Universidad Zaragoza

Trabajo Fin de Grado

ENTORNO DE SIMULACIÓN DE PLANIFICADORES TIEMPO REAL PARA MULTIPROCESADORES CON RESTRICCIONES TÉRMICAS Y DE ENERGÍA

AUTOR

Abel Chils Trabanco

DIRECTOR

José Luis Briz Velasco

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2019

RESUMEN

El diseño de algoritmos de planificación tiempo real sobre multiprocesadores es un campo en el que la industria tiene interés, pero es más complejo que el diseño tradicional sobre monoprocesadores. Esa complejidad aumenta aún más si a las restricciones temporales hay que añadir que una mala planificación puede generar puntos calientes en el procesador. Las herramientas de simulación ayudan a evaluar la corrección de las planificaciones cuando las pruebas o condiciones teóricas de planificabilidad no aseguran la existencia o no de una planificación viable, por ser generalmente condiciones suficientes pero no necesarias. El cumplimiento de restricciones térmicas o de energía aún hace más necesarios estos simuladores.

Este trabajo presenta un simulador de planificadores de tiempo real sensibles a temperatura y energía para multiprocesadores. Ha sido creado en Python haciendo uso de componentes sin restricciones de licencia, y usa Redes de Petri Continuas Temporizadas (TCPN) para simular el sistema. El entorno se basa en uno previo, al cual va a substituir, creado en Matlab, con problemas que limitan su desarrollo, y con un alto coste computacional y de ocupación de memoria.

En el nuevo entorno se mitigan los problemas del anterior logrando un speed-up del x9.1 y una reducción de la ocupación de memoria de x5.4. Estas mejoras se logran modificando la TCPN que simula el sistema, optimizando la resolución de la ecuación de estado de la misma y cambiando las estructuras de datos en las que se almacena la misma.

La arquitectura del nuevo entorno es completamente diferente de la anterior, y es ahora modular y escalable. Desacopla el simulador de TCPNs de la definición del modelo de sistema, posibilitando cambiar cualquiera de los dos de forma independiente. Además se realiza lo propio entre la definición de planificadores y el conjunto del simulador, lo que permite definirlos mediante una interfaz simple sin conocer el funcionamiento del resto del entorno.

Por otro lado, se añaden características nuevas como la posibilidad de tratar tareas aperiódicas, la inclusión de planificadores con intervalos de planificación variables y la inclusión de frecuencias variables en el procesador. Se ha añadido también un nuevo modelo de interacción con el usuario mediante línea de comandos que permite la simulación por lotes, así como nuevos planificadores.

Utilizando el nuevo entorno se realiza una comparación entre los cuatro planificadores implementados, donde se muestran y analiza su viabilidad y comportamiento en varios escenarios, incluyendo el cumplimiento de restricciones térmicas.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE CUADROS

LISTINGS

ACRÓNIMOS

TFG Trabajo de Fin de Grado

MPSoC Multiprocessor Sytem on Chip

CPU Central Processing Unit

TR Tiempo Real

TRD Tiempo Real Duro

TRB Tiempo Real Blando

TRF Tiempo Real Firme

FTP Fixed Task Priority

FJP Fixed Job Priority

DP Dynamic Priority

EDF Earliest Deadline First

G-EDF Global Earliest Deadline First

G-LLF Global Least Laxity First

OLDTFS On-line discretization of Thermal fluid schedule [10]

JDEDS Identificador arbitrario del algoritmo publicado en [35]

PPL Problema de Programación Lineal

TCPN Timed Continuous Petri Net

WCET Worst Case Execution Time

DVFS Dynamic Voltage and Frequency Scaling

mcm Mínimo común múltiplo

MCD Máximo Común Divisor

LIL List of Lists format

CSR Compressed Sparse Row format

BSR Block Compresed Sparse Row format

COO Coordinate format

CSC Compressed Sparse Column format

DIA Diagonal format

DOK Dictionary of Keys format

G-EDF Global earliest deadline first

JSON JavaScript Object Notation

PEP Python Enhancement Proposals

PEP8 Python Enhancement Proposal no. 8

1

INTRODUCCIÓN

1.1 MOTIVACIÓN

*El factor SWaP (*Space, Weight and Power*, área, peso y potencia) es un parámetro clave en el diseño actual de sistemas empotrados en automoción y aeronáutica.*

Los MPSoCs (Multiprocessor Sytems on Chip) permiten reducir el coste, tamaño, peso y consumo de muchos sistemas empotrados tiempo real (TR), en relación a soluciones sobre monoprocesador, por lo que están en el foco de interés de la industria aeroespacial y de automoción. Sin embargo entrañan nuevos retos, requiriendo algoritmos de planificación TR más complejos que los usados en monoprocesadores. Al problema inherente a todo sistema TR de estar sujeto a restricciones temporales, hay que añadir además que, por ejemplo, una mala planificación de tareas puede ocasionar puntos calientes (*hot spots*) que reduzcan la vida útil del dispositivo o provoquen fallos difíciles de prever, o también incrementar el consumo energético, factor crucial en sistemas autónomos. Por este motivo, en los últimos años existe un alto interés en incluir restricciones térmicas y de energía en planificación TR.

Debido al complejo espacio de diseño, probar nuevos algoritmos de planificación TR implementándolos sobre un sistema real, incluso sobre plataformas específicas de evaluación como el sistema operativo Litmus-RT [4], puede ser demasiado complejo, sobre todo cuando el diseño del planificador se encuentra en una etapa temprana o simplemente se están explorando vías de diseño. A menudo, como veremos, los criterios de planificabilidad TR sólo aportan condiciones necesarias, pero no suficientes. Así, el cumplimiento de una cierta condición de planificabilidad puede asegurar que un conjunto de tareas TR dado es planificable bajo cierto algoritmo, pero no que no exista alguna posible planificación correcta si no la cumple. Los entornos de simulación de planificadores TR flexibles y eficientes contribuyen a reducir los tiempos de diseño de estos planificadores.

1.2 CONTEXTO

La investigación sobre algoritmos de planificación tiempo real sensible a temperatura y energía sobre multiprocesadores es una línea de trabajo conjunta entre 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. Los responsables de esta colaboración por cada parte son el Dr. José Luis Briz y el Dr. Antonio Ramírez Treviño respectivamente. La evaluación de los modelos y

algoritmos propuestos hasta el momento en el curso de esta investigación [8-10, 12, 34, 35] se va realizando en MATLAB [24], integrándolos progresivamente en un primer entorno de simulación, públicamente disponible [11, 13], al que denominamos *entorno de referencia* en este Trabajo de Fin de Grado (TFG).

El diseño y programación de este *entorno de referencia* ha ido obedeciendo por tanto a urgencias dictadas por plazos de entrega de contribuciones y tesis, haciendo necesario un nuevo diseño para paliar sus deficiencias tanto estructurales (limitaciones en flexibilidad, dependencia de MATLAB [24]) como funcionales (alto coste computacional, ocupación de memoria y mejoras en la precisión en algunos cálculos entre otras). Este TFG cubre precisamente esa necesidad, y supone así mismo un trabajo de introducción al tipo de investigación que se realiza en el proyecto mencionado.

1.3 OBJETIVOS

El objetivo general del TFG es diseñar e implementar un entorno de simulación de planificadores TR sensibles a temperatura y energía para multiprocesadores. Este debe de ser escalable, y aumentar las características y el rendimiento del entorno anterior citado en la Sec. 1.2. Colateralmente, el TFG debe constituir una introducción a la investigación en este campo.

Los objetivos específicos son los siguientes:

- Reducir el coste computacional de las máquinas de simulación y sus necesidades de memoria dinámica.
- Conseguir una organización modular que permita la adición o modificación tanto de modelos de sistema, de máquinas de simulación o de planificadores a simular.
- Ofrecer interfaces de uso sencillo para la parametrización de las simulaciones y la representación gráfica de resultados.
- Definir interfaces de programación bien definidos que permitan la modificación o adición de nuevos planificadores, modelos térmicos o de energía.
- Demostrar la utilización del entorno mediante estudios experimentales de los planificadores TR sobre multiprocesador implementados.

1.4 ALCANCE

El proyecto ha generado un entorno de simulación conforme a los objetivos descritos, programado en Python, que pasará a ser la nueva

herramienta de trabajo para el proyecto mencionado en la Sec. 1.2 y se pondrán a disposición pública [40].

Asimismo, el proyecto ofrece un estudio comparativo de los planificadores G-EDF, G-LLF, OLDTFS [10] y JDEDS [35], realizado mediante el nuevo entorno.

1.5 ENTORNO DE TRABAJO

Las herramientas principales utilizadas han sido el entorno de desarrollo PyCharm [29] y el editor de texto Visual Studio Code [41], ambos de código abierto [28, 42]. También se ha hecho uso del entorno MATLAB [24] para poder estudiar y comparar el simulador de referencia.

El simulador se ha desarrollado en el lenguaje Python 3.7 [31] haciendo uso de las bibliotecas Scipy [22], Matplotlib [19], PyQt [30] y Qt5 [33]. Indirectamente también se hace uso de la biblioteca Intel Math Kernel Library [20], que aumenta considerablemente el rendimiento en las operaciones con matrices respecto a su alternativa (OpenBLAS) [27]. Para la utilización de Python, se ha hecho uso del gestor de paquetes Conda [7] integrado dentro de la distribución de Python Miniconda [25] debido a la gran cantidad de paquetes que posee, y a que permite crear múltiples entorno de Python, cada uno con versiones específicas de los paquetes.

El control de versiones se ha realizado mediante Git [17], y para mantener el código centralizado se ha recurrido a GitHub [18]. La escritura de la memoria se ha llevado a cabo en LaTeX sobre Overleaf [26]. El desarrollo del entorno y los experimentos de comparación de rendimiento se han realizado sobre el equipo descrito en el Anexo. 7.2.

1.6 METODOLOGÍA GENERAL

En primer lugar se procedió al estudio de las publicaciones del proyecto de referencia [8-10, 12, 34, 35], que fue necesario seguir consultando durante todo el desarrollo del TFG junto a otros materiales, bajo la tutela del director. Algunos conceptos se han ido comprendiendo mientras se han ido incluyendo o evaluando en el simulador.

La segunda fase consistió en aprender los conceptos básicos de los lenguajes MATLAB y Python. Este último fue utilizado en una asignatura de forma paralela al desarrollo del proyecto, lo que favoreció su aprendizaje.

La tercera fase consistió en realizar un análisis del *entorno de referencia* (en MATLAB). Se determinaron carencias, características que podían ser añadidas y se obtuvo una visión general de su implementación.

La cuarta fase como parte de la asignatura *Laboratorio de Sistemas Empotrados* consistió en portar el *entorno de referencia* a Python, intro-

duciendo ya algunas modificaciones de diseño, añadiendo algunas nuevas características y mejorando su rendimiento.

El desarrollo y utilización del *entorno de referencia*, guiado continuamente por nuevos objetivos, continuaba en paralelo añadiendo características, algunas importantes, por lo que esta primera versión en Python fue actualizándose en consecuencia.

La quinta fase abordaron el diseño modular del nuevo entorno, y la mejora del rendimiento y la usabilidad. En cuanto al diseño, se orientó a una arquitectura escalable, que facilita tanto la incorporación de nuevos planificadores, como la modificación o mejora del entorno en sí mismo. También se añadieron características nuevas, como el manejo de tareas aperiódicas, las frecuencias variables, la gestión del quantum, o el consumo de potencia estática. En cuanto a la usabilidad de la aplicación, se añadió un interfaz por línea de comandos que permite simulaciones por lotes, interoperable con el interfaz gráfico. Por último, se mejoró drásticamente el rendimiento, una de las principales limitaciones del *entorno de referencia*, llevando a cabo la consiguiente comparación experimental con el *entorno de referencia* (Sec. 3.7.4).

Durante la cuarta y la quinta fase se utilizó una metodología de diseño iterativo e incremental. El software a desarrollar fue dividido en pequeños componentes, diseñados, implementados, depurados y documentados por separado, refinados en iteraciones sucesivas. Esto supuso un sobre coste de desarrollo, amortizado por el bajo número de errores y la satisfactoria funcionalidad del producto final. Determinadas decisiones relativas al rendimiento se tomaron sobre el resultado de estudios experimentales (3.7).

Para reducir el número de errores finales se tuvo muy en cuenta la depuración de los componentes. Asi, en la cuarta fase las pruebas se realizaban comprobando si los resultados generados en el nuevo entorno desarrollado eran coherentes con los de la versión de referencia. Una vez obtenido el mismo resultado, una batería de pruebas permitía realizar verificaciones automáticas. La batería de pruebas automáticas creada agilizó la detección de nuevos errores especialmente durante la quinta fase.

La sexta fase consistió en un estudio experimental de los planificadores TR implementados en el entorno. La metodología específica de este estudio se detalla en la Sec. 5.1.

La séptima fase ha consistido en la escritura de la memoria.

1.7 PLANIFICACIÓN DEL PROYECTO

Las horas dedicadas al proyecto se encuentran en la Tab. 1, y el diagrama de Gantt del mismo en el Anexo.7.7.

Cuadro 1: Dedicación de horas al proyecto

Tarea	Trabajo previo	Trabajo de Fin de Grado
Estudio fundamentos TR en MPSoCs, TCPNs, modelado del procesador y planificadores TR en multiprocesadores	9	55
Estudio fundamentos MATLAB, Python y bibliotecas	22	4
Estudio entorno previo MATLAB	8	0
Implementación arquitectura interna del entorno de simulación	7	16
Implementación modelado del procesador	12	10
Implementación simuladores TCPN	20	39
Implementación simulador de planificadores	4	45
Implementación planificadores	12	25
Implementación representación gráfica de resultados	8	5
Implementación interfaces de usuario	24	22
Depuración y corrección de errores	44	105
Memoria y estudio experimental	0	141
Total	170	467

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, *Redes de Petri Continuas Temporizadas (TCPNs)* 2.1, *Modelo base de multiprocesador usado* 2.2, *Tiempo Real y planificación en tiempo real* 2.3 y *un conjunto de planificadores tiempo real sobre multiprocesadores* 2.4.

El Cap. 3 describe el *modelado del sistema mediante TCPNs* 3.1, *modificaciones que se han realizado sobre este sistema* 3.2, 3.4, y optimizaciones que se han realizado sobre él con el fin de *mejorar su rendimiento* 3.3, 3.5 así como de *reducir sus consumo de memoria dinámica* 3.6, así como los estudios experimentales que llevaron a ellas 3.7.

El Cap. 4 describe la *arquitectura del entorno anterior* 4.1, los *patrones de diseño adoptados durante la realización del nuevo entorno* 4.2, la *arquitectura del nuevo entorno* 4.3, así como una explicación más detallada de la *arquitectura del módulo que simula los planificadores* 4.4 y la *arquitectura de la interfaz de usuario* 4.5.

El Cap. 5 muestra un estudio comparativo realizado entre distintos planificadores TR haciendo uso del entorno creado.

El Cap. 6 recapitula el trabajo, aportando unas breves conclusiones y planteando vías de continuación.

2

FUNDAMENTOS

2.1 REDES DE PETRI CONTINUAS TEMPORIZADAS (TCPNS)

Una red de Petri Continua Temporizada (TCPN) es un sistema descrito por una tupla (N, λ, m_0), donde N es una Red de Petri representada por la tupla N = (P, T, *Pre*, *Post*), y P y T son conjuntos de lugares y transiciones respectivamente, siendo los primeros representados como rectángulos y los segundos como círculos. *Pre* y *Post* son matrices de incidencia, y poseen dimensiones |P| x |T|, donde *Pre*[p_i, t_j] (respectivamente *Post*[p_i, t_j]) es el peso del arco que va desde el lugar $p_i \in$ P a $t_j \in$ T (respectivamente yendo desde $t_j \in$ T a $p_i \in$ P).

La matriz de incidencia de N se define como C = *Post* - *Pre*. El vector de columnas $m_0 \in \{ \mathbb{R}^+ \cup 0 \}^{|P|}$ es el estado inicial (marcado inicial). Su *i*-esima posición representa el marcado en el lugar p_i.

En las TCPNs, tanto el marcado como los disparos de las transiciones pueden tener cualquier valor en $\mathbb{R}^+$. Una transición está habilitada con un marcado m si $\forall p_i \in^\bullet t_j, m\,[p_i] > 0$, y su grado de habilitación, $habilitación(t_j, m) = \min_{p_i \in^\bullet t_j} \frac{m[p_i]}{Pre[p_i, t_j]}$.

El vector $\lambda \in \{ \mathbb{R}^+ \cup 0 \}^{|P|}$ representa la tasa de disparo de las transiciones, el cual permite la evolución temporal del modelo.

El disparo de las transiciones ocurre a determinada velocidad, que está generalmente basada en función de la tasa de disparo de las transiciones λ y el marcado actual m, esta función depende de la semántica asociada a las transiciones. Bajo la semántica de servidores infinitos [37], el flujo en una transición es definido como $f[t_j] = \lambda[t_j]$ * $habilitación(t_j, m)$, y la matriz de frecuencia de disparos como, Λ = diag($\lambda_1, \ldots, \lambda_{|T|}$). Para que el flujo pueda ser correctamente definido, cada transición continua debe tener al menos un lugar de entrada.

Una *configuración* de una TCPN es un grupo de arcos (p_i, t_j) tal que p_i provee el ratio mínimo $\frac{m[p_i]}{Pre[p_i, t_j]}$ para todos los lugares $p_i \in^\bullet t_j$ para el marcaje m. Decimos que p_i restringe t_j para cada arco (p_i, t_j) en la configuración. La matriz de configuración es definida como:

$$\Pi(m) = \begin{cases} \frac{1}{Pre[p_i, t_j]} & si\ p_i\ restringe\ a\ t_j \\ 0 & en\ otro\ caso \end{cases} \tag{1}$$

El flujo sobre la transición es definido como $f(m) = \Lambda * \Pi(m) * m$. El comportamiento dinámico de un sistema de redes de Petri viene dado por su ecuación fundamental:

$$\dot{m} = C * \Lambda * \Pi(m) * m \tag{2}$$

La estructura de la TCPN determina la matriz de incidencia C, el marcado actual determina la matriz de configuración $\Pi(m)$ y el ratio λ determina Λ.

Para aplicar una ley de control sobre 2, se añade un término u a cada transición tal que $0 <= u <= f(m)$. Entonces, el flujo controlado de la transición pasa a ser $w[t_j] = f[t_j] - u[t_j]$. Esto implica que la ecuación de estado resultante será:

$$\dot{m} = C * (f(m) - u) = C * w \tag{3}$$

2.2 MODELO BASE DE MULTIPROCESADOR

En este trabajo, en conformidad con su contexto y alcance (Sec. 1.2) se asume como arquitectura base un multiprocesador compuesto de una o varias unidades de procesamiento indistintamente denominadas en lo que sigue CPUs o *núcleos*, carentes de mecanismos de ocultación o disminución de latencia de acceso a memoria (memorias cache, especulación) y sin conflictos en dicho acceso. Las CPUs pueden modificar su frecuencia y tensión mediante mecanismos de DVFS [43].

2.3 TIEMPO REAL Y PLANIFICACIÓN

2.3.1 *Conceptos básicos sobre sistemas TR*

En la siguiente sección se definen conceptos básicos de TR [2, 12, 38].

2.3.1.1 *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.

Se indica también el término en inglés en el caso de parámetros muy comunes (porque normalmente determina la notación más comúnmente utilizada) y en el caso de términos raramente empleados en español.

2.3.1.2 *Planificación*

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

2.3.1.3 *Planificación factible*

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

2.3.1.4 *Tarea*

Una tarea (*task*) es una secuencia de instrucciones, que en ausencia de otras actividades se ejecuta de forma continua en un procesador

hasta su terminación. Una tarea se activa normalmente varias veces con diferentes datos de entrada, generando una secuencia de instancias llamadas trabajos.

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 [2].

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 puedes 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 (Sec. 1.2) son en general TRD, excepto cuando se tratan aperiódicas, que se consideran TRB).

2.3.1.5 *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.

2.3.1.6 *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.

2.3.1.7 *Trabajo*

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

2.3.1.8 *Periodo*

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

2.3.1.9 *Hiperperiodo*

El hiperperiodo (H) se define como el mínimo común múltiplo (mcm) de todos los periodos.

Una propiedad importante es que si dividimos 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) en el mismo instante temporal relativo al inicio del hiperperiodo.

2.3.1.10 *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* (i.e. 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.

El plazo absoluto requiere mantener un contador global, lo que complica la implementación de planificadores basados en el mismo, como EDF, penalizando en general su rendimiento.

2.3.1.11 *Tiempo de ejecución máximo*

El tiempo de ejecución máximo (WCET) de una tarea es el máximo tiempo de ejecución necesitado por un núcleo de procesamiento para completar sin interrupción una tarea para cualquier dato de entrada.

2.3.1.12 *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 $U = \sum_{i=1}^{n} c_i / p_i$.

2.3.1.13 *Modelo de tareas esporádicas de tres parámetros*

Este modelo se caracteriza por definir cada tarea mediante su WCET, su periodo y su plazo. Es el modelo utilizado en este trabajo de acuerdo al contexto del mismo (1.2).

2.3.2 *Planificación TR*

2.3.2.1 *Bases de la planificación*

Los planificadores intervienen en *puntos de planificación* determinados por intervalos fijos (quantum) o variables (obedeciendo a eventos

como la finalización de un trabajo o la llegada de una aperiódica, por ejemplo). En cada punto de planificación, el planificador elige para ejecución la tarea de mayor prioridad. La asignación de prioridades puede ser estática o dinámica, y según se aplique a tareas o a trabajos, da lugar a diferentes opciones. Las más comunes son:

- Tareas de prioridad fija (FTP, *Fixed Task Priority*): Las tareas tienen prioridad fija y sus trabajos heredan dicha prioridad. Ejemplos: *Rate Monotonic, Deadline Monotonic*.
- Trabajos de prioridad fija (FJP, *Fixed Job Priority*): Se puede asignar prioridad diferente a trabajos diferentes de una misma tarea, pero una vez asignada, la prioridad no cambia. Un ejemplo muy conocido es EDF (*Earliest Deadline First*).
- Prioridad Dinámica (DP, *Dynamic Priority*): no se imponen restricciones sobre la variación de la prioridad, existiendo diferentes posibilidades. La mayor parte de los algoritmos tratados en este TFG siguen este modelo.

Los planificadores pueden actuar únicamente en tiempo de ejecución (*on-line*), o bien realizar la planificación completamente o parte con anterioridad (planificación *off-line*).

2.3.2.2 *Planificación particionada*

Modelo de planificación en el que cada núcleo de procesamiento ejecuta un subconjunto disjunto del total de tareas.

2.3.2.3 *Planificación global*

Modelo de planificación en el que cada núcleo de procesamiento puede ejecutar cualquier tarea del conjunto total de tareas. Es el modelo de planificación asumido en este trabajo, de acuerdo a su contexto (1.2).

2.3.2.4 *Planificación semi-particionada*

Existen modelos híbridos entre la planificación particionada y la global que proponen diferentes variantes, como por ejemplo dividir en subtareas las tareas que no encajan completas en una CPU tras realizar un particionado, aplicando una planificación global a dichas subtareas [6].

2.3.2.5 *Laxitud*

La laxitud (*laxity*) de un trabajo en un instante temporal t, se define cómo el tiempo máximo que puede estar sin ejecutarse, a partir de ese instante temporal t, de forma que aún pueda comenzar o reanudarse y llegar a cumplir su plazo.

2.3.2.6 *Cambio de contexto*

Un cambio de contexto se produce cuando una tarea termina o es expulsada (*preempted*), iniciándose o reanudándose una nueva.

2.3.2.7 *Quantum*

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

2.3.2.8 *Particionado por fin de plazos*

La planificación basada en quantum supone una sobrecarga excesiva en determinados sistemas de planificación TR. El *particionado por fin de plazos* (*Deadline Partitioning* [16]) es una técnica alternativa que en general reduce notablemente dicha sobrecarga. 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.

