	20
Desarrollo y análisis de comparativas experimentales	60
Lectura de artículos	50
Contribuciones en trabajos científicos	60
Redacción de la memoria	100
Reuniones	30
Total	630

B.2. Diagrama de Gantt

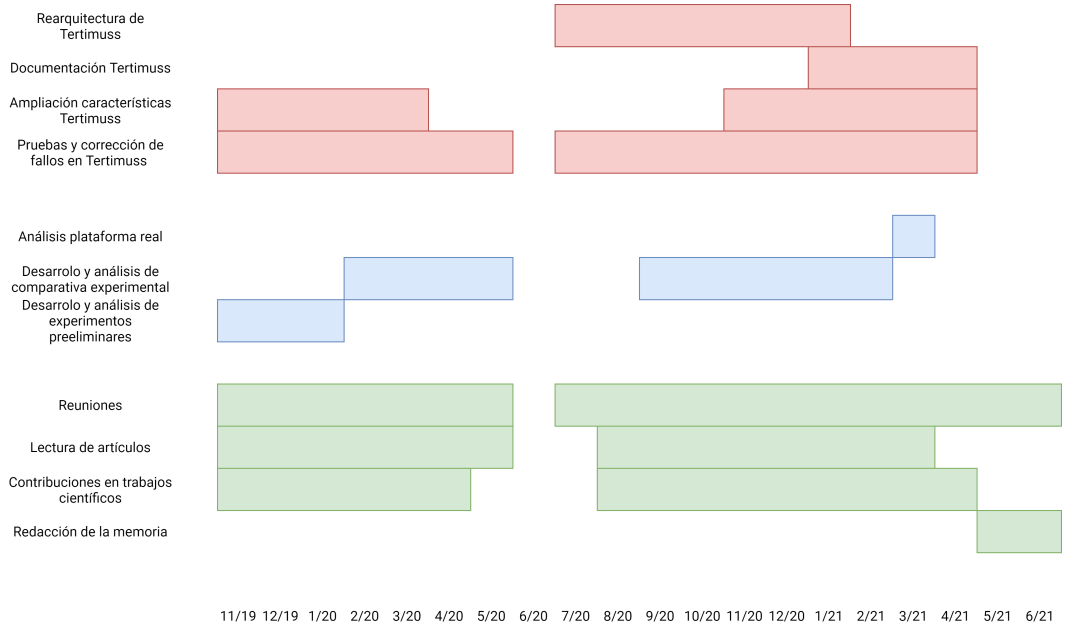


Figura B.1: Visión general de CAlECS.

Anexos C

Ejemplo de funcionamiento de CAlECS

En esta sección se ejemplifica el funcionamiento del algoritmo para facilitar su comprensión.

C.1. Definición del problema

Sea el conjunto de tareas TRD con plazos implícitos dado en el Tab. C.1, que queremos planificar en un núcleo de procesamiento con cinco núcleos disponibles. Estos núcleos pueden operar a 1 Hz, 1.5 Hz, 2 Hz, 2.5 Hz, y 3 Hz. Se han elegido unas frecuencias deliberadamente muy bajas para simplificar la explicación. No se definen restricciones térmicas ni características físicas del núcleo de procesamiento, ya que la actividad térmica se obvia en este ejemplo.

C.2. Preparación del conjunto de tareas

Como muestra la Tab. C.2, la frecuencia mínima que permite cumplir las restricciones TRD es 1 Hz, por lo que $F^* = 1\ Hz$.

Debido a que con esta frecuencia la utilización resultante es $U - tot = 4{,}4$, menor al número de núcleos de procesamiento (5), se debe añadir una o varias tareas de relleno virtual. En este ejemplo, se decide añadir una única tarea de relleno virtual tal que $\tau_{idle} = 0{,}6$, $cc_{idle} = 12$ y $d_{idle} = 20$.

Tareas	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7
cc_i	10	5	7	7	7	14	3
d_i	20	10	10	10	10	20	5

Tabla C.1: Conjunto de tareas en el ejemplo. WCET (cc_i) y plazos relativos (d_i) dados en ciclos.

