Revista Iberoamericana de Automática e Informática Industrial

Revista Iberoamericana de Automática e Informática Industrial 18 (2021) 396-406

www.revista-riai.org

Control inteligente mediante escalado dinámico voltaje-frecuencia (DVFS) de la temperatura en procesadores embebidos

Hernández-Almudi, P.[b], Suárez, D.[a], Montijano, E.[a,*], Merino, J.[b]

[a]*Instituto Universitario en Ingeniería de Aragón (I3A), Universidad de Zaragoza, c/ María de Luna 1, 50018 Zaragoza, España.*
[b]*Arm, 110 Fulbourn Road, CB1 9NJ, Cambridge, Reino Unido.*

To cite this article: Hernández-Almudi, P., Suárez, D., Montijano, E., Merino, J., 2021. Intelligent control of temperature with dynamic voltage-frequency scaling (DVFS) in embedded processors. Revista Iberoamericana de Automática e Informática Industrial. 18, 396-406. https://doi.org/10.4995/riai.2020.14200

Resumen

El aumento de la capacidad de cálculo de los procesadores embebidos ha generado una revolución en numerosos dominios de aplicación como la computación móvil o la robótica. El consumo producido por estos cálculos en una superficie tan pequeña hace que la disipación de energía suponga un problema de primer orden. Por un lado, no es posible aplicar técnicas de disipación activas y, por otro, las exigencias de diseño impiden una correcta disipación pasiva. Para resolver este problema, este trabajo presenta una metodología de control para el mantenimiento de una temperatura bajo control mediante el uso de escalado dinámico de la frecuencia del procesador (DVFS, Dynamic Voltage Frequency Scaling). La solución incluye un esquema de control de temperatura basado en realimentación junto con un supervisor que ajusta los parámetros del controlador en base al tipo de carga del trabajo. La estrategia de control propuesta se ha implementado tanto en espacio de usuario como driver dentro del *kernel* de Linux. Los experimentos realizados en una plataforma real demuestran que, comparado con el control existente en la actualidad, nuestra propuesta es capaz de gestionar la temperatura del procesador con más precisión, manteniendo niveles similares de eficiencia en la ejecución de *benchmarks* conocidos.

Palabras clave: Control inteligente de temperatura, arquitectura de computadores, DVFS.

Intelligent control of temperature with dynamic voltage-frequency scaling (DVFS) in embedded processors

Abstract

The increment in computing power of embedded processors has fueled a revolution in many application domains such as mobile computing or robotics. Dissipating the energy consumed by those processors on a very small area has made power management a first-order constraint. On one hand, it is impossible to directly apply active dissipation techniques. On the other hand, design requirements prevent the correct behaviour of known passive detection techniques. To alleviate this problem, we present a new control approach to keep the temperature of the system controlled through the Dynamic Voltage and Frequency Scaling (DVFS) system. The solution includes a feedback control scheme together with a supervisor that adjusts the control parameters based on the system load. To ease experimentation without compromising real use, the code can run in user space and as a Linux kernel driver. The experiments in a real platform prove that, compared to the current control methodology, our approach handles the processor's temperature with more precision, keeping similar performance levels in the execution of well known benchmarks.

Keywords: Intelligent control of temperature, computer architecture, DVFS.

*Autor para correspondencia: emonti@unizar.es

1. Introducción

La mejora de las prestaciones de los sistemas de computación embebidos ha sido uno de los principales hitos en la generalización de los sistemas *ciberfísicos* y el internet de las cosas (IoT - Internet of Things). Ejemplos como coches autónomos, robots, monitores médicos o sistemas de vigilancia han visto incrementada su capacidad de cálculo, lo que ha permitido que ejecuten cargas complejas, tales como aplicaciones de inteligencia artificial, que han incrementando notablemente su utilidad (Rajkumar et al., 2010).

Obviando el debate sobre si es el software quien tiende a ocupar todo el hardware disponible, o bien, si es el hardware el que se intenta adaptar al software disponible (Alastruey et al., 2006), lo cierto es que el incremento de la demanda de cálculo implica un incremento en la cantidad de calor disipado, a la vez que una mayor demanda de energía. La disipación de energía es fundamental para alargar la vida de los componentes electrónicos de cualquier sistema (López et al., 2019).

En entornos de computación clásicos tales como centros de cálculo, el calor no es siempre el problema principal, ya que mediante el uso de grandes disipadores y ventiladores se puede expulsar, y con mayores fuentes de alimentación se proporciona la demanda energética. Por supuesto estas soluciones de disipación activas también implican un mayor coste económico y medioambiental debido al elevado consumo. Por otro lado, en dispositivos más pequeños o en entornos con grandes restricciones térmicas el calor se convierte en un problema, sino en el mayor de ellos. Por ejemplo, no querríamos que un vehículo aéreo no tripulado realizando tareas de vigilancia (Madridano et al., 2020) se cayera porque el sobrecalentamiento del procesador ha deteriorado la unidad o no le ha permitido detectar un obstáculo.

Para solucionar estos problemas se utilizan soluciones de refrigeración pasivas que no requieren efectuar un trabajo para extraer la energía del sistema sino que directamente controlan la generación de calor. Muchas de estas soluciones de refrigeración pasiva se basan en actuar cuando el procesador empieza a sufrir ahogamiento térmico, o *thermal throttling* (Brooks and Martonosi, 2001; Cohen et al., 2003). Es decir, cuando el procesador sufre temperaturas excesivamente altas se activan mecanismos de reducción de temperatura, que reducen el rendimiento. El más empleado dentro de los métodos de refrigeración pasiva es la reducción de voltaje y frecuencia. La mayoría de las implementaciones se basan en heurísticas simples y puntos de activación, es decir, umbrales a partir de los cuales se empieza a reducir las frecuencias e incluso, de llegar a ser necesario, apagar por completo el procesador para evitar dañarlo.

Para tratar de mejorar las soluciones actuales nuestro trabajo presenta un nuevo gestor de frecuencia basado en teoría de control, cuyo objetivo es el mantenimiento de una temperatura constante a lo largo del tiempo. De este modo se garantiza una temperatura segura de funcionamiento, que además causa un comportamiento más suave del rendimiento del procesador dado que los cambios de frecuencia son predecibles. Este gestor además trata de paliar una de las limitaciones con las que cuentan la soluciones basadas en control de temperatura, que no se adaptan bien a los tipos de carga de trabajo, para lo que se ha desarrollado una propuesta de mecanismo de supervisión que adapta el control a los diferentes tipos de cargas.

La solución propuesta se basa en tres observaciones: 1) las cargas de trabajo que va a realizar el procesador en un sistema embebido son relativamente homogéneas a lo largo de su ejecución y además permiten clasificar sus fases de ejecución de manera aproximada en tres tipos, cada una con comportamiento propio. 2) la relación entre la temperatura y la frecuencia puede ser descrita mediante una función de transferencia de primer orden. Y, 3) el modelo puede ser utilizado para la creación de un entorno de control que unifique los controles de temperatura y frecuencia en un supervisor unificado. Esto es opuesto a como funcionan las soluciones actuales en las que las decisiones se toman individualmente sin el conocimiento del otro debido a que son soluciones legadas dentro de los sistemas operativos.

Teniendo esto en cuenta, las principal contribución de este artículo es una metodología basada en ingeniería de control que permite gestionar de forma eficiente la temperatura de trabajo de un procesador embebido. La metodología propuesta incluye una propuesta sencilla de modelado del comportamiento de la temperatura, junto con la identificación experimental de sus parámetros. En base a esta se ha desarrollado un esquema de control en bucle cerrado monobucle (SISO) para controlar la dinámica y valor de la temperatura del procesador en función de la carga de trabajo en cada instante. De este modo se previenen temperaturas excesivamente altas y la restricción de la frecuencia debido a alarmas térmicas.

