Documento de investigación

Proyecto GENIUS

GENIUS CONSUMPTION METER: SOFTWARE AND DESIGN

Autor: Jesús Morales Millán,
Javier Jesús Villaverde Ramallo

Cádiz, 4 de marzo de 2022

Índice

Índice de figuras

Índice de cuadros

1. Introducción

El proyecto GENIUS nació en el año 2020 en el seno de la Universidad de Cádiz bajo la dirección del grupo de investigación GOAL. Su objetivo es el de realizar una investigación formal orientada en la reducción del consumo energético del hardware mediante la optimización del software.

En los últimos años, hemos asistido a una explosión de desarrollo y el surgimiento cada vez más de una gran cantidad heterogénea de dispositivos hardware cada uno de ellos con una arquitectura diferente y orientados a distintas necesidades, lo que dificulta la concreción homogénea de un único conjunto de técnicas que permita reducir el consumo energético mediante software. Esta diversidad de hardware hace necesario realizar una exhaustiva experimentación de la que poder extraer una huella de consumo propia de cada arquitectura basada en su propio conjunto de instrucciones. A partir de ella, en una posterior etapa de optimización energética, se podrá seleccionar de forma autónoma qué tipo de operaciones son más eficientes para cada tipo de arquitectura.

Para establecer la huella de consumo de cada hardware es necesario idear una serie de programas básicos que sean ejecutables de forma independiente a la arquitectura. Este conjunto de programas deberá ser neutro con respecto a la arquitectura y la aplicación de las futuras técnicas.

1.1. Motivación

En el día de hoy, en vista a los estudios consultados, hemos visto como existen múltiples estudios que estudian el consumo energético, para posteriormente tratar de realizar distintos objetivos. Para ello, utilizan un monitor, un dispositivo que les permite medir el consumo energético. Sin embargo, estos estudios no suelen ser muy detallados en este aspecto, o adquieren un producto de una marca u empresa, o bien acceden a información que el sistema operativo ofrece, si procede al estudio. Algunos de estos monitores incluso suponen un considerable coste económico, o incluso tienen baja o difícil disponibilidad, además de problemas de envío en función de a qué lugares.

Debido a estos hechos, se pretende definir un equipo de medición con elementos de alta disponibilidad, así como una metodología flexible que permita cambiar de un dispositivo a otro de manera fácil, y que este proceso de cambio venga detallado en nuestra documentación. Además, se pretende que sea fácilmente adaptable, permitiendo incluir nuevas funcionalidades. Asimismo, se sigue y se pretende que este equipo tenga un bajo coste económico, y ya que se puede cambiar de dispositivo, este es personalizable a las necesidades del usuario.

Junto con el diseño de la metodología, se pretende incluir una suites de benchmarks, de forma que se tengan todos los elementos necesarios para hacer una medición de consumo sobre el equipo. Con estos, posteriormente, se podrían

hacer otros análisis, como bien puede ser la optimización energética.

2. Objetivo

El objetivo a seguir es la definición de una suite de benchmarks que nos permita medir de forma estándar el consumo energético y con el que poder comparar a futuro la optimización energética conseguida. De esta manera, se estudiarán las cuestiones que vayan surgiendo conforme la definición de este avanza.

Asimismo, se hará hincapié en el equipo de medida usado, así como otras cuestiones que surjan derivadas de su uso, con el fin de especificar cómo se ha preparado el mismo para el uso de la suite, así como servir de guía de cómo utilizar el mismo con propósitos de medición.

3. Estado del arte

Para llevar a cabo nuestro objetivo, se ha realizado una búsqueda de los trabajos previamente realizados que puedan abordar de alguna manera nuestra meta. En este sentido, se han consultado tanto suites de benchmarks, de cara a construir la nuestra propia, así como distintos elementos de medición y/o minimización de consumo, ya bien sean hardware o software.

3.1. Suites de benchmarks

Se han consultado algunas suites de benchmarks de uso múltiple, como veremos a continuación, que podrían servir de referencia para la construcción de uno propio.

3.1.1. BEEBS

BEEBS[12] es un benchmark diseñado por la Universidad de Bristol diseñado específicamente para su uso en distintos tipos de hardware, entre los que se incluyen, principalmente, los sistemas embebidos. Este, a su vez, incluye otros ya conocidos, como bien puede ser MiBench, modificados para su uso en estos sistemas. Su fin es la observación del consumo energético del dispositivo.

La modificación de estos benchmark se debe a que la mayoría de estos parten de la base de que se tienen determinados elementos tales como la existencia de un sistema operativo o de un sistema de almacenamiento, de los cuales un sistema puede prescindir. De este modo, la ejecución de estos a priori supone un problema en esas circunstancias.

Los cuatro principales aspectos a cubrir por el benchmark son:

- Realización de operaciones con enteros.
- Realización de operaciones de coma flotantes.
- Intensidad de acceso a memoria.
- Presencia de ramificación en el código (ej.: if-else).

De cara al análisis energético, las características y capacidades de cada sistema también tiene gran importancia. Por ejemplo, un benchmark tendrá un comportamiento diferente en un sistema con caché comparado con uno sin. De este modo, un conjunto de plataformas es necesario para complementar nuestras pruebas.

Siguiendo en esta línea, el número de registros tiene un gran efecto en el consumo de energía debido al alto coste de acceso a memoria. Si una variable puede almacenarse en registro, ahorraremos consumo energético al no usar la

memoria. Por razones similares, el tipo de memoria en la que se ejecuta el código puede tener un gran impacto en la energía, la memoria SRAM y la flash tienen consumos diferentes.

Para seleccionar los benchmark que forman BEEBS, se analizó un conjunto de estos en función de su compatibilidad con la caracterización del consumo energético en sistemas embebidos, las características que ofrece y la dependencia del sistema operativo.

Más tarde, sus autores señalan que, concretando en esta línea, dos parámetros han sido evaluados. El primero de ellos es la adecuación de su inclusión, basándose en lo que hace y si funciona en la arquitectura final, además de tener en cuenta el esfuerzo de "traslado/traducción" a esa arquitectura.

El otro parámetro es el tipo de operaciones que el benchmark tiene. Este se calcula observando el código fuente y categorizando las operaciones en: entero, punto flotante, memoria, y ramificación (los puntos antes mencionados).

Estas propiedades nos permiten comparar las propiedades de los distintos benchmark.

Finalmente, se escogieron para formar parte de este benchmark los que se adecúan correctamente y que tienen el menor conjunto de instrucciones.

3.1.2. Embench

Embench[7] es una suite de benchmarks adecuados para ejecutarse en sistemas embebidos que dispongan al menos de 64KB de ROM y 64KB de RAM. Sus benchmarks se derivan en gran medida de la Bristol/Embecosm Embedded Benchmark Suite (BEEBS), aunque se han añadido otros adicionales para cumplir con el objetivo de proporcionar una gran variedad de programas.

Debido a que el mercado de los dispositivos embebidos y su uso es de muy rápido cambio, las suites de benchmarks para estas plataformas deben adaptarse constantemente al desarrollo actual. Esto hace que sea necesario revisar los benchmarks incluidos y, si procede, eliminar o añadir características que muestren una carga de trabajo realista en condiciones recientes. Lamentablemente, las suites de referencia más utilizadas en la actualidad como BEEBS y CoreMark, no se han revisado desde 2013 y 2009 respectivamente, lo que disminuye su capacidad de reflejar una carga de trabajo realista para una plataforma determinada en la actualidad. Aun así, BEEBS ofrece un repertorio de programas de código abierto y fáciles de adaptar a otras arquitecturas. Por ello, la organización detrás de Embench escogieron sus más de 80 programas en función de:

- Las funcionalidad de estos

- La capacidad de carga que supone al procesador, la carga de memoria, y el nivel de branching.
- El tamaño de los programas.
- El tiempo de ejecución de cada programa.

Teniendo en cuenta estos términos, se escogieron los 20 programas que cubrirían el mayor número posible de aplicaciones distintas. Para los programas que faltan en ciertas categorías, como el cifrado SHA256, añadieron los ejemplos más adecuados de programas libres y abiertos disponibles.

Embench reporta una única puntuación de referencia resumida que se basa en el uso de la media y la desviación estándar. La puntuación otorgada puede darnos información sobre el rendimiento, el tamaño del programa ya compilado, la latencia de las interrupciones y el tiempo de cambio de contexto. Todos estos valores desempeñan un papel importante para los dispositivos IoT, ya que reflejan aspectos importantes para los fabricantes y los desarrolladores. Por ejemplo, la memoria necesaria para almacenar el código en un dispositivo IoT es un factor de coste importante para los fabricantes.

En resumen, Embench ofrece la misma variedad de benchmarks y fácil portabilidad que BEEBS, siendo al mismo tiempo ligero y actualizado. Involucrar tanto al mundo académico como a la industria en el desarrollo de los benchmarks ayuda a garantizar que los programas sean realmente importantes y se utilicen en entornos del mundo real.

Seagate Technology Inc. por ejemplo, hace patente la utilidad del Embench en un entorno real, el cual utiliza para medir el rendimiento de sus núcleos RISC-V de alto rendimiento. [8]

3.1.3. EEMBC

EEMBC[4] son las siglas de Embedded Microprocessor Benchmark Consortium, se trata de una organización sin ánimo de lucro centrada exclusivamente en el desarrollo de benchmarks bajo el amparo de un relevante número de empresas líder en la fabricación de este tipo de componentes electrónicos. Estos benchmarks están destinados a extraer información del rendimiento de cada dispositivo de forma estándar e industrial para permitir a los fabricantes comparar distintos dispositivos dedicados a la realización de la misma actividad.

Los benchmarks se agrupan bajo diferentes categorías, cada una centrada en un propósito diferente. A continuación se enumeran las más relevantes:

Bajo consumo e Internet de las Cosas
Este conjunto de benchmarks está centrado en la medición de consumo energético y rendimiento. La puntuación asociada con los benchmarks es el resultado de las mediciones tomadas por el STMicroelectronics PowerShield. Lo componen a su vez las siguientes familias de benchmarks [3]:

- ULPMark™: La medición se realiza en base a perfiles de comportamiento, esta familia de benchmarks está compuesta por los siguientes:
  - ULPMark-CoreProfile: Consumo en estado deep-sleep absoluto.
  - ULPMark-PeripheralProfile: Consumo en estado deep-sleep con los periféricos activos.
  - ULPMark-CoreMark: Consumo en actividad, utilizando CoreMark como carga de trabajo.
- IoTMark™: Se trata de un conjunto de benchmarks estandarizados para medir la eficiencia energética de las motas IoT, entendidas como dispositivos compuestos de tres partes: un sensor, un procesador y una interfaz inalámbrica. Lo componen dos benchmarks:
  - IoTMark-BLE: Para dispositivos Bluetooth Low Energy (BLE).
  - IoTMark-Wi-Fi: Para dispositivos con radio Wi-Fi (actualmente en desarrollo).
  - SecureMark™: Se trata de un framework para medir el impacto energético de las operaciones criptográficas requeridas por el protocolo Transport Layer Security (TLS), ampliamente utilizado para establecer comunicaciones en Internet.

Medición de rendimiento en procesadores mono-núcleo

- CoreMark®: Benchmark diseñado específicamente para testar la funcionalidad de un sólo núcleo de procesamiento. El resultado de la ejecución es un valor numérico que permite al usuario establecer comparaciones inmediatas entre diferentes procesadores.
- AutoBench™: Suit de benchmarks que permiten al usuario predecir el rendimiento de microprocesadores y microcontroladores industriales, automotores y de propósito general a través de una serie de cargas de trabajo predefinidas.
- DENBench™ (Digital Entertainment): Permite al usuario atisbar el rendimiento de sistemas al realizar tareas multimedia como compresión y descompresión de archivos de imágenes, vídeos y audio. También añade soporte al cifrado y descifrado típicamente utilizados en aplicaciones eCommerce (comercio electrónico) y de administración de derechos digitales (DRM, Digital Rights Management).
- Networking 2.0: Destinado a procesadores encontrados típicamente en conmutadores y encaminadores simulan de forma realista un denso tráfico de paquetes de comunicación en Internet a fin de poner a prueba el rendimiento de este tipo de dispositivos.

- OABench™ (Office Automation): Permite comparar el rendimiento de procesadores embebidos en impresoras, trazadores gráficos, y otros sistemas de automatización de tareas ofimáticas dedicados al manejo de texto e imágenes.
- TeleBench™: Centrado en el rendimiento de procesadores en módems y otras aplicaciones de telefonía fija.

Medición de rendimiento simétrico en procesadores multi-núcleo
Estas suits de benchmarks están basadas en el test de estrés multi-instancia (MITH, Multi-Instance Test Harness), que utiliza la instancia POSIX pthreads para poner a prueba el paralelismo de nivel de contexto (context) y de trabajadores (workers).

- CoreMark®-Pro: Construido a partir del benchmark CoreMark, pone a prueba todo el procesador mediante una combinación de cargas de trabajo en entero y coma flotante así como de grandes conjuntos de datos para sistemas que empleen una gran cantidad de memoria.
- AutoBench™-2.0: Implementación paralela del benchmark AutoBench que añade cargas de trabajo adicionales para incrementar la demanda de computación en unidades de control electrónicas de automóviles (ECU, Electronic Control Units).
- FPMark™: Dedicado a un intenso análisis de operaciones en coma flotante de simple y doble precisión en arquitecturas multiprocesador mediante paquetes de cargas de trabajo predefinidas.
- MultiBench™: Implementa el análisis de microprocesadores multinúcleo mediante tres formas de concurrencia:
  - Descomposición de datos: Múltiples hilos cooperan para conseguir una meta común mediante técnicas de paralelismo de grano fino.
  - Procesamiento múltiple de flujos de datos: Ejecuta programas básicos sobre múltiples hilos y pone a prueba la escalabilidad del procesador frente a entradas de datos sobredimensionadas.
  - Procesamiento múltiple de cargas de trabajo: Pone a prueba la concurrencia tanto del programa como de los datos y la escalabilidad de un procesamiento de uso general.

Computación heterogénea
En esta categoría se agrupan frameworks de alto nivel para computación asimétrica:

- ADASMark™: Mide el rendimiento del pipeline de visión artificial empleado en plataformas de sistemas de asistencia avanzada a la conducción (ADAS, Advanced Driver-Assistance Systems) con OpenCL.

- MLMark™: Este benchmark está dedicado a categorizar y analizar varias clases de las redes neuronales ML.

Es importante destacar, para finalizar, que **cada benchmark de EEMBC tiene un número recomendado de iteraciones** para su ejecución. Este número de iteraciones **será asumido como una única ejecución de un benchmark**.

3.2. Elementos hardware para la medición de consumo

3.2.1. Nordic Semiconductor Power Profiler Kit

Serie de productos realizados por la empresa Nordic Semiconductor diseñados como herramientas de fácil uso para la medición precisa de consumo energético así como su optimización. Estos productos se venden como un kit de hardware y software, donde este último ofrece una interfaz gráfica para todas las funciones que el hardware ofrece. Tras medir, ofrecen la opción de exportar los datos. Usuarios de GitHub han desarrollado APIs para manejar su funcionamiento, de manera que se puede automatizar su uso prescindiendo de una interfaz gráfica [18] [10]. A continuación, se realiza una comparativa en la tabla 3.2.1 de los dos kit disponibles hasta la fecha, especificando algunas de sus características.