Utilización de las tareas	u_1	u_2	u_3	u_4	u_5	u_6	u_7	U_{tot}
F = 1.0 Hz	0.5	0.5	0.7	0.7	0.7	0.7	0.6	4.4
F = 1.5 Hz	$0.\overline{3}$	$0.\overline{3}$	$0,4\overline{6}$	$0,4\overline{6}$	$0,4\overline{6}$	$0,4\overline{6}$	0.4	$2,9\overline{3}$
F = 2.0 Hz	0.25	0.25	0.35	0.35	0.35	0.35	0.3	2.2
F = 2.5 Hz	0.2	0.2	0.28	0.28	0.28	0.28	0.24	1.76
F = 3.0 Hz	$0,1\overline{6}$	$0,1\overline{6}$	$0,2\overline{3}$	$0,2\overline{3}$	$0,2\overline{3}$	$0,2\overline{3}$	0.2	$1,4\overline{6}$

Tabla C.2: Utilización de los conjuntos de tareas a diferentes frecuencias.

C.3. *Clustering* de tareas

En esta fase se aplica el Alg. 1 para asignar las tareas a los cinco núcleos de procesamiento garantizando a la vez una planificación factible con el sistema con utilización máxima.

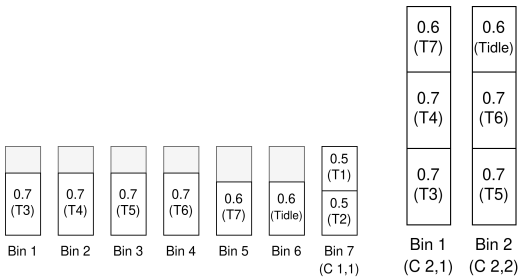
La primera iteración del Alg. 1 resuelve el BPP con contenedores de tamaño 1 (volumenContenedor = 1) usando una heurística Best fit descending (BFD). Esto genera siete contenedores (Fig. C.1 a). De ellos, solamente uno (Bin 7, $(C1, 1)$) tiene utilización máxima, por lo cual es el único seleccionado por la heurística. La ejecución de este *cluster* se asigna a un único núcleo (CPU_1), que es retirado del conjunto de núcleos de procesamiento disponibles.

A continuación se incrementa en 1 el tamaño de los *clusters* a buscar. Como este número es aún inferior al número de núcleos de procesamiento restantes (4), se vuelve a ejecutar el BPP con contenedores de tamaño 2 (volumenContenedor = 2). Esto produce dos *clusters* (Contenedor 1, $(C2, 1)$) y (Contenedor 2, $(C2, 2)$) (Fig. C.1 b), cada uno de los cuales requiere dos núcleos de procesamiento para su ejecución. Se asignan los núcleos de procesamiento CPU_2 y CPU_3 a (C 2,1) (tareas τ_3, τ_4, τ_7), y CPU_4 y CPU_5 a (C 2,2) (tareas $\tau_5, \tau_6, \tau_{idle}$). Se puede planificar cada uno de estos *clusters* mediante un planificador global óptimo debido a que el número de núcleos de procesamiento disponibles es igual a su utilización.

Tras esta iteración, el número de núcleos disponibles (0) es menor que el tamaño del cluster a buscar (3), por lo que se finaliza el algoritmo.

C.4. Planificado por *cluster* y obtención del ejecutivo cíclico

En el siguiente paso, se aplica un planificador específico sobre cada *cluster*, con el resultado mostrado en la Fig. C.2. El *cluster* $(C1, 1)$ (tareas τ_1, τ_2, hiperperíodo igual a $20s$) sólo requiere un núcleo de procesamiento (CPU_1), por lo que se aplica EDF. Los *clusters* $(C2, 1)$ y $(C2, 2)$ requieren dos núcleos de procesamiento cada uno por lo que



(a) Volumen contenedor 1 (b) Volumen contenedor 2

Figura C.1: BBP en alg. 1: a) Iteración 1; b) Iteración 2

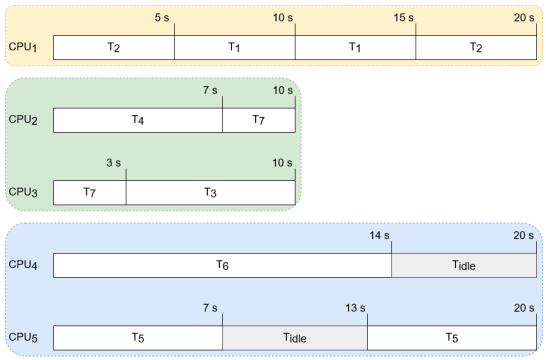


Figura C.2: Planificación de los clusters en sus respectivos hiperperíodos

se les aplica la etapa de planificado del planificador global AlECS (hiperperíodo igual a $10s$ y $20s$ respectivamente).