Esta metodología de caracterización y posterior aplicación en un control puede ser fácilmente aplicada a diferentes configuraciones de procesador y carga de trabajo reduciendo el trabajo de optimización manual que realizan los integradores de sistemas antes de sacar los productos al mercado procesador. La estrategia de control propuesta se ha implementado tanto en espacio de usuario como driver dentro del *kernel* de Linux.

El resto del artículo se organiza de la siguiente manera: La sección 2 presenta el estado del arte en el problema estudiado. La sección 3 describe la estrategia de control de temperatura y la sección 4 el supervisor del control basado en la clasificación de programas. La sección 5 describe la metodología del trabajo y la sección 6 los resultados experimentales obtenidos. Finalmente, en la sección 7 se presentan las conclusiones del trabajo.

2. Estado del Arte

El control térmico de computadores ha suscitado mucho interés en el ámbito científico, generando un gran número de propuestas. Por un lado, tenemos soluciones basadas en hardware tanto pasivas como activas, como disipadores dedicados, o el uso de otros elementos del dispositivo como la batería para disipar el exceso de calor, ventiladores para forzar la entrada de aire o refrigeración líquida. Para centros de datos donde no hay tantas limitaciones de espacio y se puede emplear refrigeración activa, (Xu, 2007) evalúa el uso de refrigeración líquida en servidores y (Chen et al., 2020) propone un control dinámico con también refrigeración líquida para balancear el consumo energético de la refrigeración y de los procesadores. La refrigeración líquida no siempre es una opción en los entornos que funcionan los procesadores embebidos. (Yueh et al., 2015) propone un sistema de refrigeración por agua que prueba consumir menos que otras técnicas mientras logra mantener un alto desempeño. Aunque

estas técnicas mitigan el problema de la disipación tienen un coste extra que no siempre es posible.

Por otro lado, tenemos las soluciones software pasivas, las cuales, están mejor enfocadas en los dispositivos embebidos. Entre ellas podemos distinguir entre soluciones basadas en heurísticas y soluciones que consideran herramientas de ingeniería de control. Entre el primer tipo encontramos técnicas que se enfocan en el consumo de energía. (Park et al., 2018) propone priorizar las tareas que se ejecutan en primer plano, parando incluso las tareas de fondo cuando el consumo es excesivamente alto. Esto por supuesto supone que el sistema deja de hacer cosas que pueden ser importantes por lo que estaríamos perdiendo la utilidad que tiene el dispositivo. CoScale (Deng et al., 2012) mejora el consumo en aplicaciones con muchos accesos a memoria mediante la coordinación de las frecuencias tanto de la CPU como de la memoria. Pese a ser una buena solución, solo se centra en ese tipo de aplicaciones mientras que la solución que proponemos en este trabajo se adapta a todo tipo de tareas.

Algunas soluciones basadas en teoría de control son las propuestas por (Pothukuchi et al., 2016; Pothukuchi et al., 2020), usando un controlador MIMO, e Isci et al. (2006), proponiendo distintas propuestas que buscan el mejor compromiso entre rendimiento y consumo. Maggio et al. (2010) propone un controlador PID enfocado en mantener una calidad de servicio (QoS) constante, mediante el uso de bibliotecas que insertan señales en los programas que se ejecutan. Las propuestas anteriores solo han sido probadas mediante simuladores, por lo que no han sido evaluadas en una plataforma real, mientras que la última propuesta no considera la temperatura, solo calidad de servicio.

Otras propuestas están más orientadas al mantenimiento de un buen QoS. La metodología principal consiste en ejecutar varios experimentos para comprender como cambia el QoS bajo ciertos parámetros para poder proponer distintos modelos de consumo y algoritmos de control. SPECTR (Rahmani et al., 2018) usa un controlador MIMO basado en máquina de estados para mantener la QoS medida en imágenes por segundo a la par que se busca la mejor relación con el consumo. Rahmani et al. (2015) también propone un control retro-alimentado para proteger sistemas multi-core contra picos de energía usando un PID para realizar una estimación de la potencia disponible. De manera similar, IPA de Arm (Wang, 2017) implementa un governor térmico basado en PID para restringir las frecuencias disponibles en base a un modelo de potencia. Todos estos trabajos tienen en común el uso de la estimación de potencia disponible para calcular un valor de frecuencia aceptable. En nuestra propuesta trabajamos directamente con la temperatura simplificando el problema de control evitando la necesidad de modelos de energía complejos.

Finalmente, el uso de teoría de control para la gestión de la temperatura ya ha sido explorado por Leva et al. (2018) mediante el uso de un esquema más sencillo con un controlador PI. Ellos prueban su propuesta usando ejecuciones de *LINPACK* en un Intel core I5-6600K con un TDP de 91W con una implementación realizada en espacio de usuario. Nosotros vamos a enfocarnos en procesadores móviles de bajo consumo que no tienen demandas tan grandes de energía ni de disipación. Además vamos a realizar el control dentro del kernel del sistema operativo simplificando su uso por los usuarios. Por otra parte, el *governor Intelligent Power Allocator* (Wang, 2017) utiliza un modelo dinámico, basado en PID, de la potencia que puede disipar el procesador sin saltarse los límites de temperatura. Es un control centrado en aplicaciones cortas que no realiza un control directo sobre las acciones que modifican el comportamiento del sistema, como se plantea en este trabajo.

2.1. Gestión de temperatura en Linux

En Linux, el sistema que gestiona la temperatura se conoce como *thermal governor*. Su tarea es muy simple, comprueba la temperatura del dispositivo, y en base a unos puntos de disparo activa distintos mecanismos de enfriamiento activo. En base a estos eventos, este sistema controla por ejemplo, la velocidad de ventiladores o limita la frecuencia máxima.

La Figura 1 muestra un resumen de los elementos del subsistema de temperatura. Por un lado, en el sistema operativo en espacio de usuario tenemos una interfaz para modificar el comportamiento del sistema térmico. Este se encuentra en espacio de *kernel* donde también están los dispositivos de enfriamiento y el *governor*. Por último en el hardware tenemos los sensores y dispositivos mecánicos de enfriamiento.

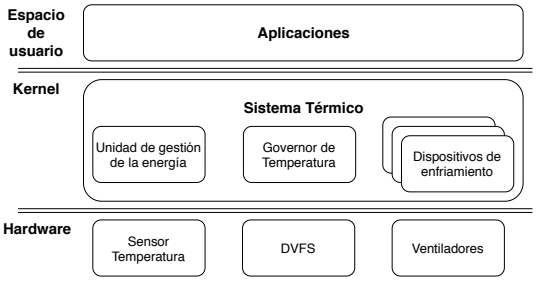


Figura 1: Esquema componentes del subsistema térmico en Linux

El *governor* de temperatura utiliza la temperatura medida para actuar sobre los dispositivos de enfriamiento que, a su vez, puede que controlen elementos mecánicos como ventiladores. Entre las acciones que puede tomar se encuentra limitar las frecuencias disponibles del *governor* de frecuencias. Los principales *governor*s de temperatura son:

- **step_wise** Si la temperatura está por encima del umbral, incrementa la acción del dispositivo de enfriamiento (e.g. baja la máxima frecuencia). Si la temperatura está por debajo del umbral, disminuye la acción del dispositivo de enfriamiento (e.g. incrementa la máxima frecuencia)
- **fair_share** Actua sobre los diferentes dispositivos de enfriamiento en función de unos pesos. Estos pesos definen la contribución de cada dispositivo, pero son estáticos, es decir, no dependen de la carga de trabajo actual en el sistema.
- **user_space** Se limita a informar al espacio de usuario de la temperatura actual y si se sobrepasa ciertos umbrales. El kernel no hace nada por disminuir la temperatura de esta zona térmica y deja que sea el espacio de usuario el que actúe.
- **power_allocator** El governor de Intelligent Power Allocator, calcula la frecuencia máxima disponible en base

un modelo de potencia dinámico y la distribuye entre los diferentes clusteres de CPU y la GPU.