	Dispositivos	
Características	P.P.K. I[15]	P.P.K. II [16]
Rango de medida	0 mA - 70 mA	200nA - 1A
Resolución de medida	0.2 µA	Varía entre 100nA - 1mA
Frecuencia muestreo	77 kHz	10 veces más
Dispositivos compatibles	nRF51 DK, nRF52 DK, placas personalizadas	Todas las placas Nordic DK, placas personalizadas
Tiempo medición	20s	No limitado

Cuadro 1: Comparativa de las prestaciones de los dos dispositivos Nordic Semiconductor Kit.

3.2.2. Enchufes inteligentes

Estos dispositivos están formados por uno o más enchufes hembras por las que se pueden realizar múltiples operaciones de control. Entre estas operaciones se encuentran el control del paso de corriente o no por cada uno de los enchufes, así como la monitorización del consumo utilizado en cada uno de estos a lo largo de un intervalo de tiempo definido.

Existen múltiples marcas que los fabrican, ofreciendo a sus usuarios el acceso a las operaciones mediante una aplicación en dispositivos móviles inteligentes.

3.3. MAGEEC: Proyecto de optimización energética

MAGEEC[9] es un proyecto que utiliza las opciones de compilación, en definiva los conocidos como "flags", junto a técnicas de aprendizaje computacional con el fin de crear un framework de compilación que sea capaz de generar código que mejore la eficiencia energética, y por lo tanto, reduzca el consumo energético.

Sus metas principales son las siguientes:

- Conseguir la optimización energética.
- Utilización de valores de consumo obtenidos a través de su medición para cumplir la optimización, no predicciones de consumo mediante modelos de aprendizaje computacional.
- Uso de GCC y LLVM.
- Conseguir un framework de compilación funcional.

Este proyecto tuvo lugar en 2013, y tuvo una duración inicial de 18 meses, para más tarde continuar su desarrollo al formar parte de otro proyecto mayor, llamado TSERO [1]. Este último a su vez es un proyecto para el reporte y optimización de consumo energético desde el punto de vista software. Tiene como socios a la agencia de innovación del gobierno inglés, así como algunas empresas, entre las que se encuentra Embecosm [5]. Es la encargada de la continuación del desarrollo MAGEEC. Están desarrollando una versión gratuita y de código abierta de MAGEEC enfocado al uso junto a computadores de alto rendimiento.

3.4. Estudios de obtención de consumo energético y/o posterior análisis

Existen múltiples trabajos que monitorean el consumo energético de diferentes dispositivos, bien definiendo una metodología propia, o usando elementos hardware preparados para la monitorización, como los ya vistos. Otros tratan incluso de predecir este consumo energético. Tras el monitoreo, con esta información se pueden realizar múltiples objetivos. A continuación veremos algunos estudios relacionado con estos aspectos.

- Review of Computer Energy Consumption and Potential Savings [2]:

  Estudio de diferentes computadores personales y monitores que trata de reportar el consumo energético de cada uno de estos. Para ello, hace uso tanto de las opciones que ofrece el SO, así como hacer hincapié en el esfuerzo manual y buena educación por parte de los usuarios.

- Energy Consumption of Personal Computer Workstations [17]:

Estudio del año 1994 sobre múltiples workstations de uso personal, impresoras de red, faxes y fotocopiadoras. Se hicieron mediciones sobre estos durante 24h, de manera que se construyó un perfil estándar de consumo energético sobre el que trabajar. Se hicieron múltiples pruebas adicionales, contemplando la inactividad de teclados y ratones, el apagado de monitores, etc. De esta manera, se hace un reporte completo de consumo en múltiples circunstancias, así como el coste económico que supone un ordenador. Para todo este estudio, se utiliza un monitor de consumo en el que se conectan múltiples dispositivos, para más tarde monitorear la corriente que utilizan cada uno de ellos con transformadores. Más información se encuentra en su metodología.

- Estimation of energy consumption in machine learning [6]:

  Estudio en el que se ofrecen múltiples acercamientos de aprendizaje computacional para la predicción de consumo energético en diferentes niveles, desde una capa de aplicación, hasta la predicción en capa de hardware utilizado. Del mismo modo, se ofrece información sobre simuladores con los que conseguir los valores de consumo. Se ofrece mucha más información y comparativa de técnicas en detalle en el documento.

- Characterizing the energy consumption of data transfers and arithmetic operations on x86-64 processors [11]:

  En este trabajo, se presenta una metodología de medición, así como unos benchmarks, para analizar el consumo energético de microarquitecturas de x86 y x64 de AMD e Intel. Se caracterizan, además, el consumo de las operaciones individuales y transferencia de datos con respecto del consumo total. Para ello, se aplica una carga de trabajo constante, compensando así las limitaciones de resolución presentes en algunos medidores.

  La mayor contribución de este trabajo es llegar a medir de manera real el consumo energético de estos procesadores, ya que generalmente este valor se suele conseguir a través del uso de simuladores. Esto es debido a dos factores. El primero de ellos es la ya mencionada limitación de resolución temporal de algunos medidores. El otro es la dificultad de medir únicamente el procesador, el cual está alimentado por varias vías no accesibles y compartidas con otros componentes.

4. Metodología

4.1. Elección de elementos hardware para la medición

Antes de comenzar, queremos poner en consideración los ya mencionados elementos hardware diseñados para la medición de consumo, de cara a valorar si alguna de estos ha de ser usado a lo largo de este proyecto.

- Kits de Nordic:
  Para comenzar, los kits mencionados de Nordic parecen altamente relacionados con nuestro objetivo actual. Podrían ser de alta utilidad, sin embargo, carecen de una documentación significativa más allá de sus especificaciones técnicas de prestaciones. Son un sistema cerrado, ya definido, sin posibilidad de "indagar" en su funcionamiento, ya que únicamente se ofrece su uso.

  Las APIs existentes para el manejo de estos mediante lenguajes de programación son externas a la empresa. El hecho de que sean externas no es ideal, y aunque pudiesen ser funcionales, carecen de soporte.

  Por otro lado, estos dispositivos establecen un rango de medición ya establecido. Sin embargo, al no saber qué dispositivos vamos a medir, no podemos conocer cuál es el rango de consumo que obtendremos.

  Por todos estos factores, el uso de estos kits queda descartado a priori.

- Enchufes inteligentes:
  Uno de los inconvenientes encontrados con estos dispositivos es su diseño, el monitoreo que ofrecen es a lo largo de una gran ventana temporal, generalmente horas. En contraposición, un software que se ejecute durante horas sería uno de larga duración, y evidentemente, podemos encontrarnos en otra situación de que queramos medir el consumo de uno de escasa duración, incluso menor a un segundo.

  Incluso sin tener en cuenta el ya mencionado inconveniente, el mayor que estos presentan es que tampoco permiten el manejo de estos datos de monitoreados, o siquiera su exportación.

  Se conoce que la comunidad de desarrolladores de GitHub ha logrado, mediante una larga investigación en un modelo concreto de enchufe inteligente, el exportado de la información monitoreada. Incluso así, este modelo ya no está en fabricación al estar obsoleto, de manera que es de difícil obtención incluso en el mercado de segunda mano.

Por estos motivos, finalmente de nuevo se descarta este dispositivo.

Como apunte final, se han consultado otros dispositivos especializados para la medida de corriente de forma, algunos que incluso no requieren de cableado. Debido a los requisitos temporales establecidos del proyecto, los tiempos de envío de estos dispositivos debido a la situación actual, además de los requisitos económicos, se tienen en consideración estos dispositivos pero no se hace uso de ellos actualmente.

Como conclusión de esta elección de elementos hardware, se descartan los mencionados previamente fijados a empresas y el soporte que estas ofrezcan. Se escoge el uso de computadores desde los que manejar la información que nos proporcionen los sensores, como veremos más adelante en 4.5.

De esta manera, podemos realizar una metodología general y genérica, sin fijarnos a la utilización de un dispositivo concreto.

4.2. Preparación de los benchmarks

Para comenzar a definir nuestra suite, necesitamos tener benchmarks funcionales que puedan formar parte de la nuestra. De esta manera, se hace uso de los benchmarks de las suites mencionadas en el estado del arte, BEEBS, Embench y las de EEMBC. De este último, en [13], documento del que haremos uso más adelante, se hace uso de un subconjunto de suites del total disponible, las cuales formarán parte a priori de la nuestra. Pese a que las suites que usemos, como es el caso de las suites de EEMBC, no están enfocadas al consumo energético, pueden ser útiles en función de los criterios de caracterización definidos. Descartamos el uso de BEEBS frente al uso de Embench por las mejorías que este último presenta, como se puede ver en el apartado correspondiente, 3.1.2, del estado del arte.

El inconveniente que se ha encontrado es que, por lo general, las suites de benchmarks suelen tener una estructura de proyecto compleja, donde para compilar se utilizan herramientas propias de la suite, tales como los Makefile, que enlanzan toda la estructura del proyecto. Sería conveniente que podamos tener el código fuente disponible para la compilación de manera sencilla, sin herramientas ajenas ni necesitar enlazar una estructura compleja de archivos. De esta manera, podemos controlar la compilación: qué herramienta se usa, qué nivel de optimización se aplica, entre otras ventajas. Por consiguiente, los códigos fuentes de los benchmarks han sido adaptados manual y exhaustivamente con este propósito, enlazando manualmente las librerías y solucionando errores de compilación hasta su correcto funcionamiento.

4.3. Método de caracterización de los mismos

Una vez disponemos de nuestros benchmarks candidatos a formar parte de nuestra suite, necesitamos un método para distinguir los que son relevantes para nuestro propósito. En otras palabras, necesitamos de unas caracterizar cada uno de los candidatos para, sobre estas, poder tomar decisiones.

Para ello, hemos planteado el método de caracterización del **grafo araña**, tomando como referencia el **método propio de caracterización de EEMBC** [13]. En este, se ha conseguido describir la aplicación de un programa independientemente de la plataforma donde este se ejecute. Para ello, recogieron datos de distintas arquitecturas, compilando bajo las opciones de optimización por defecto.

La caracterización que realizan desde EEMBC está enfocada desde dos perspectivas distintas:

- Caché, recogiendo datos de trazas de programa, usando un simulador de caché que evaluaba múltiples configuraciones, usando a su vez el algoritmo LRU.
- **Nivel de requisito de las distintas unidades funcionales de la CPU**, usando un simulador MIPS en condiciones ideales. Se mide el nivel de requisito de estos a lo largo del 85 % del tiempo de ejecución. Como resultado, tenemos un diagrama de araña, también conocidos como Kiviat o radar, con distintas unidades funcionales, y su uso en un intervalo cerrado 0 a 10, como se muestra en la figura 1.

Tomando como base esta última mencionada perspectiva, hemos definido nuestro propio método de caracterización, el del grafo araña, mencionado al inicio. En este método, determinamos el porcentaje de uso de las unidades funcionales mediante el código fuente del benchmark, utilizando, de nuevo, la arquitectura MIPS. Hemos dividido el conjunto de instrucciones de MIPS en distintas categorías que corresponden a las unidades funcionales mostradas en la figura 1. Cada instrucción solo puede pertenecer a una categoría, escogida en función de qué unidad funcional hace uso. De esta manera, se calculan los porcentajes de cuántas instrucciones hay de cada categoría con respecto del total, obteniendo un resultado como el mostrado en la figura 1.

Además, añadimos un dato adicional al grafo, el consumo energético del benchmark. Para ello, se hace necesaria la obtención del consumo de un determinado benchmark, como veremos a continuación. De esta manera, con este criterio obtenemos una "huella" de cada benchmark.

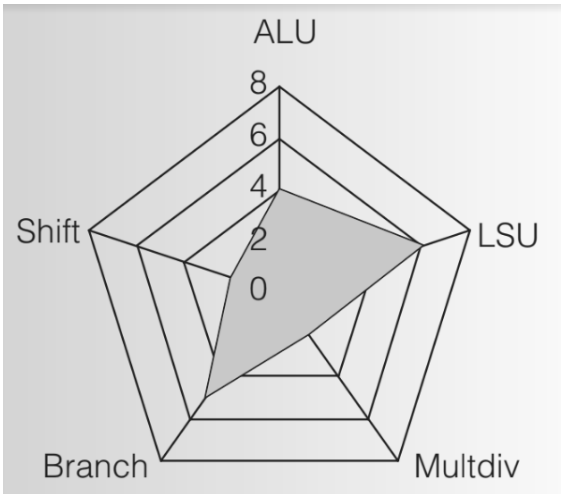

Figura 1: Diagrama de araña que muestra el uso de las distintas unidades funcionales. [13]

4.3.1. Determinación del consumo de un benchmark

Para la determinación del consumo de un benchmark, el proceso realizado consiste en medir el consumo de corriente de este, en miliamperios, tantas iteraciones como sean necesarias hasta cumplir el criterio de parada definido. Mediante este, se exige que la suma del tiempo de ejecución de las iteraciones debe igual o superior a una hora. Teniendo los miliamperios resultantes y sabiendo el tiempo de ejecución total, definido como criterio de parada, se aplica un cálculo para conseguir el consumo energético del benchmark durante una hora. Dado que puede existir cierta incertidumbre a la medición, se decide hacer 50 veces el experimento anteriormente descrito. Finalmente, **determinamos el consumo energético de un benchmark como el promedio conseguido de 50 experimentos de consumo de una hora**.

Ahora sí, para determinar el consumo energético, hemos de irnos a su propia definición. En esta, se especifica como:

$$E = P \cdot T$$

Donde T será el tiempo total del experimento, una hora en nuestro caso, y P la potencia total consumida.

Sin embargo, en nuestra metodología se presenta una particularidad, y es que tenemos medidas de intensidad a lo largo de todo el experimento, de forma separada en el tiempo, describiendo una función discontinua. De estas mediciones debemos extraer nuestra potencia. Para ello, comenzamos por suponer que la intensidad medida en un instante de tiempo se mantiene constante hasta la

siguiente medida. Se ilustra esta suposición en la figura 2. Se supone, además, que el voltaje tiene un valor constante de 5V.

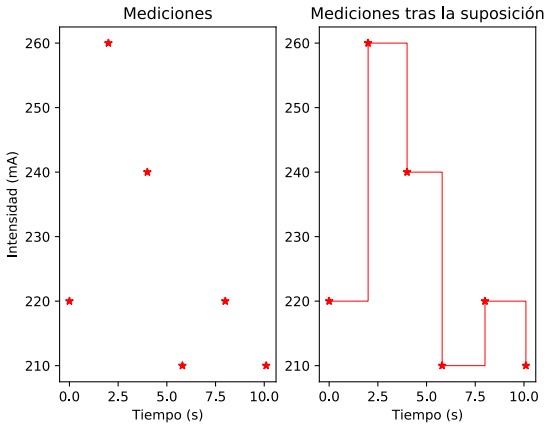

Figura 2: Comprobación del consumo del experimento propuesto usando los tres métodos diferentes de espera planteados.

Ahora sí, para cada par de muestras, podemos definir la potencia de una sola medida como:

$$P_i = I[i] \cdot 5V \cdot T_i$$

Donde para una medida "i", la potencia de i, P_i, es igual a la intensidad de i, $I[i]$, por el voltaje, 5V, durante el tiempo hasta la siguiente medida, T_i.

Siguiendo de esta fórmula, definimos la potencia total como:

$$P = \frac{\sum_{i=0}^{n-1} m[i] \cdot 5V \cdot T[i]}{T}$$

Donde n será el número de muestras totales conseguidas durante nuestro experimento de una hora.

De nuevo, $T[i]$ corresponde al tiempo hasta la siguiente muestra, o periodo de muestreo. Este periodo depende la frecuencia con la que el dispositivo de medición demande datos al dispositivo que nos proporciona las mediciones, además de la frecuencia con la que este último es capaz de ofrecer datos.