2.3.2.9 *Planificación fluida*

Este concepto se utiliza no sólo en planificación TR sino en planificadores como el Complete Fair Scheduler, *opción por defecto en las distribuciones actuales de Linux*

La planificación fluida es un concepto matemático consistente en el reparto instantáneo de las CPUs disponibles entre las tareas a ejecutar. Por ejemplo, con una única CPU disponible, dos tareas se ejecutarían simultáneamente sobre ella utilizando cada una un 50 % de la misma, u otra proporción si se desea priorizar una tarea sobre la otra. Obviamente no es realizable en la práctica, y una discretización basada en el reparto infinitesimal en el tiempo de las CPUs es inviable. Sin embargo permite implementar planificadores óptimos en utilización TRD, especialmente cuando se incorporan restricciones térmicas además de temporales. Su factibilidad se basa en hacer que el *error fluido* (la diferencia entre el tiempo realmente ejecutado por un trabajo y el tiempo previsto por la planificación fluida) sea cero en los puntos de planificación. Estos últimos se pueden definir mediante quantum, o mediante particionado por fin de plazos. Se utiliza con éxito en planificadores globales tratados en este TFG, como OLDTFS [10] y JDEDS [35].

2.4 PLANIFICADORES TR EN MULTIPROCESADORES

En esta sección se resume el comportamiento de los planificadores TR relevantes para este trabajo (Sec. 1.2). Todos ellos han sido implementados en el nuevo entorno de simulación y comparados experimentalmente en el Cap. 5.

2.4.1 *G-EDF (Global Earliest Deadline First)*

G-EDF es un planificador global guiado por eventos tipo FJP, basado en EDF. Es óptimo únicamente para conjuntos de tareas TRB con plazos implícitos, pero se utiliza como referencia habitual para evaluar otros planificadores. Los eventos a los que obedece son:

- Llegada de una nueva tarea
- Finalización de una tarea
- Llegada de una tarea aperiódica

Al ocurrir uno de estos eventos, se asigna a los trabajos pendientes de finalización una prioridad mayor cuanto más próximo queda el cumplimiento de su plazo absoluto.

Este algoritmo no especifica la asignación de tareas a CPUs. En la implementación seguida en este trabajo, la asignación de tareas a núcleos se hace de tal forma que se minimicen los cambios de contexto. Las tareas aperiódicas se tratan, una vez recibidas, como el resto de tareas.

La descripción exhaustiva de estos planificadores, sus propiedades y las referencias en las que se basan está fuera del ámbito de este TFG. La información sobre los mismos ha sido extraída de las publicaciones relacionadas con el proyecto en el que se enmarca (Sec. 1.2), y puede consultarse también por ejemplo en [2] o [5].

2.4.2 *G-LLF (Global Least Laxity First)*

G-LLF es también un planificador global *on-line* de política FJP guiado por eventos, considerando los siguientes:

- Llegada de una nueva tarea
- Finalización de una tarea
- Cuando la laxitud de un trabajo que no está ejecutándose en ningún núcleo de procesamiento (y no ha completado su ejecución), es menor a la laxitud de alguno de los trabajos que se están ejecutando en alguno de los núcleos de procesamiento
- Llegada de una tarea aperiódica

Tras estos eventos, su política de asignación marca que los trabajos que aún no hayan completado se ejecución, obtendrán una prioridad acorde a su laxitud (menor laxitud, mayor prioridad). Las tareas aperiódicas se tratan, una vez recibidas, como el resto de tareas.

2.4.3 *OLDTFS*

OLDTFS (*On-line discretization of Thermal fluid schedule* [10]) se basa en el concepto de planificación fluida (Sec. 2.3.2.9) y se divide en dos fases, *off-line* y *on-line*. Este planificador no considera tareas aperiódicas.

Este además de encargarse de crear una planificación factible, asegura que la ejecución de esa planificación sobre la plataforma de cómputo no genera temperaturas superiores a un límite dado.

Durante la fase *off-line* se resuelve un problema de programación lineal (PPL) a partir de la especificación de las tareas y del modelo térmico de la Sec. 3.1.3. Este PPL calcula la porción fluida de cada tarea a ejecutar en cada CPU durante el hiperperiodo, para cumplir tanto restricciones térmicas como restricciones de plazos.

Durante la fase *on-line*, los puntos de planificación vienen dados por un quantum que se calcula por particionado de plazos (Sec. 2.3.2.8), como el máximo común divisor (MCD) del conjunto de plazos de todos los trabajos y sus múltiplos.

*El control de superficie deslizante con realimentación (*Sliding surface feedback controller*) es un control tipo* on-off *que se ha encontrado idóneo en los planificadores propuestos en la investigación de referencia de este trabajo (Sec. 1.2). Aunque se implementa en este TFG, su análisis y explicación queda fuera de los objetivos del mismo.*

En cada punto de planificación se simula un modelo TCPN reducido del sistema (*TCPN** Sec. 3.1). Un control de modo deslizante asegura que si hay una perturbación (reproducida en el modelo TCPN), las tareas retrasadas por la misma recuperen tiempo de ejecución. Esto se consigue priorizando la asignación de esas tareas de modo que se minimice el denominado *error fluido* (diferencia entre el tiempo acumulado de ejecución real y el previsto por la solución *off-line*): a mayor error fluido, mayor prioridad. Al cumplir *on-line* la ejecución en el hiperperiodo de las porciones de tiempo fluidas, calculadas *off-line*, se asegura el cumplimiento de la restricción térmica, garantizado por ser dichas porciones las soluciones del PPL.

La solución *off-line* fluida del PPL proporciona ya una asignación de tareas a *CPU*s, lo que evita tener que hacerlo al discretizar en la fase *on-line*.

2.4.4 *JDEDS*

Como en el planificador anterior, JDEDS [35] se basa en planificación fluida (Sec. 2.3.2.9) y consta de dos fases, *off-line* y *on-line*.

Además de calcular una planificación factible, JDEDS asegura que el resultado de aplicarla 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 primero la frecuencia mínima a la que deben ejecutar el conjunto de tareas TRD para cumplir las restricciones temporales maximizando la utilización, y la frecuencia máxima que el sistema soporta sin violar la restricción térmica. Esta frecuencia mínima asegura que la energía consumida por el procesador sea mínima. A continuación, se resuelve un PPL que aplicando particionado por plazos calcula la porción de cada tarea que hay que ejecutar en cada intervalo de planificación.

A diferencia del planificador OLDTFS, las porciones calculadas en la fase *off-line* de JDEDS no son fluidas sino que ya corresponden a

ciclos de CPU, por lo que no requieren discretización *on-line* posterior. Tampoco proporciona una asignación de tareas a CPUs, sino que la realiza en la fase *on-line*, ahora dirigida por los siguientes eventos, y no por quantum como en OLDTFS:

- Inicio/fin de un intervalo
- Un trabajo pendiente alcanza laxitud 0
- Llegada de una tarea aperiódica

Dado un evento, se asigna la máxima prioridad a los trabajos de laxitud 0. Los trabajos 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.

Tras la llegada de una tarea aperiódica (TRB) 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.

3

MODELADO Y SIMULACIÓN DE UN SISTEMA MULTIPROCESADOR TR SOBRE TCPNS

En la Sec. 1.3 se planteó como objetivo general de este proyecto la necesidad de disponer de un entorno de simulación más eficiente y mejor estructurado que el existente hasta ahora. Este capítulo expone el modelado original del sistema, las modificaciones y optimizaciones llevadas a cabo en el mismo y las diferencias respecto al *entorno de referencia*. Posteriormente, el Cap. 4 describirá el diseño y arquitectura de este nuevo entorno.

3.1 MODELADO DEL SISTEMA

Esta sección resume en primer lugar el modelado de un sistema TR multiprocesador mediante TCPNs que incluye el modelado térmico, tal como se ha ido definiendo en [8-10, 12, 34, 35]. Este es el modelo de partida sobre el que, en el nuevo entorno, se realizan las modificaciones que se exponen posteriormente.

3.1.1 *Modelado de tareas*

El modelo de tareas se muestra en la Fig. 1.

El período ω_i de la tarea T_i implica que llegan $\frac{1}{\omega_i}$ tareas por segundo de media. Esto es modelado como la tasa de disparos de la transición t_i^{ω}.

El tiempo de ejecución de la tarea T_i (WCET) en ciclos, es nombrado como cc_i. Este es modelado como el peso del arco que une la transición t_i^{ω} con el lugar p_i^{cc} y cómo el marcado inicial del lugar p_i^{cc}.

El plazo de la tarea T_i, es nombrado cómo d_i. Este es modelado como el marcado inicial del lugar p_i^{d}.

Como se asume que los plazos son implícitos, se utiliza ω_i en vez de d_i para los cómputos.

3.1.2 *Modelado de CPUs*

El modelo de tareas se muestra en la Fig. 2.

La transiciones $t_{i,j}^{alloc}$ modelan la asignación de los trabajos de la tarea T_i al procesador CPU_j. Su ratio de disparo $\lambda_{i,j}^{alloc}$, es el ratio de asignación de la tarea al procesador.

El lugar $p_{i,j}^{busy}$ representa el estado ocupado del procesador CPU_j, por un trabajo de una tarea T_i.

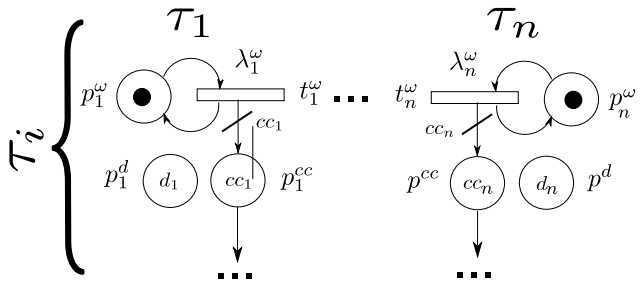


Figura 1: Modelo de tareas. Tomado de la Fig. 2.1 de [12]

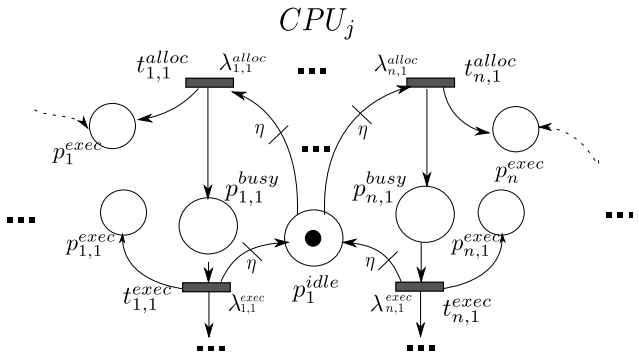


Figura 2: Modelo de procesadores. Tomado de la Fig. 2.2 de [12]

La transición $t_{i,j}^{exec}$ representa la ejecución de la tarea T_i al procesador CPU_j. El modelo considera que las tareas comienzan a ejecutarse inmediatamente después de asignarse. Por razones de simulación, $\lambda_{i,j}^{exec}$ puede ser definida como $\lambda_{i,j}^{exec} = \eta * F$, donde F representa la frecuencia de la CPU_j y η es un parámetro de modelado que garantiza que p_j^{idle} restringe a la transición $t_{i,j}^{alloc}$ (un valor suficiente es η >10). El ratio de disparo $\lambda_{i,j}^{alloc}$ puede ser considerado como $\lambda_{i,j}^{alloc} = \eta * \lambda_{i,j}^{exec}$. De esta forma la transición $t_{i,j}^{alloc}$ no limitará la activación de la transición $t_{i,j}^{exec}$.

El peso η de los arcos de la transición t_{exec} al lugar p_j^{idle} y del lugar p_j^{idle} a la transición $t_{i,j}^{alloc}$ se elige arbitrariamente grande, para limitar el flujo en la transición $t_{i,j}^{alloc}$ según la capacidad del procesador modelado por el lugar p_j^{idle}, asegurando así una configuración única de la TCPN (Sec. 2.1).

3.1.3 *Modelado del comportamiento térmico*

El modelo del comportamiento térmico se muestra en la Fig. 3.

Se utiliza el modelo térmico presentado en [8] donde se divide un multiprocesador en componentes sólidos prismáticos con capacidad de generación de calor, conducción de calor y convección de calor. Pudiendo obtenerse así una ecuación general de estado, y una representación del espacio de estados del sistema.

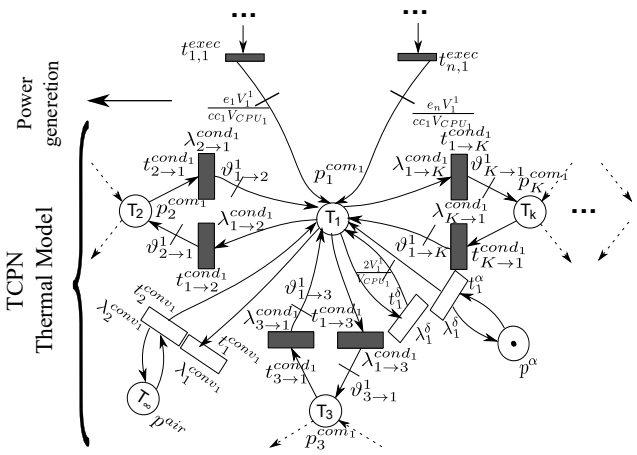


Figura 3: Modelo térmico. Tomado de la Fig. 2.5 de [12]

Por otro lado, en este modelo la energía consumida por un procesador (igual al calor generado) tiene dos componentes, la energía de fuga y la energía dinámica. La energía dinámica depende directamente de la ejecución de los trabajos, y la energía de fuga puede ser modelada como una función lineal de la temperatura para un rango de estas [23].

La generación de calor solamente se aplica a los prismas que representan los núcleos del procesador, al ser los que lo generan, y la convección solamente se aplica a los prismas que representan la capa superior de la placa, al ser los que están en contacto con el aire.

3.1.3.1 *Conducción térmica*

La conducción térmica entre dos prismas adyacentes (p_1^{com} y p_2^{com}) es modelada acorde a la ecuación que se encuentra en la primera fila de la Tab. 5.

Estos prismas se asume que tienen propiedades isotrópicas. Es decir, los coeficientes de conductividad k_1 y k_2, los volúmenes V_1 y V_2, densidades ρ_1 y ρ_2 y las capacidades de calor específico cp_1 y cp_2 son iguales en todas las direcciones.

El marcado de los lugares (p_1^{com} y p_2^{com}) representa la temperatura media en estos prismas, respectivamente.

3.1.3.2 *Convección térmica*

La convección de un prisma (p_1^{com}) es modelado acorde a la ecuación que se encuentra en la segunda fila de la Tab. 5.

El coeficiente h indica el coeficiente de convección del prisma.

El marcado del lugar (p_1^{com}) representa la temperatura media en el prisma.

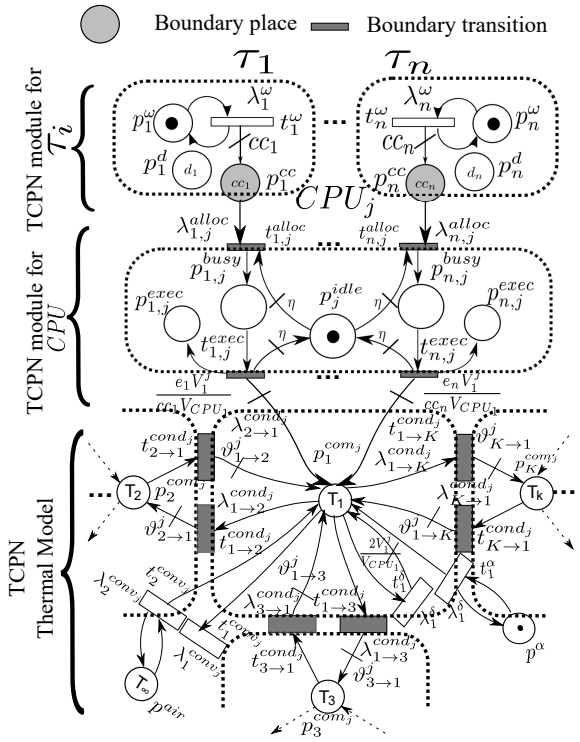


Figura 4: Modelo integral del sistema. Tomado de la Fig. 2 de [10]

3.1.3.3 *Generación de calor debido a la energía de fuga*

La generación de calor debido a la energía de fuga de un prisma (p_1^{com}) es modelada acorde a la ecuación que se encuentra en la segunda fila de la Tab. 6.

Los coeficientes δ y α dependen de la tecnología de fabricación y del tipo de subcircuito (componentes lógicos, memoria SRAM).

La técnica de modelo utilizada permite establecer valores específicos para cada uno de los prismas, teniendo un esquema más preciso si es necesario. Unos valores razonables para estas variables son δ = 0.1 y α = 0.001 [1].

El marcado del lugar (p_1^{com}) representa la temperatura media en el prisma.

3.1.3.4 *Generación de calor debido a la energía dinámica*

La generación de calor debido a la energía dinámica de un prisma (p_1^{com}) es modelada acorde a la ecuación que se encuentra en la primera fila de la Tab. 6.

Los prismas que representan los núcleos, aumentan su temperatura debido a la ejecución de trabajos.

El marcado del lugar (p_1^{com}) representa la temperatura media en el prisma.

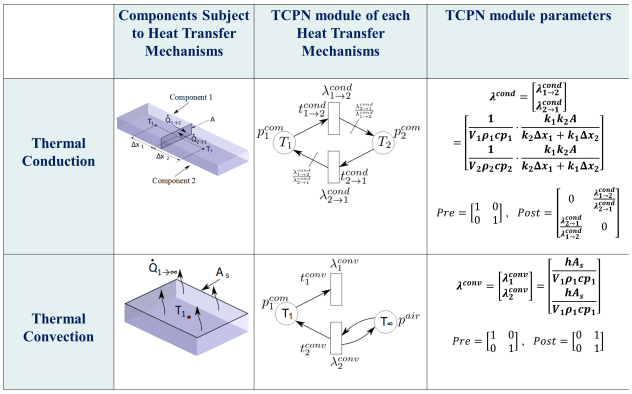


Figura 5: Mecanismos de conducción y convección y sus modelos de TCPN. Tomado de la Fig. 3 de [10]

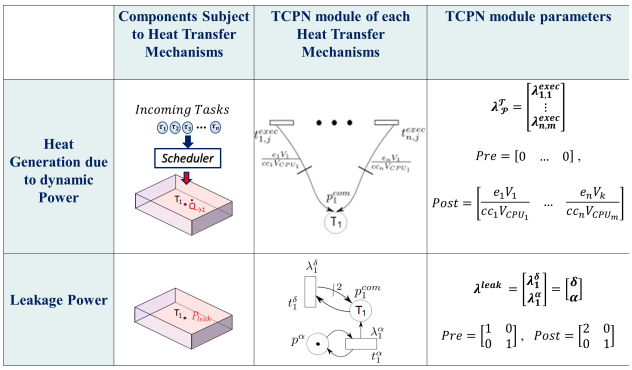


Figura 6: Mecanismos de generación de calor y sus modelos de TCPN. Tomado de la Fig. 4 de [10]

3.1.4 *Modelo integral del sistema*

El modelo integral del sistema se muestra en la Fig. 4.

Para crearlo, se combinan todos los lugares y transiciones compartidos de los 3 sub-modelos.

3.2 MODIFICACIONES EN LA ESTRUCTURA DEL MODELO

Como se ha visto en la Sec. 3.1 la TCPN que modela el sistema completo (tareas, CPUs y dinámica térmica) se genera y simula en el entorno original de manera monolítica, integrando tres sub-modelos: tareas, procesadores, y comportamiento térmico. Este modelo monolítico lo dividimos, y en lo que sigue denotaremos como $TCPN^*$ la TCPN que modela el conjunto *tareas-procesadores* (Fig. 7 para tareas periódicas, Fig. 8 para las aperiódicas), y como $TCPN^T$ la TCPN que modela el comportamiento térmico del multiprocesador (Fig. 9).

La separación es posible porque $TCPN^*$ no recibe retroalimentación del $TCPN^T$. Por ello, modificamos la red $TCPN^T$ añadiendo a la misma una copia de las transiciones t^{exec} de $TCPN^*$. Ambos conjuntos

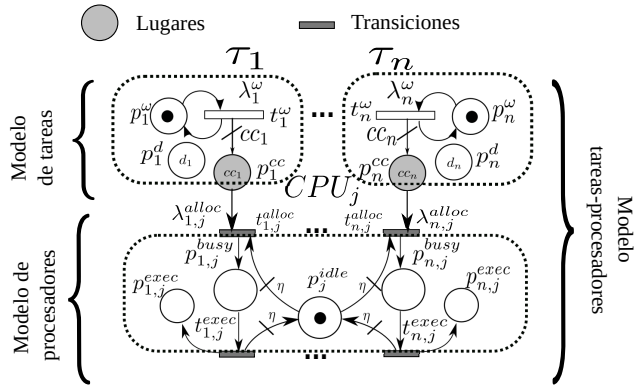


Figura 7: Sub-modelo *TCPN** (tareas-procesadores). Basada en la Fig. 2 de [10]

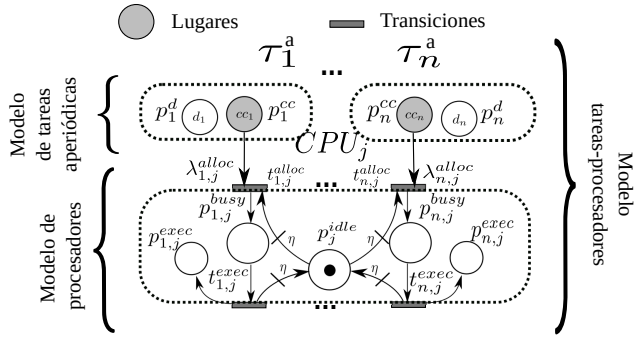


Figura 8: Sub-modelo *TCPN** (tareas-procesadores) para tareas aperiódicas. Basada en la Fig. 2 de [10]

de transiciones, originales y copia, han de realizar los mismos disparos en ambos modelos para que el $TCPN^T$ que las incorpora se comporte igual que en el modelo monolítico de partida. Para conseguir que estas transiciones realicen el mismo número de disparos, se añaden al $TCPN^T$ réplicas de los lugares p^{busy} y se hace que la evolución en el marcado de estos sea igual a la correspondientes del $TCPN^*$, como se explicará en la Sec. 3.2.1.

La separación presenta las siguientes ventajas:

- Permite una simulación mucho más eficiente, como mostraremos experimentalmente en la Sec. 3.7. Esta mejora proviene de dos factores:
  - La reducción del número de operaciones. La aplicación de técnicas de control sobre el modelo monolítico generado en el entorno de partida suponen actuar sobre todas las transiciones. Sin embargo, tras la división solamente se ha de aplicar control sobre las transiciones de $TCPN^*$, mientras que en el modelo $TCPN^T$ se puede aplicar el control directamente sobre el marcado m_0.
  - La reducción de los tamaños de los grafos. La estructura de cada una de las matrices *Pre* y *Post* pasan de tener un

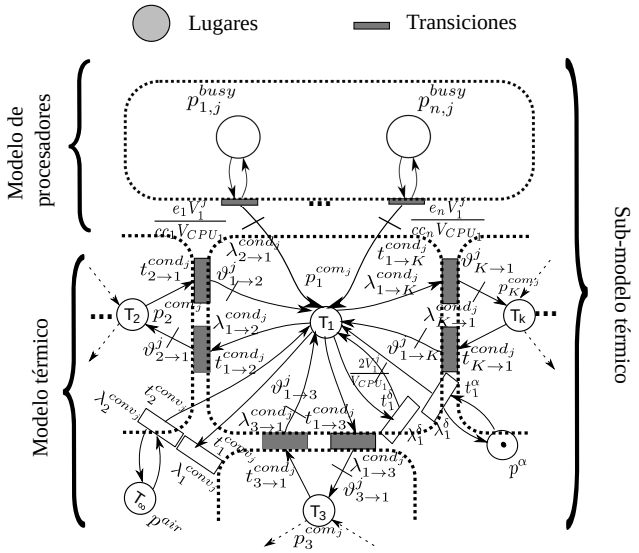


Figura 9: Sub-modelo $TCPN^T$ (térmico). Basada en la Fig. 2 de [10]

tamaño $(|P_{TCPN^*}| + |P_{TCPN^T}|) * (|T_{TCPN^*}| + |T_{TCPN^T}|)$ cada una, a un tamaño conjunto igual a $|P_{TCPN^*}| * |T_{TCPN^*}| + (|P_{TCPN^T}| + m * n) * (|T_{TCPN^T}| + m * n)$, siendo m y n respectivamente el número de procesadores y el de tareas. Los vectores m_0 y λ aumentan en tamaño, pero solamente en $m * n$ unidades. Además esto ahorra una multiplicación de matrices y una suma, ambas operaciones muy costosas debido a las dimensiones de las matrices.