El *governor* de frecuencias es un sistema que se encarga de elegir las frecuencias de trabajo del procesador. La elección se realiza de acuerdo a distintos parámetros, en general se emplea la carga de trabajo del procesador calculado mediante el tamaño de las colas de procesos, y siempre eligiendo entre las frecuencias que se encuentran disponibles.

3. Estrategia de control

En esta sección se describe la metodología seguida para el control de temperatura para una carga de trabajo constante.

3.1. Modelo del sistema

Dada la naturaleza de la computación toda la potencia que se introduce en un sistema se transforma en calor. Esto es debido a que un procesador no es más que un circuito de conmutación y por tanto, el consumo se puede dividir en dos componentes, estático y dinámico,

$$P_{total} = P_{est} + P_{dyn}. \tag{1}$$

El consumo estático es el causado por las fugas producidas en los transistores cuando no están conmutando y puede modelarse mediante

$$P_{est} = I_{sub} V_{dd}, \tag{2}$$

donde I_{sub} es la corriente subumbral y V_{dd} la tensión de alimentación. Por otro lado, la mayor fuente de calor de un procesador es la producida por las conmutaciones, también conocida como potencia dinámica. En un circuito que conmuta constantemente podemos expresar este consumo como,

$$P_{dyn} = \frac{1}{2} C V_{dd}^2 f, \tag{3}$$

donde C es el valor de capacitancia, V_{dd} la tensión de alimentación y f la frecuencia de conmutación.

De acuerdo a (Mudge, 2001), la frecuencia es dependiente de la tensión mediante una relación que se puede aproximar por una función lineal, $f \propto V_{dd}$. Por lo tanto, introduciendo (2) y (3) en (1) podemos aproximar la potencia consumida como

$$P_{total} = I_{sub} V_{dd} + \frac{1}{2} C V_{dd}^2 f \propto I_{sub} f + \frac{1}{2} C f^3 \simeq \alpha f^3, \tag{4}$$

donde en el último paso hemos despreciado el término lineal en f por estar dominado por el término cúbico y hemos agrupado en α la capacitancia y la relación (desconocida) de proporcionalidad entre V_{dd} y f.

Por otra parte, a la hora de disipar el calor generado existen dos variables importantes a tener en cuenta. Por un lado la cantidad de superficie que tengamos disponibles para expulsar el calor, sobre lo cual el diseño del dispositivo es un factor limitante, y por otro lado la temperatura exterior. Cuanta mayor sea la diferencia de temperaturas entre el exterior y el procesador, mejor se podrá disipar el calor. La disipación del calor se rige por la ecuación (Stephanopoulos, 1984),

$$P_{total} = m C_p \frac{dT}{dt} - \frac{T_{ext} - T}{R_t} \tag{5}$$

donde:

- T es la temperatura del procesador ($^\circ C$)
- T_{ext} es la temperatura del aire ($^\circ C$)
- m es la masa total del procesador (Kg)
- C_p es el calor especifico, medido como la cantidad de energía que hay que suministrar a una cantidad de masa para incrementar un grado su temperatura ($\frac{J}{^\circ C Kg}$)
- R_t es la resistencia térmica entre el procesador y el aire, medida como la diferencia de temperatura cuando una unidad de potencia los atraviesa ($\frac{^\circ C s}{J}$)

El primer término de la ecuación describe el procesador como tal, junto con el disipador. El segundo término describe la expulsión al exterior de la temperatura. Para incrementar la disipación se pueden usar técnicas hardware y software. Mediante hardware podemos usar disipadores más grandes para incrementar la masa, así como una mayor superficie para disminuir la resistencia térmica, también se pueden usar mejores materiales para reducir el calor específico. Con el uso de ventiladores lo que haríamos sería incrementar el flujo de aire para poder sustituir el aire caliente cerca del disipador por aire más frío para incrementar la diferencia de temperatura.

Combinando las ecuaciones (4) y (5) obtenemos el modelo dinámico no lineal que rige el comportamiento del sistema, donde T es la variable a controlar, f la variable de control y T_{ext} se entiende como una perturbación del sistema. Linealizando mediante Taylor en un punto (f_0, T_0, T_{ext0}), despreciando las derivadas de orden mayor que 1 se obtiene

$$3\alpha f_0^2 \Delta f = m C_p \frac{d\Delta T}{dt} + \frac{\Delta T_{ext} - \Delta T}{R_t}, \tag{6}$$

donde $\Delta f = f - f_0, \Delta T = T - T_0$ y $\Delta T_{ext} = T_{ext} - T_{ext0}$ representan las variaciones de las variables dinámicas con respecto al punto de linealización.

Aplicando el principio de superposición se obvia de momento la influencia de ΔT_{ext}. Usando la transformada de Laplace se calcula la relación entre las variaciones de la temperatura del procesador $\Delta T(s) = \mathcal{L}[\Delta T]$ y la frecuencia de entrada $\Delta F(s) = \mathcal{L}[\Delta f]$ de la forma

$$3\alpha f_0^2 \Delta F(s) = \left(m C_p s + \frac{1}{R_t} \right) \Delta T(s) \tag{7}$$

Despejando se obtiene la función de transferencia utilizada como modelo de la planta a controlar

$$G(s) = \frac{\Delta T(s)}{\Delta F(s)} = \frac{k_p}{1 + T_p s}, \tag{8}$$

donde $k_p = 3\alpha f_0^2 R_t$ y $T_p = R_t m C_p$.

En resumen, se obtiene que el comportamiento dinámico de la temperatura se puede modelar aproximadamente con una función de transferencia de primer orden con respecto a la frecuencia de trabajo del procesador. La cantidad de parámetros físicos relacionados con la ganancia de la función y su constante de tiempo hace que resulte más fácil e intuitivo realizar una identificación experimental de estos dos parámetros. En la Sección 6.1 se realiza dicha identificación y se muestra la precisión del modelo.

3.2. Esquema de control

El esquema de control escogido se muestra en la Figura 2 en azul (linea discontinua), mientras que en rojo (fotografia dentro de linea continua) se muestra el procesador embebido a controlar. El controlador toma como entradas el valor deseado de temperatura, T^* y la temperatura real medida, T, ambas del procesador. Al haber aplicado linealización, el modelo utilizado solo tiene en cuenta variaciones en la salida con respecto al punto de trabajo por lo que el algoritmo de control está expresado en esos cambios. Esto se realiza restando T_0 a la temperatura deseada y la temperatura real. Entonces, la frecuencia seleccionada es computada por la suma de dos términos, un término de pre-alimentación, ajustado con ganancia $1/k_p$ y un control PID retro-alimentado.

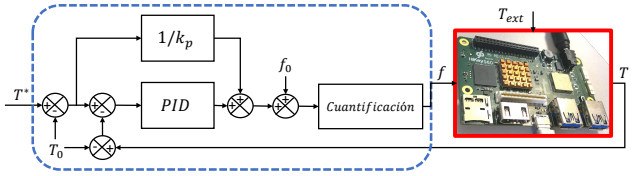


Figura 2: Esquema de control. En línea continua roja se muestra el sistema bajo control y en línea discontinua azul el controlador utilizado.

El término en prealimentación es usado para compensar las variaciones de la temperatura deseada con respecto a las condiciones de trabajo. El controlador PID utiliza las acciones proporcional y derivativa para controlar la dinámica de la temperatura, mientras que la acción integral se encarga de eliminar la influencia de las perturbaciones, así como las posibles imprecisiones en el modelado del sistema, asegurando que T converge a T^* en estado estacionario. El ajuste de estos parámetros se realiza en base al modelo de la ecuación (8), e.g., mediante ajuste sencillo de un PI. A la acción calculada se le añade la frecuencia en el punto de trabajo f_0. Finalmente se aplica al procesador la frecuencia más cercana permitida, seleccionada por el bloque de cuantificación, ya que el procesador únicamente funciona en un conjunto discreto de valores de frecuencia.