Una vez realizadas estas planificaciones, se replica la planificación de aquellos *clusters* cuyo hiperperíodo sea inferior al hiperperíodo de $\mathcal{T}$ tantas veces sea necesario para obtener un ejecutivo cíclico completo. En el caso de $(C2, 1)$ la planificación debe ser replicada una única vez. El ejecutivo cíclico resultante se muestra en la Fig. C.3. El hiperperíodo de cada *cluster* es siempre divisor del hiperperíodo total, por lo que el número entero de replicaciones necesarias es siempre entero.

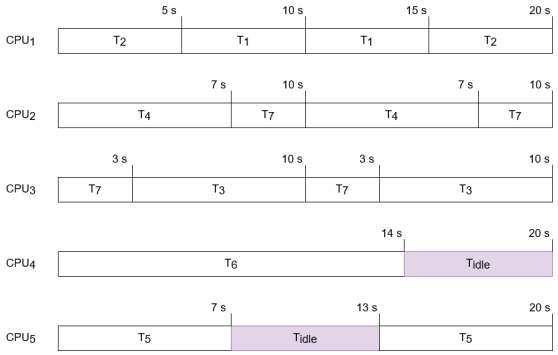


Figura C.3: Ejecutivo cíclico para las tareas tiempo real duro de la tabla C.1

Anexos D

Ejemplo de flujo de trabajo para la aplicación de un ejecutivo cíclico

Tareas	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7
cc_i	15000	7000	7000	7000	7000	14000	3000
d_i	20	10	10	10	10	20	5

Tabla D.1: Conjunto de tareas en el ejemplo. WCET (cc_i) derivado del cómputo de las tareas de forma aislada y plazos de finalización relativos (d_i) dados en ciclos y segundos respectivamente.

En esta sección se muestra un ejemplo cuya finalidad es comprender el flujo de trabajo.

D.1. Definición del problema

Se cuenta con el conjunto de tareas de la tabla D.1, que se ejecuta sobre un microprocesador de 5 núcleos con una frecuencia de núcleo de procesamiento de 1000 Hz. Esto implica que el conjunto de tareas inicialmente tiene una utilización de 4,85.

El WCET que se muestra en la tabla es el obtenido en el primer componente del flujo de trabajo, por lo que solamente se contabilizan los costes que tienen las tareas cuando son ejecutadas en exclusividad sobre los recursos de cómputo.

Se asume que el sistema tiene un tiempo de cambio de contexto de 10 ciclos y un tiempo de migración de 20 ciclos (no se incluye el coste del cambio de contexto previo a la migración). Además, las tareas 1 y 6, tienen un recurso compartido que se accede en exclusión mutua, teniendo la región crítica un coste de 15 ciclos. Las tareas 4 y 7 comparten un recurso diferente con una región crítica que tiene el mismo coste que la anterior.

D.2. Cálculo de la solución

La resolución del ejemplo, se ha automatizado mediante un script de Python que hace uso de la biblioteca Tertimuss para la obtención del ejecutivo cíclico y el análisis del mismo.

Este tarda 3 iteraciones en encontrar una planificación que cumple los requisitos. El WCET final de las tareas tras la planificación se encuentra en la tabla D.2, mientras en la tabla D.5 aparece la evolución de los sobre-costes sobre el WCET de las tareas en las distintas iteraciones, y en la tabla D.4 aparece la evolución del número de cambio de contextos, migraciones y sobre-costes por exclusión mutua de los trabajos en las diferentes iteraciones del algoritmo.

La figura D.1 muestra la planificación que se realiza en cada una de las etapas del algoritmo por tareas y por trabajos, mientras que en la tabla D.3 aparece la relación entre las tareas y los trabajos.

D.3. Análisis de la solución

La planificación de las diferentes iteraciones es muy distinta, aunque debido a la baja tasa de migraciones y cambios de contexto que presenta CAlECS, el algoritmo converge de una forma rápida.

Entre las iteraciones, el mayor aumento del WCET se produce en la transición de la primera a la segunda iteración. Esto se debe a que la planificación entre ambas cambia muy abruptamente, por lo que las tareas que presentan un sobrecoste son distintas a las que lo presentan en la primera iteración. Además, en la segunda iteración CAlECS no consigue clusterizar el conjunto de tareas, y esto produce sobre-costes mayores.