- Facilita simular sólo $TCPN^*$ para guiar la planificación fluida *on-line* como en [10, 12], recurriendo al modelo integral sólo para la resolución *off-line*.

- Permite una implementación modular. De este modo, los datos de la actividad térmica pueden ser obtenidos a partir de otros modelos térmicos diferentes (basados en elementos finitos, por ejemplo).

3.2.1 *Evolución autónoma de* TCPN^T

Definimos como *paso de simulación* a la unidad mínima de avance (discretización) del tiempo en el simulador.

En el modelo $TCPN^*$, si tenemos un η suficientemente elevado, el marcado en p^{busy} es constante (con un valor $\frac{1}{\eta}$) mientras la tarea esté asignada, debido a que el parámetro λ de las transiciones t^{alloc} es proporcional a η^2 y el de las transiciones t^{exec} a η.

La tarea permanecerá asignada al mismo procesador durante todo el *paso de simulación* siempre que el marcado en p^{cc} sea superior al

tamaño del propio *paso de simulación*, al comienzo de la simulación del dicho paso.

Esto nos permite (si asumimos que siempre es superior) separar la evolución de $TCPN^T$ de la de $TCPN^*$, si mientras la tarea está asignada, el marcado en p^{busy} de $TCPN^T$ es $\frac{1}{\eta}$.

Con ese fin se añade una transición entre t^{exec} y p^{busy} en $TCPN^T$ que permita que un marcado que se establezca en ese lugar se conserve. Como la ejecución de tareas se controla en $TCPN^*$ mediante la aplicación de un control sobre t^{alloc}, cuando se aplique este, se establecerá el marcado $\frac{1}{\eta}$ en p^{busy} de $TCPN^T$, y en el momento que se deje de aplicar se restablecerá el marcado a 0, logrando así el objetivo.

Cuando p^{cc} es inferior al trabajo que se ejecuta durante el *paso de simulación*, se comete un error, ya que se simula que la tarea se está ejecutando más tiempo del que le correspondería en la red $TCPN^T$. Sin embargo este error es despreciable, porque sucede como máximo una vez por período, y además está acotado por el tamaño del *paso de simulación*.

3.3 OPTIMIZACIÓN DE LA RESOLUCIÓN DE LA ECUACIÓN DE ESTADO

En esta sección, se expone la optimización aplicada a la resolución de la ecuación de estado vista en el Cap. 2.1 con el fin de reducir el tiempo de simulación de la TCPN.

3.3.1 *Resolución de la ecuación de estado en el entorno de referencia*

La ecuación de estado, es una ecuación diferencial ordinaria que puede ser resuelta mediante diferentes métodos de integración numérica. Esta se debe resolver en cada *paso de simulación*, integrando sobre el intervalo $[0, s]$, donde s es el tamaño del *paso de simulación*, para obtener tanto las temperaturas como el resultado de la ejecución de las tareas durante ese paso. Para integrar, el método usado en la última versión del *entorno de referencia* de este TFG es el método de Euler, porque permite equilibrar precisión y tiempo de ejecución de la simulación.

En el método de Euler la precisión viene marcada por la fragmentación que se aplique en el intervalo de resolución de la integral, llamado a partir de ahora *fragmentación del paso*. Una *fragmentación del paso* más alta indica mayor precisión y mayor tiempo de ejecución necesario para resolver la integral.

El algoritmo usado en el *entorno de referencia* para aplicar este método, en cada *paso de simulación* es el siguiente:

para $i = 0$ **a** 10 **hacer**
  $m_0 = (A * m_0 + \text{const}_1 * w_{\text{alloc}} + \text{const}_2) * \frac{s}{10} + m_0$
fin

Donde s es el tamaño del *paso de simulación*, A es un invariante que se define como $A = C * \Lambda * \Pi(m)$, y w_{alloc} es el vector que marca la asignación de tareas a procesadores. El bucle va en el rango 0 a 10 porque se asume una *fragmentación del paso* fija de 10 unidades. Esto hace que el tiempo de ejecución en las simulaciones del *entorno de referencia* sea directamente proporcional a la *fragmentación del paso*.

3.3.2 *Modificación de la ecuación de estado*

Para realizar la optimización se modifica en primer lugar la Ec. 3 transformando el término u (control) en un término r tal que $0 <= r <= 1$ y $u[t_j] = f[t_j] * (1 - r)$. La ecuación de estado resultante es:

$$\dot{m} = C * (f(m) - f(m) * (1 - r)) = C * (r * f(m)) \tag{4}$$

Cuando cada transición solo tiene un lugar de entrada (como sucede en el modelo $TCPN^T$) o su disparo está limitado siempre por el mismo lugar de entrada (como sucede en el modelo $TCPN^*$, donde el término η cumple esta función), $\Pi(m)$ será constante, porque solamente existirá una configuración (Sec. 3.1.2).

Además, en el modelo $TCPN^T$ no se aplica control sobre las transiciones, por lo que se puede definir una matriz $A = C * \Lambda * \Pi(m)$ que es invariante. En caso de que se aplique un control *Cont*, haciendo uso de la Ec. 4, se puede definir esta misma matriz como $A = C * Cont * \Lambda * \Pi(m)$.

Tras esto, la resolución del paso número n de una fragmentación de k pasos mediante el método de Euler, si se integra en el intervalo $[0, s]$ sobre la ecuación de estado que hemos modificado (Ec. 4) puede formularse como:

$$m_n = A * \frac{s}{k} * m_{n-1} + m_{n-1} = (A * \frac{s}{k} + I) * m_{n-1} \tag{5}$$

Donde s representa la duración del *paso de simulación*, e I representa la matriz identidad de orden $|A|$, por lo que la suma $A * \frac{s}{k} + I$ es constante también. En esta ecuación, si se toma m_0 como el marcado de la TCPN en un instante cualquiera, m_k correspondería al marcado de la misma TCPN, transcurridos s unidades de tiempo (un *paso de simulación*). Aplicando recursividad sobre esta ecuación, se puede obtener que:

$$m_k = (A * \frac{s}{k} + I)^k * m_0 \tag{6}$$

3.3.3 *Aplicación de la modificación de la ecuación de estado sobre* TCPN^T

La red *TCPN*^{T} nos permite usar la Ec. 6, donde k es la *fragmentación del paso* y s el tamaño del *paso de simulación*. Entonces, $(A * \frac{s}{k} + I)^k$ (denotado A_{MS} a partir de ahora) solamente se tiene que calcular una vez. La complejidad de su cálculo (una exponenciación de matrices) es $O(\log f)$, si se realiza mediante el algoritmo de exponenciación binaria, siendo f la *fragmentación del paso*. Esto permite aumentar esta última, mejorando la precisión en la resolución de la integral considerablemente.

Por otra parte, el coste de cada *paso de simulación* sobre la red *TCPN*^{T} queda reducido a una sola multiplicación y a una modificación del vector m_0. Es decir, el tiempo de cálculo es constante, en lugar de incrementarse linealmente con f como en el *entorno de referencia* (Sec. 3.3.1). Esta diferencia queda de manifiesto experimentalmente en la Sec. 3.7.

3.4 DESACOPLE DEL MODELO TCPN Y DE LA MÁQUINA DE SIMULACIÓN

En el simulador de partida, el avance de la TCPN se lograba resolviendo la ecuación de estado correspondiente al modelo monolítico(Ec. 3, Sec. 3.3.1). Por razones de eficiencia, esta ecuación era modificada y transformada en la suma de tres productos.

El primer producto obtenía el avance de la red sin aplicar control, multiplicando A por m_0. Con el segundo se obtenía los cambios de marcado causados por la asignación de tareas, multiplicando una matriz derivada de la matriz de generación del modelo térmico por un vector que indicaba en qué procesador se ejecutaba cada tarea. El tercer producto calculaba el cambio de marcado debido a la convección, mediante el producto de una matriz derivada de la matriz de convección del modelo térmico.

Esta solución, además de ser más costosa que la solución que proponemos, tenía el inconveniente de precisar un simulador *ad-hoc* para la TCPN concreta que modela el sistema. En cambio, la nueva solución propuesta desacopla totalmente la máquina de simulación de la TCPN. Esto permite cambiar la TCPN que se simula sin modificar el simulador, o bien optimizar el simulador sin modificar la TCPN. Además, permite especificar la TCPN de modo más simple, únicamente mediante las matrices *Pre*, *Post*, Π y λ, obviando matrices auxiliares.

3.5 OPTIMIZACIÓN MEDIANTE DOS MÁQUINAS DE SIMULACIÓN

Dado que ahora partimos de un simulador de TCPNs desacoplado de estas (Sec. 3.4), y se tienen dos sub-modelos (*TCPN** ,*TCPN*^{T}) que pueden ser simulados de forma independiente (Sec. 3.2), se ha podido

mejorar el rendimiento diseñando un simulador optimizado para cada uno.

Los dos simuladores de TCPNs se han creado haciendo uso de la semántica de servidores infinitos, y del método de Euler para resolver la ecuación de estado optimizada previamente (Ec. 6).

La diferencia entre ambos reside en que el simulador que gestiona el modelo de tareas-procesadores ($TCPN^*$) permite la aplicación de un control sobre el marcado, en cambio el que simula el modelo térmico ($TCPN^T$) no tiene que gestionar esta característica. Es decir A (y por tanto A_{MS}) en $TCPN^T$ permanece constante, solo se calcula una vez al crear el simulador, en cambio en $TCPN^*$ se ha de recalcular cada vez que existe un cambio de control (i.e. cuando se realice un cambio de contexto). Al ser bastante pequeño el grafo del modelo $TCPN^*$, la sobrecarga causada por el recálculo de A_{MS} en cada cambio de control es inapreciable.

3.6 OPTIMIZACIÓN DE LOS REQUERIMIENTOS DE MEMORIA

Los dos simuladores específicos para cada sub-modelo ($TCPN^*$, $TCPN^T$) están diseñados para guardar el menor número de variables de estado posible, minimizando la ocupación en memoria y disminuyendo el tiempo de ejecución. Esto es especialmente importante en el caso del modelo $TCPN^T$ debido al tamaño del grafo. En ambos se consigue realizar la simulación almacenando solamente las matrices A_{MS}, Pre, λ, C y Π.

3.6.1 *Almacenamiento del modelo en memoria*

Se ha optado por representar parte de ambos modelos mediante matrices dispersas a fin de minimizar su ocupación de memoria, un serio problema en el entorno de simulación previo.

El espacio ocupado por una matriz dispersa es proporcional al número de componentes distinto de cero, mientras que el ocupado por una matriz densa es proporcional a su número total de componentes.

Las matrices Pre, Post y Π se representan como matrices dispersas de listas (LIL) durante su definición. Una vez creadas, estas son transformadas a matrices dispersas comprimida por filas (CSR) excepto en el caso de A_{MS}, que se almacena siempre como una matriz densa. Este criterio obedece a los resultados experimentales mostrados en la Sec. 3.7.

La disposición en memoria de los modelos se ha elegido cuidadosamente de tal forma que concatenando matrices identidades, matrices de 1s y matrices de 0s, se pudiesen formar tanto *Pre* como *Post*, sea en el modelo $TCPN^*$ como en el $TCPN^T$. Esto reduce drásticamente el tiempo de generación automática del modelo.

Cuadro 2: Análisis de densidad de las matrices

Pre	Post	Π	A	A_{MS}
0.000369	0.000391	0.000369	0.002307	0.997047

3.7 ESTUDIO DE OPCIONES DE REPRESENTACIÓN DE MATRICES Y DE RESOLUCIÓN DE LA ECUACIÓN DE ESTADO

En esta sección se evalúan alternativas de representación de las matrices, y dos métodos para la resolución de la ecuación de estado (*resolución optimizada de la ecuación de estado*, Sec. 3.3.2, y *resolución iterativa de la ecuación de estado*, Sec. 3.3.1).

Durante el experimento se ha utilizado el simulador de TCPNs desacoplado de estas (Sec. 3.4) y la división del modelo monolítico en dos sub-modelos (Sec. 3.2). Se compara solo el rendimiento del modelo $TCPN^T$, porque es el que condiciona tanto el tiempo de ejecución, como la ocupación de memoria de la simulación debido a su gran tamaño.

3.7.1 *Condiciones del experimento*

Durante el experimento se ha realizado una simulación del modelo $TCPN^T$ representado y simulado de diferentes formas para evaluar la más conveniente. El experimento se ha llevado a cabo especificando un procesador con placa de tamaño 50 mm x 50 mm x 1 mm, 2 núcleos de tamaño 10 mm x 10 mm x 2 mm, con un paso de malla de 1 mm, 3 tareas, una *fragmentación del paso* de 100 unidades y 2000 *pasos de simulación*. Esta especificación es suficientemente representativa por tiempo de ejecución y ocupación de memoria.

Para mejorar la fiabilidad de los resultados solamente se ha simulado la evolución del modelo térmico en un ambiente en el cual el planificador asigna siempre las mismas tareas a los mismos procesadores (el algoritmo usado se encuentra en el Anexo. 7.5), ya que el cambio de las leyes de control no afectan en el rendimiento del modelo $TCPN^T$.

3.7.2 *Caracterización de la densidad de las matrices*

La densidad de una matriz se define como la tasa de lugares no nulos respecto a los lugares totales. Se ha realizado en primer lugar un análisis de densidad de las matrices Pre, Post, Π, A y A_{MS}, cuyos resultados figuran en la Tab. 2.

Cuando la densidad de la matriz es inferior a 0,5 en las implementaciones más eficientes, y 0,3 en las menos eficientes, se considera que la representación como matriz dispersa supone un ahorro significativo de memoria. Las matrices Pre, Post, Π presentan densidades inferiores

a 0,3, por lo que considerarlas matrices dispersas supone un ahorro de memoria. Por el contrario, A_{MS} presenta una alta densidad, por lo que una implementación como matriz densa es más adecuada. Esto quedará patente posteriormente en el experimento.

3.7.3 *Tipos de matrices comparados*

Los tipos de matrices comparados son los siguientes:

Las implementaciones de matrices dispersas en Python pueden ser consultadas en [36].

- Matriz densa: Es el tipo de matriz usada previamente. Ocupa en memoria el mísmo número de elementos que posee.
- CSR: Matriz dispersa comprimida por filas. Está compuesta por tres vectores. El primero almacena los datos no nulos de la matriz introducidos por filas. El segundo almacena los índices de las columnas de estos datos. El tercero almacena los índices del primer elemento no nulo de cada fila.
- BSR: Matriz dispersa de bloques. Formato similar a CSR, en el cual en vez de guardarse elementos, se almacenan bloques densos de elementos.
- COO: Matriz dispersa en formato de coordenadas. Está compuesta por tres vectores. El primero almacena los datos no nulos de la matriz. El segundo almacena los índices de las filas de cada uno de estos datos. El tercero almacena los índices de las columnas de cada uno de los datos.
- CSC: Matriz dispersa comprimida por columnas. Está compuesta por tres vectores. El primero almacena los datos no nulos de la matriz introducidos por columnas. El segundo almacena los índices de las filas de cada uno de estos datos. El tercero almacena los índices del primer elemento no nulo de cada columna.
- DIA: Matriz dispersa de diagonales. Está compuesta por una matriz que almacena las diagonales con elementos no nulos, y por un vector que almacena los índices de las diagonales correspondientes a cada una de las columnas de la matriz anterior.
- DOK: Matriz dispersa de diccionario de claves. Se almacena un diccionario donde los valores son los elementos no nulos, y las claves son la fila y la columna de cada uno de estos.
- LIL: Matriz dispersa de listas. Se almacenan dos vectores. En el primero de ellos los elementos no nulos de la matriz, y en el segundo las coordenadas de estos elementos.

Cuadro 3: Resultados comparativos por tipo de matriz. Tamaño en memoria de las matrices imprescindibles para simular el modelo $TCPN^T$

Tipo de matriz	Tamaño en memoria (GB)				
	Pre	Pi	C	A	A_{MS}
Array C	2.75E-01	2.75E-01	2.75E-01	5.46E-02	5.46E-02
CSR	1.62E-04	2.03E-04	3.21E-04	1.99E-04	8.17E-02
BSR	1.62E-04	2.03E-04	3.21E-04	1.99E-04	8.17E-02
COO	2.03E-04	2.03E-04	4.15E-04	1.99E-04	8.17E-02
CSC	1.62E-04	2.03E-04	3.21E-04	1.99E-04	8.17E-02
DIA	1.62E-04	2.03E-04	3.21E-04	1.99E-04	8.17E-02
DOK	1.62E-04	2.03E-04	3.21E-04	1.99E-04	8.17E-02
LIL	1.62E-04	2.03E-04	3.21E-04	1.99E-04	8.17E-02

Cuadro 4: Resultados comparativos por tipo de matriz. Tiempo y memoria consumidos en la simulación del modelo $TCPN^T$ mediante la resolución iterativa de la ecuación de estado

	Tiempo de creación del simulador de TCPN (s)	Tiempo de simulación de 2000 pasos (s)	Tiempo total (s)	Memoria total (GB)
Array C	6.639	1735.322	1741.961	0.879520
CSR	1.941	71.715	73.655	0.000886
BSR	1.944	73.035	74.979	0.000886
COO	1.942	71.451	73.392	0.001020
CSC	1.941	71.534	73.475	0.000886
DIA	1.941	71.502	73.443	0.000886
DOK	1.942	71.582	73.522	0.000886
LIL	1.941	71.565	73.506	0.000886

3.7.4 *Resultados experimentales*

Los resultados obtenidos sobre el equipo descrito en el Anexo. 7.2 se recogen en las Tabs. 3, 4 y 5.

Cuando se representa $TCPN^T$ mediante matrices densas, la *resolución optimizada de la ecuación de estado* produce un ahorro de un factor x53,16 en el tiempo de ejecución respecto a la *resolución iterativa de la ecuación de estado*. En el caso de la representación mediante matrices dispersas, este tiempo de ejecución siempre empeora, aumentando en un factor x3,45 en el caso de los tipos CSR, COO, CSC, DIA, DOK y LIL, y en un factor x4,12 en el caso del tipo BSR.

Las representaciones mediante matrices dispersas asociadas a ejecuciones más rápidas son las producidas por los tipos CSR, COO, CSC, DIA, DOK y LIL mediante la *resolución iterativa de la ecuación de estado*. Comparando con la representación mediante matrices densas que utiliza la *resolución optimizada de la ecuación de estado*, esta última produce un

Cuadro 5: Resultados comparativos por tipo de matriz. Tiempo y memoria consumidos en la simulación del modelo $TCPN^T$ mediante la resolución optimizada de la ecuación de estado

	Tiempo de creación del simulador de TCPN (s)	Tiempo de simulación de 2000 pasos (s)	Tiempo total (s)	Memoria total (GB)
Array C	15.578	17.192	32.771	0.93416
CSR	95.801	161.062	256.862	0.08261
BSR	95.994	213.352	309.346	0.08261
COO	95.828	160.919	256.747	0.08274
CSC	95.661	161.041	256.701	0.08261
DIA	95.716	160.967	256.683	0.08261
DOK	95.727	160.997	256.724	0.08261
LIL	95.684	161.006	256.69	0.08261

Cuadro 6: Tiempo y memoria consumidos en la simulación del modelo $TCPN^T$ con representación optimizada (CSR y matriz densa) mediante la resolución optimizada de la ecuación de estado

	Tiempo de creación del simulador de TCPN (s)	Tiempo de simulación de 2000 pasos (s)	Tiempo total (s)	Memoria total (GB)
Array C - CSR	11.626	17.478	29.104	0.10996

speed-up de x2,24 en el tiempo de ejecución respecto a las primeras. En cambio, las primeras disminuyen la ocupación de memoria dinámica por un factor de x1054,35. Estas representaciones mediante matrices dispersas suponen un ahorro de x992,68 respecto a la configuración de partida (representación mediante matrices densas que utiliza la *resolución iterativa de la ecuación de estado*).

A la vista de estos resultados parece razonable utilizar únicamente matrices dispersas debido a su ahorro de memoria dinámica.

Sin embargo, penalizan el tiempo de ejecución. Por ello se ha diseñado otro experimento utilizando ambos tipos de matrices (CSR y matrices densas, además de usar la *resolución optimizada de la ecuación de estado*). Las matrices *Pre*, Π y C se almacenan como matrices dispersas, mientras que A se calcula inicialmente como una matriz dispersa, y es transformada después a una matriz densa antes de proceder al cálculo de A_{MS}, que queda así almacenada en forma de matriz densa. Almacenar A_{MS} en forma de matriz densa aligera los cálculos de evolución del sistema, con impacto notable en el tiempo de ejecución, mientras que almacenar *Pre*, Π y C (menos utilizadas) como matrices dispersas supone un considerable ahorro en memoria.

Los resultados experimentales (Tab. 6) confirman los beneficios de esta opción. La ocupación de memoria disminuye en un factor x8 y

se disminuye el tiempo de ejecución en un factor $x59{,}85$ respecto a la configuración de partida.

Para configurar la representación de las matrices se priorizó el tiempo de ejecución respecto al consumo de memoria dinámica, ya reducida en un factor que consideramos suficiente ($x8$). Si es preciso, este criterio puede cambiarse sin afectar al resto del entorno, debido al desacople de la máquina de simulación (Sec. 3.4).

3.8 COMPARATIVA DEL NUEVO ENTORNO Y DEL ENTORNO ANTERIOR

En esta sección se compara el rendimiento de las distintas versiones del simulador. En la comparación se usa el entorno completo para simular la ejecución del planificador G-EDF en un experimento con las mismas características de procesador y tareas mostradas en la Sec. 3.7.1. Sin embargo, los períodos de las tareas hacen que el número de pasos de simulación alcance los 2400.

Por otro lado, se comparan dos escenarios. El primero utiliza un paso de malla de 2 mm (precisión baja), y el segundo un paso de malla de 1 mm (precisión media). No se ha podido realizar la comparativa con un paso de malla de 0.5 mm (precisión media-alta) debido a que esta configuración sólo podía ejecutarse utilizando Python v2.0 sin agotar la memoria del equipo de pruebas (Anexo. 7.2).

3.8.1 *Versiones del simulador comparadas*

Se han comparado las siguientes versiones del simulador:

- MATLAB v2.3: Versión inicial del entorno de simulación. Hace uso del método de Dormand-Prince [14] para resolver la ecuación de estado de la TCPN.
- MATLAB v3.0: Versión del entorno de simulación desarrollada por el grupo de investigación en paralelo a la realización de este TFG. Hace uso del método de Euler para resolver la ecuación de estado de la TCPN. La *fragmentación del paso* usada es de 10 unidades.
- Python v1.0: Versión del entorno de simulación desarrollada en el marco de la asignatura Laboratorio de Empotrados como preparación previa para este TFG. Hace uso del método de Dormand-Prince para resolver la ecuación de estado de la TCPN.
- Python v2.0: Versión del entorno de simulación desarrollada íntegramente en este TFG. Hace uso del método de Euler para resolver la ecuación de estado de la TCPN y de las optimizaciones mostradas en este capítulo. La *fragmentación del paso* usada es

Cuadro 7: Resultados comparativos de uso de memoria entre las diferentes versiones del entorno de simulación (menor mejor) y el factor de ahorro logrado en la versión Python v2.0

	Uso memoria máximo (KB)		
	Interfaz y entorno	Malla precisión baja	Malla precisión media
MATLAB v2.3	904524 (x6.3)	769400 (x20.6)	2528096 (x8.4)
MATLAB v3.0	1041000 (x7.3)	255764 (x6.9)	1609016 (x5.4)
Python v1.0	100852 (x0.7)	107932 (x2.9)	1238620 (x4.1)
Python v2.0	142752 (x1.0)	37340 (x1.0)	299872 (x1.0)

Cuadro 8: Resultados comparativos de tiempo de ejecución entre las diferentes versiones del entorno de simulación (menor mejor) y el factor de ahorro logrado en la versión Python v2.0

	Tiempo de ejecución (segundos)	
	Malla precisión baja	Malla precisión media
MATLAB v2.3	143.54 (x63.0)	1566.611 (x45.0)
MATLAB v3.0	11.09 (x4.9)	317.371 (x9.1)
Python v1.0	26.03 (x11.4)	980.711 (x28.1)
Python v2.0	2.28 (x1.0)	34.843 (x1.0)

de 10 unidades para poder compararlo con la versión MATLAB v3.0.