De cara a este último factor, la frecuencia de muestreo puede no ser constante, ya que, generalmente, los dispositivos no funcionan de manera ideal en un tiempo constante, admitiendo ligeras desviaciones temporales. De igual manera, nuestro dispositivo medidor, puede no demandar datos a una frecuencia constante por múltiples factores, entre los que se puede encontrar la presencia

de un sistema operativo de carga significativa.

Debido a esto, se decide tomar el tiempo entre muestras como una constante. Esta decisión responde al hecho de que se podrían producir demoras adicionales al obtener, para cada medición, el tiempo en el que esta se ha obtenido. La operación de toma de tiempo tiene un coste en tiempo computacional que, de cara a obtener el máximo de muestras, puede ser un factor que decremente nuestro valor de muestras totales obtenidas. Del mismo modo, el almacenamiento de todas estas muestras temporales de tiempo aumentaría significativamente el coste en memoria.

Dado este hecho, la fórmula resulta en:

$$P = \frac{\sum_{i=0}^{n-1} m[i] \cdot 5V \cdot T[i]}{T} = \frac{\sum_{i=0}^{n-1} m[i] \cdot 5V \cdot \frac{T}{n}}{T} = \frac{(\sum_{i=0}^{n-1} m[i]) \cdot (5V \cdot \frac{T}{n})}{T} = (\sum_{i=0}^{n-1} m[i]) \cdot \frac{5V}{n}$$

Y volviendo al cálculo de la energía, queda:

$$E = P \cdot T = (\sum_{i=0}^{n-1} m[i]) \cdot \frac{5V}{n} \cdot T = \frac{(\sum_{i=0}^{n-1} m[i]) \cdot 5V \cdot T}{n}$$

Quedando la energía total consumida como Ws o J, mWs, mWh o Wh según las unidades de las medidas, A o mA, y el tiempo, en segundos o milisegundos.

4.4. Estudio de ejecución de la suite

Ya que nuestro objetivo se basa en la definición de nuestra suite, nos preguntamos cómo deberíamos de ejecutar la misma. Para ello, hemos seguido una serie de pautas para definir cómo hacer una correcta ejecución del mismo.

4.4.1. Longitud mínima de la ejecución de cada benchmark

Para comenzar, se plantea el determinar cuantas veces como mínimo se ha de ejecutar un benchmark. Esta mencionada longitud es medida en iteraciones, por lo que el objetivo es medir un número de iteraciones mínimo. Para ello, se hace uso de las iteraciones calculadas durante los 50 experimentos de una hora. Usando estas, se obtiene el consumo energético del benchmark durante 2^i iteraciones, variando i desde 0 hasta el mayor número posible en función de las iteraciones de las que se disponga, así como el número de iteraciones múltiplo de 100. Estos valores de consumo de distintas iteraciones son comparados con nuestros valor de referencia de cuánto consume ese benchmark en concreto, conseguido con anterioridad. De esta manera, podemos determinar que con qué número de iteraciones conseguimos un consumo de una representatividad cercana a la real.

4.4.2. Búsqueda de la ejecución óptima de la suite

Una vez obtenidos los resultados mencionados en el apartado anterior, podemos dar paso a hacer un planteamiento sobre cómo ejecutar la suite completa. En concreto, definiremos qué número de iteraciones hemos de definir cada benchmark de todos los disponibles. Para ello, haremos uso de un algoritmo genético cuyos individuos serán un vector tan largo como tantos benchmarks se dispongan en nuestra suite. De esta forma, cada posición corresponde a un vector. Los valores de cada posición indicarán el número de iteraciones que el benchmark correspondiente a esa posición se ejecuta.

Se podría pensar que, para calcular el consumo correspondiente al vector, si se dispone del consumo de cada uno de los benchmarks para su número correspondiente de iteraciones, se podría consultar estos consumos y sumarlos para averiguar el consumo total correspondiente a las iteraciones reflejadas en el vector. Sin embargo, la ejecución de un benchmark pueda afectar al consumo de uno posterior. A pesar de este hecho, se asumirá a priori que el consumo de manera individual tanto como conjunta es igual. De esta manera, sabiendo el consumo de una iteración de un benchmark, podemos saber el consumo de n iteraciones, n veces el de una iteración.

La función de fitness valorará tres factores:

- El consumo total correspondiente a nuestro vector individuo, que contiene el número de iteraciones de cada benchmark.
- La diferencia entre el tiempo total de ejecutar los benchmarks tantas veces como indique el vector, a un tiempo objetivo.
- El número de benchmarks que se ejecuten en este individuo.

De esta manera, se busca que la función de fitness valore positivamente el acercamiento del tiempo ejecución total al tiempo objetivo, el mayor número posible de benchmarks con respecto del total disponible, y minimizar o maximizar el consumo, en función de lo que se desee. La función únicamente penaliza en un único caso, que es el de que el consumo conseguido sea menor que el consumo basal del sistema operativo durante tanto tiempo como indique el tiempo objetivo.

Con la función de fitness se persigue, pues, maximizar o minimizar el consumo, en el número de iteraciones que sea necesario, siempre siendo similar al tiempo objetivo, y usando la mayor cantidad de benchmarks posibles, consiguiendo un consumo mayor al basal.

Un problema adicional que aparece es la estructura de los benchmarks. Las suites que los engloban, en algunos casos, hacen uso de funciones, variables o estructuras de igual nombre en cada uno de ellos. Al compilar todos los programas con funciones de igual nombre, tenemos un evidente problema de redefinición. Sin embargo, a pesar de las múltiples definiciones de estos elementos ya

mencionados, en algunos de estos su contenido es diferente en cada uno de los benchmarks, de modo que no simplemente se puede dejar la primera ocurrencia de cada uno de estos.

La solución que se ha encontrado es la de hacer un proceso iterativo de compilación y comprobación de errores, donde si se encuentra una redefinición, se sustituye el elemento redefinido en cada benchmark por el nombre de este elemento acompañado del nombre del benchmark.

Como se puede seguir de este hecho, el nombre de los benchmarks debe de ser único, hecho que se comprueba al inicio de nuestra implementación del medidor, añadiendo a su nombre el de la suite a la que pertenece, si fuese necesario.

Finalmente, se muestra en la figura 3 un diagrama de flujo relativo al software implementado. Algunas de las funciones mostradas en este se explicarán más adelante en la implementación de la medida. Asimismo, se incluye en la figura 4 un diagrama de comportamiento de la implementación descrita.

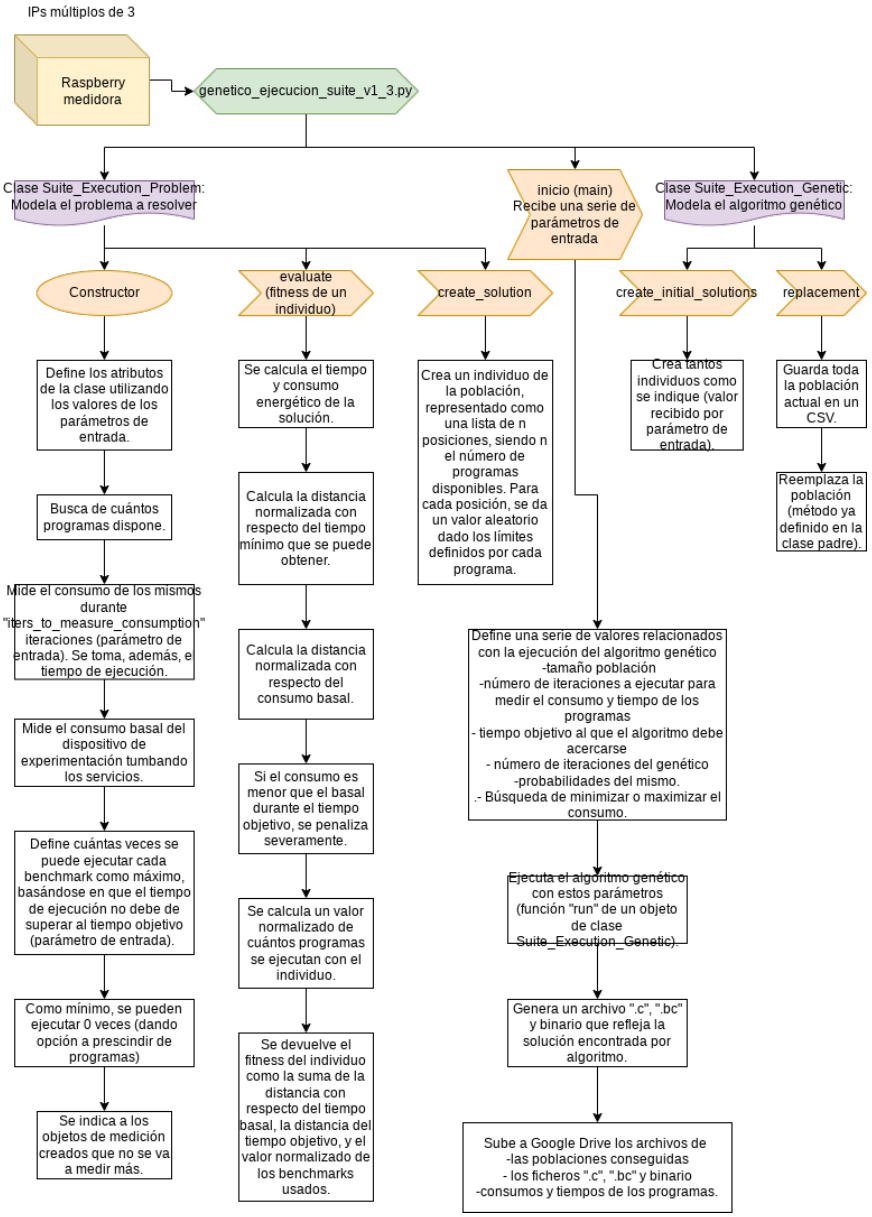

Figura 3: Diagrama del genético correspondiente a la MediPi.

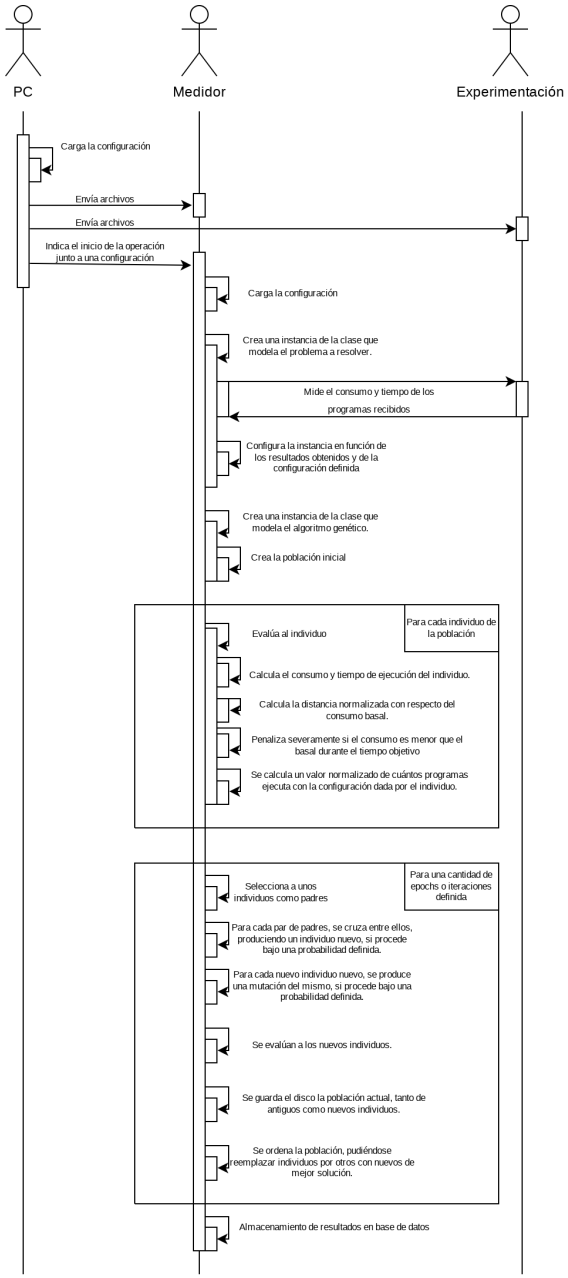

Figura 4: Diagrama de comportamiento del genético.

4.5. Equipo de medida

Definimos equipo de medida como uno o más computadores donde podamos ejecutar programas y observar el consumo energético del computador donde este se ejecuta.

Nuestro primer acercamiento de este equipo se sigue de otros trabajos previos, donde se planteaba mediciones con un ADC de alta resolución conectado a un sistema Raspberry Pi que se optimizó más adelante con el uso de los circuitos integrados INA219 e INA226, cada uno con distinta resolución, controlados mediante placas de desarrollo compatibles con Arduino como la Adafruit Metro M4 (SAMD51J20A) o la novedosa Raspberry Pi Pico RP2040. Estos sistemas integrados recolectaban la medición hasta llenar su memoria y la volcaban en un PC una vez se llenasen los buffers de medición. Más adelante, se decidió prescindir de este método para empezar a utilizar en su lugar una Raspberry Pi modelo 3B, que hiciera las veces de medidor y de orquestador, sin tener la necesidad de disponer de un PC para tal fin. Esta elección de cambio de hardware, además, fue motivada por su bajo coste, alta disponibilidad de stock y la presencia de una gran variedad de software compatible. En este segundo acercamiento, la Raspberry Pi, para realizar la medición, pasaría a controlar directamente los integrados INA219, INA226 o INA228 mediante los protocolos IIC o SPI.

Así se pudo montar un primer prototipo de un sistema medidor básico 5 que consta de dos Raspberries. Una de ellas es la encargada de ejecutar los benchmarks y por tanto es objeto de poder ser sustituida por cualquier otro hardware sobre el que se quiera realizar una medición. Mientras que la otra es fija y se encuentra dentro de la caja negra que delimita el equipo de medición propiamente dicho y se encarga de medir el consumo de la primera y almacenarlo en disco de forma ordenada. De esta manera, se consigue aislar la medición de la ejecución. Para diferenciarlas, se las conoce como "Experimentación" o "ExperiPi" y "Medidora" o "MediPi", respectivamente.

"MediPi" se encarga también de realizar la fase previa a la medición de compilación de los programas para evitar añadir más carga de trabajo durante la experimentación y acortar la duración de la misma. Para realizar la medición cuenta con cuatro circuitos integrados INA219 que le permiten realizar hasta cuatro mediciones simultáneas, con la limitación de tener una reducción del número de mediciones obtenidas por segundo. Esto es debido a que todos ellos comparten el mismo bus IIC y sólo se puede realizar una petición de datos a la vez.

"ExperiPi" en realidad puede ser una Raspberry Pi, un sistema Arduino, un teléfono móvil o en definitiva cualquier micro-arquitectura hardware.