4. Planificación de ganancias mediante reconocimiento de programas en tiempo de ejecución

Las constantes del modelo descrito en la ecuación (8) son muy dependientes del tipo de programa ejecutado en el procesador en cada momento[1]. Con el objetivo de dotar de flexibilidad al sistema de control, en esta sección se propone la utilización de un supervisor que identifica la carga de trabajo en función de contadores hardware y permite el ajuste de las ganancias del control en tiempo de ejecución para obtener un mejor rendimiento.

4.1. Funcionamiento del clasificador

A la hora de clasificar las fases de ejecución existen múltiples soluciones, entre las que se incluyen la inserción de marcas en programas, llevar un registro de los programas ejecutados o tener una base de datos con nombres y tipos (Dhodapkar and Smith, 2003; Hamerly et al., 2005; Park et al., 2018). Nuestra propuesta pretende hacer una identificación dinámica, buscando una visión general del comportamiento que tiene el procesador para una mejor adaptación del control. En particular, se plantea una división de los tipos de programas en tres clases, programas de cálculo en coma flotante, programas con operaciones enteras y programas de acceso a memoria.

La Figura 3 muestra el funcionamiento general de la metodología propuesta para la clasificación de los programas en tiempo de ejecución. La clasificación esta basada en una regresión logística multinomial sobre diversos contadores hardware proporcionados por el procesador. Estos han sido el número de instrucciones ejecutadas en total, instrucciones de acceso a memoria, operaciones de tipo entero y operaciones especiales SIMD (*single-instruction multiple-data*), normalmente usadas por programas de cálculo. Se han elegido porque a priori cuentan los eventos relacionados con cada una de las fases que queremos clasificar. Por ejemplo, el contador de instrucciones a memoria puede dar una buena idea de si el programa está haciendo varios accesos a memoria. Estos eventos se cuentan por segundo mediante el módulo creado para ello y se usa un programa automático que va recolectando sus valores.

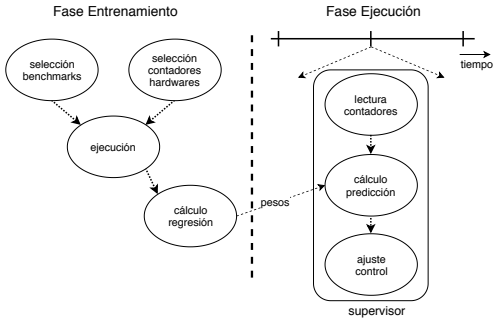


Figura 3: Esquema general de la metodología propuesta

Para obtener los pesos del clasificador se ha aplicado regresión logística multinomial a los valores extraídos de los contadores en la fase de entrenamiento, donde se ejecutan programas característicos de los tipos a clasificar. Conociendo la clase del programa ejecutado, $y_i \in \{\text{flotante, entero, memoria}\}$, y los valores de los contadores hardware durante la ejecución del mismo,

$$X = (\#\text{Instrucciones}, \#\text{enteros}, \#\text{memoria}, \#\text{SIMD}), \quad (9)$$

la fórmula de la regresión es

$$P(Y = y_i \mid X) = \frac{e^{\Sigma \beta_{y_i} X}}{e^{\Sigma \beta_{y_1} X} + e^{\Sigma \beta_{y_2} X} + e^{\Sigma \beta_{y_3} X}}, \quad (10)$$

siendo β_{y_i} los pesos asociados a la clase y_i. Estos pesos se calculan realizando el ajuste del modelo usando los datos de todos los programas de entrenamiento.

[1] En en esta sección, al hablar de tipos programas se hará en sentido amplio y también se referirá a sus fases de ejecución.

4.2. Control Supervisado

Posteriormente durante la ejecución, de manera periódica, se emplean los parámetros calculados junto a los contadores hardware para determinar el tipo del programa en ejecución. Esta metodología no requiere ningún tipo de memoria sobre el estado del sistema y tiene una sobrecarga muy pequeña.

Esta tarea la lleva a cabo un supervisor, encargado también de modificar los parámetros del controlador en función de la clasificación del tipo de programa, ver Figura 4.

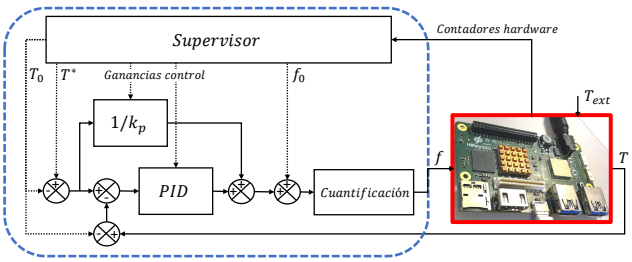


Figura 4: Esquema de control con supervisor. En función de las lecturas de los contadores hardware el supervisor clasifica la fase de ejecución y ajusta las ganancias del control.

La metodología propuesta para el supervisor está basada en planificación de ganancias *Gain Scheduling*. El supervisor puede modificar tanto las ganancias del PID, como los puntos de trabajo. Además, también puede modificar la temperatura objetivo, permitiendo al sistema alcanzar temperaturas más altas en función de la carga de trabajo para mejorar el rendimiento.

5. Metodología

Esta sección describe la plataforma hardware, el software, la clasificación de programas y la identificación de la planta.

5.1. Hardware

Para la evaluación y pruebas del control hemos seleccionado la plataforma HiKey 960, mostrada en la Figura 5. HiKey es una plataforma de desarrollo con un *sistema en chip* (SoC). Este SoC es un *HiSilicon Kirin 960* de 8 núcleos con arquitectura *big.LITTLE* de *ARM*, lo que significa que tiene 4 procesadores de alto rendimiento (Cortex-A73) y 4 de bajo consumo (Cortex-A53), todo ello fabricado con un proceso de 16nm.

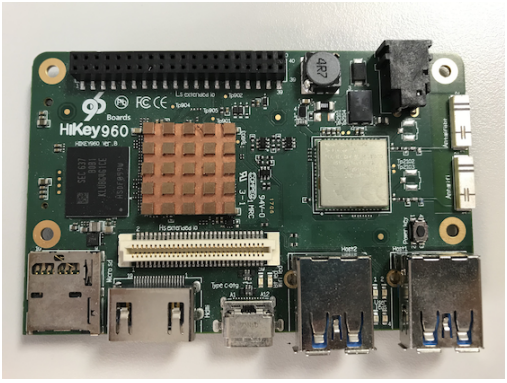


Figura 5: Plataforma de pruebas

Para los experimentos y la evaluación se han utilizado los núcleos de alto rendimiento, porque son los que más calor generan y además permiten realizar operaciones de coma flotante, muy empleadas en sistemas como drones o robots autónomos. La plataforma cuenta además con un pequeño disipador de cobre, como se puede ver en el centro de la Figura 5. Este disipador no es suficiente para disipar el calor generado por los 4 núcleos de alto rendimiento, resultando necesario aplicar el control de temperatura propuesto en el artículo.

5.2. Software

5.2.1. Plataforma

Sobre la plataforma se ejecuta Android 8.0 del proyecto Android Open Source Project (AOSP), en la que todo el software ejecutado es software libre. La posibilidad de ejecutar Android hace que nuestra solución pueda ser usada directamente también en dispositivos móviles. En cuanto al Kernel, usamos Linux en la versión 4.9 modificada para poder ejecutar las aplicaciones sobre un sistema operativo con nuestros módulos de detección de fases y el *governor* de frecuencia.

5.2.2. Implementación

La estrategia de control completa se ha implementado tanto en espacio de usuario como dentro del *kernel* de Linux de AOSP. La implementación en espacio de usuario permite la realización de pruebas individuales de control, útiles para el ajuste de los parámetros sin necesidad de recompilar el módulo del *kernel* con cada modificación. Esta versión del controlador también resulta interesante en otros ámbitos de aplicación, como puede ser la docencia tanto en asignaturas de control como en asignaturas de arquitectura de computadores. Por otro lado, la inclusión del control dentro del *kernel* facilita la transparencia de uso y la facilidad de inserción en sistemas reales sin requerir interacción por parte de los usuarios.