El speed-up *puede traducirse como* factor de mejora *y es la relación entre el rendimiento de un sistema respecto a otro de referencia, donde rendimiento puede ser tiempo de ejecución o cualquier métrica de productividad. Se expresa en porcentaje o como factor de multiplicación. En el caso de la ocupación de memoria, le denominamos aquí* factor de ahorro.

3.8.2 *Resultados experimentales*

Los resultados de la experimentación se resumen en las Tabs. 7 y 8. En las mediciones se ha recogido solamente el tiempo consumido en la realización de la simulación, se ha excluido el tiempo de creación de la interfaz y el tiempo de generación de los resultados en forma de gráficos.

La ocupación de memoria se reduce de forma drástica respecto al *entorno de referencia* (Tab. 7), superando una de las limitaciones de este último. En una malla de baja precisión la reducción es de $x6{,}9$, y en una malla de precisión media de $x5{,}4$, respecto a la versión MATLAB v3.0, sin contar la sobrecarga producida por el entorno de MATLAB y la interfaz con respecto a las versiones en MATLAB.

El incremento de rendimiento en la nueva versión también es destacable (Tab. 8), alcanzando un *speed-up* de $x4{,}86$ en una malla de baja precisión y de $x9{,}11$ en una malla de precisión media respecto a la versión MATLAB v3.0. La versión MATLAB v3.0 es la que obtiene un rendimiento más cercano en tiempo de ejecución, debido a la aproximación que se realiza para resolver la ecuación de estado mediante el método de Euler con una *fragmentación del paso* de 10 unidades. Sin

Cuadro 9: Resultados comparativos de tiempo de ejecución con fragmentación del paso de 64 unidades (menor mejor) y el factor de ahorro logrado en la versión Python v2.0

	Tiempo de ejecución (segundos)	
	Malla precisión baja	Malla precisión media
MATLAB v3.0	32.851 (x15.3)	1616.146 (x41.6)
Python v2.0	2.152 (x1.0)	38.883 (x1.0)

embargo hay una mejora cualitativa añadida en el nuevo entorno, al poder establecer una *fragmentación del paso* mayor a fin de incrementar la precisión al resolver la ecuación de estado mediante Euler. Así, si se utiliza una *fragmentación del paso* de 64 unidades (Tab. 9) el incremento de precisión penaliza la versión MATLAB v3.0 respecto a la nueva, que alcanza un *speed-up* de x15,26 en una malla de baja precisión y de x41,56 en una malla de precisión media respecto a la anterior.

4

DISEÑO Y ARQUITECTURA DEL ENTORNO DE SIMULACIÓN

En la Sec. 1.2 exponíamos como objetivo general de este proyecto el diseño de un entorno de simulación orientado a la investigación sobre planificadores TR sensibles a temperatura y energía en multiprocesadores, utilizando técnicas de control, a fin de mejorar la modularidad y el rendimiento de un entorno anterior. En el Cap. 3 se ha expuesto el diseño de las nuevas máquinas de simulación y el rediseño tanto del modelo como de las estructuras de datos. En este capítulo, se describen los componentes de este nuevo entorno, además del rediseño en la simulación de planificadores y en la interfaz de usuario.

4.1 ARQUITECTURA DEL ENTORNO ANTERIOR

El entorno previo presentaba una arquitectura totalmente acoplada (Fig. 10 y Fig. 11) que tenía como punto de apoyo el módulo de interfaz gráfica de usuario.

Este módulo, además de generar la interfaz gráfica, se encargaba de la mayor parte de la lógica del programa, incluyendo la gestión y almacenamiento de los datos introducidos en la entrada (almacenados en un objeto propio), la creación del modelo TCPN integral, la ejecución del planificador y la gestión de resultados, con módulos auxiliares.

Los módulos que implementaban los planificadores también estaban sobrecargados, simulando tanto el planificador como el sistema real mediante el modelo TCPN integral.

4.2 PATRONES EN EL DISEÑO DEL NUEVO ENTORNO

En esta sección se presentan los patrones usados en el diseño del nuevo entorno.

4.2.1 *Paradigmas de programación*

Uno de los principales problemas que se encuentran en la arquitectura del entorno de partida es la excesiva interdependencia de los módulos, que dificulta añadir, substituir o mejorar sus componentes.

Por ello, como primera medida, se optó por utilizar para el nuevo diseño un paradigma de orientación a objetos. De este modo, el entorno se basa en módulos intercambiables, con estructuras similares. Por ejemplo, todos los planificadores implementan una interfaz conocida por el resto de la aplicación, que establece los únicos vínculos

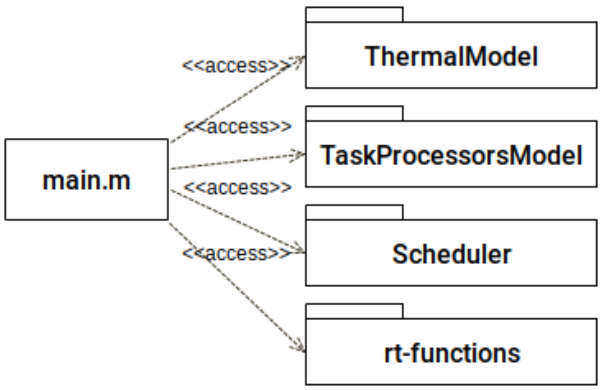


Figura 10: Diagrama arquitectura previa

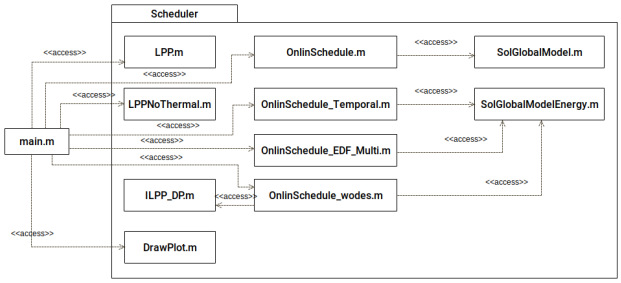


Figura 11: Diagrama arquitectura previa, paquete planificadores

entre ambos. Para mantener la coherencia durante el desarrollo del proyecto, las clases abstractas y sus implementaciones han sido colocadas en directorios separados (directorio `templates` para las abstractas, `implementation` para las implementaciones).

Se recurre también al uso de *high order functions* de Python (e.g. `map`), porque aceleran la ejecución del código y lo hace más legible [32].

Las Python Enhancement Proposals *(PEP) describen características o nuevas funcionalidades para la mejora del lenguaje Python y aportan información sobre su uso.*

4.2.2 *Codificación*

El proyecto se atiene a la propuesta PEP8 de codificación en Python [15], excepto en reglas que no se han considerado adecuadas. Se ha usado tipado durante todo el proyecto aunque Python no lo requiere, ya que facilita la detección de errores de ejecución.

También se ha documentado completamente el proyecto con instrucciones de uso de cada una de las clases en el propio código. Código y documentación se ha escrito en inglés para internacionalizar su uso.

4.3 ARQUITECTURA DEL NUEVO ENTORNO

En esta sección se resume la estructura interna del entorno de simulación. La descripción formal de sus clases se encuentra en el

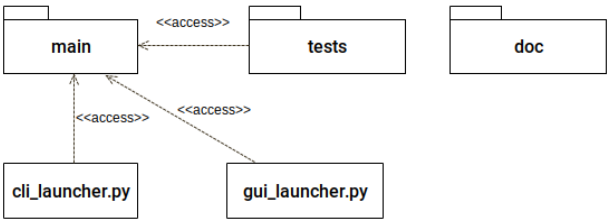


Figura 12: Diagrama general

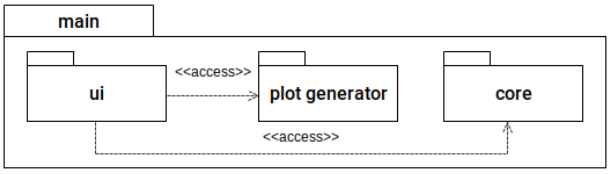


Figura 13: Diagrama de la aplicación

Anexo. 7.1. Los nombres de las clases describen bien su contenido, y la propia codificación es autoexplicativa.

El código del entorno está estructurado en tres directorios y dos ficheros (Fig. 12) como es habitual en Python. Los directorios contienen respectivamente el paquete principal de la aplicación desarrollada (`main`), las pruebas realizadas a esta (`tests`), y parte de la documentación de la aplicación y pruebas de rendimiento que no se pueden especificar en el propio fuente (`doc`). Los dos ficheros son los responsables de ejecutar la aplicación, bien en modo gráfico (`gui_launcher.py`) o por línea de comandos (`cli_launcher.py`).

El paquete de aplicación se estructura en tres paquetes (Fig. 13), interacción con el usuario (`ui`), generación de gráficos (`plot_generator`) y núcleo de la aplicación (`core`). La generación de gráficos y el núcleo de la aplicación están desacoplados. Su único vínculo de comunicación es el formato en el que el núcleo de la aplicación devuelve los resultados de asignación de tareas y temperaturas del procesador tras una ejecución, que han de ser interpretados por el paquete de generación de gráficos. Por su parte, el paquete de interfaz con el usuario hace uso tanto del núcleo de la aplicación como de la generación de gráficos, aunque el diseño posibilita que las interfaces de comunicación sean mínimas. Esto consigue minimizar el acoplamiento.

El paquete de pruebas se estructura en tres paquetes (Fig. 27 del Anexo. 7.1): pruebas sobre el núcleo de la aplicación (`core`, con la misma estructura que en el paquete principal), ejemplos para probar la interfaz por línea de comandos (`cli`, con las especificaciones en formato JSON) y pruebas de rendimiento realizadas para optimizar el entorno (`performance`).

Las siguientes secciones describen los principales componentes del entorno: núcleo, generación de gráficos e interfaz.

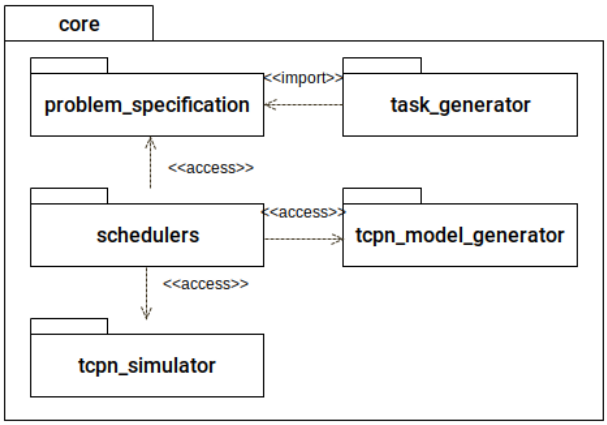


Figura 14: Diagrama del núcleo de la aplicación

4.3.1 *Núcleo*

El núcleo del entorno (Fig. 14) realiza una simulación y devuelve los resultados, a partir de una especificación. Se estructura en los cinco siguientes paquetes.

4.3.1.1 *Especificación del problema y generación automática de tareas*

El paquete `problem_specification` (Fig. 28 del Anexo. 7.1) contiene las clases usadas para especificar los parámetros de la simulación, así como una clase envoltorio (`GlobalSpecification`), que sirve como contenedor para facilitar su uso desde el exterior del paquete.

El paquete de generación automática de tareas (`task_generator`, Fig. 29 del Anexo. 7.1) incluye los métodos de generación automática de tareas que posee el entorno. Este depende del paquete de especificación del problema, concretamente de la clase que representa una tarea.

Actualmente se ha implementado el método UUniFast [3]. Método que genera automáticamente un conjunto de tareas a partir de una utilización dada.

4.3.1.2 *Planificadores*

El paquete de planificadores (`schedulers`, Figs. 30 y 31 del Anexo. 7.1) define dos clases abstractas (Sec. 4.4) y contiene cuatro planificadores ya implementados (los presentados en la Sec. 2.4), además de dos variantes de uno de ellos (G-EDF).

La primera de las clases abstractas (`AbstractBaseScheduler`) sirve como base para implementar los diferentes planificadores sin tener que conocer el resto del entorno. Esta actúa a modo de capa de abstracción, ofreciendo una interfaz de tres funciones, que sirven para describir el comportamiento de los planificadores que la implementen.

En caso de que algún planificador no pueda ser implementado con esta interfaz, deberá de implementar la segunda clase abstracta (`AbstractScheduler`), que requiere conocer el resto de componentes del núcleo de la aplicación.

La clase `AbstractBaseScheduler` hereda de la clase `AbstractScheduler`. De esta forma `AbstractScheduler` es la clase que representa un planificador cualquiera.

4.3.1.3 *Generación y simulación de la TCPN*

El paquete de creación del modelo de TCPN (Fig. 32 del Anexo. 7.1) define clases que especifican las diferentes partes del modelo simulado, descrito en el Cap. 3.1.3. Cada modelo se especifica mediante sus matrices Pre, Post, Π, Λ y m_0. Los sub-modelos $TCPN^*$ y $TCPN^T$ se integran mediante la clase `GlobalModel`.

El paquete de simuladores de TCPN (Fig. 33 del Anexo. 7.1) está compuesto por los diferentes simuladores de TCPN implementados. Todos ellos utilizan un interfaz común que permite intercambiarlos de forma sencilla o crear una optimización derivada de alguno.

4.3.2 *Generación de gráficas*

El paquete de generación de gráficas (Fig. 34 del Anexo. 7.1) está compuesto por los diferentes generadores de resultados, accesibles mediante una interfaz común. Se ha utilizado para implementarlos la biblioteca `Matplotlib` debido a su excelente integración con la biblioteca Scipy [22], usada para el cálculo.

Los resultados gráficos generados por el entorno pueden ser consultadas en el Anexo. 7.4.

4.3.3 *Interfaz de usuario*

El paquete de interacción con el usuario (Fig. 35 del Anexo. 7.1), que se explica con detalle en la Sec. 4.5, está compuesto por los paquetes de interfaz por línea de comandos (`cli`), interfaz gráfica (`gui`) y por un paquete de características comunes (`common`).

4.3.3.1 *Interfaz por línea de comandos*

El paquete de interfaz por línea de comandos (Fig. 36 del Anexo. 7.1) está compuesto por la clase `CliController`, que gestiona este interfaz, y la especificación del fichero de entrada en JSON (Fig. 38 del Anexo. 7.1).

4.3.3.2 *Interfaz gráfica*

El paquete de interfaz gráfica (Fig. 37 del Anexo. 7.1) está compuesto por la clase `GuiController`, que se encarga de gestionar la interfaz gráfica, y por las implementaciones de las diferentes vistas que posee la aplicación.

4.4 ARQUITECTURA DEL SIMULADOR DE PLANIFICADORES

En esta sección se explica el diseño del paquete de simulación de planificadores, centrándose en la clase `AbstractBaseScheduler`, por ser la más relevante.

4.4.1 *Mejoras funcionales del simulador*

En el nuevo entorno de simulación, se ha añadido el soporte para la gestión de tareas aperiódicas, admisión de quantum variable y variación de frecuencia durante la fase *on-line* del planificador. Estas características son específicas de este TFG en relación al entorno de partida. Ya se utilizan en los planificadores incluidos, y pueden ser utilizadas para los que se incorporen en el futuro.

Permitir la gestión de tareas aperiódicas ha supuesto modificar el modelo TCPN creando un nuevo modelo específico de tareas (Fig. 8), integrado en el modelo *TCPN** (Sec. 3.1).

Tanto el quantum variable como la variación de frecuencia cada quantum se controlan desde la clase abstracta `AbstractBaseScheduler`. Para lograr una frecuencia variable cada quantum, además, se ha modificado la representación del modelo TCPN en memoria, a fin de realizar esta operación de forma rápida y eficiente.

4.4.2 *Diseño de la clase (`AbstractBaseScheduler`)*

En la versión de partida del simulador, la implementación de los planificadores estaba totalmente acoplada a la simulación de la TCPN. Esto impedía implementar un nuevo planificador sin tener conocimiento del funcionamiento de esta, además de imposibilitar cambiar la TCPN sin tener que cambiar los planificadores también.

Para solucionarlo se ha implementado una capa de abstracción entre el simulador y los planificadores (clase `AbstractBaseScheduler`). Esta ofrece una interfaz de las tres funciones que los planificadores han de implementar para definir su comportamiento:

- Comportamiento *off-line* del planificador: Tareas que ha de realizar el planificador antes de que el sistema TR comience su ejecución. Para su implementación se ofrece toda la información relativa a las tareas que se ha de procesar, a los procesadores y al modelo térmico usando. El planificador podrá realizar todos

los cálculos necesarios, y guardarse toda la información necesaria para su ejecución. Además tendrá que elegir el quantum y las frecuencia de cada uno de los procesadores durante la simulación.

- Comportamiento *on-line* del planificador: Tareas que ha de realizar el planificador al finalizar cada quantum. Para su implementación se ofrece toda la información relativa a las tareas que aún no han completado su ejecución y la temperatura de los núcleos. El planificador deberá decidir qué tareas se ejecutan el siguiente quantum. Podrá cambiar el tamaño del quantum y la frecuencia de cada uno de los procesadores establecida previamente.
- Tratamiento de aperiódicas: Tareas que ha de realizar el planificador al ser notificado de la activación de una tarea aperiódica. Para su implementación se ofrece toda la información relativa a la tarea aperiódica que ha llegado. El planificador deberá decidir si realizar una re-planificación inmediata, o esperar al vencimiento del quantum.

Todos los planificadores presentados en el Cap. 2.4 han sido implementados usando esta interfaz.

4.5 ARQUITECTURA DE LA INTERFAZ DE USUARIO

En el entorno de partida, la interacción con el usuario se producía a través de una interfaz gráfica interactiva, que no permitía almacenar o recuperar la especificación de las simulaciones, impidiendo automatizarlas.

Por otro lado, esta interfaz se encontraba totalmente integrada con el resto del simulador (Sec. 4.1). De acuerdo con los objetivos planteados (Sec. 1.3) se ha independizado el interfaz del resto del entorno, de modo que ahora es posible utilizar o implementar diferentes interfaces. En concreto, se han desarrollado dos métodos de interacción con el usuario, una interfaz por línea de comandos y una interfaz gráfica. En el Anexo. 7.3 se explica la utilización de estas interfaces.

4.5.1 *Interfaz por línea de comandos*

Esta interfaz permite realizar la simulación utilizando como entrada un fichero de texto que especifica el problema a simular en formato JSON. Esto permite reutilizar especificaciones de las simulaciones y automatizar la ejecución de varias simulaciones mediante un *script*.

Para especificar el formato de la entrada se ha usado JSON Schema [21].

La definición de la entrada mediante este formato permite además:

- Utilizar la especificación del formato de la entrada como documentación de esta, al ser no ambigua.
- Validar la entrada mediante bibliotecas externas haciendo uso de su especificación.

JSON Schema es una notación que permite definir el formato de ficheros JSON, ampliamente usada y con gran soporte, tanto en lenguajes de programación (bibliotecas que permiten validar si un fichero cumple la especificación), como en editores de texto (validación y ayuda en la creación de documentos que cumplan la especificación). Aún se encuentra en desarrollo, aunque tiene aspiraciones de convertirse en un estándar. La versión usada en este TFG ha sido el borrador nº 7 del formato, el último disponible cuando se inició el desarrollo del proyecto.

Debido a la complejidad de la entrada, la especificación se ha dividido en diferentes partes y se ha usado una arquitectura similar a la vista en la especificación del problema (Fig. 38 del Anexo. 7.1) para mantener coherencia con esta, facilitando las modificaciones en ambas.

4.5.2 *Interfaz gráfica de usuario*

La interfaz gráfica desarrollada replica el mismo método de interacción con el usuario usado en el entorno de partida, pero implementado de acuerdo a los objetivos planteados.

En primer lugar se recurrió a la biblioteca TkInter [39], que permite definir interfaces gráficas mediante código en Python. Sin embargo, la definición mediante código del diseño de una interfaz gráfica es complicado, debido a que para ver los resultados hay que ejecutar cada vez el programa. Por ello se decidió reescribir la interfaz haciendo uso de la biblioteca Qt5 [33].

Qt5 posee un diseñador de interfaces. Puede ser instalado como un paquete de Python, y permite realizar el diseño de forma gráfica, viendo el resultado de cualquier modificación interactivamente. Además, permite establecer el método que será llamado para atender ciertos eventos de la interfaz, como pulsar un botón.

Los diseños realizados con esta herramienta se conservan en ficheros XML en el paquete `ui_specification/design` (Fig. 37 del Anexo. 7.1), y después son transformados a código Python haciendo uso de la biblioteca PyQt [30], almacenando el resultado en `ui_specification/implementation` (Fig. 37 del Anexo. 7.1).

Mediante herencia y polimorfismo, se crean clases herederas de los ficheros transformados, y en ellas se sobrescriben los métodos definidos para atender los diferentes eventos de la interfaz. De esta forma se separa la definición de la interfaz de la implementación de su comportamiento, lo que permite realizar cambios visuales en la interfaz sin tener que modificar el código del comportamiento.

La interfaz ha sido diseñada para permitir exportar e importar la especificación del problema en ficheros JSON en el mismo formato que usa la interfaz por línea de comandos. Esto asegura una compatibilidad entre las dos interfaces, además de permitir reutilizar la validación de la entrada y las interfaces creadas para la ejecución de la simulación que usa la interfaz por línea de comandos.

5 ESTUDIO EXPERIMENTAL DE PLANIFICADORES

En este capítulo se utiliza el entorno creado para realizar una comparación experimental entre los diferentes planificadores implementados en el mismo, descritos en el Cap. 2.4. Se analizan sólo los principales resultados, que pueden consultarse completos en el Anexo. 7.6.

5.1 METODOLOGÍA Y ENTORNO EXPERIMENTAL

Los cinco experimentos realizados obedecen al objetivo de subrayar determinadas características de los planificadores. Todos comparten la definición del procesador, del entorno y las características de simulación. Solo se han variado las definiciones de tareas. El comportamiento del simulador es determinista, por lo que las simulaciones se han ejecutado solo una vez. Las características de los elementos invariantes se definen a continuación.

El procesador simulado se basa en el utilizado en los artículos citados en la Sec. 1.2, compuesto por dos CPUs de silicio sobre un disipador de cobre con las características mostradas en la Tab. 10.

Las frecuencias disponibles en el procesador son 1000000 Hz, 850000 Hz, 600000 Hz, 400000 Hz y 150000 Hz. La frecuencia al inicio de ejecución es siempre 1000000 Hz. Se asume temperatura ambiental constante a 45 ºC, con coeficiente de convección $1000 \frac{W}{m^2} K$. El tamaño del paso de malla elegido ha sido de 1 mm, el *paso de simulación* de 0.01 segundos, y la *fragmentación del paso* en ambos modelos de TCPN de 128 unidades.

5.2 RESULTADOS

En esta sección se describen y analizan los resultados de los cinco experimentos realizados.

5.2.1 *Experimento Nº1: Control de la temperatura máxima mediante OLDTFS*

El primer experimento compara los planificadores OLDTFS (global fluido sensible a temperatura) y G-EDF (global no fluido, no sensible a temperatura) sobre el conjunto de tareas TRD descrito en la Tab. 11. El umbral de temperatura para OLDTFS durante en este experimento se ha establecido en 60 ºC.