La comunicación principal entre las dos Raspberry Pi, medidora y experimentación, se realiza mediante el estándar Ethernet a través de los protocolos ssh o telnet para transferir los benchmarks compilados. Una vez entregados,

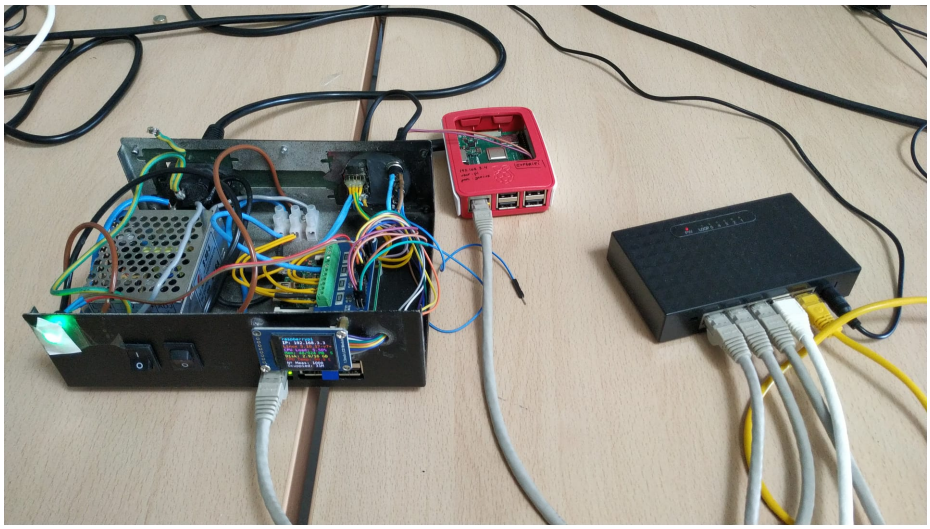

Figura 5: Prototipo inicial del equipo de medida, donde podemos ver de izquierda a derecha: la fuente de alimentación del equipo controlada por conmutadores, la medidora junto a una pantalla con información relevante, la experimentadora y un switch para las conexiones Ethernet.

los dos equipos se sincronizan mediante señales digitales GPIO para iniciar y detener la experimentación y la medición respectivamente.

El equipo de medición está diseñado para ser manejado tanto de forma manual como automática. De forma manual, el usuario puede interactuar con el equipo de experimentación y con el de medición, mediante el protocolo ssh, para preparar su experimentación, o bien programar una ejecución automática para que ambos equipos inicien y detengan la medición.

Finalmente, se incluye en la figura 6 diagrama de la arquitectura hardware mínima necesaria, así como su conexión. Otra figura 7 muestra la arquitectura software del equipo de medida.

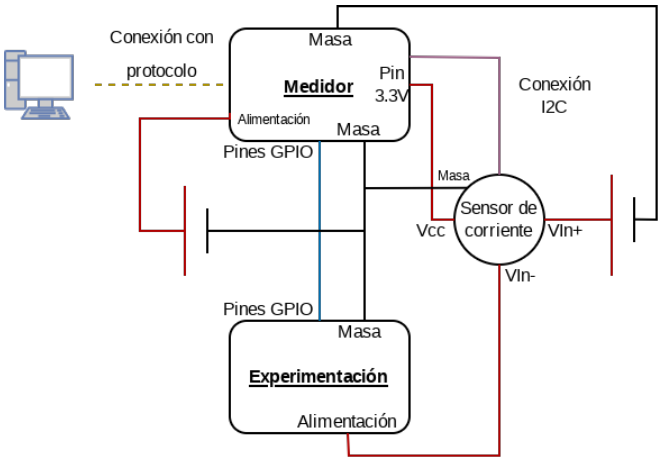

Figura 6: Diagrama de la mínima arquitectura hardware necesaria.

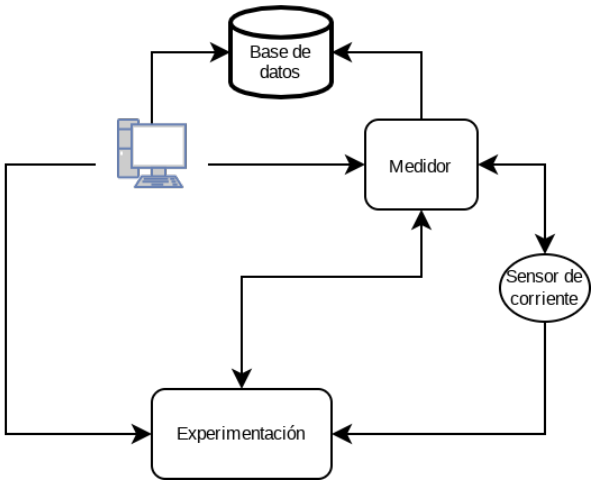

Figura 7: Diagrama de la mínima arquitectura software.

4.5.1. Introducción al software de la Raspberry Pi

La elección de la placa de desarrollo Raspberry Pi 3B, conlleva la elección de un sistema operativo que pudiera ser ejecutado por esta arquitectura, que sirva tanto para el equipo medidor como para el de experimentación.

El mismo fabricante ofrece en el momento de la redacción de este documento tres versiones de su sistema operativo Raspberry Pi OS, basadas en el kernel de Linux 5.1 y lanzadas el pasado 7 de Mayo de 2021:

– Raspberry Pi OS con entorno de escritorio y software recomendado, con un peso de instalación de 2867MB.

– Raspberry Pi OS con entorno de escritorio, con un peso de instalación de 1180MB.

– Raspberry Pi OS Lite, con peso de instalación de 440MB.

Sin embargo, la comunidad ofrece de forma alternativa y gratuita diferentes variantes que compiten con las oficiales. Tras un estudio del estado del arte del software disponible, a continuación se enumeran las distribuciones más populares sin entorno de escritorio que podrían tener un menor consumo:

– Ubuntu Server 20.04, con un peso de instalación de 730MB.

– FreeBSD 13.0, con un peso de instalación de 465MB.

– OpenSUSE 15.2, con un peso de instalación de 251MB.

– DietPi v7.5, con un peso de instalación de 129MB.

– piCore/Tiny Core Linux 12.0, con un peso de instalación de 88MB.

La motivación de centrarnos sólo en sistemas operativos sin entorno de escritorio está fundamentada en que, a priori por pruebas empíricas realizadas, asumimos que tendrán menos servicios activos, y por ende, un menor consumo que distorsione nuestras mediciones. Se hará una comparativa del consumo base de cada sistema operativo y nos quedaremos con aquel que tenga el menor. Dado que este benchmark forma parte del proyecto GENIUS, donde se hace necesario el uso de Python, LLVM y el software de control de los periféricos nativos (GPIO, SPI, IIC, UART) del hardware Raspberry, buscaremos un sistema operativo donde tengamos soporte a estas herramientas.

4.5.2. Implementación de la medición

Para la medida, se hacen uso de tres pines GPIO de la Raspberry Pi. A continuación los listamos, explicando su funcionalidad.

- Pin asociado al GPIO 19, controlado por la ExperiPi, se encarga de indicar si se está ejecutando una iteración de un programa en la misma. Si es así, este tiene un valor en alto.

- Pin asociado al GPIO 13, controlado por la ExperiPi, se encarga de indicar si la misma tiene alguna iteración o programa adicional que ejecutar. Si ese es el caso, tiene aún algo que ejecutar, tiene un valor en alto.

- Pin asociado al GPIO 26, controlado por la MediPi, se encarga de indicar si se está escribiendo en disco. En tal caso, tiene un valor en alto.

Como se puede observar por la funcionalidad descrita, los pines funcionan como si fuesen, haciendo comparativa con la programación tradicional, booleanos. Indican en qué estado se encuentra la Raspberry en función de su valor, de modo que el resto puede actuar en consecuencia de un modo u otro. En definitiva, mediante el uso de estos pines GPIO podemos sincronizar los dispositivos de medición y experimentación. El propósito que buscamos es que la ExperiPi ejecute uno o más programas tantas iteraciones como se desee, indicando la ejecución de una programa con el pin 19, mientras que la MediPi lee y almacena en memoria los valores leídos del sensor de corriente INA219. Cuando la ejecución finaliza, el pin 19 cambio su estado de estar en alto a estar en bajo, y se almacenan las lecturas en disco. Mientras se almacenan, la ExperiPi ha de esperar a que finalice la lectura antes de continuar ejecutando otros programas, información que obtiene mediante el valor del pin 26. Se repite este ciclo mediante el pin 13 esté en alto.

Para llevar a cabo las funcionalidades descritas, se ha creado una clase llamada "Measurer", que implementa las funcionalidades descritas previamente. Sin embargo, no se incluye la funcionalidad de compilar los programas, enviarlos al dispositivo de experimentación, e iniciar la ejecución de los mismos. La idea es la de tener una clase con la funcionalidad general de la medición, y se especifique detalles más específicos, como bien pueden ser de la arquitectura usada, en una clase hija que heredan de "Measurer". De esta manera, en esta clase hija ha de implementarse tres métodos: "compile_send_all", "send_and_compile_file" y "execute_file". Entre los dos primeros métodos, se debe de conseguir la funcionalidad de compilar los ficheros que sean necesarios, que estos sean enviados al dispositivo de experimentación. La diferencia entre estos dos es que el primero solo se llama una única vez, en el constructor de clase, mientras que el otro se llama para cada programa que se desee un programa. El último, "execute_file", manda al dispositivo de experimentación a ejecutar un programa.

Además, se incluye un archivo, "pin_initialization_experipi", que es llamado por la clase "Measurer" para inicializar los tres pines ya mencionados. La funcionalidad del dispositivo de experimentación también ha de ser implementada por usuario, así como la responsabilidad de que todos los scripts estén en los dispositivos correspondientes.

Como es evidente, se ha hecho una implementación correspondiente para la medición de los benchmarks, la cual se describe en las figuras 8, 9, 10, 11, 12, 13 y 14.

Como se puede observar, en la figura 13 y 14 se diseña cómo un equipo puede interaccionar con los distintos dispositivos de medición y experimentación.

Asimismo, se incluye en la figura 16 y 15 un diagrama de comportamiento de la implementación descrita.

La comprobación de valor de los pines, mencionada previamente, se ha implementado como un polling, que comprueba el valor del pin, espera un milisegundo, y lo comprueba de nuevo. Esta espera y segunda comprobación se realizan con el fin de evitar los posibles rebotes que pudiesen aparecer, esperando el tiempo suficiente para asegurarnos de que el valor que se retorna no sea debido a un rebote de cambio de valor del pin.

Relacionado con los valores de los pines, se introducen algunas esperas tras algunos cambios de valores de pines para asegurar la sincronización de ambos dispositivos.

Como apunte final, se planteó el posible uso de el aislamiento de uno o varios núcleos de la CPU para lanzar la medición de manera aislada al sistema operativo. Los problemas que podemos encontrar vienen dados por el propio aislamiento: se prescinde del planificador del sistema operativo, de modo que la medición se realiza en un solo núcleo, obteniendo un peor rendimiento y mayores tiempos. Finalmente, debido a los requisitos temporales de las mediciones y experimentos diseñados, se decide no hacer uso del aislamiento.

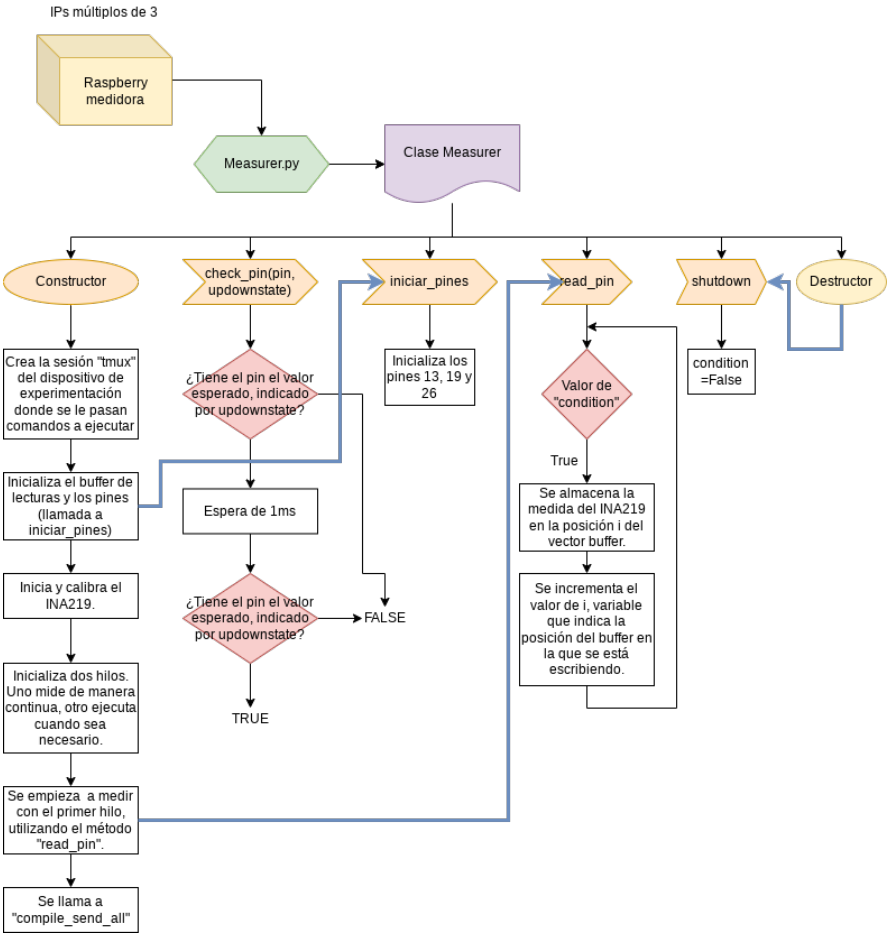

Figura 8: Diagrama de la clase “Measurer” de la MediPi.

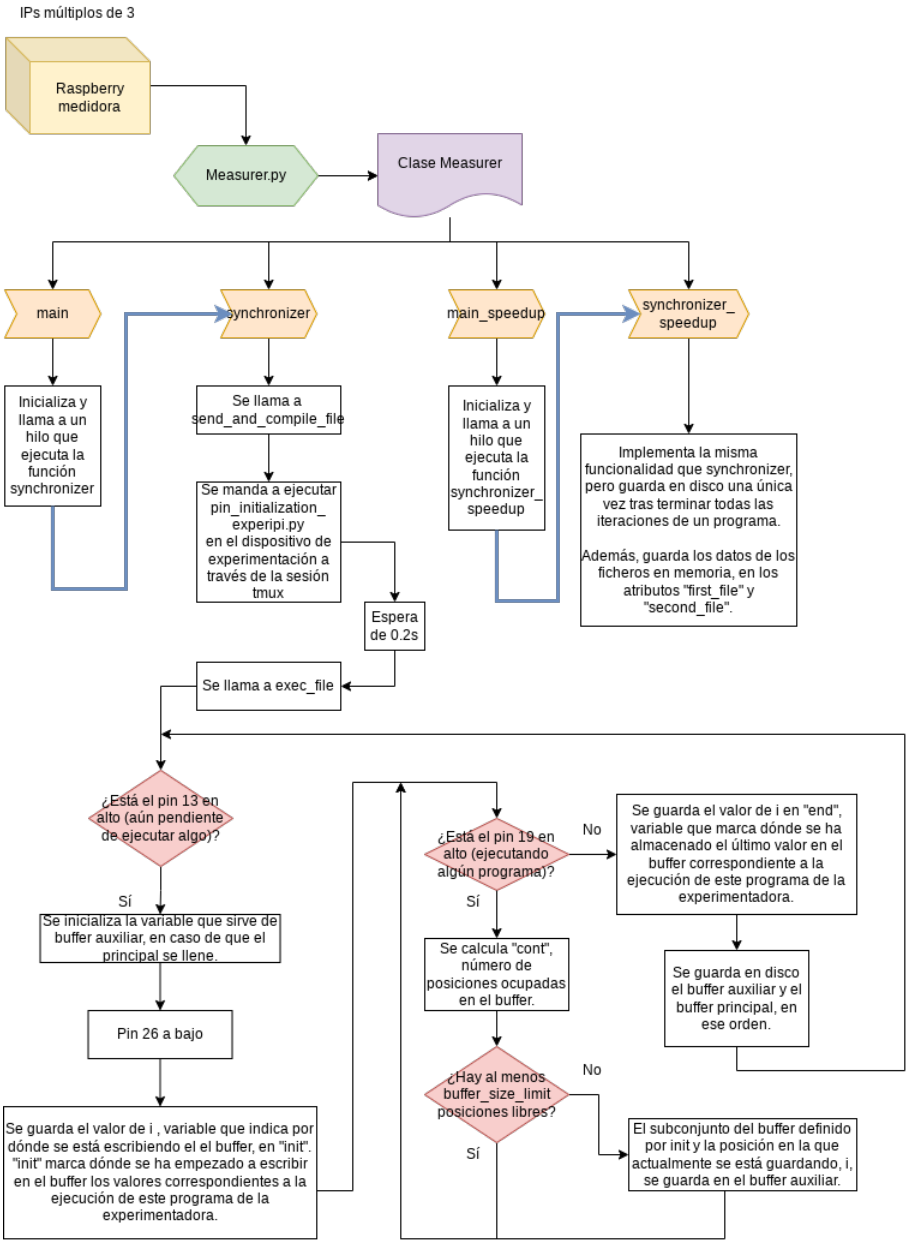

Figura 9: Otro diagrama de la clase “Measurer” de la MediPi.