5.2.3. Benchmarks

Para realizar las distintas pruebas y experimentos se han elegido varios programas que buscan simular cargas de trabajo comunes en procesadores embebidos en móviles o robots.También se ha usado un software comercial ampliamente utilizado, que realiza un conjunto variado de test para probar todas las características de los sistemas. Los test escogidos se han agrupado de acuerdo a las tres clases que se pretende que reconozca el clasificador de programas.

Por un lado, tenemos los programas que realizan una gran carga de trabajo en la jerarquía de memoria. Este tipo de programas incluirían servidores de ficheros y bases de datos. Para probar este tipo de ejecuciones usamos los *benchmarks Memcpy* y *Sysbench* (Kopytov). *Memcpy* es un programa sintético que realiza copias de datos entre dos zonas de la memoria principal. *Sysbench* por el contrario es un conjunto de herramientas para la evaluación de sistemas operativos, entre las cuales se encuentra la posibilidad de probar la memoria, opción que hemos escogido.

Por otro lado, tenemos programas orientados al cálculo de operaciones de coma flotante. En este tipo de aplicaciones se incluye el aprendizaje automático, la visión por computador y el análisis de datos. Para su aprendizaje se ha utilizado *Stress-ng* (Canonical), un conjunto de herramientas parecido a *Sysbench* en el cual se ha elegido la opción de estresar la coma

flotante. También se ha querido simular la ejecución de aplicaciones de aprendizaje automático, para lo cual se ha realizado un programa sintético que realiza multiplicaciones de matrices, las cuales son la base de cualquier convolución en una red neuronal. Este programa hace uso de la biblioteca matemática de código abierto *Eigen* (Guennebaud et al.) para ejecutar multiplicaciones de matrices haciendo uso de instrucciones vectoriales SIMD, las cuales pueden realizar varias operaciones de coma flotante a la vez. El programa a su vez es altamente configurable permitiendo la selección de varios hilos de ejecución, su asignación a distintos núcleos, tamaño de matrices y duración de experimentos.

Por último, se han utilizado programas más generalistas que representan una carga más relajada, en la que podemos incluir la mayoría de aplicaciones actuales. Para realizar los experimentos hemos seleccionado también dos programas distintos: *Stress_ng*, como en el caso de coma flotante pero con la opción de solo enteros, y *Dhrystone*. *Dhrystone* es un *benchmark* sintético veterano creado para servir de referencia como carga de trabajo entera para los procesadores.

Además de estas cargas de trabajo sintéticas se ha añadido una conocida aplicación comercial de benchmarking largamente usada en una gran variedad de dispositivos y sistemas. *Geekbench* (LABS) es una de las mayores referencias a la hora de comparar dispositivos. Las distintas pruebas que realiza son aplicaciones reales, como abrir una página web, aplicar filtros a fotografías o medir la latencia de la memoria. *Geekbench* al igual que nuestro trabajo también clasifica los programas dependiendo de si son enteros, de coma flotante o de memoria. *Geekbench* no proporciona información sobre en que momentos se esta ejecutando cada uno de los tests para poder clasificarlos. Para poder usar *Geekbench* como datos de prueba para la clasificación de tipos de programa se ha utilizado una traza que el programa deja en los logs del sistema. Esto es debido a que la aplicación muestra un mensaje por pantalla en el que indica en qué momento se va ejecutando cada aplicación, y Android registra esos cambios. Se ha desarrollado una pequeña herramienta Perl que parsea todos los logs y registros y los sincroniza.

Para ayudar a automatizar todas las pruebas se ha hecho uso del *framework* de ejecución de pruebas y recolección de datos de *ARM* llamado Workload Automation (WA) (arm). Este *framework* permite ejecutar lo que llama agendas, ficheros en formato YAML, donde se describen los experimentos y se modifican los ajustes de configuración de la plataforma. Mediante esto se puede seleccionar la cantidad de núcleos que se usan, la frecuencia de funcionamiento, instrumentación a usar, tiempos de ejecución, etc. Uno de los instrumentos usados es un programa llamado *poller* que permite monitorizar ficheros de sistema y extraer sus valores periódicamente.

5.3. *Identificación del sistema y diseño de los controladores*

La metodología propuesta requiere de un proceso de identificación de la planta y de un ajuste de las ganancias del controlador para cada una de las clases de programas. Durante este proceso se van a utilizar los *benchmarks* de carácter específico, mientras que *Geekbench* será el programa utilizado para la evaluación final del sistema y la comparación con el controlador de estado del arte. Es importante recalcar que todo el proceso se realiza de forma independiente para cada una de las tres clases de programas (entero, memoria y flotante), obteniendo por tanto tres modelos y tres conjuntos de ganancias para el control.

Para la identificación, se ha elegido el punto de linealización $(f_0, T_0) = (1421MHz, 70°C)$ ya que representa valores medios observados empíricamente. Llevando el sistema a dicho punto, para la identificación de las constantes del modelo en la Ecuación (8) se han aplicado escalones positivos y negativos en la frecuencia (entrada a la planta) midiendo la temperatura en todo momento para finalmente utilizar la Toolbox de identificación de sistemas de MatLab.

Para el ajuste de las ganancias del controlador, empíricamente se ha observado que en el hardware empleado, un controlador PI es capaz de obtener resultados aceptables sin necesidad de utilizar la acción derivativa. Al considerar un modelo de primer orden, la técnica de asignación de polos permite calcular el resto de ganancias de forma sencilla. En concreto, en una primera etapa se ha utilizado ajuste sencillo, cancelando el polo de la planta con el cero del PI, y ajustando la ganancia del regulador para obtener el transitorio deseado. Posteriormente, se ha buscado una mejora en el rendimiento aumentando el valor del cero empíricamente hasta obtener un comportamiento lo más rápido posible sin sobreoscilación.

6. Resultados

Esta sección describe los experimentos realizados y los resultados observados.

6.1. *Identificación del sistema*

Empezamos mostrando la caracterización del modelo planteado. Para ello hemos ejecutado los distintos *Benchmarks* de entrenamiento con escalones unitarios de frecuencia mantenidos en el tiempo para poder comprobar como es el comportamiento tanto dinámico como en régimen permanente. Al ejecutar estos *Benchmarks* por separado hemos podido obtener diferentes medidas de como se comporta la frecuencia y la temperatura.

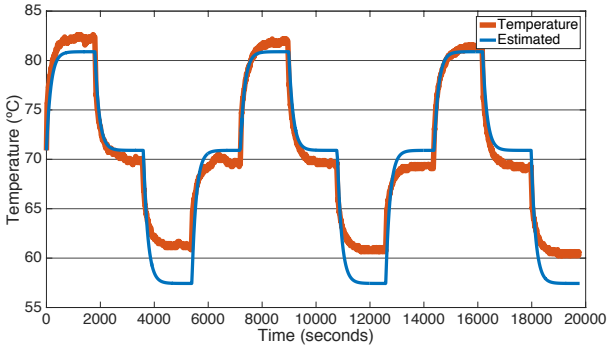


Figura 6: Caracterización *benchmark* de matrices.

Con los valores obtenidos, se ha elegido como punto de trabajo $(f_0, T_0) = (1421MHz, 70°C)$, escogido tras varias ejecuciones de prueba donde se observaban la relación entre ambos valores y considerando que $70°C$ es un valor de temperatura común y que suele poder ser mantenido de manera indefinida. Con estos datos se han identificado los parámetros de la ecuación (8) para cada

uno de los tipos de ejecución. En la Figura 6 se muestra el resultado superponiendo ambos comportamientos para el *benchmark* de matrices, en rojo el real y en azul el simulado. Se puede ver el comportamiento dinámico y estacionario de la temperatura para cada frecuencia de trabajo y las inmediatamente anterior y posterior. Las diferencias se deben a la no linealidad existente entre frecuencia y temperatura pero, como el control debe centrarse en controlar las temperaturas más altas que son las que pueden causar ahogamiento térmico, el ajuste es suficientemente bueno como para darlo como válido con solo un 2 % de error.