Por una parte, la Fig. 15 representa la evolución de la temperatura a lo largo del hiperperiodo para cada CPU en ambos planificadores. En (a) se observa que el planificador OLDTFS nunca supera la temperatura máxima en ninguna de las dos CPUs, manteniendo la temperatura

Cuadro 10: Características físicas del procesador

Característica	Placa	Núcleos
Dimensiones	50 mm x 50 mm x 1 mm	10 mm x 10 mm x 2 mm
Densidad (Density)	8933 $\frac{Kg}{cm^3}$	2330 $\frac{Kg}{cm^3}$
Calor específico (Specific heat capacities)	385 $\frac{J}{Kg}K$	712 $\frac{J}{Kg}K$
Conductividad (Thermal conductivity)	400 $\frac{W}{m}{}^oC$	148 $\frac{W}{m}{}^oC$

Cuadro 11: Conjunto de tareas del experimento Nº 1

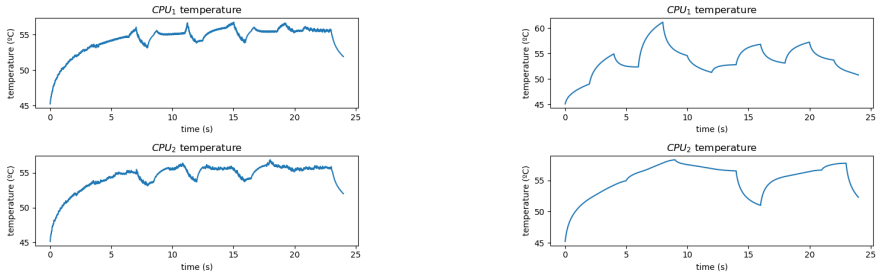
	Tiempo de ejecución (Ciclos)	Periodo (Segundos)	Potencia consumida (Vatios)
T_1	2000000	4	3.4
T_2	5000000	8	8
T_3	6000000	12	9.6
T_4	2000000	24	15.4

en torno a 56 ºC. En (b), G-EDF produce variaciones mayores, con picos de temperatura que alcanzan la temperatura límite (60 ºC). El planificador OLDTFS respeta la restricción térmica, mientras que G-EDF la viola.

Por otra parte, la Fig. 16 ilustra la diferente naturaleza (fluida y no fluida) de ambos planificadores y sus consecuencias. En cada caso se muestra el tiempo de ejecución acumulado de cada una de las cuatro tareas (de izquierda a derecha) sobre cada una de las dos CPUs (arriba /abajo) a lo largo del hiperperiodo. Una rampa significa ejecución activa, y una horizontal la suspensión de la tarea. Como se resumió en la Sec. 2.4.3, en OLDTFS la discretización *on-line* aproxima una ejecución fluida calculada *off-line* consiguiendo una ocupación óptima de ambas CPUs a lo largo de todo el hiperperiodo, además de un control correcto de la temperatura. Por el contrario, en G-EDF la CPU_2 nunca ejecuta ninguna porción de τ_1 ni τ_4, y la CPU_1 nunca ejecuta τ_2, por lo que estas tareas *presionan* más las otras CPUs en cada caso. El control de temperatura es peor (de hecho, es incorrecto). Como contrapartida, OLDTFS genera un gran número de cambios de contexto (transiciones rampa - recta en las gráficas).

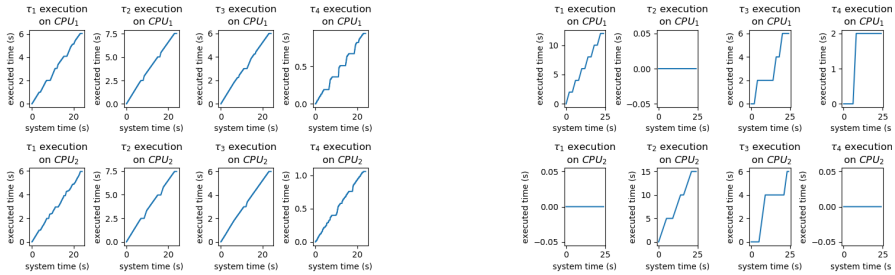
5.2.2 *Experimento Nº2: Control de la temperatura mediante JDEDS*

En este experimento se comparan los planificadores JDEDS (fluido, control de temperatura) y G-LLF (no fluido, sin control de temperatura) sobre el conjunto de tareas TRD descrito en la Tab. 12. El experimento comprueba que el planificador JDEDS reduce la frecuencia para minmizar potencia, maximizar la utilización, y complir la restricción



(a) OLDTFS (b) G-EDF

Figura 15: Temperaturas máximas de los procesadores en el experimento nº1



(a) OLDTFS (b) G-EDF

Figura 16: Ejecución de las tareas en los procesadores en el experimento nº1

térmica. La comparación se realiza con un planificador sin control de temperatura, no óptimo en cuanto a utilización.

La Fig. 17 muestra las frecuencias establecidas por cada planificador a lo largo del hiperperiodo. Dado que no hay aperiódicas en esta prueba, la frecuencia calculada en la fase *off-line* de JDEDS no varía durante la ejecución. Con 850000 Hz, JDEDS logra una utilización próxima a 1, concretamente 0,96 (Fig. 18(a), sin intervalos de desocupación en ninguna CPU salvo ligeramente al final), mientras que G-LLF mantiene desocupada la CPU_2 (abajo) en los intervalos temporales $[6, 8]$, $[10, 12]$ y $[14, 16]$ logrando una utilización de 0,81. Esto repercute en la temperatura (Fig. 19), que se incrementa de forma progresiva y constante en JDEDS.

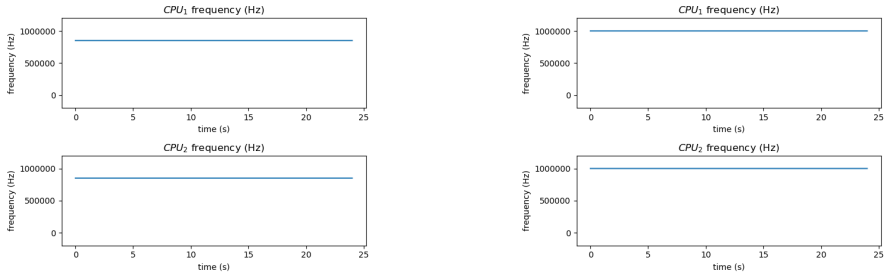
Por otro lado, la Fig. 18 evidencia que G-LLF produce un número de cambios de contexto mayor que JDEDS, debido a que la menor prioridad se asigna alternativamente a τ_2 y τ_3, que compiten en laxidad.

5.2.3 *Experimento Nº3: Estudio de planificabilidad bajo incumplimiento de condición suficiente*

En este experimento se ha elegido un conjunto de tareas (Tab. 13) que ejemplifica la utilidad de simuladores como este para analizar la planificabilidad TRD en casos en los que la teoría no alcanza a asegu-

Cuadro 12: Conjunto de tareas del experimento Nº 2

	Tiempo de ejecución (Ciclos)	Periodo (Segundos)
T_1	2000000	4
T_2	5000000	8
T_3	6000000	12



(a) JDEDS (b) G-LLF

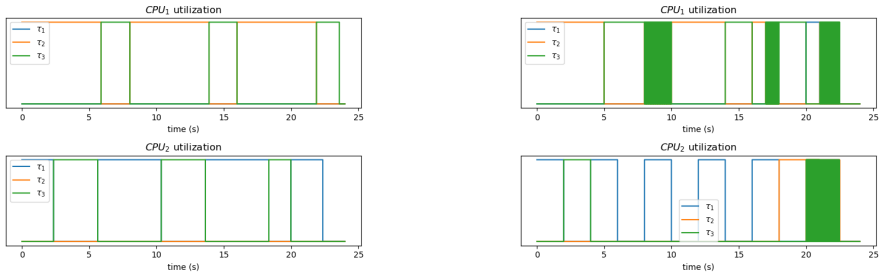
Figura 17: Frecuencia de los procesadores en el experimento nº2

rarla. Se ha desactivado la simulación térmica por no ser relevante en este caso.

En este caso el conjunto incumple el test de densidad de Gossens (*density test*) que para el caso de plazos implícitos establece que un conjunto de tareas TRD es planificable *si* su utilización total U_t es tal que $U_t \leq m - (m-1) \cdot U_m$, donde m es el número de CPUs y U_m la máxima de las utilizaciones del conjunto [5]. Al no ser condición necesaria, el test deja abierta la posibilidad de que exista una planificación viable. Se podría recurrir al método de la *problem window* de Baker [5], pero además de resultar muy costoso podría no ser determinante tampoco, ya que establece otra condición suficiente pero no necesaria.

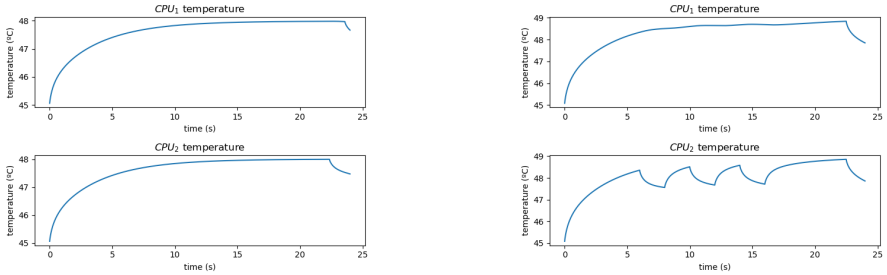
Los resultados de la Fig. 20(a) muestran que el planificador G-EDF incumple los plazos en la tarea τ_3, que no llega a completarse al 100 % en ninguna de sus dos instancias. La Fig. 21(a) permite analizar la causa. En los instantes $t = 0$ s y $t = 16$ s las tareas más prioritarias por el criterio EDF son τ_1 y τ_2, que se ejecutan completamente ocupando las dos CPUs durante 1 s impidiendo que entre τ_3, cuya laxidad en esos instantes es 0,9. La CPU_2 está desocupada en los intervalos $[1, 8]$, $[10, 16]$ y $[17, 24]$, por lo que no es un problema de capacidad de cálculo, sino de fallo del criterio de planificación.

Por el contrario, tanto los planificadores fluidos como OLDTFS y JDEDS como G-LLF realizan en este caso una planificación satisfactoria, como evidencian las Figs. 20(b) y Fig. 21(b) para el caso JDEDS (los resultados completos del experimento pueden consultarse en el Anexo. 7.6.3). Es decir, el experimento demuestra que existe varias planificaciones TRD viables aunque no se cumpla el test de densidad.



(a) JDEDS (b) G-LLF

Figura 18: Utilización de los procesadores en el experimento n°2



(a) JDEDS (b) G-LLF

Figura 19: Temperaturas máximas de los procesadores en el experimento n°2

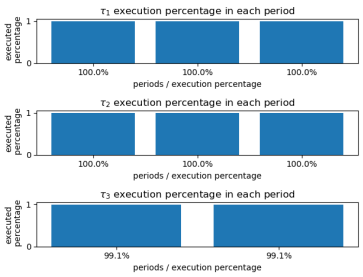
5.2.4 *Experimento N°4: Manejo de aperiódicas de JDEDS*

En el experimento se muestra la capacidad de JDEDS para incrementar la frecuencia del procesador tras aceptar una tarea aperiódica a la mínima de las frecuencias superiores disponibles que permita cumplir todos los plazos, regresando a la frecuencia inicial tras la finalización de la aperiódica. Se utilizan los conjuntos de tareas descritos en la Tab. 14. Ambos se diferencian solamente en el tiempo de ejecución de la aperiódica, de 1000000 ciclos en el primer conjunto y de 2000000 ciclos en el segundo. Las tareas periódicas son TRD y la aperiódica TRB (puede ser rechazada).

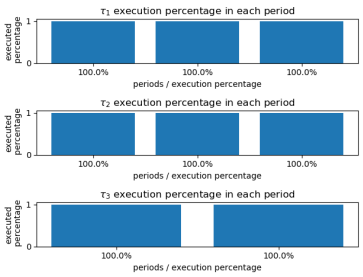
En los resultados de la Fig. 22 se aprecia que JDEDS aumenta la frecuencia en el instante 10 s en ambos casos, al recibir la aperiódica. En el primero, esta se termina de ejecutar en menos intervalos debido a su menor tiempo de ejecución, restituyendo la frecuencia anterior en el instante 16 s. En el segundo caso la reducción se produce en el instante 20 s debido a que su ejecución abarca más intervalos. La Fig. 23 muestra que el ajuste de frecuencia permite mantener la máxima utilización durante todo el hiperperiodo. También se puede comprobar como en los intervalos previos a la llegada de la tarea aperiódica la asignación de tareas en ambos casos es igual, mientras que en los intervalos posteriores esto no sucede. Esto se debe a la

Cuadro 13: Conjunto de tareas del experimento Nº 3

	Tiempo de ejecución (Ciclos)	Periodo (Segundos)
T_1	1000000	8
T_2	1000000	8
T_3	11100000	12



(a) G-EDF



(b) JDEDS

Figura 20: Cumplimiento de plazos en el experimento nº3

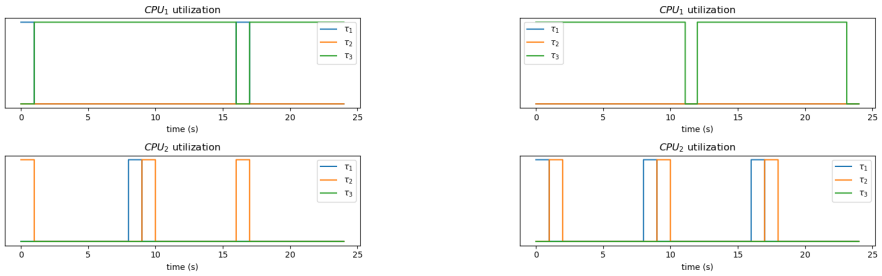
política de minimización de cambios de contexto aplicada en JDEDS, que asigna mayor prioridad a las tareas que ya están en ejecución respecto a las pendientes (salvo que estas últimas alcancen laxitud cero).

5.2.5 *Experimento Nº5: Utilización máxima*

El quinto experimento tiene objetivo comparar los cambios de contexto en los planificadores OLDTFS (Sec. 2.4.3) y JDEDS (Sec. 2.4.4), ambos globales fluidos y sensibles a temperatura. Se obvia la gestión de temperatura, perturbaciones y aperiódicas, no relevantes para lo que se pretende analizar aquí.

El conjunto de tareas TRB elegido (Tab. 15) tiene utilización máxima. Esto último supone el incumplimiento de plazos si cualquier CPU queda ociosa en cualquier intervalo de tiempo, lo que ocurre en este caso con G-EDF y G-LLF (ver resultados en Anexo 7.6.5). Por el contrario, OLDTFS y JDEDS encuentran una planificación factible (Fig. 24), pero su comportamiento es diferente como vamos a estudiar con este experimento.

El número de puntos de planificación *npp* en OLDTFS viene dado por un quantum Q calculado como el MCD del conjunto de todos los plazos y sus múltiplos en el hiperperiodo H (*Partición por plazos*, Sec. 2.3.2.8). Eso determina que el máximo número de cambios de contexto sea $npp = m \cdot H/Q$, donde m es el número de CPUs, es decir, un cambio de contexto por CPU cada Q unidades de tiempo. En JDEDS el número *npp* es directamente el conjunto obtenido por partición por



(a) G-EDF (b) JDEDS

Figura 21: Utilización de los procesadores en el experimento nº3

Cuadro 14: Conjuntos de tareas del experimento Nº 4

	Tiempo de ejecución (Ciclos)	Periodo (Segundos)	Llegada aperiódica (Segundos)	Fin de plazo aperiódica (Segundos)
T_1	2000000	4	-	-
T_2	5000000	8	-	-
T_3	6000000	12	-	-
T_1^a	1000000	-	10	20

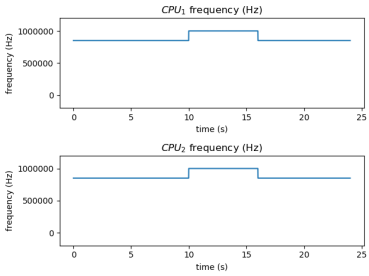
(a)

	Tiempo de ejecución (Ciclos)	Periodo (Segundos)	Llegada aperiódica (Segundos)	Fin de plazo aperiódica (Segundos)
T_1	2000000	4	-	-
T_2	5000000	8	-	-
T_3	6000000	12	-	-
T_1^a	2000000	-	10	20

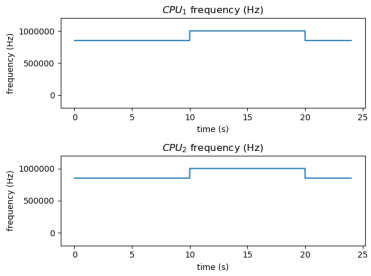
(b)

plazos. Pueden diseñarse conjuntos de tareas en donde *npp* coincida en ambos planificadores, pero en general es menor en JDEDS que en OLDTFS. Con independencia de ello, nuestra implementación de ambos planificadores tiene en cuenta la CPU en la que está en ejecución una tarea para evitar migraciones innecesarias (criterio de afinidad). Para el conjunto de tareas que nos ocupa, la Fig. 25 permite ver que el número de expulsiones (inflexiones rampa - horizontal) es mayor en OLDTFS (a) que en JDEDS (b). Sin embargo, en OLDTFS (a) τ_1 nunca migra a CPU_2, ni τ_2 lo hace a CPU_1, migraciones que sí se dan en JDEDS (b). Es decir, para este conjunto concreto de tareas TRD, OLDTFS genera más cambios de contexto y menos migraciones, frente a JDEDS, que genera menos cambios de contexto y más migraciones.

La Fig. 26 visualiza muy bien el mayor número de cambios de contexto en OLDTFS. La elección de uno u otro planificador vendrá en cada

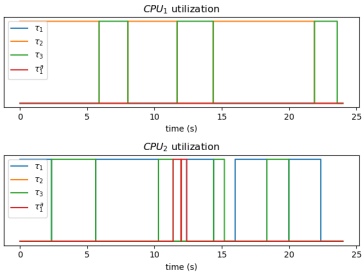


(a) JDEDS, CC_a = 1000000 ciclos

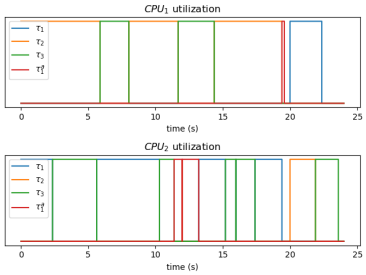


(b) JDEDS, CC_a = 2000000 ciclos

Figura 22: Frecuencia de los procesadores en el experimento nº4



(a) JDEDS, CC_a = 1000000 ciclos



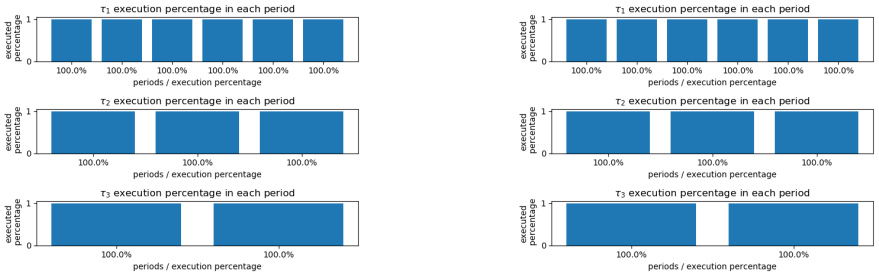
(b) JDEDS, CC_a = 2000000 ciclos

Figura 23: Utilización de los procesadores en el experimento nº4

caso determinada por las necesidades de gestión de perturbaciones o aperiódicas o la minimización de migraciones.

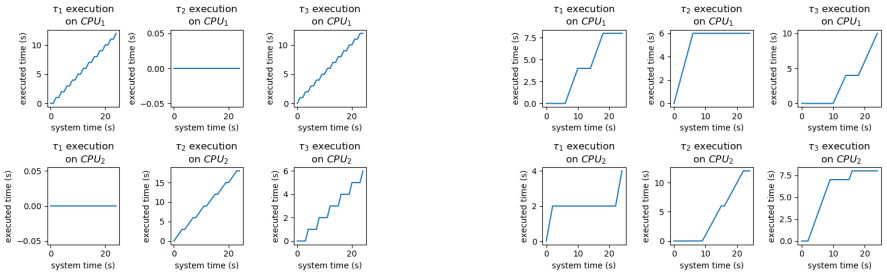
Cuadro 15: Características de las tareas durante el experimento N° 5

	Tiempo de ejecución (Ciclos)	Periodo (Segundos)
T_1	2000000	4
T_2	6000000	8
T_3	9000000	12



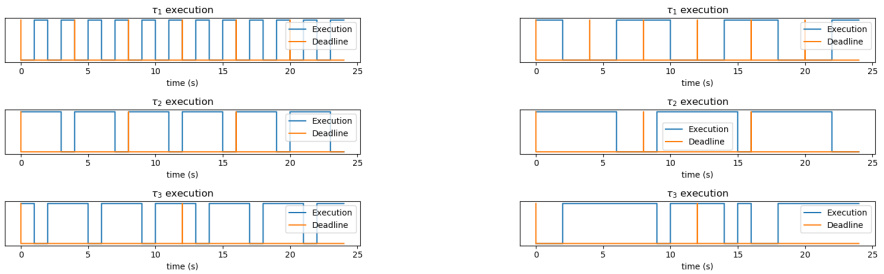
(a) OLDTFS (b) JDEDS

Figura 24: Cumplimiento de plazos en el experimento n°5



(a) OLDTFS (b) JDEDS

Figura 25: Ejecución de las tareas en los procesadores en el experimento n°5



(a) OLDTFS (b) JDEDS

Figura 26: Ejecución de las tareas en el experimento n°5

6 CONCLUSIONES Y TRABAJO FUTURO

6.1 RECAPITULACIÓN

Se ha diseñado un entorno en condiciones de substituir al utilizado hasta ahora en el proyecto en el que se enmarca el TFG (Sec. 1.2). Este entorno se ha diseñado con una arquitectura escalable y con unos métodos de interacción con el usuario que mejoran sustancialmente la usabilidad del anterior.

En el proceso, se han alcanzado mejoras drásticas de rendimiento en términos de tiempo de ejecución y ocupación de memoria, un factor este último que limitaba seriamente el entorno de referencia. Ello ha conllevado algunos cambios en la forma de modelar el sistema, y sobre todo cambios importantes en la implementación de las estructuras de datos, la resolución de ecuaciones, y la estructura en general del entorno. Adicionalmente, se han integrado nuevos planificadores.

Finalmente, se ha mostrado la utilidad y funcionalidad del entorno mediante estudios comparativos del comportamiento de los cuatro planificadores implementados bajo diferentes aspectos.

6.2 CONCLUSIONES

El nuevo método de interacción con el usuario introducido en el entorno y el rediseño del anterior, permiten a los investigadores definir escenarios de prueba de una forma más rápida (Sec. 4.5.1). Unido al aumento de rendimiento del propio entorno (Sec. 3.8) ayuda a minimizar el tiempo dedicado a la parametrización de experimentos y el procesamiento de resultados, permitiendo centrarse más fácilmente en la discusión de los mismos.

El principal factor que ha contribuido al aumento del rendimiento es la optimización de la resolución de la ecuación de estado (Sec. 3.3.2). Por otro lado, la reducción de memoria conseguida tras hacer uso de matrices dispersas (Sec. 3.6) permite aumentar el tamaño y complejidad de los experimentos considerablemente.

Además de utilizar y ampliar conocimientos adquiridos en la titulación, en especial sobre planificación TR (limitada a monoprocesadores en el Grado), el TFG me ha acercado a la metodología y estilo de trabajo de un grupo de investigación con el que he podido interaccionar activamente.