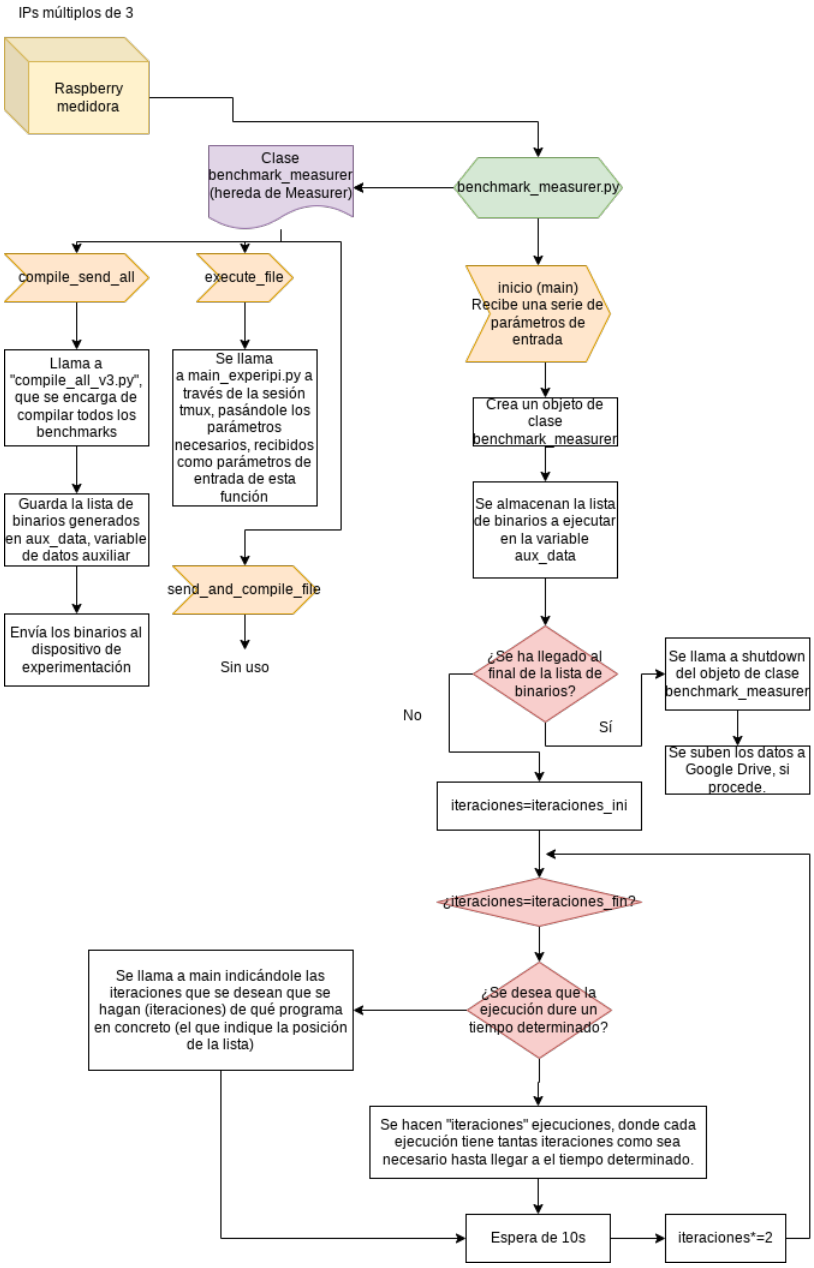

Figura 10: Diagrama de la clase “benchmark_measurer” de la MediPi.

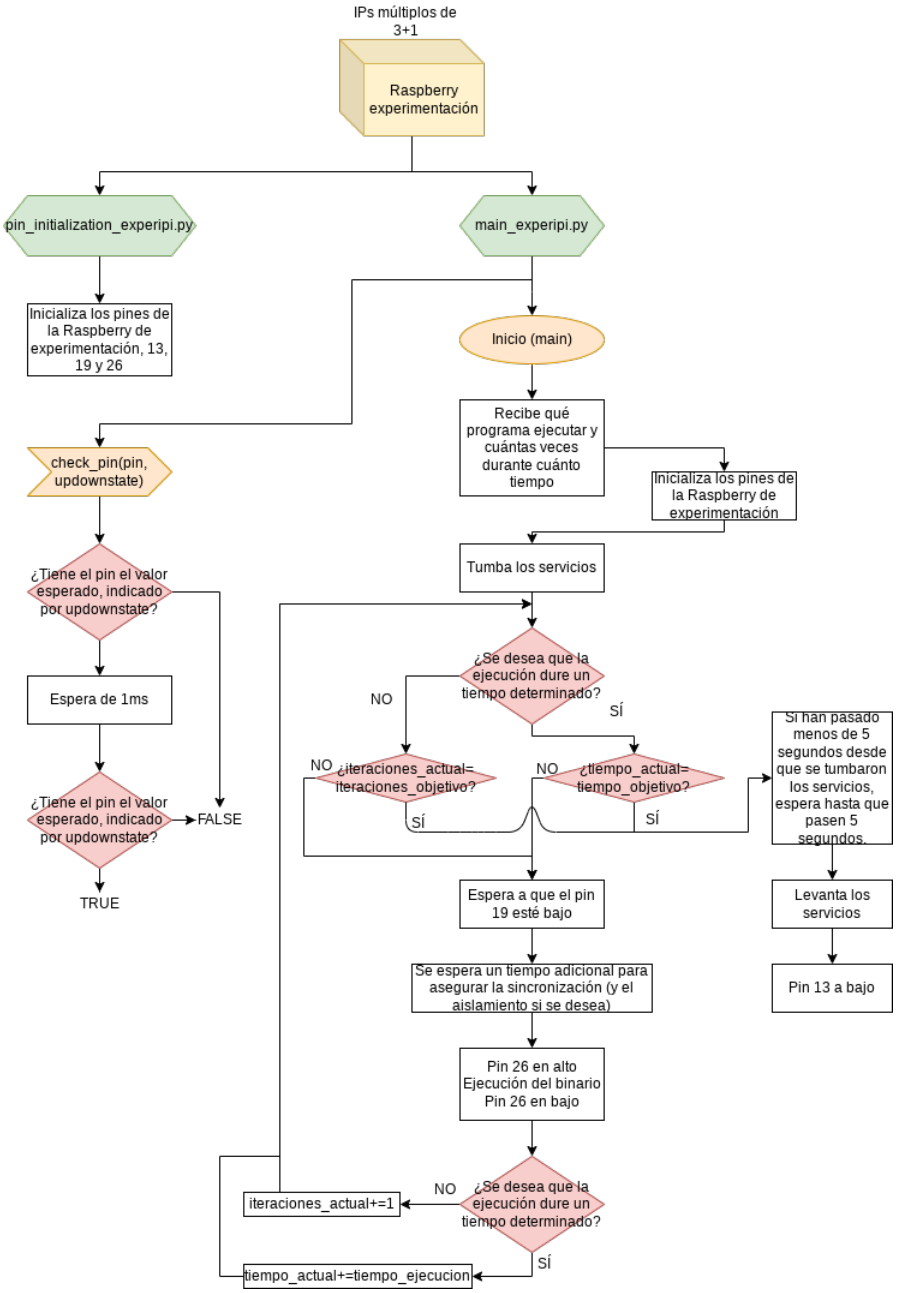

Figura 11: Diagrama de la clase correspondiente a la ExperiPi.

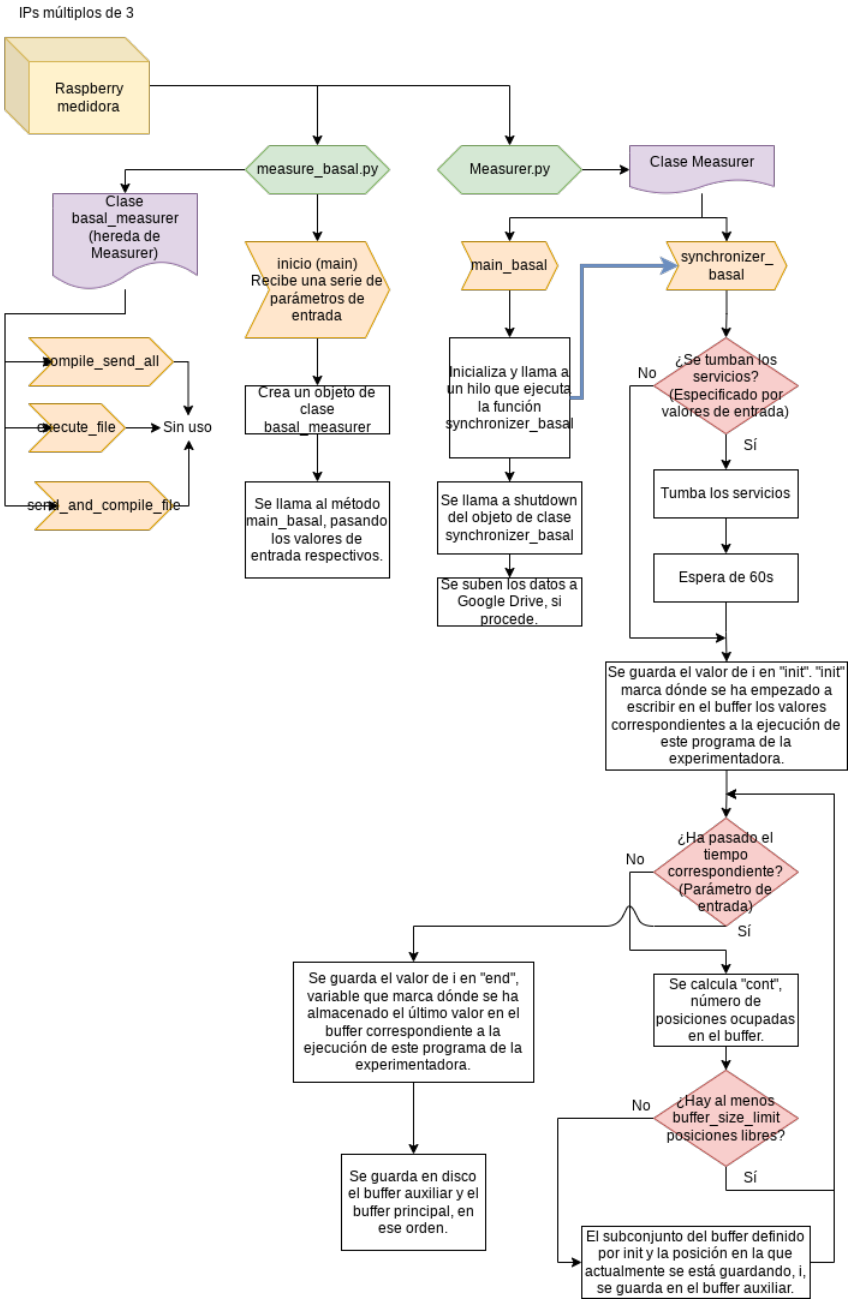

Figura 12: Diagrama de la medición basal correspondiente a la MediPi.

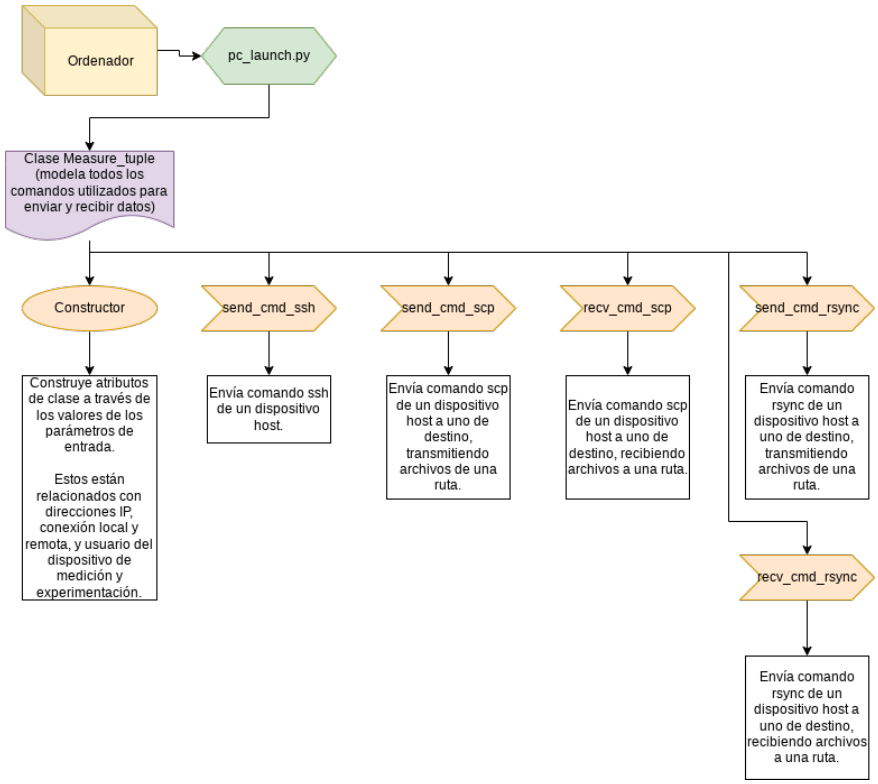

Figura 13: Diagrama del computador.