Los resultados para las diferentes clases de programas se resumen en la Tabla 1. Se observa que el comportamiento dinámico para cada clase de programa es sustancialmente diferente, lo que motiva el uso del supervisor en el control. Los programas de memoria tienen una dinámica más lenta y una ganancia más pequeña que los programas de coma flotante. Se ve también como el ajuste es bastante bueno para los tres casos, disminuyendo en el caso de la memoria, al ser computacionalmente menos intensa.

Tabla 1: Identificación experimental de parámetros

Clase de programa	k_p	T_p	Fit %
Flotante	$2.21 \cdot 10^{-5}$	133	78.02
Entero	$1.74 \cdot 10^{-5}$	164	78.96
Memoria	$1.92 \cdot 10^{-5}$	234	71.90

Con el objetivo de validar la complejidad del modelo propuesto, se ha intentado realizar un ajuste de los datos a funciones de transferencia de orden superior al propuesto, incluyendo la presencia de un retraso en el modelo. En la Tabla 2 se muestra el resultado de dicho ajuste para el caso de programas flotantes, observando que el valor de los polos adicionales se encuentra claramente dominado por el primer polo, por lo que no se tienen en consideración.

Tabla 2: Ajuste de diferentes funciones de transferencia para la clase Flotante

Polos	k_p	T_{p1}	T_{p2}	T_{p3}	T_d	Fit %
1	$2.21 \cdot 10^{-5}$	133	0.0	0.0	0.0	78.02
2	$2.21 \cdot 10^{-5}$	133	0.049	0.0	0.0	78.02
3	$2.21 \cdot 10^{-5}$	133	0.0036	0.007	0.0	78.02

6.2. *Análisis de contadores*

Para el análisis de contadores hemos usado las herramientas explicadas en la Sección 5: el lector de contadores vía el pseudo-sistema de ficheros *sysfs* y el módulo de adquisión de estadísticas de uso de los contadores escogidos. Para una clasificación precisa y rápida, hemos utilizado 4 contadores; el número de instrucciones ejecutadas en total, instrucciones de acceso a memoria, operaciones de tipo entero y operaciones especiales SIMD (single-instruction multiple-data) normalmente usadas por programas de cálculo.

Los experimentos realizados han consistido en la ejecución de los *benchmarks* específicos para cada clase de programa con distintas configuraciones del procesador, tanto de frecuencia como de número de núcleos usados. En la Tabla 3 se incluye el porcentaje de uso de cada una de las instrucciones especializadas relativo a las instrucciones totales. Los valores mostrados representan la media de varias ejecuciones de los *benchmarks*, ejecutados de manera individual con la intención de obtener valores realistas de la ejecución de un programa.

En los programas de la clase Memoria dominan las instrucciones de acceso a memoria, representando aproximadamente el 80 % del total de instrucciones, lo que ayuda en su clasificación. En todas las categorías, el porcentaje de SIMD es pequeño, incluso 0 % en memoria, porque estas instrucciones realizan operaciones con datos vectoriales y consiguen que una única instrucción SIMD sea equivalente a múltiples instrucciones escalares. El resultado es su aportación pequeña al total, aunque el procesador dedique muchos ciclos a su ejecución. Además, el porcentaje de instrucciones enteras es mayor en la clase Flotante porque estos programas suelen requerir de múltiples operaciones con índices enteros para después acceder a memoria. La clase Enteros también suelen ejecutar mas instrucciones de control, lo que hace que el porcentaje de instrucciones enteras baje en su caso y permite ayudar a discernir si un programa ejecuta muchas instrucciones de control que ralentizarán la ejecución y tenderán a bajar la temperatura.

Tabla 3: Resumen contadores por tipo de programa, número de cores y frecuencia

Clase	Núcleos	f MHz	%Mem	%Int	%SIMD
Flotante	1	903	12.22	69.49	0.55
Flotante	1	1421	9.63	70.80	0.41
Flotante	2	903	9.57	68.94	0.56
Flotante	2	1421	9.64	71.34	0.42
Entero	1	903	14.68	51.07	0.58
Entero	1	1421	12.64	54.80	0.43
Entero	2	903	20.20	52.50	0.59
Entero	2	1421	10.10	52.94	0.43
Memoria	1	903	79.70	10.24	0.00
Memoria	1	1421	79.80	10.17	0.00
Memoria	2	903	79.71	10.24	0.00
Memoria	2	1421	79.80	10.18	0.00

Tabla 4: Resumen precisión regresión

	Precisión (%)	
Variables	Entrenamiento	Test
% todas	79.5	33.88
% por instrucciones	79.3	51.97
% memoria y SIMD	79.3	43.03
% memoria y enteros	79.6	58.62

Para realizar la regresión logística multinomial, descrita en la Sección 4, hemos empleado el software de análisis estadístico *R* con los datos de los *benchmarks* específicos, mientras que hemos utilizado *Geekbench* como test para verificar la clasificación, donde el etiquetado del *ground truth* se ha hecho a nivel de programa, al no disponer del código fuente de cada fase de ejecución para poder realizar un análisis más preciso. Los ajustes se han probado para cuatro configuraciones: usando todos los contadores, usando el porcentaje de uso por instrucciones, usando memoria y SIMD, y memoria y enteros. Los resultados pueden observarse en la Tabla 4, donde se muestra la precisión