6.3 TRABAJO FUTURO

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

- Divulgación del nuevo entorno, tanto mediante su disponibilidad pública como mediante la presentación de un tutorial en un workshop específico.
- Contabilización del número de cambios de contexto y migraciones diferenciando entre cambios de contexto obligatorios (independientes del planificador) y los causados específicamente por cada algoritmo.
- Incorporación del tiempo de cambio de contexto y migración de las tareas, a menudo conocido o caracterizable en sistemas reales. Aunque se obvia normalmente en estudios formales de planificación, es interesante tenerlo en cuenta en simulación.
- Incorporación de un analizador automático de resultados generando estadísticas sobre cumplimiento de plazos, cambios de contexto o utilización total. Esto también permitiría realizar comparaciones automáticas entre dos o más planificadores.
- Gestión de procesadores heterogéneos. Esta característica no se ha implementado debido a que tanto OLDTFS como JDEDS asumen que los procesadores son homogéneos, pero la arquitectura del entorno ha sido preparada para facilitar la ampliación del entorno hacia la simulación de heterogéneos como los basados en la arquitectura ARM big.LITTLE por ejemplo.
- Implementación de mecanismos de sincronización de tareas (e.g para simular conflictos de acceso a memoria).
- Crear un visualizador 3D del calentamiento del procesador.
- Incorporar un módulo de *floor planning* que permita explotar la capacidad del modelo $TCPN^T$ para modelar con precisión las características térmicas de cada parte de un multiprocesador.
- Aunque el rendimiento ha dejado de ser un factor limitante, aún existe margen de mejora paralelizando determinados cálculos mediante aceleradores (CUDA /OpenCL).
- El entorno podría ser ampliado para permitir su uso distribuido mediante una arquitectura cliente servidor, ejecutando el núcleo de la aplicación en el servidor y las interfaces de usuario en el cliente. El poco acoplamiento de las interfaces al propio núcleo de la aplicación y el hecho de que ambas interfaces sean interoperables con el formato JSON permite utilizar este mismo formato para las comunicaciones, manteniendo el formato de la

especificación del sistema. Debido a que los resultados de las simulaciones no son instantáneos se podría usar la tecnología websocket, aunque ello implicaría tener que añadir mecanismos de seguridad y control de acceso al servidor.

7 ANEXOS

7.1 CLASES Y DIAGRAMA DE CLASES

En la siguiente sección se define las clases que conforman la aplicación. Se omiten las clases que conforman las pruebas, los ficheros que contienen la especificación del formato de la entrada JSON en la interfaz por línea de comandos, los que conforman la definición de la interfaz de usuario en XML, y su respectiva traducción a Python. Si se desea, estos pueden consultarse en el repositorio Github del proyecto [40].

Todos los ficheros contienen solamente una clase del mismo nombre, salvo excepciones como los lanzadores de la interfaz gráfica y la interfaz por línea de comandos. En la descripción que aparece en las siguientes páginas se ha usado el nombre de los ficheros omitiendo la extensión (*.py*).

7.1.1 *Clases*

Cuadro 16: Clases del paquete principal

Fichero	Descripción
`cli_launcher`	Ejecuta la interfaz por línea de comandos
`gui_launcher`	Ejecuta la interfaz gráfica

Cuadro 17: Clases del paquete `main.core.problem_specification`

Fichero	Descripción
`GlobalSpecification`	Almacena la especificación del problema

Cuadro 18: Clases del paquete `main.core.problem_specification.tasks_specification`

Fichero	Descripción
`AperiodicTask`	Representación de una tarea aperiódica
`PeriodicTask`	Representación de una tarea periódica
`Task`	Clase abstracta que representa una tarea
`TasksSpecification`	Almacena el conjunto de tareas definidos en el problema

Cuadro 19: Clases del paquete `main.core.problem_specification.cpu_specification`

Fichero	Descripción
`BoardSpecification`	Representación de la placa donde se ubican los núcleos de cómputo
`CoreGroupSpecification`	Representación de un grupo de núcleos de cómputo con las mismas características
`CpuSpecification`	Representación de un procesador
`EnergyConsumptionProperties`	Propiedades relativas a la energía que consumen los núcleos de cómputo
`MaterialCuboid`	Representación de un objeto material con forma de cuboide
`Origin`	Representación de la ubicación de un objeto material con forma de cuboide

Cuadro 20: Clases del paquete `main.core.problem_specification.simulation_specification`

Fichero	Descripción
`SimulationPrecision`	Definición de la precisión de los cálculos de la simulación (Uso de reales de 32 bits o de 64 bits)
`SimulationSpecification`	Definición de variables relativas a la simulación (Tamaño del paso de malla, tamaño mínimo del quantum)
`TCPNModelSpecification`	Especificación del modelo térmico a usar

Cuadro 21: Clases del paquete `main.core.problem_specification.environment_specification`

Fichero	Descripción
`EnvironmentSpecification`	Especificación de variables ambientales (Temperatura del aire y coeficiente de convección del aire)

Cuadro 22: Clases del paquete `main.core.schedulers.implementations`

Fichero	Descripción
`G_EDF_AFA`	Implementación de una revisión del planificador G-EDF. En ella se minimiza el número de cambios de contexto, y se asigna la tarea con más prioridad al procesador que tenga una mayor frecuencia en caso de que no posean una frecuencia homogénea
`G_EDF_A`	Implementación de una revisión del planificador G-EDF. En ella se minimiza el número de cambios de contexto.
`G_EDF`	Implementación del planificador G-EDF
`G_LLF_AFA`	Implementación de una revisión del planificador G-LLF. En ella se minimiza el número de cambios de contexto, y se asigna la tarea con más prioridad al procesador que tenga una mayor frecuencia en caso de que no posean una frecuencia homogénea
`JDEDS`	Implementación del planificador JDEDS
`OLDTFS`	Implementación del planificador OLDTFS

Cuadro 23: Clases del paquete `main.core.schedulers.templates.abstract_base_scheduler`

Fichero	Descripción
`AbstractBaseScheduler`	Clase abstracta que representa un planificador. Esta clase es usada como una capa de abstracción para implementar planificadores, ofreciéndose una interfaz simple
`BaseSchedulerAperiodicTask`	Representación de un trabajo de una tarea aperiódica. En esta clase también se almacena información de la tarea aperiódica
`BaseSchedulerPeriodicTask`	Representación de un trabajo de una tarea periódica. En esta clase también se almacena información de la tarea periódica
`BaseSchedulerTask`	Clase abstracta que representa un trabajo

Cuadro 24: Clases del paquete `main.core.schedulers.templates.abstract_scheduler`

Fichero	Descripción
`AbstractScheduler`	Clase abstracta que representa un planificador
`SchedulerResult`	Representación de los resultados de la ejecución de un planificador

Cuadro 25: Clases del paquete `main.core.schedulers.utils`

Fichero	Descripción
`GlobalModelSolver`	Se encarga de simular los dos modelos de TCPN de una forma coordinada. Se puede ver como el simulador del comportamiento del procesador. Es llamada en cada paso de simulación con las tareas que ha de ejecutar, y devuelve cuanto han avanzado estas tareas, y la temperatura del procesador tras este avance.

Cuadro 26: Clases del paquete `main.core.task_generator.implementation`

Fichero	Descripción
`UUniFast`	Implementación del algoritmo de generación de tareas UUniFast

Cuadro 27: Clases del paquete `main.core.task_generator.template`

Fichero	Descripción
`AbstractTaskGeneratorAlgorithm`	Clase abstracta que representa un algoritmo de generación de tareas

Cuadro 28: Clases del paquete `main.core.tcpn_model_generator`

Fichero	Descripción
`GlobalModel`	Encapsula las TCPN relativas a los dos modelo de TCPN usados por el planificador
`ProcessorModel`	Crea la TCPN del modelo de CPUs
`TasksModel`	Crea la TCPN del modelo de tareas
`ThermalModelEnergy`	Crea la TCPN del modelo térmico. En esta alternativa, el usuario especifica el consumo energético que tiene cada tarea. A partir de este se obtiene la potencia dinámica.
`ThermalModelFrequency`	Crea la TCPN del modelo térmico. En esta alternativa, la potencia dinámica se obtiene a partir de la frecuencia de los procesadores (se asume que estos aplican DVFS).
`ThermalModel`	Crea la TCPN del modelo térmico. Esta es una clase abstracta.
`ThermalModelSelector`	Permite seleccionar el método de generación de la potencia dinámica.

Cuadro 29: Clases del paquete `main.core.tcpn_simulator.implementation.numerical_integration`

Fichero	Descripción
`TcpnSimulatorIntegrationFixedStep`	Simulador de TCPN basado en el método de Euler
`TcpnSimulatorIntegrationVariableStep`	Simulador de TCPN basado en la fórmula de Runge-Kutta
`TcpnSimulatorOptimizedTasksAndProcessors`	Simulador de TCPN basado en el método de Euler optimizado para el modelo de tareas-procesadores
`TcpnSimulatorOptimizedThermal`	Simulador de TCPN basado en el método de Euler optimizado para el modelo térmico

Cuadro 30: Clases del paquete `main.core.tcpn_simulator.implementation.second_semantic`

Fichero	Descripción
`TcpnSimulatorAccurateOptimized`	Simulador de TCPN basado en una semántica de disparos alternativa. Optimizado para el caso de que Π sea variable
`TcpnSimulatorAccurateOptimizedTasks`	Simulador de TCPN basado en una semántica de disparos alternativa. Optimizado para el modelo de tareas-procesadores
`TcpnSimulatorAccurate`	Simulador de TCPN basado en una semántica de disparos alternativa. Optimizado para el caso de que Π sea invariable

Cuadro 31: Clases del paquete `main.core.tcpn_simulator.template`

Fichero	Descripción
`AbstractTcpnSimulator`	Clase abstracta que representa un simulador de TCPN

Cuadro 32: Clases del paquete `main.plot_generator.implementations`

Fichero	Descripción
`AccumulatedExecutionTimeDrawer`	Genera la gráfica que representa el tiempo de ejecución de las tareas en las CPUs a lo largo del tiempo de ejecución del programa
`EnergyConsumptionDrawer`	Genera la gráfica que representa el consumo de energía de las CPUs a lo largo del tiempo de ejecución del programa
`ExecutionPercentageDrawer`	Genera la gráfica que representa los cumplimientos de plazos de los trabajos a lo largo del tiempo de ejecución del programa
`FrequencyDrawer`	Genera la gráfica que representa la frecuencia de las CPUs a lo largo del tiempo de ejecución del programa
`HeatMatrixDrawer`	Genera la animación que representa el calentamiento del procesador a lo largo del tiempo de ejecución del programa
`MaxCoreTemperatureDrawer`	Genera la gráfica que representa la temperatura máxima de las CPUs a lo largo del tiempo de ejecución del programa
`TaskExecutionDrawer`	Genera la gráfica que representa la ejecución de las distintas tareas a lo largo del tiempo de ejecución del programa
`UtilizationDrawer`	Genera la gráfica que representa la utilización de las CPUs a lo largo del tiempo de ejecución del programa

Cuadro 33: Clases del paquete `main.plot_generator.templates`

Fichero	Descripción
`AbstractResultDrawer`	Clase abstracta que representa un generador de gráficas

Cuadro 34: Clases del paquete `main.ui.cli`

Fichero	Descripción
`CliController`	Gestiona la interfaz por línea de comandos
`ProgressBarCli`	Gestiona el progreso de una simulación en la interfaz por línea de comandos

Cuadro 35: Clases del paquete `main.ui.common`

Fichero	Descripción
`AbstractProgressBar`	Gestiona el progreso de una simulación
`JSONGlobalModelParser`	Implementa funciones que permiten la lectura, escritura y validación mediante JSON-Schema de un fichero JSON
`OutputSelector`	Permite seleccionar la gráfica a generar a partir de su nombre
`SchedulerSelector`	Permite seleccionar el planificador a usar a partir de su nombre
`TaskGeneratorSelector`	Permite seleccionar la generación de tareas a usar a partir de su nombre
`TCPNThermalModelSelector`	Permite seleccionar el modelo térmico a usar a partir de su nombre

Cuadro 36: Clases del paquete `main.ui.gui`

Fichero	Descripción
`GuiController`	Gestiona la interfaz gráfica

Cuadro 37: Clases del paquete `main.ui.gui.implementation`

Fichero	Descripción
`AddAutomaticTaskDialog`	Implementación de la ventana que permite la generación automática de tareas
`AddFrequencyDialog`	Implementación de la ventana que permite el añadido de nuevas frecuencias
`AddOriginDialog`	Implementación de la ventana que permite el añadido de las ubicaciones de las CPUs
`AddOutputDialog`	Implementación de la ventana que permite el seleccionado de las gráficas que se desean generar
`AddTaskDialog`	Implementación de la ventana que permite el añadido de nuevas tareas
`MainWindow`	Implementación de la ventana principal de la interfaz gráfica

7.1.2 *Diagramas de clases y paquetes*

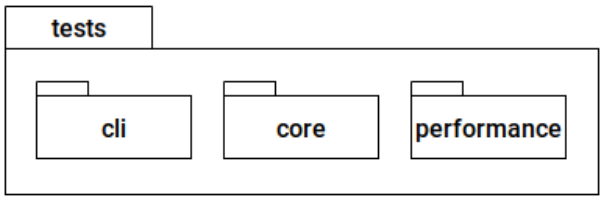


Figura 27: Diagrama del módulo de pruebas

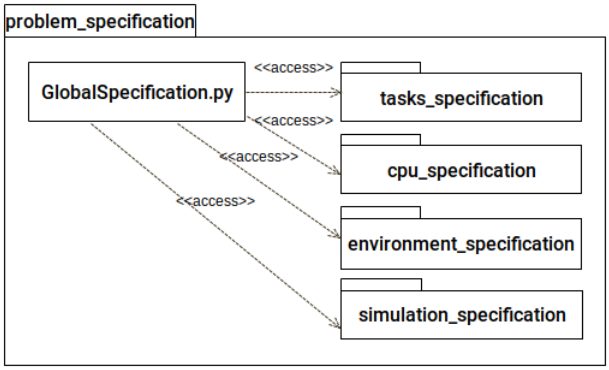


Figura 28: Diagrama del módulo de especificación del problema

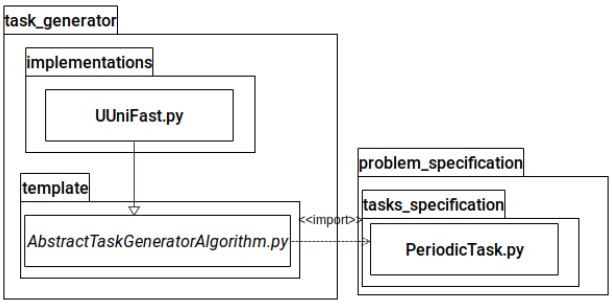


Figura 29: Diagrama del módulo de generación automática de tareas

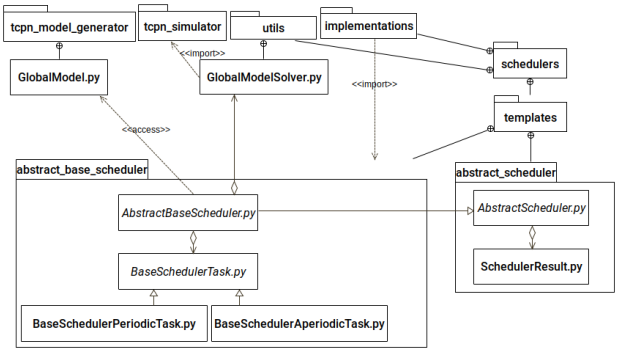


Figura 30: Diagrama del módulo de planificadores

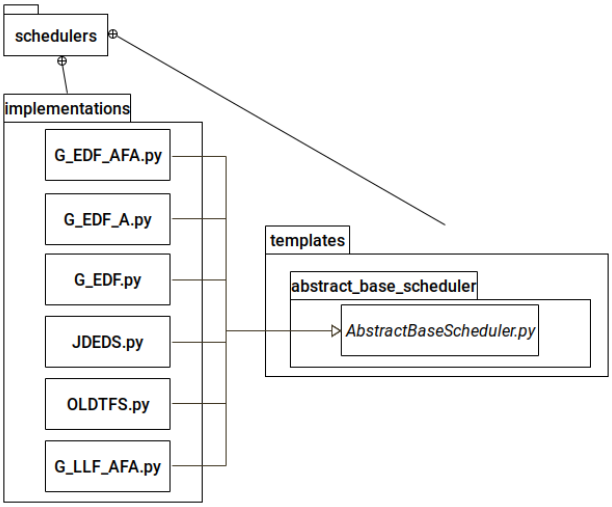


Figura 31: Detalle del módulo de planificadores

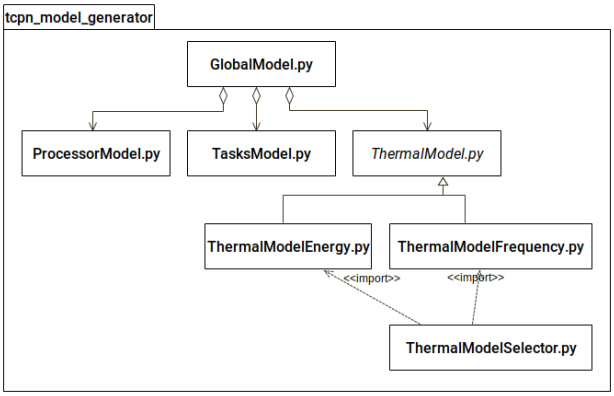


Figura 32: Diagrama del módulo de generación de los modelos de TCPN

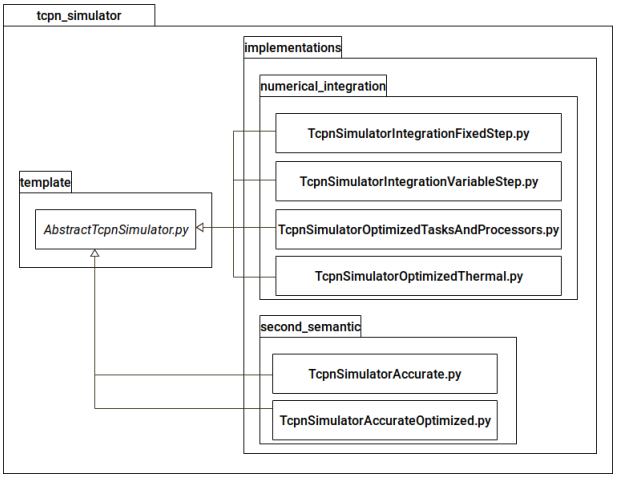


Figura 33: Diagrama del módulo de simulación de TCPN

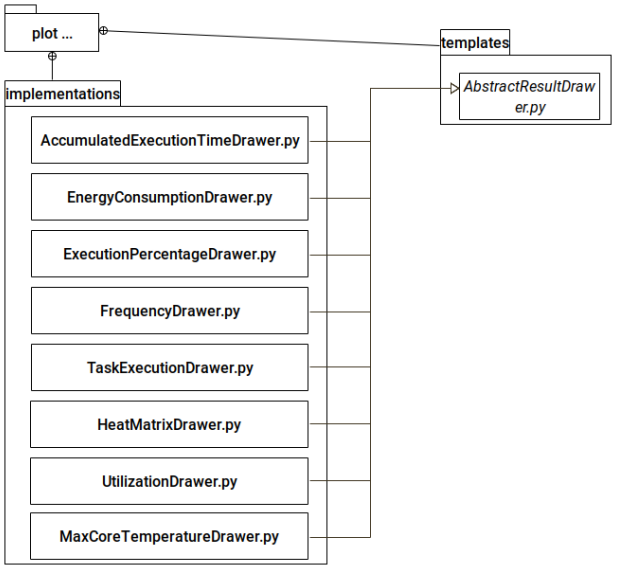


Figura 34: Diagrama del módulo de generación de resultados

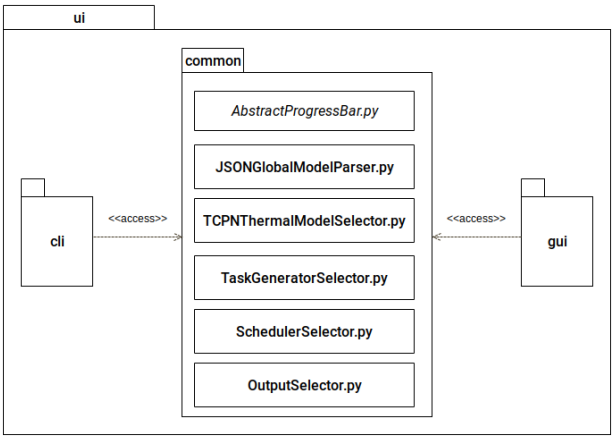


Figura 35: Diagrama del módulo de interacción con el usuario

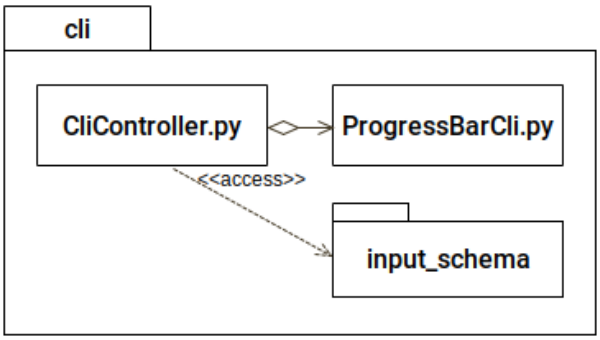


Figura 36: Diagrama del módulo de interacción con el usuario mediante línea de comandos

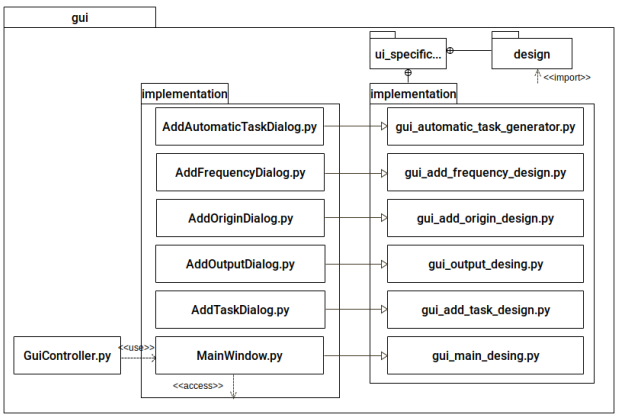


Figura 37: Diagrama del módulo de interacción con el usuario mediante interfaz gráfica

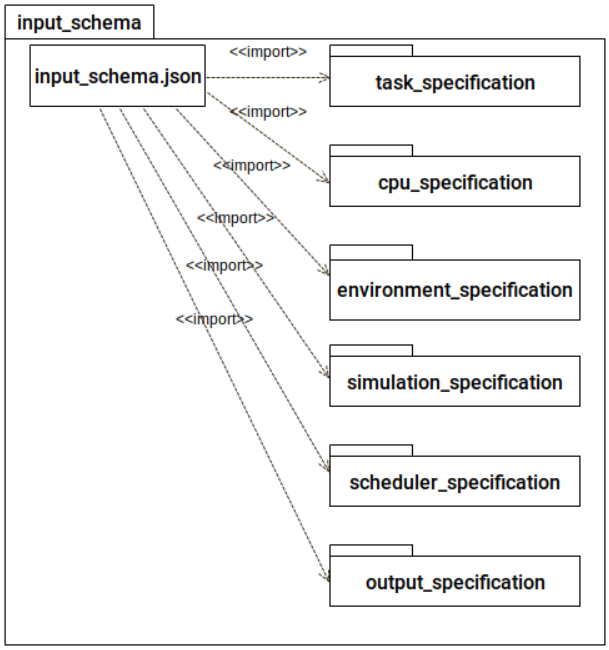


Figura 38: Diagrama del módulo de definición del formato de la entrada mediante línea de comandos

7.2 EQUIPO DE PRUEBAS

El equipo en el que se han ejecutado todos los experimentos posee la siguiente configuración:

- Sistema operativo: Ubuntu 18.04
- Versión conda: 4.7
- Versión Python: 3.7
- Versión Scipy: 1.2.1
- Procesador: Intel® Core™2 Quad Q6600 (caché de 8 M, frecuencia de 2.40 GHz y FSB de 1066 MHz)
- Memoria RAM: 2 x 4 GB, 1600 MHz

Python se ha instalado mediante la distribución Anaconda, y todos los paquetes mencionados han sido instalados desde el canal por defecto.