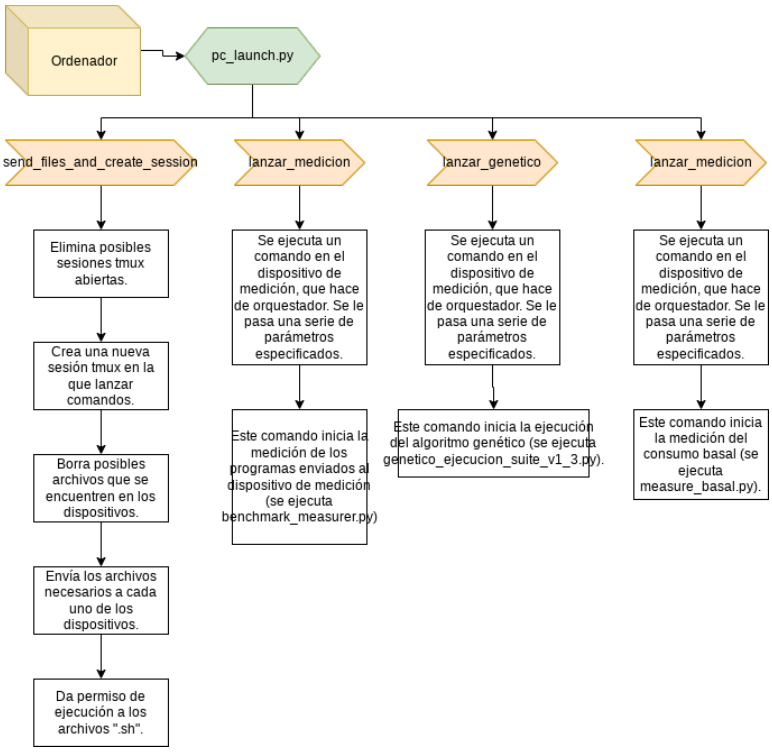

Figura 14: Segundo diagrama del computador.

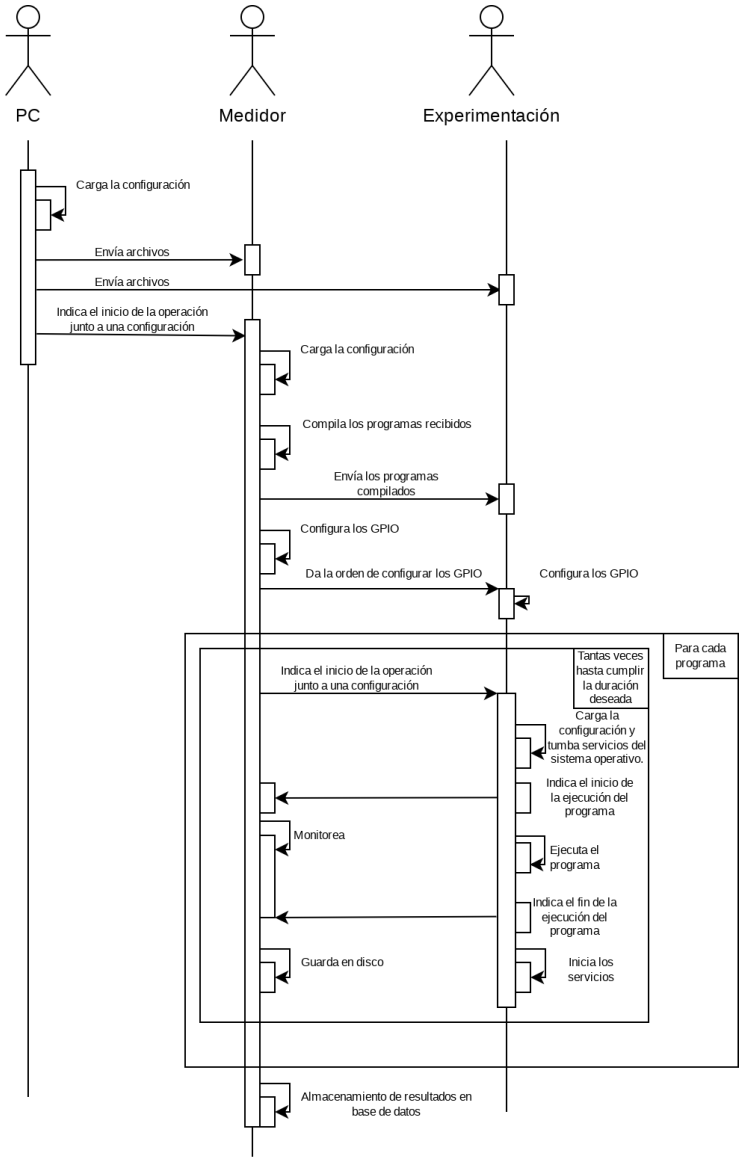

Figura 15: Diagrama de comportamiento para la medición de un benchmark.

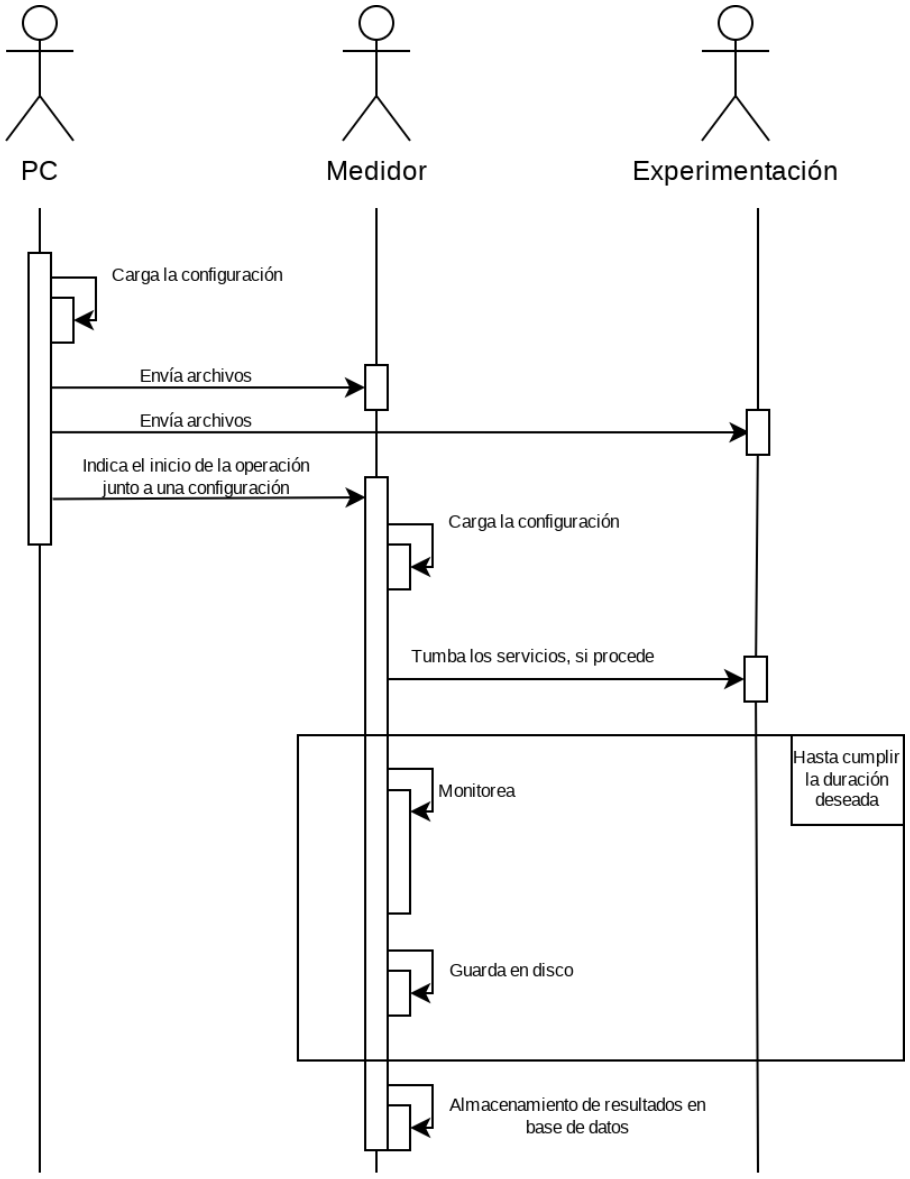

Figura 16: Diagrama de comportamiento para una medición del sistema operativo inactivo, también conocido como basal.

5. Discusión

5.1. Software y reducción del consumo de la Raspberry Pi

5.1.1. Elección del sistema operativo de Raspberry Pi

El empleo de las placas Raspberry Pi 3B+ para realizar mediciones y experimentación conlleva la elección de un sistema operativo liviano que disponga de soporte para el software necesario, y cuyo consumo energético sea lo suficientemente bajo y con pocas variaciones como para que el ruido de este no perturbe las mediciones de los benchmarks.

5.1.2. Criterios y procedimiento para la selección del sistema operativo

Esta elección se llevará a cabo mediante los siguientes criterios:

- **Disponibilidad del software para hacer la medición**. Se valorará positivamente aquellas distribuciones que den soporte nativo a `LLVM` y `Python3`, así como a librerías que faciliten el control del hardware nativo de Raspberry Pi (`Wiringpi`).
- **Consumo energético medio más la desviación típica**. Se busca el sistema operativo con el consumo más estable, por ello se estudiará el consumo energético base a lo largo del tiempo y se valorará positivamente aquel con el menor consumo medio y menor desviación típica.
- **Posibilidad de optimización**. De nada sirve tener un bajo consumo base si no lo podemos optimizar aún más. Se valorará positivamente que el sistema operativo integre herramientas para deshabilitar servicios o funcionalidades irrelevantes en las mediciones peor que aumenten la huella de consumo.

Más adelante se estudiará el consumo base de distintos sistemas operativos, para realizar la medición energética se empleará un sistema RaspberryPi 3B+ con el sistema operativo Raspberry Pi OS Lite dotado de un módulo INA219 (en adelante MediPi). Se utiliza este sistema operativo porque no es necesario un entorno gráfico para el control IIC del módulo medidor, pero bien se podría haber utilizado cualquier otro sistema.

Este sistema monitorizará el consumo energético de otro sistema RaspberryPi 3B+ (en adelante ExperiPi) al que se le irá cambiando el sistema operativo. La duración de esta medición estará fijada en 60 minutos ya que consideramos tiempo suficiente como para que el sistema se estabilice y ofrezca una buena perspectiva del consumo. Esto se ratificó al hacer varias pruebas independientes y una prueba extensa de 24 horas.

Esta tupla de interconexionado básico MediPi-ExperiPi supone el prototipo de lo que más adelante será un medidor dedicado.

5.1.3. Validación del medidor

Para validar el driver escrito en Python, y verificar que las lecturas del módulo ***INA219*** sean correctas, se compararán con el consumo de un motor DC paso a paso. Este motor paso a paso, modelo ***HANPOSE 17HS2408***, por su hoja de datos, presenta un consumo máximo por paso de 600 mA. El motor es controlador por el driver ***Allegro A4988*** integrado en el módulo ***StepStick*** de ***RepRap.org***. Este módulo está conectado a la placa controladora ***RepRap RAMPS 1.4*** , que se encarga de proveer de alimentación tanto al módulo como al motor, así como de actuar de interfaz de los GPIO del driver con el microcontrolador ATMega2560 de una placa ***Arduino Mega***.

La placa Arduino Mega, que controla el motor paso a paso, está programada con la librería `StepperDriver` de ***Laurentiu Badea***. Se realizan 6 pruebas diferentes de una duración de 5 minutos medidas con diferentes intervalos de resolución en el medidor INA219, esto es, a 400mA, 800mA, 1000mA, 1600mA y 2400mA.

- **stepperBasal**: Contiene una medición de 5 minutos del consumo basal del motor paso a paso detenido.
- **stepperRotationDirect**: Contiene una medición de 5 minutos del consumo de programar una rotación de 360 grados, realizarla directamente, sin esperas entre pasos, y tras finalizar la rotación actual volver realizar una nueva. Así sucesivamente.
- **stepperRotationDirect_5s**: Contiene una medición de 5 minutos del consumo de programar una rotación de 360 grados, realizarla directamente, y esperar 5 segundos al finalizar la rotación antes de volver a realizar una nueva. Así sucesivamente.
- **stepperRotationSteps**: Contiene una medición de 5 minutos del consumo de programar una rotación de 360 grados, realizarla paso a paso, sin esperas entre pasos, y tras finalizar la rotación actual volver realizar una nueva. Así sucesivamente.
- **stepperRotationSteps_5s**: Contiene una medición de 5 minutos del consumo de programar una rotación de 360 grados, realizarla paso a paso, sin esperas entre pasos, y esperar 5 segundos al finalizar la rotación antes de volver a realizar una nueva. Así sucesivamente.
- **stepperRotationSteps_5s_10steps_1s**: Contiene una medición de 5 minutos del consumo de programar una rotación de 360 grados, realizarla paso a paso, con una espera de 1 segundo cada 10 pasos, y esperar 5 segundos al finalizar la rotación antes de volver a realizar una nueva. Así sucesivamente.

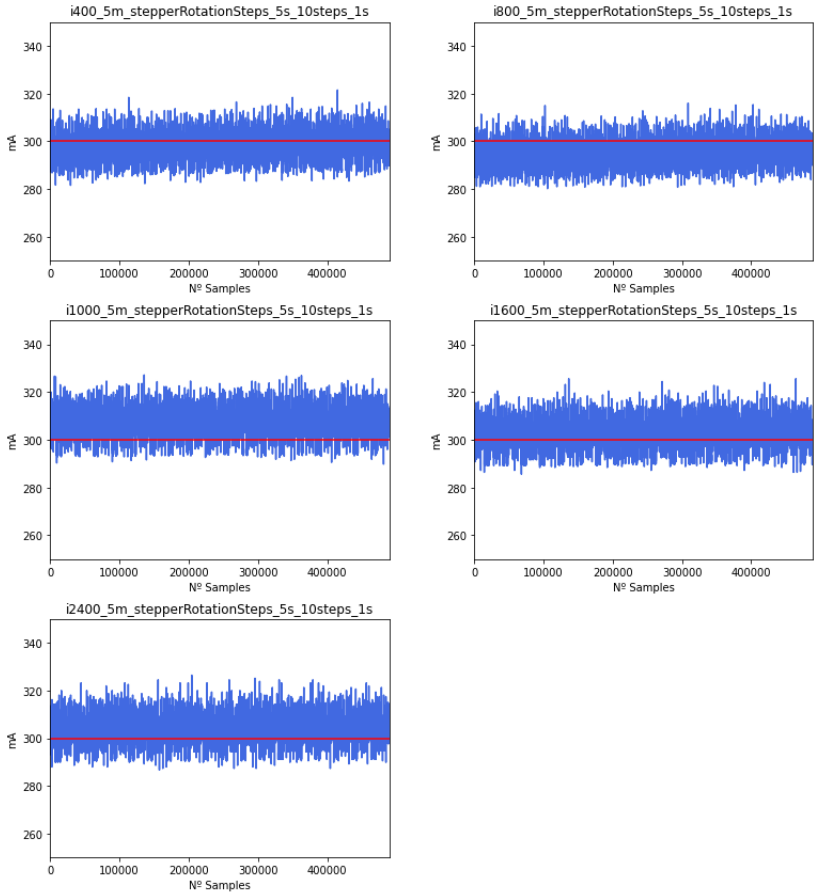

Figura 17: Comparación de los rangos de consumo para el programa `stepperRotationSteps_5s_10steps_1s` frente al consumo del medidor de referencia

En esta figura, hemos trazado una línea vertical que corresponde a lo que hemos estimado que es el consumo medio de nuestro motor DC. Como podemos observar, con todas las resoluciones utilizadas, la media de consumo queda dentro del rango de consumos medido, aumentando la precisión conforme más desciende la resolución y se acerca al consumo medio estimado. De esta manera, afirmamos que el motor, a nivel general, está correctamente calibrado.

5.1.4. Pruebas de consumo de los sistemas operativos

En esta sección se pondrá a prueba el consumo base de distintos sistemas operativos compatibles con la placa de desarrollo Raspberry Pi 3B+.