En la tercera iteración, aunque el número de migraciones y cambios de contexto de los trabajos aumenta, el de las tareas no lo hace, lo que permite que el algoritmo converja.

Tareas	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7
cc_i	15035	7070	7040	7070	7040	14055	3085
d_i	20	10	10	10	10	20	5

Tabla D.2: Conjunto de tareas en el ejemplo. WCET (cc_i) final y plazos de finalización relativos (d_i) dados en ciclos y segundos respectivamente.

Tareas	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7
Trabajos	j_1	j_2, j_3	j_4, j_5	j_6, j_7	j_8, j_9	j_{10}	j_{11}, j_{12}, j_{13}, j_{14}

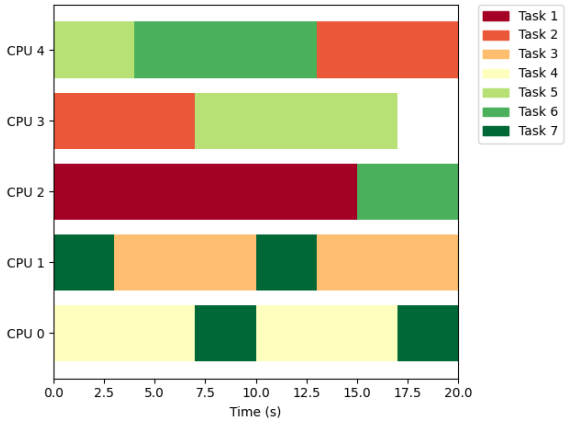
Tabla D.3: Relación tareas y trabajos.

Trabajo	Iteración 1			Iteración 2			Iteración 3		
	Número cambios de contexto	Número migraciones	Sobrecarga por exclusión mutua en ciclos	Número cambios de contexto	Número migraciones	Sobrecarga por exclusión mutua en ciclos	Número cambios de contexto	Número migraciones	Sobrecarga por exclusión mutua en ciclos
1	1	0	15	2	0	15	2	0	15
2	1	0	0	3	2	0	3	2	0
3	1	0	0	2	1	0	2	1	0
4	1	0	0	1	0	0	2	1	0
5	1	0	0	2	1	0	2	1	0
6	1	0	15	2	1	30	2	1	30
7	1	0	15	1	0	30	1	0	30
8	2	1	0	2	1	0	1	0	0
9	1	0	0	2	1	0	2	1	0
10	2	1	15	2	1	15	2	1	15
11	1	0	15	2	1	15	2	1	15
12	1	0	0	1	0	15	1	0	15
13	1	0	15	3	2	15	3	2	15
14	1	0	0	1	0	15	1	0	15

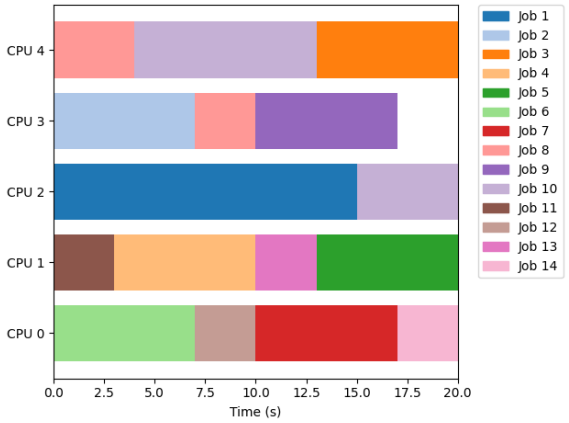
Tabla D.4: Número de cambios de contexto, migraciones y sobrecarga debida a la exclusión mutua experimentada en las distintas iteraciones.