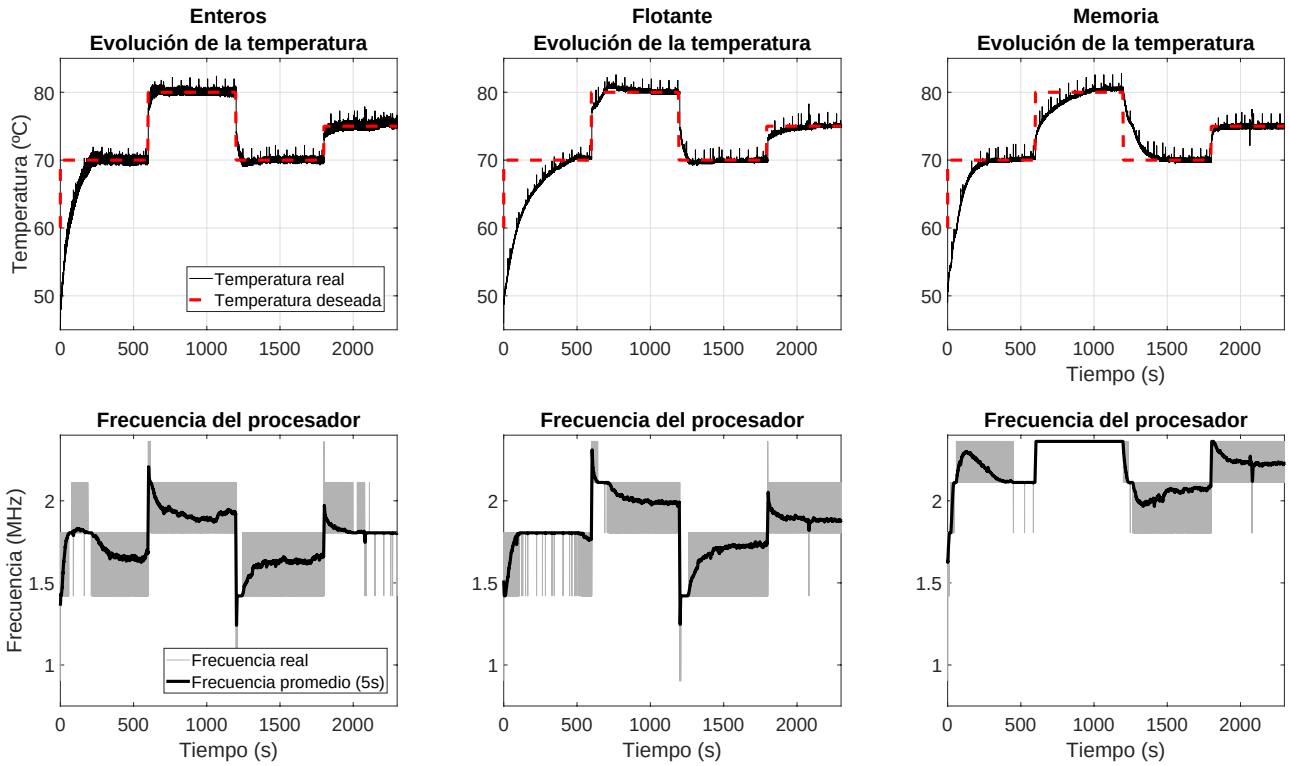


Figura 7: Pruebas de control de las distintas clases de programa. La primera fila muestra la evolución de la temperatura real en el procesador (línea sólida negra) y la temperatura objetivo (línea de rayas roja). En la segunda fila se muestra la frecuencia aplicada al procesador por el sistema de control (gris). Las figuras también incluyen la media móvil de la frecuencia en un intervalo de 5 segundos (negro).

obtenida con cada uno de los dos conjuntos de datos, entrenamiento y test. Usar una medida independiente de la velocidad a la que funciona el procesador, segunda fila, hace que mejore la precisión en los datos de test. Por el contrario, la utilización de todos los contadores devuelve peores resultados.

Teniendo en cuenta estos resultados, para las pruebas de control supervisado se utilizará la clasificación que considera las instrucciones de memoria y enteros por instrucción ejecutada, cuarta fila, por ser la que mejor precisión consigue, además de simplificar el cálculo de la regresión, utilizando solo dos variables.

6.3. *Pruebas de control individuales*

En este apartado se analiza el comportamiento del sistema de control. La Figura 7 muestra los resultados de control individuales para cada clase de programa evaluadas con los *benchmarks* específicos, utilizando únicamente las ganancias de control ajustadas para la propia clase. La primera fila muestra la evolución de la temperatura en el tiempo (línea negra continua), incluyendo cambios en la temperatura de consigna (línea de rayas roja). Se observa que el control responde adecuadamente en las tres situaciones. La segunda fila muestra las frecuencias aplicadas en cada periodo de muestreo (línea gris). Al disponer de un número limitado de frecuencias el controlador necesita realizar gran cantidad de cambios en la acción para controlar el sistema. En las figuras se incluye también en línea negra la frecuencia promedio aplicada en los últimos 5 segundos con el objetivo de visualizar mejor la frecuencia efectiva de trabajo del procesador. Un caso interesante se produce en la clase *Memoria* cuando el sistema de control necesita reducir la temperatura deseada. Al utilizar activamente una mayor superficie de la placa (la zona de memoria) la disipación de energía es más lenta, por lo que la diferencia en el comportamiento dinámico entre calentamiento y disipación sea mayor que en los otros casos.

6.4. *Pruebas de control con el supervisor*

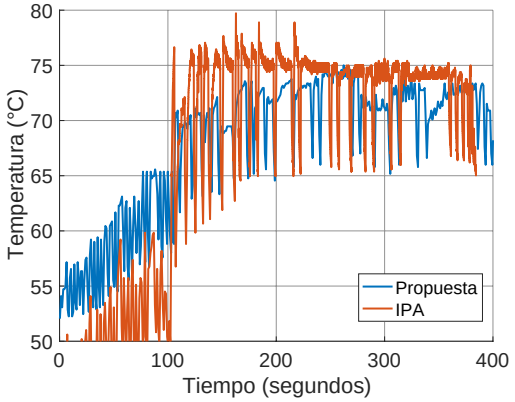


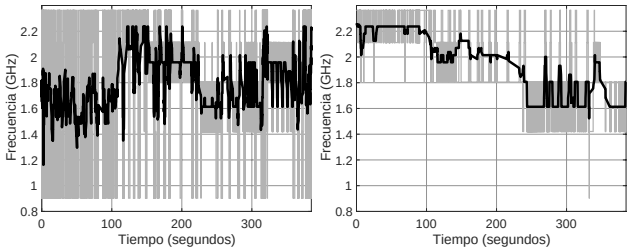
Figura 8: Comparación Temperaturas *Geeckbench*.

Para terminar se realiza una comparación entre IPA y nuestra propuesta de control completa en la ejecución de *Geekbench*. Este *benchmark* realiza una carga de trabajo variada e intensa, siendo necesario el supervisor para determinar las ganancias de

control adecuadas en cada momento. En la Figura 8 se muestra la evolución de la temperatura a lo largo de la ejecución usando ambos controles.

Tabla 5: Comparación de IPA y nuestra propuesta ejecutando Geeekbench

Medidas	IPA	Propuesta	Reducción (%)
Puntos unicore	1894	1766	-6.8
Puntos multicore	2690	2365	-12.1
Tª media ($°C$)	73.2	67.8	-7.2
Cambios frec.	2600	1000	-61.5
Desviación (MHz)	392.8	312.3	-20.5



(a) IPA (b) Propuesta

Figura 9: Comparación de frecuencias ejecutando *Geekbench*. En gris se muestra la frecuencia instantánea y en negro la media móvil de los últimos 5 segundos

Como se puede ver al comienzo de la ejecución, el control permite la utilización de frecuencias más altas para alcanzar la temperatura de consigna. Mas tarde, pese a que aparecen oscilaciones causadas por la alta variabilidad de las pruebas, se consigue mantener una temperatura inferior a la de IPA. La Tabla 5 muestra los resultados de rendimiento obtenidos, donde destaca una reducción del 61.5 % en los cambios de frecuencia, a pesar de obtener una puntuación en el test de *Geekbench* ligeramente inferior.

En la Figura 9 se muestra la comparación de las frecuencias seleccionadas por ambos controladores. La gran cantidad de cambios de frecuencia se debe a la separación entre el *governor* térmico y el de frecuencia, y la sobrecarga que esto supone. El *governor* térmico selecciona una frecuencia máxima y activa al *governor* de frecuencia, que selecciona la frecuencia en base a la carga de trabajo. Esto puede generar que en ocasiones no se llegue a tiempo a acelerar la ejecución, o que la reducción de frecuencia debida a alarmas térmicas sea demasiado alta.

También es interesante observar en la Figura 10 las diferentes secciones que *Geekbench* ejecuta y que el supervisor clasifica para seleccionar el mejor tipo de control en cada momento. Cada una de estas fases tiene un comportamiento térmico distinto tal y como mostramos en la sección 6.3, por lo que la habilidad de poder modificar el comportamiento del control mejora el comportamiento durante toda la ejecución.

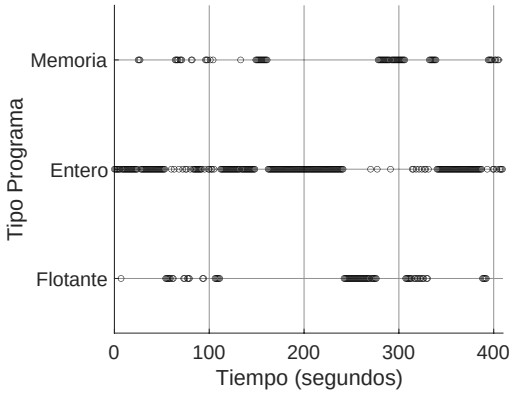


Figura 10: Clasificación de la ejecución que realiza el supervisor durante la ejecución de *Geeckbench*.