Se medirá el consumo en miliamperios durante 60 minutos desde que el sistema arranca y se eliminarán los 5 primeros minutos. La mediciones de los primeros 5 minutos no serán representativas ya que durante el arranque del sistema se suelen producir altos picos de consumo debidos a las operaciones que se ejecutan durante el arranque. El estado de los sistemas operativos medidos será limpio, es decir, no se realizará ninguna modificación a estos más allá de haberlos actualizado previamente a la última versión. Esto quiere decir que estarán sin optimizar y que tendrán servicios activos en segundo plano consumiendo energía que pueden ser irrelevantes. La Raspberry Pi permanecerá durante toda la medición conectada a Internet mediante un cable Ethernet RJ45 y se mantendrá una sesión SSH abierta por él. Algunas distribuciones podrían tener activa la interfaz UART por defecto, se indicará explícitamente si esto fue así.

La prueba se realizará en dos tandas, primero se probarán los sistemas operativos que el fabricante de los sistemas Raspberry Pi pone a disposición de los usuarios: tres variantes del sistema operativo RaspberryPiOS. La diferencia entre estas tres distribuciones es la presencia o no de entorno gráfico y la cantidad de software pre-instalado.

En la segunda tanda se probarán los sistemas operativos más populares y livianos y que ofrezca la comunidad y se comparará su consumo con el menor de los consumos de los sistemas operativos oficiales de Raspberry Pi.

Así partir de aquí, las gráficas mostradas representan: en el eje de abscisas el número de muestras tomadas en 55 minutos y en el eje de ordenadas el consumo medido expresado en miliamperios.

En la figura 18 se observa la comparativa entre las tres variantes del sistema operativo oficial RaspberryPiOS. Se pueden apreciar diferencias de consumo significativamente altas de las dos primeras variantes con entorno gráfico (RaspberryPiOS y RaspberryPiOS_Full) frente a la que no dispone de este, Raspberry OS Lite. Estas se explican, en primer lugar, porque albergan una mayor cantidad de software preinstalado (Full) y servicios extra que la variante más liviana y, en segundo lugar, por la presencia misma del entorno gráfico y la salida de imagen continua por la interfaz HDMI. Esto produce un alto impacto en el consumo energético, al presentar una carga de trabajo adicional en el procesador para mantener activa continuamente la interfaz gráfica. También cabe destacar la variabilidad de las mediciones en las que se producen numerosos picos de consumo a lo largo de toda la medición. Estos picos se achacan a servicios que

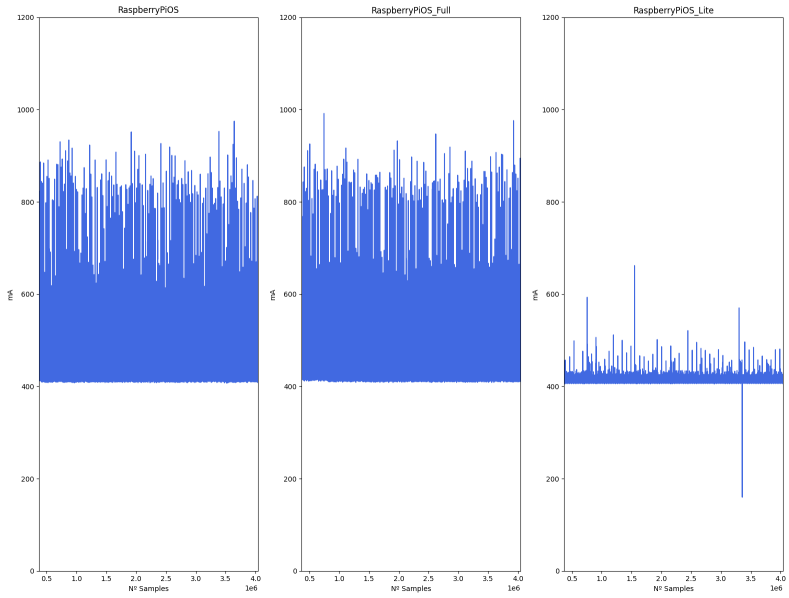

Figura 18: Representación del consumo en reposo de los sistemas operativos oficiales.

se encuentran en segundo plano realizando alguna tarea del sistema operativo.

En vista de los resultados anteriores, se concluye que para tener una mayor precisión del consumo real de los programas de experimentación es necesario tener un sistema operativo lo más liviano posible, en otras palabras, sin interfaz gráfica y carente, en la medida de lo posible, de servicios activos que interfieran en la medición. A su vez, deberán de permitir una instalación sencilla del software requerido anteriormente justificado en el apartado 4.5.1 de la metodología.

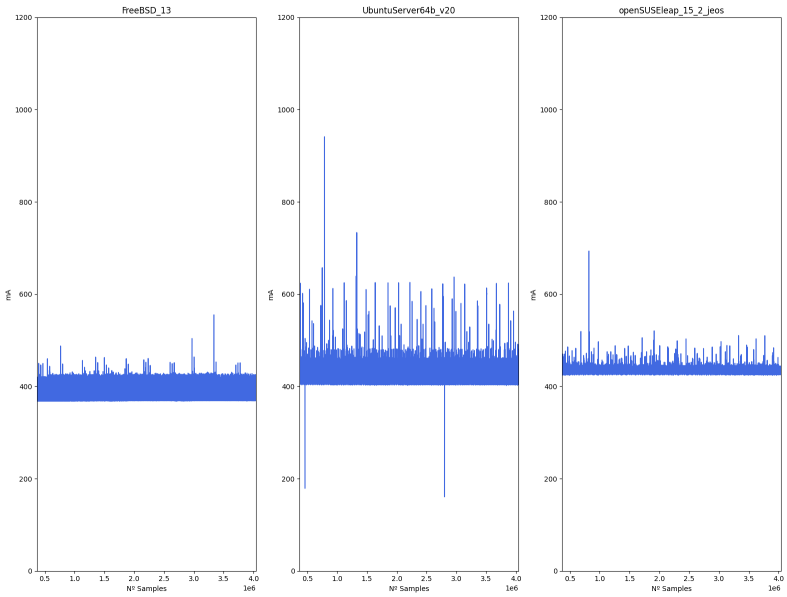

Figura 19: Representación del consumo de los sistemas operativos FreeBSD 13, Ubuntu 20 y openSUSE Leap 15.2

En la figura 19 se observa la comparativa entre los sistemas operativos FreeBSD 13, Ubuntu 20 y openSUSE Leap 15.2. En este caso, los tres sistemas operativos tienen por defecto habilitada la interfaz uart de forma adicional a la ssh. El consumo más estable lo obtuvo la distribución openSUSE con una media de 428.75 mA y desviación típica de 1.441 mA.

Finalmente en la figura 20 se representan los diferentes consumos básicos los sistemas operativos ligeros DietPi y tinyCore/PiCore enfrentados a Raspberry Pi OS Lite. Raspberry Pi OS Lite resultó tener un consumo medio de 410.62

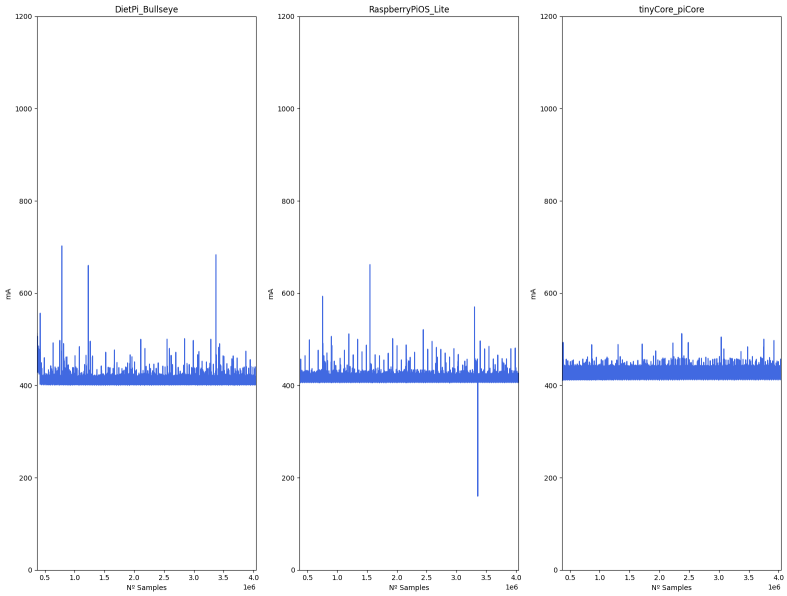

Figura 20: Representación del consumo de los sistemas operativos DietPi y piCore frente a Raspberry OS Lite.

mA y una desviación típica de 1.817 mA, que es el menor de los consumos de entre todas las distribuciones puestas a prueba.

Por tanto, de ahora en adelante para la medición y para el sistema de experimentación se utilizará el sistema operativo Raspberry Pi OS Lite tras realizar una optimización del consumo del mismo, como se verá más adelante.

5.1.5. Pruebas de consumo de los protocolos de comunicación

Los sistemas Raspberry Pi ponen a disposición del usuario una gran variedad de protocolos de comunicación para transferir archivos u órdenes al sistema. Para implementar un sistema autónomo de medición energética entre un equipo medidor y de experimentación surge la necesidad de utilizar un protocolo de comunicación que permita iniciar los experimentos y transferir los resultados. Los principales protocolos de comunicación disponibles son:

- **UART (Universal Asynchronous Receiver-Transmitter)**: Mediante los pines GPIO Tx-Rx permite iniciar sesión y transferir órdenes de línea de comandos. Con el software y la gestión de paridad adecuada se podría utilizar para transferir archivos. El coste energético debería ser muy bajo pero también lo es su velocidad de transferencia, que como máximo alcanzaría unos 2500000 baudios, esto es, aproximadamente 2.5Mbps. Si bien puede ser eficiente para sincronizar la experimentación, esta velocidad puede resultar demasiado lenta para transmitir los archivos y resultados de la experimentación.

- **SSH (Secure SHell)**: Protocolo de transferencia de archivos y de inicio de sesión seguro con cifrado basado en el protocolo TCP/IP. Este protocolo hace uso de la interfaz Ethernet del sistema Raspberry Pi para transmitir información. Tradicionalmente la velocidad es FastEthernet 100Mbps, aunque los modelos recientes, como la Raspberry Pi 3B+ implementan Gigabit Ethernet con velocidades de 300Mbps a 1000Mbps. El coste energético puede ser elevado debido a que, en este tipo de sistemas, la interfaz Ethernet depende de la alimentación de todo el hub USB.

- **Telnet**: Al igual que SSH es un protocolo de transferencia de archivos y de inicio de sesión basado en el protocolo TCP/IP. La principal diferencia es que no implementa cifrado por lo que se espera que el consumo energético sea menor al carecer de etapa de cifrado. Nuestras comunicaciones no resultan críticas para la privacidad por lo que puede ser una alternativa a SSH si presenta un menor consumo.

- Otros protocolos: También se dispone de la presencia de otros protocolos como el SPI (Serial Peripheral Interface) o el IIC (Inter-Integrated Circuit), pero no se van a entrar a valorar ya que no permiten el inicio de sesión, sólo transferencia simple de ordenes.

A continuación, se analizará el consumo del coste energético de los diferentes protocolos de comunicación, para ello se realizó una medición de 60 minutos, de la cual se descartaron las muestras a los primeros 5 minutos debido a que se ve reflejado el arranque del sistema operativo y las mediciones obtenidas no resultan representativas.

En la figura 21 se compara la diferencia del consumo basal, sin tener conectado ningún cable Ethernet, de tener iniciada una sesión UART de tener desactivado el periférico. Así, el consumo basal sin ningún protocolo de comunicación activo ni haber iniciado sesión es de 368.798mA con una desviación típica de 2.267mA. Manteniendo el Ethernet desconectado e iniciando sesión mediante UART se obtiene un consumo medio de 381.96mA con una desviación típica de 2.376mA.

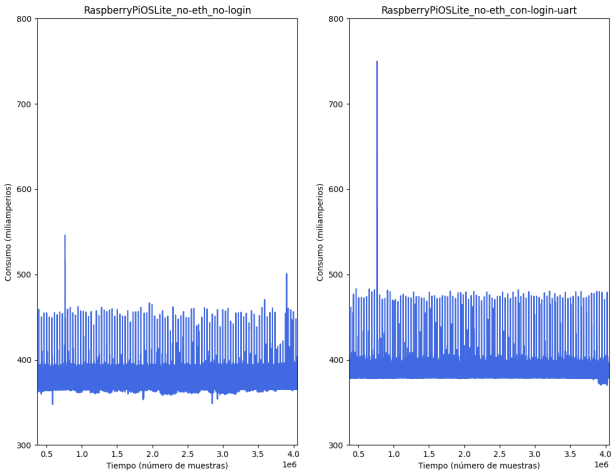

Figura 21: Comparativa con Ethernet desconectado entre no iniciar sesión e iniciar de sesión por UART.

En la figura 21 se compara la diferencia del consumo basal de los periféricos UART, SSH y Telnet, teniendo conectada la interfaz Ethernet y conexión a Internet. En cada una de las pruebas se deshabilitó el resto de protocolos para que no interfiriesen en la medición.

En la tabla 2, se muestra una comparativa de la media y desviación estándar de la corriente medida usando los distintos protocolos de comunicación.

Como se puede observar en la ya mencionada tabla 2, los consumos obtenidos

Protocolo	Media de corriente (mA)	std (mA)
Sin protocolo ni iniciar sesión, con Ethernet	410.856	1.814
Inicio de sesión usando UART	424.558	1.983
Inicio de sesión usando Telnet	410.323	1.986
Inicio de sesión usando ssh	410.626	1.817

Cuadro 2: Comparativa de los valores conseguidos, donde se muestra la media y desviación estándar (std) de cada protocolo.

son similares unos con respecto de los otros, tanto en media como en desviación, a excepción del uso de UART que es ligeramente superior al resto.

Por lo tanto, el protocolo que hemos escogido es el de SSH, ya que permite una comunicación tanto de órdenes como de archivos a la vez que cifra las comunicaciones. Sorprendentemente la implementación de las funciones de cifrado que lo diferencian del protocolo Telnet no suponen un consumo energético mucho mayor por lo que resulta una mejor elección.

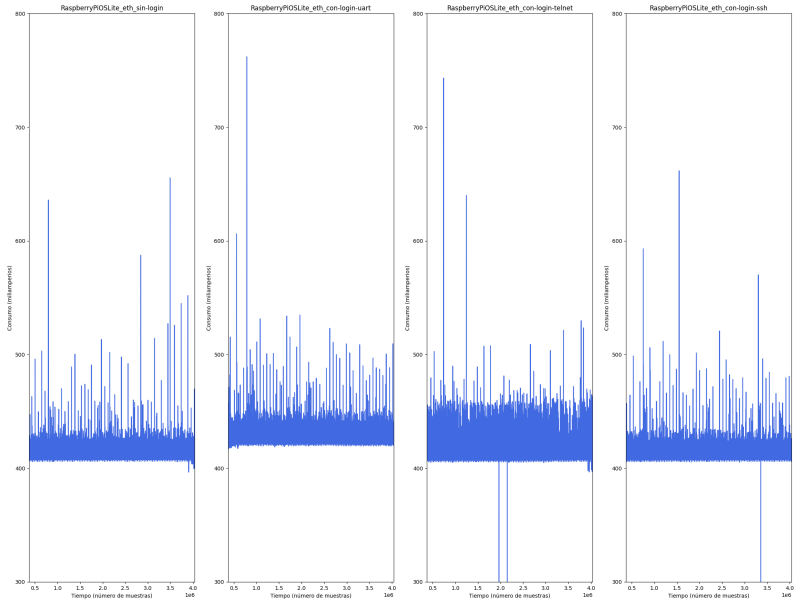

Figura 22: Comparativa con Ethernet conectado entre no iniciar sesión e iniciar sesión por UART, SSH y Telnet.