Tarea	Sobrecarga producida por la planificación en ciclos		
	Iteración 1	Iteración 2	Iteración 3
1	25	35	35
2	10	70	70
3	10	40	40
4	25	70	70
5	40	40	40
6	55	55	55
7	25	85	85

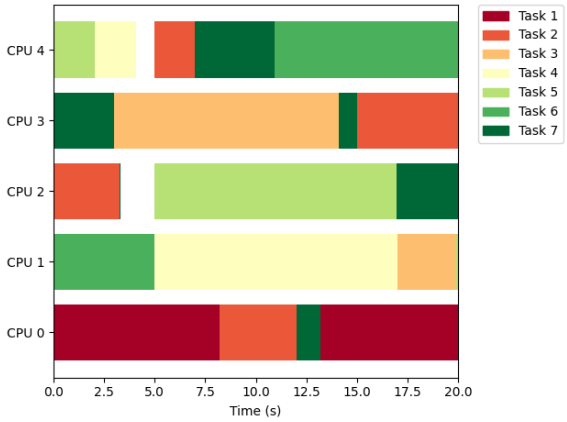
Tabla D.5: Sobrecarga de las tareas en ciclos tras las diferentes iteraciones.



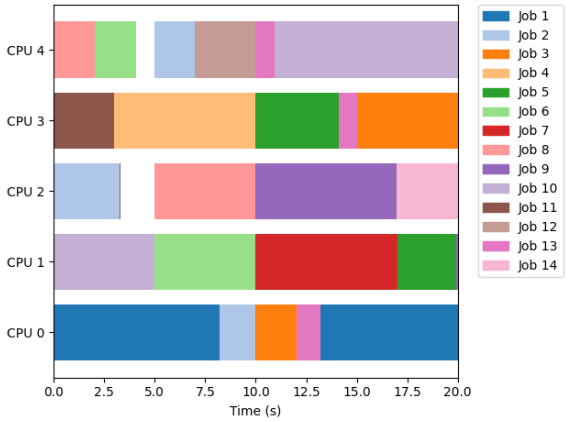
(a) Iteración 1 / Tareas



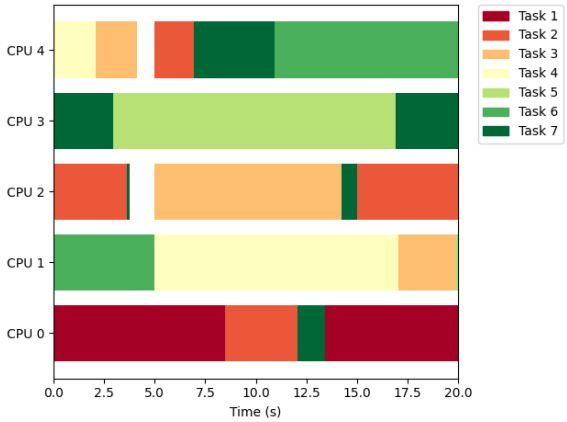
(b) Iteración 1 / Trabajos



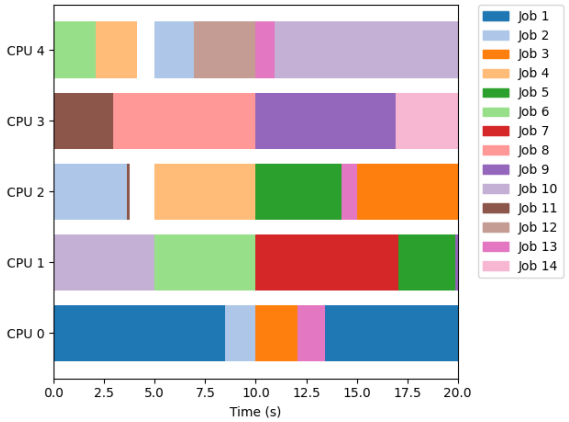
(c) Iteración 2 / Tareas



(d) Iteración 2 / Trabajos



(e) Iteración 3 / Tareas



(f) Iteración 3 / Trabajos

Figura D.1: Ejecutivo cíclico producido en las diferentes iteraciones (vistas de tareas y trabajos).

Anexos E

Características Raspberry Pi 3B

En la tabla E.1 se describen alguna de las características de la placa de desarrollo Raspberry Pi 3B relacionadas con el análisis realizado sobre ella [59] [60].

Número de procesadores	4
Frecuencia procesadores	1.2 GHz
Memorias cache	write-back write-allocate exclusivas
Cache de primer nivel	16 KB de instrucciones (asociatividad 2) 6 KB de datos (asociatividad 4)
Cache de segundo nivel	compartida entre todos los núcleos 512 KB unificada (asociatividad 16)
TLB	512 entradas
Unidades vectoriales	unidad NEON por núcleo
Memoria principal	1 GB

Tabla E.1: Características Raspberry Pi 3B.