Geekbench al tratarse de un *benchmark* con una carga muy variable no permite ver las diferencias de usar un control formal y no es totalmente representativo de cargas más homogéneas más habituales en coches autónomos, robots, etc., por lo que también se ha realizado la comparación entre propuestas utilizando el *benchmark* de multiplicación de matrices. En la Tabla 6 se muestra la media de instrucciones totales y SIMD ejecutadas por segundo usando IPA y nuestra propuesta de control. Dado que IPA está configurado para limitar la ejecución a $75°C$ hemos usado nuestro control con dos temperaturas de consigna, 72 y $75°C$. Se puede observar que con la misma limitación, nuestra propuesta ofrece un rendimiento superior. Esto es debido a los mecanismos que entran en funcionamiento en cada propuesta. IPA al ser un *governor* térmico, realiza cálculos de estimación de energía cada vez que la temperatura cambia. Posteriormente, el *governor* de frecuencia modifica la frecuencia a la deseada o máxima permitida. Por el contrario, nuestra propuesta realiza los cálculos y modifica la frecuencia simultáneamente en cada periodo de muestreo siendo adecuada para cargas más homogéneas en procesadores embebidos.

Tabla 6: Comparación de IPA y propuesta ejecutando el *benchmark* de Matrices

	Inst. ($x10^9$)	Inst. SIMD ($x10^9$)
IPA	2.12	1.39
Propuesta $T^* = 75°C$	2.26	1.48
Propuesta $T^* = 72°C$	2.03	1.33

7. Conclusiones

Este trabajo presenta un estudio completo de un control supervisado para la gestión de la temperatura mediante el escalado dinámico de la frecuencia del procesador/es en sistemas embebidos. Además, se presenta el modelado de la plataforma y el diseño del control. Para adaptar el control a las fases de ejecución de los programas, el modelo propone emplear regresión logística multinomial sobre contadores *hardware* para la detección automática de las fases.

Los resultados demuestran la utilidad de adaptar el control en base a las fases de ejecución, permitiendo un mejor comportamiento y posibilidad de ajuste. En comparación con los sistemas estándar de gestión de temperatura, se ha visto que en pruebas

con una alta variabilidad de carga como *Geekbench*, la propuesta es capaz de mantener la temperatura en un rango más bajo con una pérdida de rendimiento de 7 %. Con cargas de trabajo más homogéneas se han conseguido *speed-up* de hasta el 6 %.

Todo el código utilizado se encuentra disponible en un repositorio público de Github (Hernández).

Agradecimientos

Este artículo ha sido financiado parcialmente por los proyectos PGC2018-098817-A-I00 (MCIU/AEI/FEDER, UE), MIG-20201006, PID2019-105660RB-C21 (MINECO/AEI/FEDER), Grupos T58_20R y T45_20R (Gobierno de Aragón) y FEDER 2014-2020 "Construyendo Europa desde Aragón". Los autores agradecen el apoyo recibido.

Referencias

Alastruey, J., Briz, J.L., Ibanez, P., Vinals, V., 2006. Software demand, hardware supply. IEEE Micro 26, 72–82.

arm, . Workload automation. URL: https://github.com/ARM-software/workload-automation.

Brooks, D., Martonosi, M., 2001. Dynamic thermal management for high-performance microprocessors, in: Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture, IEEE. pp. 171–182.

Canonical, . Stress-ng. URL: https://wiki.ubuntu.com/Kernel/Reference/stress-ng.

Chen, H., Han, Y., Tang, G., Zhang, X., 2020. A dynamic control system for server processor direct liquid cooling. IEEE Transactions on Components, Packaging and Manufacturing Technology 10, 786 – 794.

Cohen, A., Finkelstein, F., Mendelson, A., Ronen, R., Rudoy, D., 2003. On estimating optimal performance of cpu dynamic thermal management. IEEE Computer Architecture Letters 2, 6–6.

Deng, Q., Meisner, D., Bhattacharjee, A., Wenisch, T.F., Bianchini, R., 2012. Coscale: Coordinating cpu and memory system dvfs in server systems, in: MICRO, pp. 143–154.

Dhodapkar, A.S., Smith, J.E., 2003. Comparing program phase detection techniques, in: Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, IEEE Computer Society, Washington, DC, USA. pp. 217–.

Guennebaud, G., Jacob, B., et al., . Eigen v3. URL: http://eigen.tuxfamily.org.

Hamerly, G., Perelman, E., Lau, J., Calder, B., 2005. Simpoint 3.0: Faster and more flexible program phase analysis. Journal of Instruction Level Parallelism 7, 1–28.

Hernández, P., . spider. URL: https://github.com/Pablololo12/sPIDer.

Isci, C., Buyuktosunoglu, A., Cher, C.Y., Bose, P., Martonosi, M., 2006. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget, in: MICRO, pp. 347–358.

Kopytov, A., . Sysbench. URL: https://github.com/akopytov/sysbench.

LABS, P., . Geekbench. URL: https://www.geekbench.com/.

Leva, A., Terraneo, F., Giacomello, I., Fornaciari, W., 2018. Event-based power/performance-aware thermal management for high-density microprocessors. IEEE Transactions on Control Systems Technology 26, 535–550.

López, M.G., Ponce, P., Soriano, L.A., Molina, A., Rodriguez, J.J., 2019. Mejora de la vida Útil en los módulos de electrónica de potencia de un bldcm mediante la optimización de un control difuso. Revista Iberoamericana de Automática e Informática industrial 16, 66–78.

Madridano, A., Campos, S., Al-Kaff, A., García, F., Martín, D., Escalera, A., 2020. Vehículo aéreo no tripulado para vigilancia y monitorización de incendios. Revista Iberoamericana de Automática e Informática industrial 17, 254–263.

Maggio, M., Hoffmann, H., Santambrogio, M.D., Agarwal, A., Leva, A., 2010. Controlling software applications via resource allocation within the heartbeats framework, in: 49th IEEE Conference on Decision and Control (CDC), pp. 3736–3741.

Mudge, T., 2001. Power: a first-class architectural design constraint. Computer 34, 52–58. doi:10.1109/2.917539.

Park, J., Lee, S., Cha, H., 2018. App-oriented thermal management of mobile devices, in: ISLPED, pp. 36:1–36:6.

Pothukuchi, R.P., Ansari, A., Voulgaris, P., Torrellas, J., 2016. 2016 acm ieee 43rd annual international symposium on computer architecture (isca), in: ISCA, pp. 658–670.

Pothukuchi, R.P., Pothukuchi, S.Y., Voulgaris, P.G., Torrellas, J., 2020. Control systems for computing systems: Making computers efficient with modular, coordinated, and robust control. IEEE Control Systems Magazine 40, 30–55.

Rahmani, A., Haghbayan, M., Kanduri, A., Weldezion, A.Y., Liljeberg, P., Plosila, J., Jantsch, A., Tenhunen, H., 2015. Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach, in: ISLPED, pp. 219–224.

Rahmani, A.M., Donyanavard, B., Mück, T., Moazzemi, K., Jantsch, A., Mutlu, O., Dutt, N., 2018. Spectr: Formal supervisory control and coordination for many-core systems resource management, in: ASPLOS, pp. 169–183.

Rajkumar, R., Lee, I., Sha, L., Stankovic, J., 2010. Cyber-physical systems: The next computing revolution, in: Design Automation Conference, pp. 731–736.

Stephanopoulos, G., 1984. Chemical process control: an introduction to theory and practice.

Wang, X., 2017. Intelligent Power Allocator. Technical Report. ARM.

Xu, G., 2007. Evaluation of a liquid cooling concept for high power processors, in: Twenty-Third Annual IEEE Semiconductor Thermal Measurement and Management Symposium, pp. 190–195.

Yueh, W., Wan, Z., Joshi, Y., Mukhopadhyay, S., 2015. Experimental characterization of in-package microfluidic cooling on a system-on-chip, in: 2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), pp. 43–48. doi:10.1109/ISLPED.2015.7273488.