7.3 UTILIZACIÓN DEL ENTORNO

En la siguiente sección se muestran los dos métodos de entrada que posee el entorno de simulación y se realiza una ligera descripción de su uso. Si se desea una descripción más profunda de algún campo se puede acceder al repositorio Github del proyecto [40].

Para obtener el entorno también se ha de acceder al repositorio Github del mismo. En él se encuentran instrucciones de la instalación de las dependencias que este posee.

7.3.1 *Interfaz por línea de comandos*

La interfaz por línea de comandos se ejecuta mediante el *script* de Python `cli_launcher.py`.

Este posee dos parámetros de entrada que se describen a continuación:

- `-f o -file`: Este parámetro es obligatorio y recibe un argumento. Este argumento es la ruta del fichero que especifica la simulación que se quiere realizar.
- `-v o -verbose`: Este parámetro es opcional y no recibe argumentos. Sólo en caso de activación, mostrará durante la simulación una barra del progreso de esta en la terminal.

7.3.1.1 *Especificación de la simulacion*

La entrada se especifica en un fichero de formato JSON. Un ejemplo de este se encuentra en la Fig. 7.3.1.1.

Aunque los nombres de los campos son descriptivos, se podría consultar la definición de los mismos en el paquete `input_schema`, que sirve como documentación del mismo.

```
{
    "$schema": "main/ui/cli/input_schema/global-schema.json",
    "$id": "example.specification",
    "title": "Example specification",
    "simulate_thermal": true,
    "tasks_specification": {
      "task_generation_system": "Manual",
      "task_consumption_model": "Frequency based",
      "tasks": [
        {
          "type": "Periodic",
          "worst_case_execution_time": 2000000,
          "period": 4
        },
        {
          "type": "Periodic",
          "worst_case_execution_time": 5000000,
```

```
          "period": 8
        },
        {
          "type": "Periodic",
          "worst_case_execution_time": 6000000,
          "period": 12
        }
      ]
    },
    "cpu_specification": {
      "board_specification": {
        "physical_properties": {
          "x": 50,
          "y": 50,
          "z": 1,
          "density": 8933,
          "specific_heat_capacity": 385,
          "thermal_conductivity": 400
        }
      },
      "cores_specification": {
        "physical_properties": {
          "x": 10,
          "y": 10,
          "z": 2,
          "density": 2330,
          "specific_heat_capacity": 712,
          "thermal_conductivity": 148
        },
        "energy_consumption_properties": {
          "leakage_alpha": 0.001,
          "leakage_delta": 0.1,
          "dynamic_alpha": 1.52,
          "dynamic_beta": 0.08
        },
        "available_frequencies": [
          150000,
          400000,
          600000,
          850000,
          1000000
        ],
        "operating_frequencies": [
          1000000,
          1000000
        ],
        "cores_origins": "Automatic"
      }
    },
    "environment_specification": {
      "environment_temperature": 45,
      "maximum_temperature": 110,
```

```
    "convection_factor": 0.001
  },
  "output_specification": {
    "output_path": "./results/experiment_n_1_frequency",
    "output_naming": "g_edf",
    "selected_output": [
      "Accumulated execution time",
      "Energy consumption",
      "Execution percentage",
      "Frequency",
      "Cores maximum temperature",
      "Task execution",
      "Processor utilization"
    ]
  },
  "scheduler_specification": {
    "name": "G-EDF"
  },
  "simulation_specification": {
    "mesh_step": 1,
    "dt": 0.01
  }
}
```

Listing 1: Ejemplo de definición de entrada en formato JSON

7.3.2 *Interfaz gráfica*

La interfaz gráfica se ejecuta mediante el *script* de Python `gui_launcher.py`, que no acepta ningún parámetro.

Las Figs. 39, 40, 41, 42, 43 y 44, muestra la interfaz desarrollada.

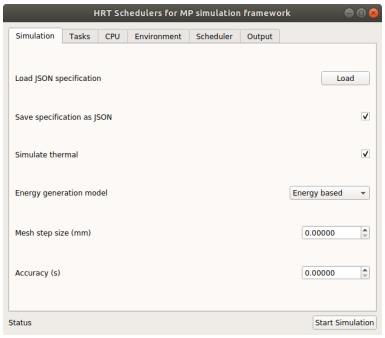
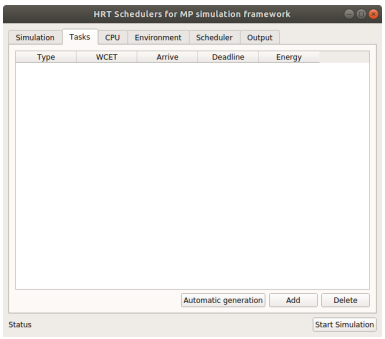
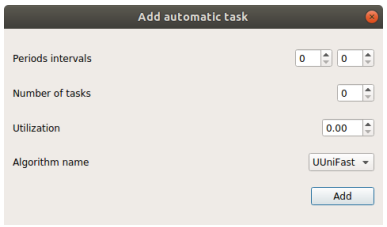


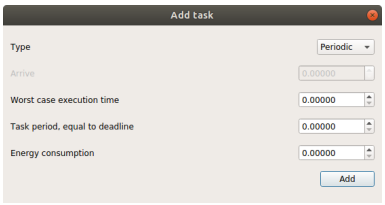
Figura 39: Interfaz gráfica: Definición de características de la simulación



(a) Listado de tareas a simular

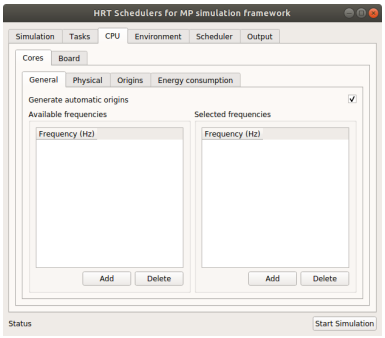


(b) Generar tareas automáticamente



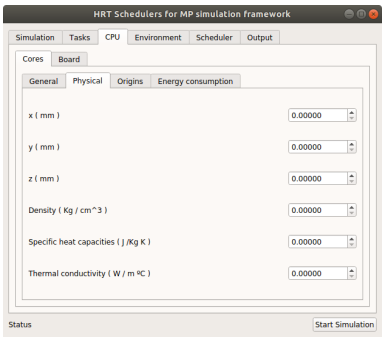
(c) Añadir nueva tarea

Figura 40: Interfaz gráfica: Definición de tareas

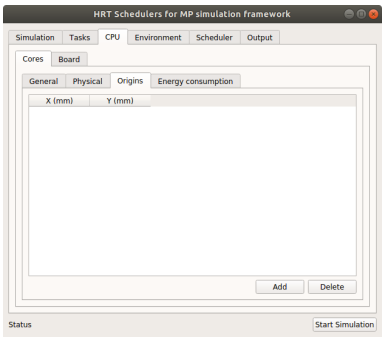


(a) Frecuencias de los núcleos

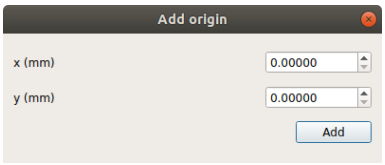
(b) Añadir nueva frecuencia



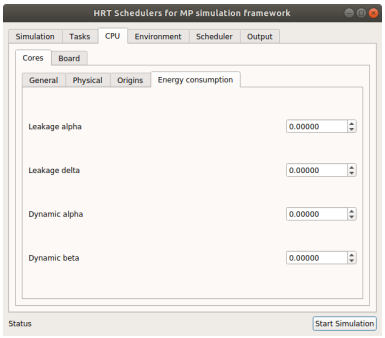
(c) Características físicas de los núcleos



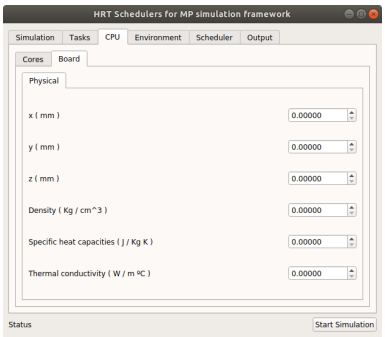
(d) Posiciones de los núcleos



(e) Añadir posición de un núcleo



(f) Características energéticas



(g) Características físicas de la placa

Figura 41: Interfaz gráfica: Definición del procesador

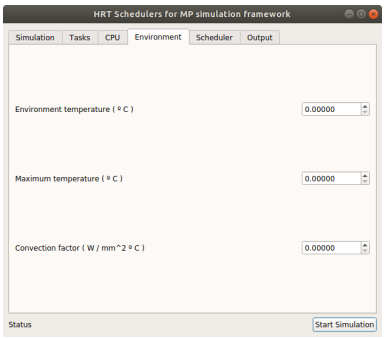


Figura 42: Interfaz gráfica: Definición del entorno

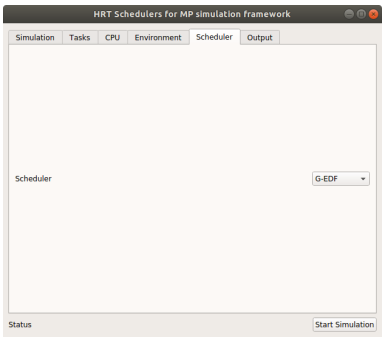
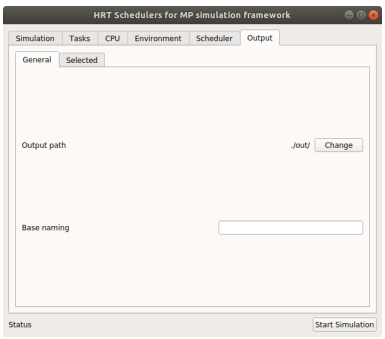
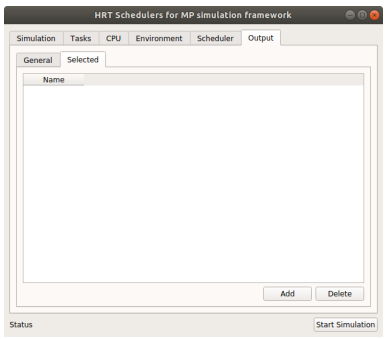


Figura 43: Interfaz gráfica: Selección del planificador



(a) Seleccionar ruta de guardado de los gráficos



(b) Gráficos seleccionados

(c) Seleccionar nuevo gráfico

Figura 44: Interfaz gráfica: Selección de los gráficos a generar

7.4 GRÁFICAS GENERADAS POR EL ENTORNO

En la siguiente sección se exponen los diferentes resultados gráficos que puede generar el entorno desarrollado tras una simulación.

Estas son listadas a continuación:

- Ejecución de tareas (Fig. 45 (a)): En esta gráfica se muestra la ejecución de las diferentes tareas a lo largo del hiperperiodo. En color azul se muestran los intervalos en los que se ejecutan dichas tareas, y en color naranja el conjunto de plazos de sus trabajos. En cada una de las subgráficas, el eje x marca la evolución del tiempo y el eje y indica si la tarea está en ejecución (*alto* implica estar en ejecución y *bajo* implica no estarlo).

- Cumplimiento de plazos (Fig. 45 (b)): En esta gráfica se muestra la tasa $\frac{ciclos\ ejecutados}{WCET}$ de cada uno de los trabajos del conjunto de tareas. De ella se puede derivar si un trabajo a cumplido su plazo (tasa igual a 100 % implica haberlo cumplido e inferior no haberlo hecho).

- Utilización (Fig. 45 (c)): En esta gráfica se muestra la asignación de las diferentes tareas a las CPUs a lo largo del hiperperiodo, pudiéndose ver la utilización de estos. En cada una de las subgráficas el eje x marca la evolución del tiempo y el eje y indica si la tarea está en ejecución (*alto* implica estar en ejecución y *bajo* implica no estarlo).

- Tiempo de ejecución de tareas en núcleos de procesamiento (Fig. 45 (d)): En esta gráfica se muestra el tiempo de ejecución acumulado por cada tarea, en cada núcleos de procesamiento, a lo largo del hiperperiodo. En cada una de las subgráficas el eje x marca la evolución del tiempo y el eje y indica el tiempo de ejecución acumulado que lleva la tarea en la CPU.

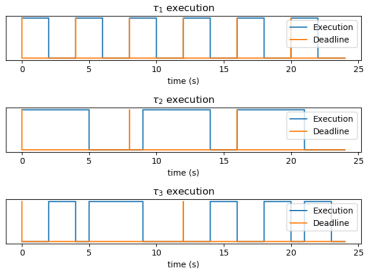
- Temperatura máxima de los núcleos de procesamiento (Fig. 45 (e)): En esta gráfica se muestra la temperatura a lo largo del hiperperiodo de cada CPU. En cada una de las subgráficas el eje x marca la evolución del tiempo y el eje y indica la temperatura de la CPU.

- Mapa de temperatura del procesador (Fig. 45 (f)): En esta animación (en la figura solamente se ha mostrado un fotograma de ella) se muestra la evolución de la temperatura en las diferentes partes del procesador a lo largo del hiperperiodo. El mapa de temperatura es tomado desde el lado del procesador que contiene las CPU y en la esquina superior izquierda se muestra el tiempo.

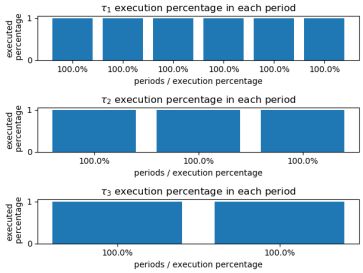
- Consumo energético de los núcleos de procesamiento (Fig. 45 (g)): En esta gráfica se muestra el consumo energético de cada

CPU debido a la energía dinámica (no se tiene en cuenta la energía de fuga) a lo largo del hiperperiodo. En cada una de las subgráficas el eje x marca la evolución del tiempo y el eje y indica el consumo energético de la CPU.

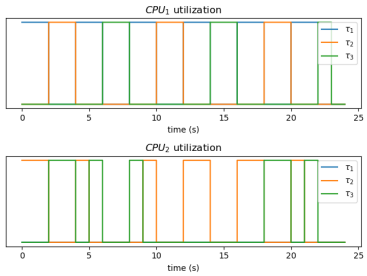
- Frecuencia de los núcleos de procesamiento (Fig. 45 (h)): En esta gráfica se muestra la frecuencia a la que opera cada CPU a lo largo del tiempo de simulación. En cada una de las subgráficas el eje x marca la evolución del tiempo y el eje y indica la frecuencia de la CPU.



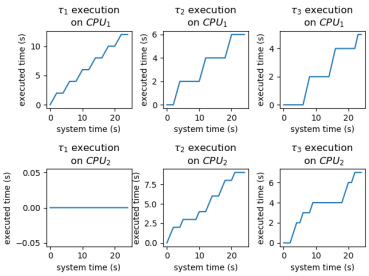
(a) Ejecución (Tareas)



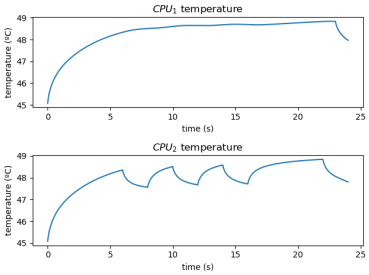
(b) Cumplimiento de plazos



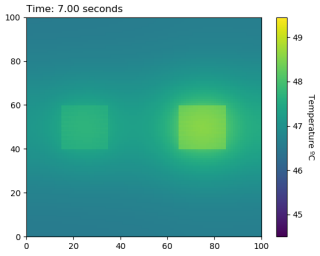
(c) Utilización



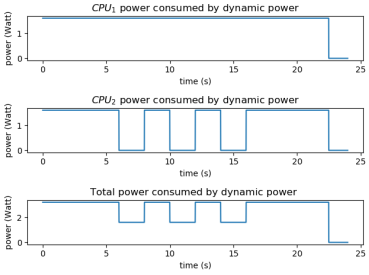
(d) Ejecución (Tareas, Procesadores)



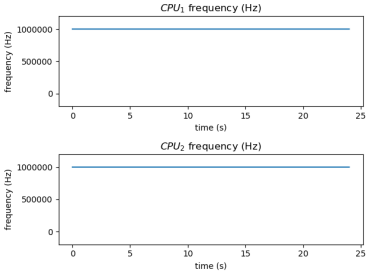
(e) Temperatura máxima



(f) Mapa de temperatura



(g) Consumo energético



(h) Frecuencia

Figura 45: Resultados gráficos generados por el entorno de simulación

7.5 ALGORITMO DE COMPARACIÓN DE MATRICES DISPERSAS

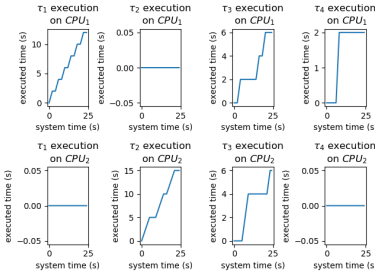
El algoritmo 2 ha sido el usado en la comparación de matrices dispersas.

Datos: pasos de simulación, fragmentación del paso de simulación, tipo de matriz
Resultado: tiempo de creación, tiempo de simulación
Pre, Post, λ, Π, m_0 = Crear modelo térmico () ;
marcado = m_0 ;
control = set de tareas a ejecutar ;
tiempo inicio = medir tiempo () ;
simulador TCPN = Crear simulador TCPN (Pre, Post, λ, Π, fragmentación del paso de simulación, tipo de matriz) ;
tiempo tras creación = medir tiempo () ;
para $i = 0$ **a** *pasos de simulación* **hacer**
| marcado = simulador TCPN.simular(marcado, control)
fin
tiempo tras simulación = medir tiempo () ;
tiempo de creación = tiempo tras creación - tiempo inicio ;
tiempo de simulación = tiempo tras simulación - tiempo tras creación ;

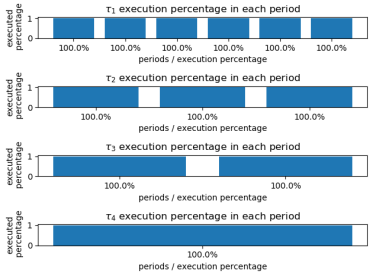
Algoritmo 1: Comparador del rendimiento de tipos de matrices en la simulación del modelo $TCPN^T$

7.6 RESULTADOS COMPLETOS DE LOS EXPERIMENTOS

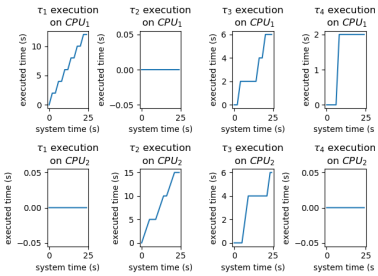
7.6.1 *Resultados completos del experimento 1*



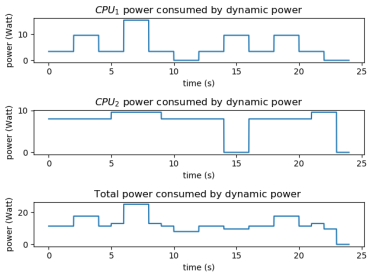
(a) Ejecución (Tareas)



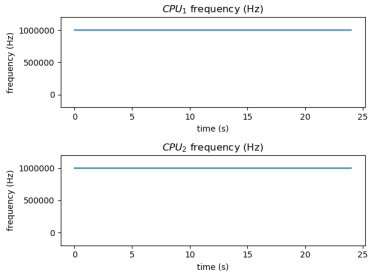
(b) Cumplimiento de plazos



(c) Utilización

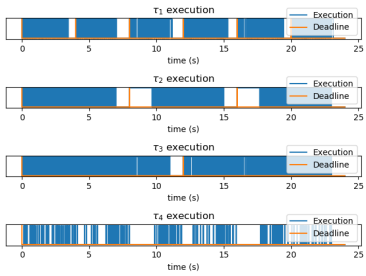


(d) Consumo energético

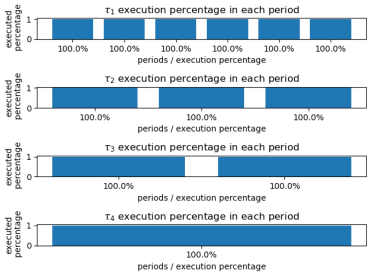


(e) Frecuencia

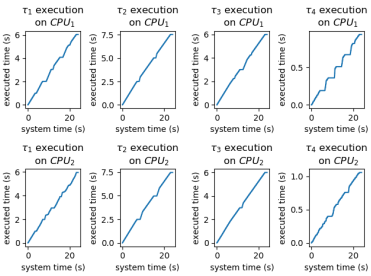
Figura 46: Resultados del planificador G-EDF en el experimento nº1



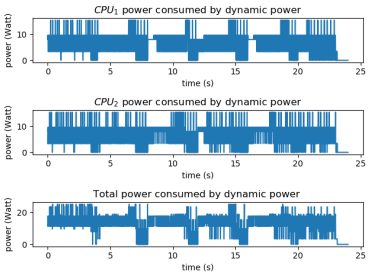
(a) Ejecución (Tareas)



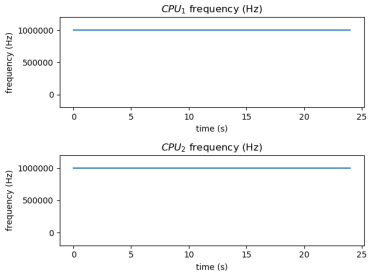
(b) Cumplimiento de plazos



(c) Utilización



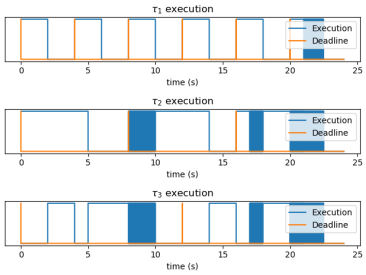
(d) Consumo energético



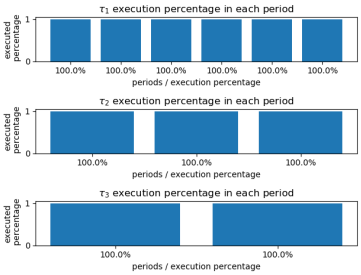
(e) Frecuencia

Figura 47: Resultados del planificador OLDTFS en el experimento nº1

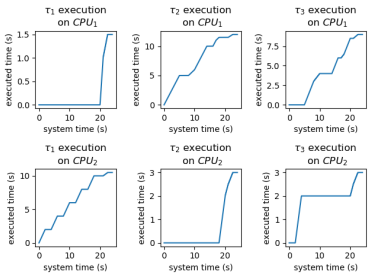
7.6.2 *Resultados completos del experimento 2*



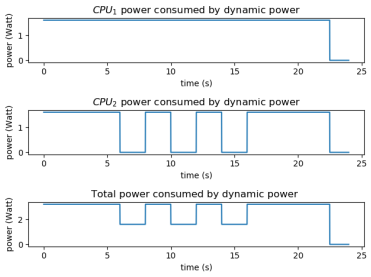
(a) Ejecución (Tareas)



(b) Cumplimiento de plazos

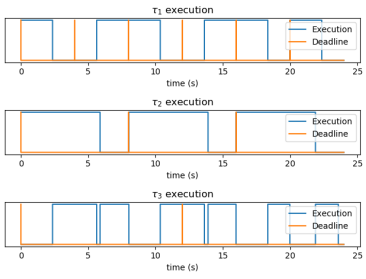


(c) Ejecución (Tareas, Procesadores)

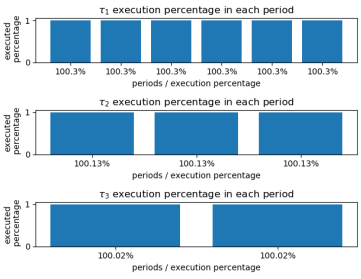


(d) Consumo energético

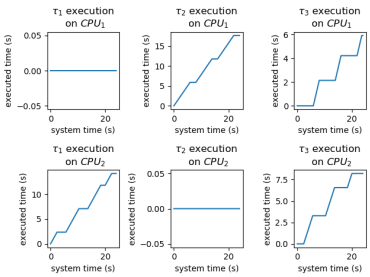
Figura 48: Resultados del planificador G-LLF en el experimento nº2



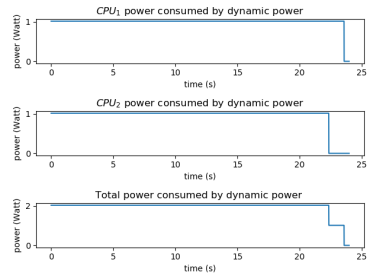
(a) Ejecución (Tareas)



(b) Cumplimiento de plazos



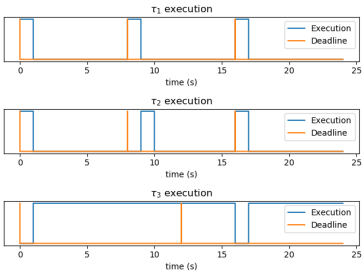
(c) Ejecución (Tareas, Procesadores)



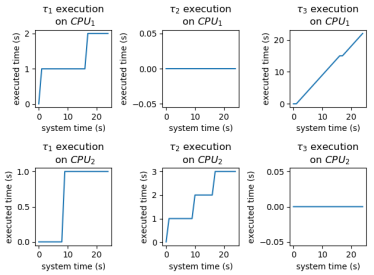
(d) Consumo energético

Figura 49: Resultados del planificador JDEDS en el experimento nº2

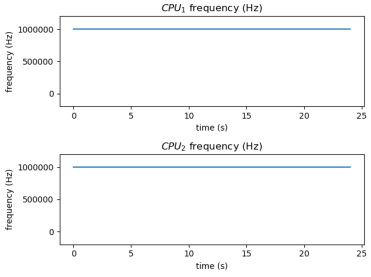
7.6.3 *Resultados completos del experimento 3*



(a) Ejecución (Tareas)

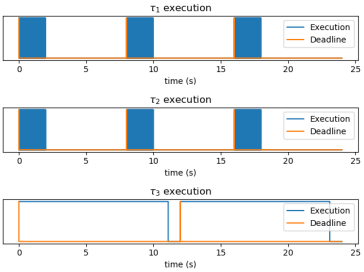


(b) Ejecución (Tareas, Procesadores)

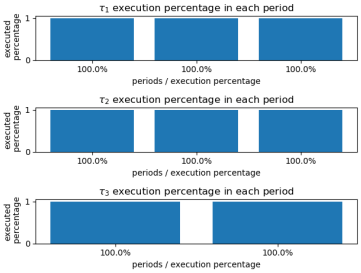


(c) Frecuencia

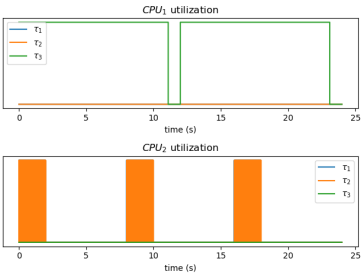
Figura 50: Resultados del planificador G-EDF en el experimento nº3



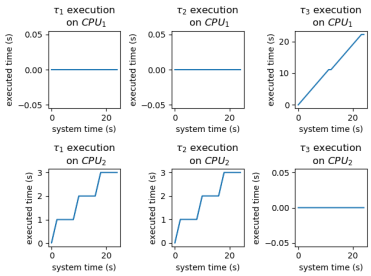
(a) Ejecución (Tareas)



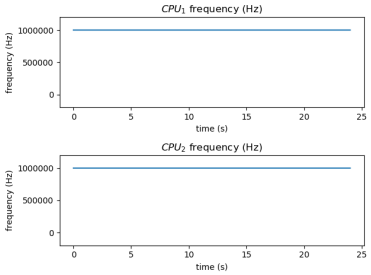
(b) Cumplimiento de plazos



(c) Utilización

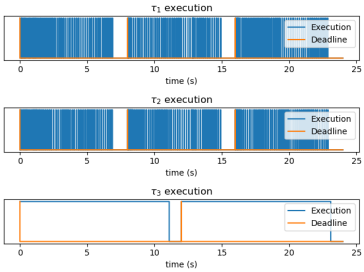


(d) Ejecución (Tareas, Procesadores)

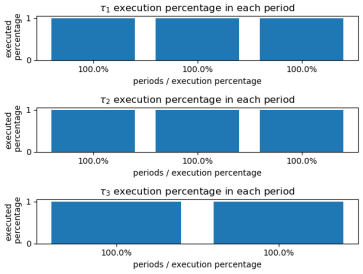


(e) Frecuencia

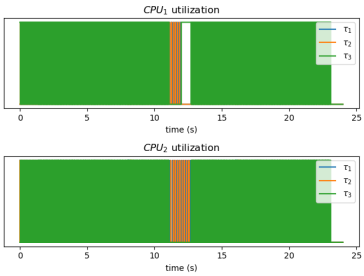
Figura 51: Resultados del planificador G-LLF en el experimento nº3



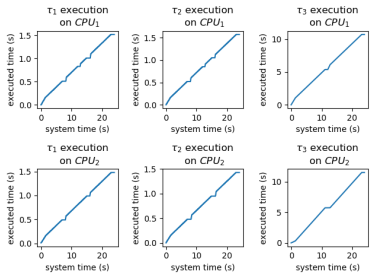
(a) Ejecución (Tareas)



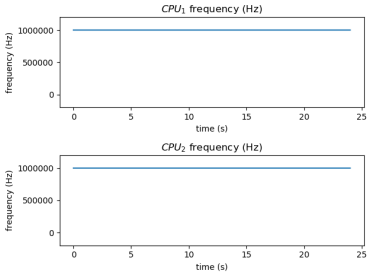
(b) Cumplimiento de plazos



(c) Utilización

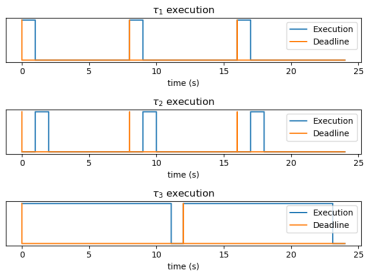


(d) Ejecución (Tareas, Procesadores)

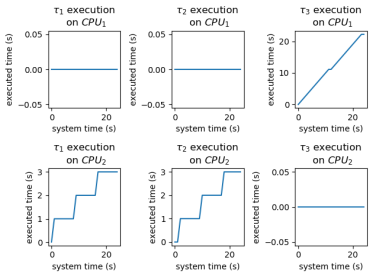


(e) Frecuencia

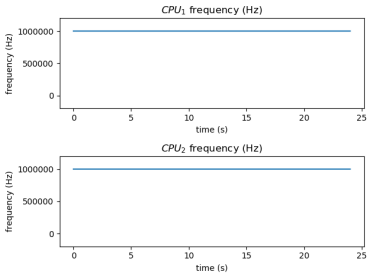
Figura 52: Resultados del planificador OLDTFS en el experimento nº3



(a) Ejecución (Tareas)



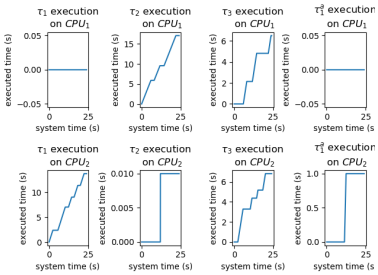
(b) Ejecución (Tareas, Procesadores)



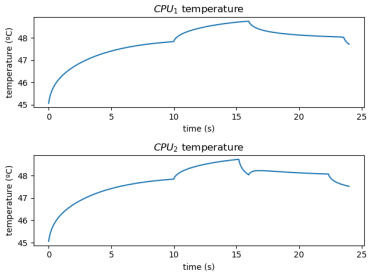
(c) Frecuencia

Figura 53: Resultados del planificador JDEDS en el experimento nº3

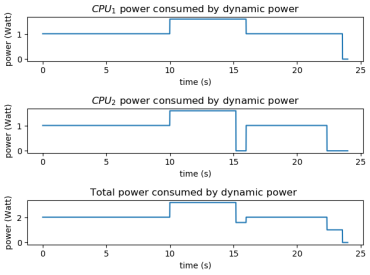
7.6.4 *Resultados completos del experimento 4*



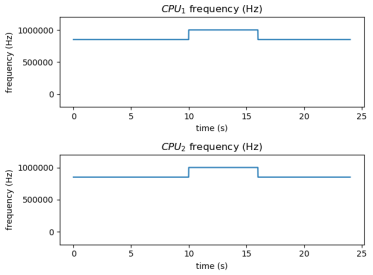
(a) Ejecución (Tareas, Procesadores)



(b) Temperatura máxima

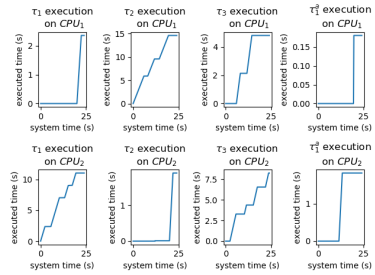


(c) Consumo energético

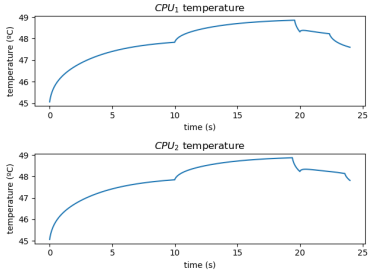


(d) Frecuencia

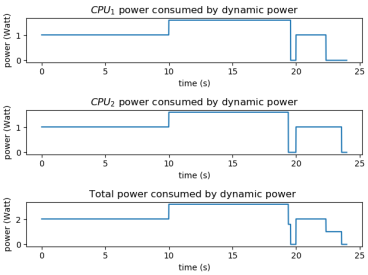
Figura 54: Resultados del planificador JDEDS con aperiódica de 1s en el experimento nº4



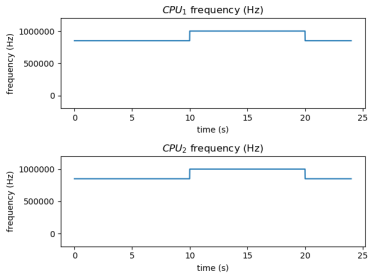
(a) Ejecución (Tareas, Procesadores)



(b) Temperatura máxima



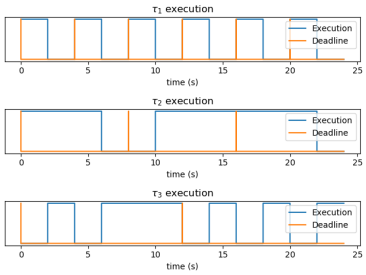
(c) Consumo energético



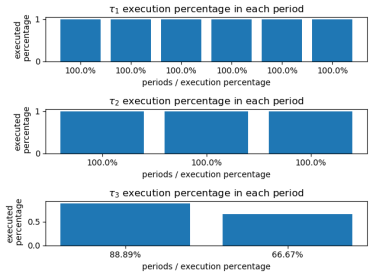
(d) Frecuencia

Figura 55: Resultados del planificador JDEDS con aperiódica de 2s en el experimento nº4

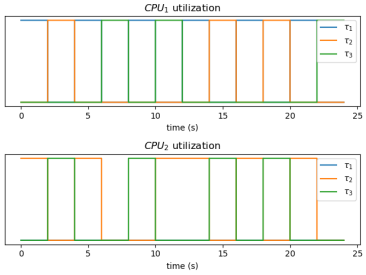
7.6.5 *Resultados completos del experimento 5*



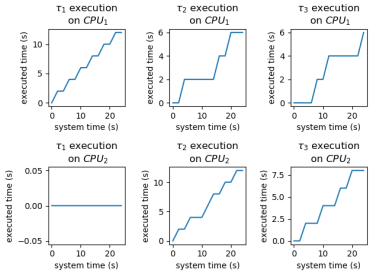
(a) Ejecución (Tareas)



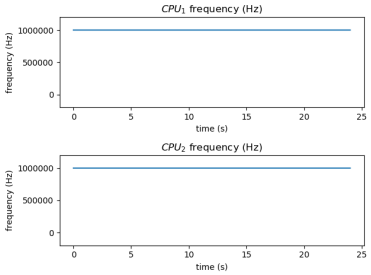
(b) Cumplimiento de plazos



(c) Utilización

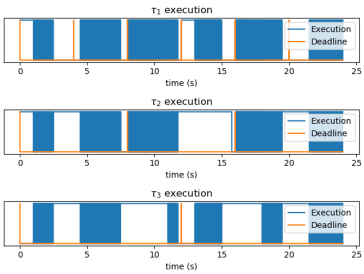


(d) Ejecución (Tareas, Procesadores)

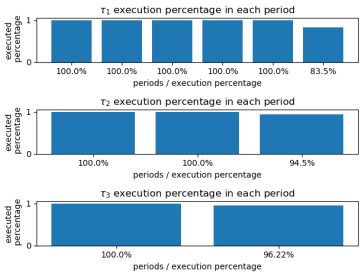


(e) Frecuencia

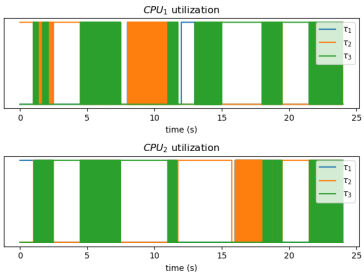
Figura 56: Resultados del planificador G-EDF en el experimento nº5



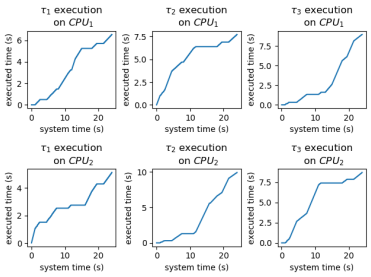
(a) Ejecución (Tareas)



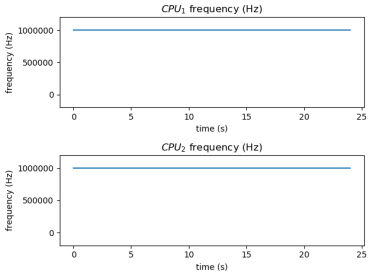
(b) Cumplimiento de plazos



(c) Utilización

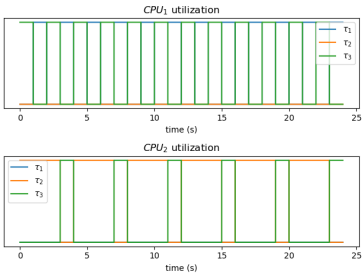


(d) Ejecución (Tareas, Procesadores)

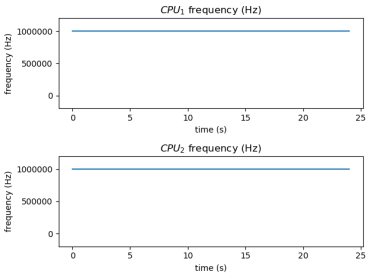


(e) Frecuencia

Figura 57: Resultados del planificador G-LLF en el experimento nº5

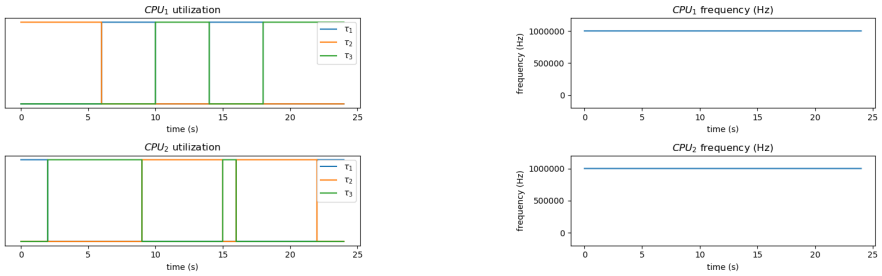


(a) Utilización



(b) Frecuencia

Figura 58: Resultados del planificador OLDTFS en el experimento nº5



(a) Utilización (b) Frecuencia

Figura 59: Resultados del planificador JDEDS en el experimento nº5

7.7 DIAGRAMA DE GANTT

En esta sección se muestra el diagrama de Gantt del proyecto. En él solamente se incluye la planificación relativa al TFG obviándose la planificación del trabajo previo desarrollado en la asignatura *Laboratorio de Sistemas Empotrados*.

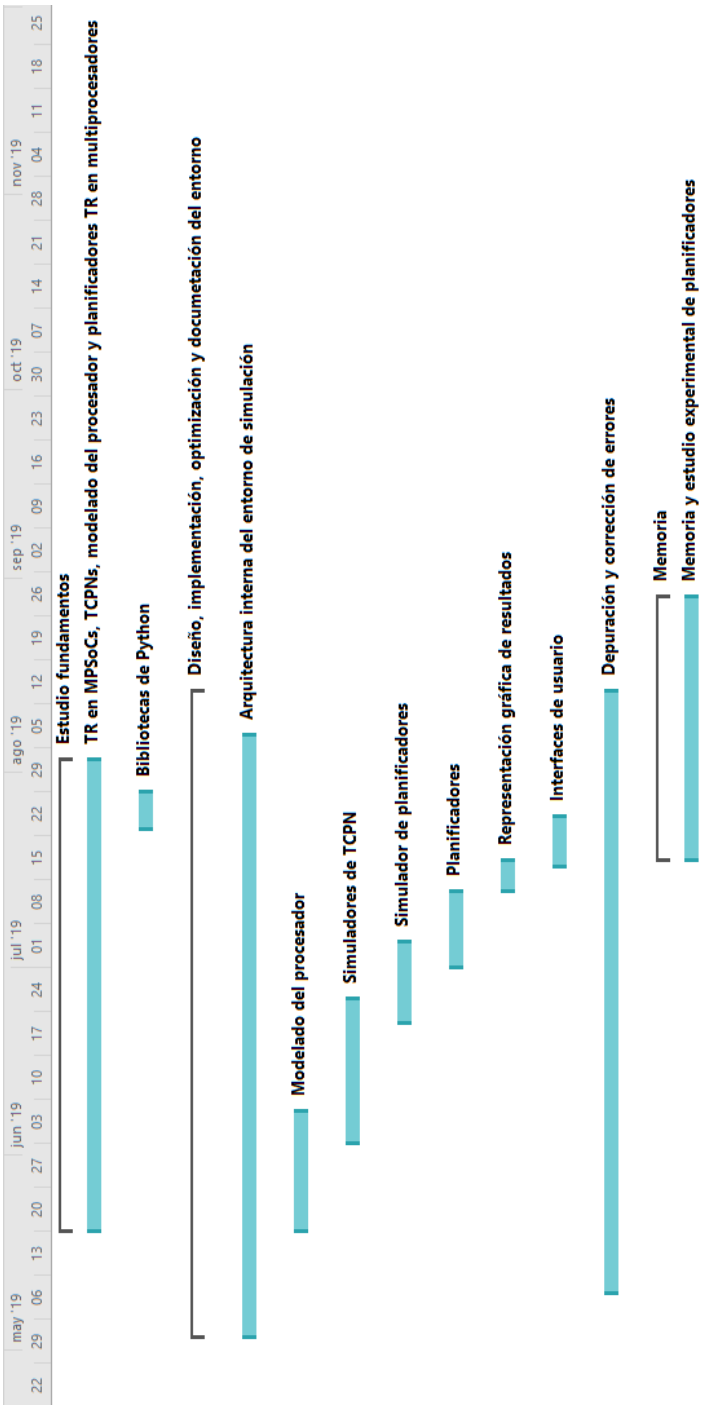


Figura 60: Diagrama de Gantt del proyecto

BIBLIOGRAFÍA

[1] 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.

[2] 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.

[3] 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.

[4] Nathan Otterness Björn B. Brandenburg Namhoon Kim. *LITMUS-RT: Linux Testbed for Multiprocessor Scheduling in Real-Time Systems*. https://www.litmus-rt.org/. (Visitado 20-08-2019).

[5] Bjorn B. Brandenburg. «Scheduling and Locking in Multiprocessor Real-time Operating Systems». AAI3502550. Tesis doct. Chapel Hill, NC, USA: University of North Carolina at Chapel Hill, 2011. ISBN: 978-1-267-25618-8.

[6] Daniel Casini, Alessandro Biondi y Giorgio Buttazzo. «Semi-Partitioned Scheduling of Dynamic Real-Time Workload: A Practical Approach Based on Analysis-Driven Load Balancing». En: *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*. Ed. por Marko Bertogna. Vol. 76. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 13:1-13:23. ISBN: 978-3-95977-037-8. DOI: 10.4230/LIPIcs.ECRTS.2017.13. URL: http://drops.dagstuhl.de/opus/volltexte/2017/7165.

[7] *Conda*. URL: https://conda.io/en/latest/ (visitado 20-08-2019).

[8] G Desirena-López, LR Vazquez, A Ramírez-Ireviño y David Gómez-Gutiérrez. «Thermal modelling for temperature control in MPSoC'S using timed continuous petri nets». En: *Control Applications (CCA), 2014 IEEE Conference on*. IEEE. 2014, págs. 2135-2140.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] G. Desirena, L. Rubio, A. Ramirez y J.L. Briz. *Thermal-Aware HRT Scheduling simulation framework*. https://www.gdl.cinvestav.mx/art/uploads/TCPN-Thermal-Aware_Real-Time_Scheduling.zip. 2019. (Visitado 08-03-2019).

[14] P. J. Prince Dormand J. R. «A family of embedded Runge-Kutta formulae». En: (). URL: https://doi.org/10.1016/0771-050X(80)90013-3.

[15] The Python Software Foundation. *Python Enhancement Proposals*. https://www.python.org/dev/peps/. 2000. (Visitado 07-08-2019).

[16] 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.

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

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

[19] 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.

[20] *Intel® Math Kernel Library*. URL: https://software.intel.com/en-us/mkl (visitado 21-08-2019).

[21] *JSON Schema specification*. URL: https://json-schema.org/.

[22] Eric Jones, Travis Oliphant, Pearu Peterson y col. *SciPy: Open source scientific tools for Python*. http://www.scipy.org/. 2001–. (Visitado 12-08-2019).

[23] Yongpan Liu, Robert P Dick, Li Shang y Huazhong Yang. «Accurate temperature-dependent integrated circuit leakage power estimation is easy». En: (2007), págs. 1526-1531.

[24] MATLAB. *version 9.4 (R2018a)*. Natick, Massachusetts: The MathWorks Inc., 2018.

[25] *Miniconda*. URL: https://conda.io/en/latest/miniconda.html (visitado 20-08-2019).

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

[27] *Performance Comparison of OpenBLAS* and Intel® Math Kernel Library in R*. URL: https://software.intel.com/en-us/articles/performance-comparison-of-openblas-and-intel-math-kernel-library-in-r (visitado 21-08-2019).

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

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

[30] *PyQt5 library*. URL: https://www.riverbankcomputing.com/static/Docs/PyQt5/.

[31] *Python Language*. URL: https://www.python.org/ (visitado 20-08-2019).

[32] *Python Performance Tips*. URL: {https://wiki.python.org/moin/PythonSpeed/PerformanceTips} (visitado 20-08-2019).

[33] *Qt library*. URL: https://www.qt.io/.

[34] 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.

[35] 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.

[36] *SciPy: SciPy 2-D sparse matrix package for numeric data*. https://docs.scipy.org/doc/scipy/reference/sparse.html. (Visitado 12-08-2019).

[37] Manuel Silva, Jorge Júlvez, Cristian Mahulea y C. Renato Vázquez. «On fluidization of discrete event models: observation and control of continuous Petri nets». En: *Discrete Event Dynamic Systems* 21.4 (2011), págs. 427-497. ISSN: 1573-7594.

[38] 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).

[39] *TkInter library*. URL: https://docs.python.org/3/library/tkinter.html#module-tkinter.

[40] Abel Chills Trabanco. *Fuentes del TFG*. URL: {https://github.com/AbelChT/RT-scheduler-simulation-environment} (visitado 21-08-2019).

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

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

[43] Mark Weiser, Brent Welch, Alan Demers y Scott Shenker. «Scheduling for Reduced CPU Energy». En: *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*. OSDI '94. Monterey, California: USENIX Association, 1994. URL: http://dl.acm.org/citation.cfm?id=1267638.1267640.