5.1.6. Optimización energética del sistema operativo

Una vez elegido el sistema operativo y la forma de comunicación, surge la necesidad de su optimización, a fin de reducir el número de servicios y funcionalidades en segundo plano que puedan afectar a la medición. Esta optimización se hizo de forma escalonada, es decir, acumulando progresivamente la desactivación de funcionalidades. Primero se probó a deshabilitar los periféricos inalámbricos (Bluetooth y Wi-Fi), todos los periféricos básicos (Audio, IIC, UART, SPI), a continuación los LEDs asociados a la conectividad Ethernet, seguidamente el hardware HDMI, para más adelante seguir con toda la alimentación del hub USB (de la cual depende el módulo que controla la interfaz Ethernet), y por último todos los servicios del sistema que se ha podido. A continuación, se muestran y analizan las figuras correspondientes a mediciones de 60 minutos de los consumos para cada una de las etapas de la optimización ya descritas, con el objetivo de observar cómo afecta cada funcionalidad al consumo del sistema. La medición de las etapas se realiza a una resolución de 1000 milliamperios con el módulo INA219, teniendo las Raspberry Pis conectadas a internet mediante cable Ethernet RJ45 e interconectadas entre sí con un pin de masa (GND). Ambos equipos se encuentran conectados a una misma regleta eléctrica.

5.1.6.1 Etapa 0: Consumo basal sin optimizar

En esta etapa, quisimos establecer una base desde la cual poder comparar con el resto de etapas. Podemos observar en la figura 23 como el consumo, por lo general, está entre los 400 y 450 mAh. Por otra parte, vemos que se producen picos consumo de 500 mA de manera periódica, otros superiores a 600mA y alguna ocasional caída de consumo. Nuestro objetivo es el de, entre las distintas optimizaciones que vamos a aplicar, poder conseguir reducir la aparición de estos, consiguiendo estabilidad, así como reducir los valores de consumo obtenidos. En concreto, en la medición de la figura 23 en un consumo basal sin optimizar observamos un consumo medio de 411.743 mA con una desviación típica de 1.925 mA. Este leve incremento de consumo con respecto a lo analizado en las pruebas del apartado 5.1.4 podría explicarse por la instalación de todo el software necesario para realizar las mediciones.

5.1.6.2 Etapa 1: Consumo basal deshabilitando los servicios inalámbricos (WiFi / Bluetooth)

Tras deshabilitar los servicios inalámbricos activos disminuye el consumo medio a 388.140 mA con una desviación típica de 2.276 mA.

5.1.6.3 Etapa 2: Consumo basal deshabilitando los servicios básicos (Audio, IIC, UART, SPI)

Tras deshabilitar los periféricos básicos no se produce una caida del consumo demasiado significativa, quizá porque ya estuvieran en su mayoría desactivados

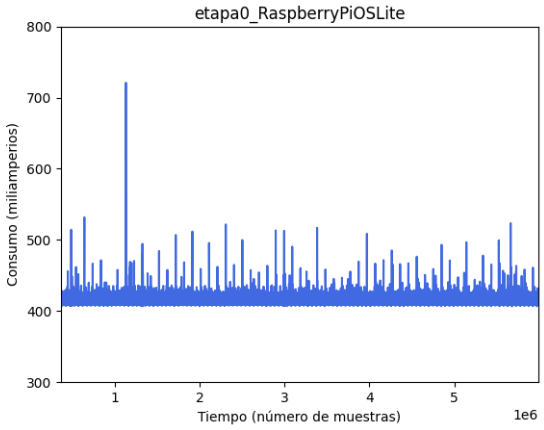

Figura 23: Etapa 0 - Consumo basal sin optimizar

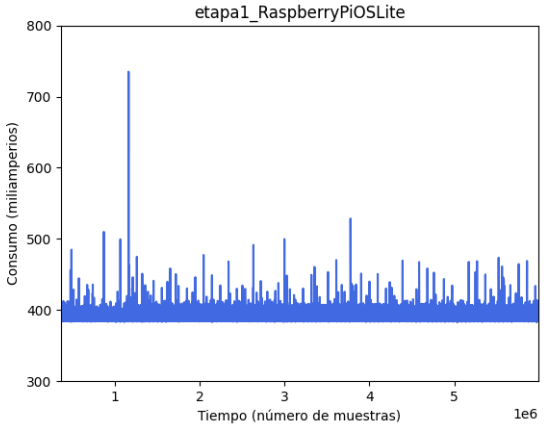

Figura 24: Etapa 1 - Consumo basal deshabilitando los servicios inalámbricos (WiFi / Bluetooth)

por defecto. El consumo medio es de 387.868mA con una desviación típica de 1.898 mA.

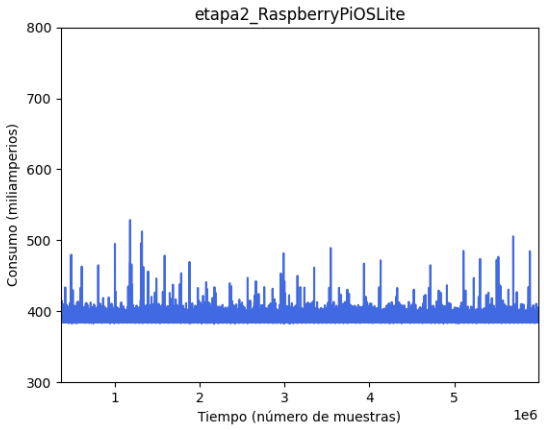

Figura 25: Etapa 2 - Consumo basal deshabilitando los servicios básicos (Audio, IIC, UART, SPI)

5.1.6.4 Etapa 3: Consumo basal deshabilitando los LEDs del sistema (Encendido, Actividad y Ethernet)

Sorprendentemente deshabilitar los LEDs produce un pequeño aumento de consumo. Pasamos a tener un consumo medio de 387.5946 mA con una desviación típica de 2.173 mA. Esta optimización de nuevo también resulta ser casi inapreciable.

5.1.6.5 Etapa 4: Consumo basal deshabilitando la interfaz HDMI

Al deshabilitar la interfaz HDMI se aprecia un nuevo descenso leve de consumo. Ahora el consumo medio es de 371.666 mA con una desviación típica de 1.800 mA.

5.1.6.6 Etapa 5: Consumo basal deshabilitando el concentrador USB interno

La Raspberry Pi se encuentra equipada de un hub USB interno que alimenta los cuatro puertos usb posteriores y a la interfaz Ethernet. Por ello, deshabilitar el hub USB interno es sin duda con diferencia la mejor optimización de consumo que se puede realizar en un sistema Raspberry Pi. El consumo medio descendió drásticamente hasta los 133.664 mA con una desviación típica de 2.1873 mA, más de 200 mA son consumidos por este subsistema.

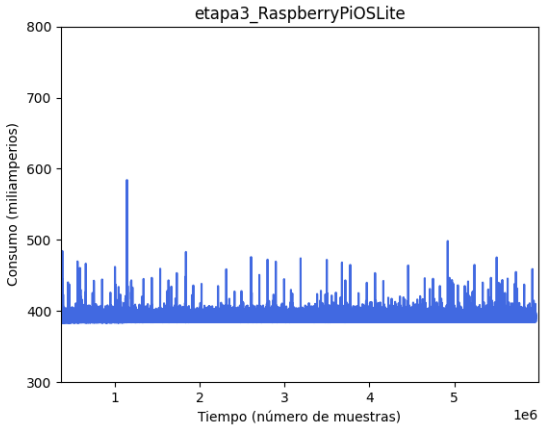

Figura 26: Etapa 3 - Consumo basal deshabilitando los LEDs del sistema (Encendido, Actividad y Ethernet)

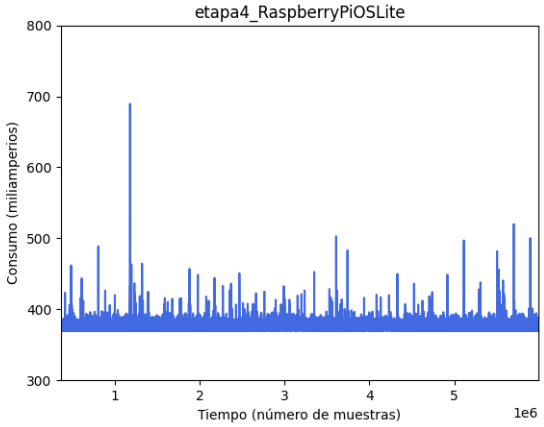

Figura 27: Etapa 4 - Consumo basal deshabilitando la interfaz HDMI

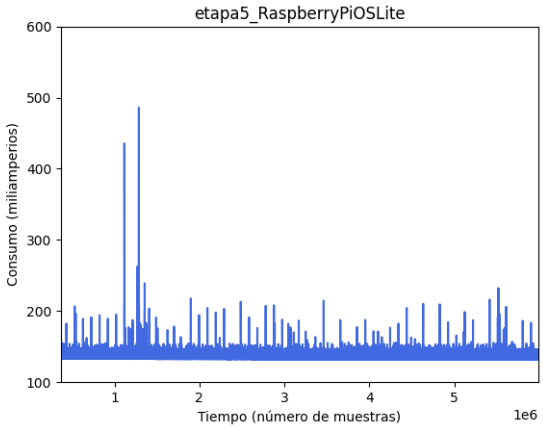

Figura 28: Etapa 5 - Consumo basal deshabilitando hub USB interno

5.1.6.7 Etapa 6: Consumo basal deshabilitando los principales servicios del sistema

Deshabilitando la mayor parte de los servicios del sistema se consigue estabilizar la desviación típica de las mediciones a 1.369 mA, manteniendo un consumo medio de 133.449 mA.

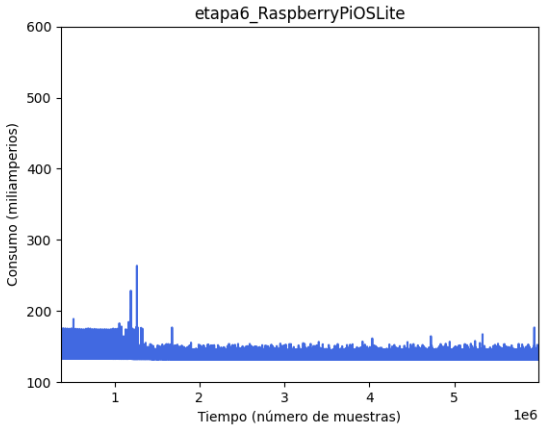

Figura 29: Etapa 6 - Consumo basal deshabilitando los principales servicios del sistema

5.1.6.8 Etapa 7: Consumo basal deshabilitando los servicios de inicio de sesión (`getty`, `session` y `user`)

Finalmente, pretendimos analizar el impacto energético de los servicios de inicio de sesión. Si estos servicios se detienen con ellos también caen las sesiones de red que hayan quedado abiertas, lo cual puede interrumpir el progreso de las ejecuciones programadas. Esta etapa es puramente opcional y sólo se realizará si resulta viable ya que el su consumo es bastante despreciable con una media de 133.112 mA y una desviación típica de 1.382 mA.

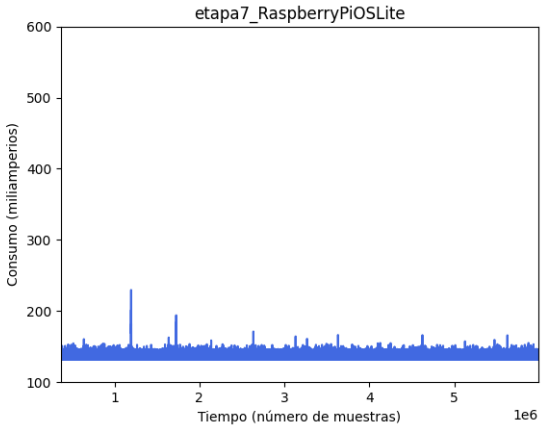

Figura 30: Etapa 7 - Consumo basal deshabilitando servicios de inicio de sesión

5.1.7. Observaciones iniciales de las mediciones de consumo

A lo largo del desarrollo del proyecto, se han hecho múltiples experimentos y mediciones de consumo. En todos ellos, hemos podido discernir el suceso mostrado en la figura 31.

Hemos mostrado las mediciones de consumo de 1.25 minutos tras el arranque, viendo como se producen unos picos de consumo irregulares y diferentes en cada experimento. De esta manera, decidimos eliminar este segmento de tiempo de las mediciones para obtener un resultado estable.

Del mismo modo, se observa como, durante los primeros 15 minutos tras el arranque de los dispositivos usados, pueden aparecer irregularidades de consumo sin periodicidad, en otras palabras, de manera única. Se ha probado que este suceso ocurre independientemente del protocolo de comunicación usado, mediante las pruebas pertinentes de los protocolos. Únicamente con dos especificaciones hemos comprobado la ausencia de esta irregularidad: usando el sistema operativo FreeBSD y cuando no tenemos sistema operativo. De este modo, se asume esta irregularidad como parte del funcionamiento del sistema operativo y se mide tras 15 minutos del arranque.

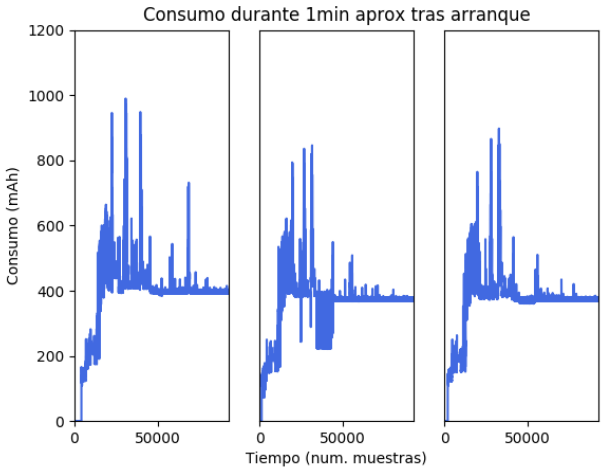

Figura 31: Medición del consumo dado por nuestro equipo durante el primer minuto aproximadamente tras el arranque.

5.1.7.1 Conclusiones a extraer

Las caídas de consumo en las gráficas se deben a que las tierras entre la MediPi y la ExperiPi estaban desconectadas, por tanto, deben de estar conectadas y esto es una advertencia importante a tomar.

Que MediPi y ExperiPi estén conectadas mediante fuentes de alimentación externas e independientes a la misma zapatilla eléctrica afecta a las mediciones. En las pruebas realizadas se ha detectado la presencia de ruido eléctrico en mediciones realizadas de esta maneras en comparación a mediciones realizadas en líneas eléctricas independientes.

5.1.8. Ejecución directa sin sistema operativo

Siguiendo el proyecto `raspberry-pi-os` de `s-matyukevich` en GitHub [14] se consiguió ejecutar un *Hola Mundo*, escrito en C, directamente sobre el microprocesador de la Raspberry Pi sin necesidad de sistema operativo. El resultado de su medida se puede observar en la figura 32.

Esto plantea una serie de ventajas e inconvenientes: al ser una ejecución directa eliminamos de la medición de energía toda interferencia de la ejecución del sistema operativo; sin embargo, al carecer de sistema operativo perdemos la funcionalidad de comunicación por red para cargar nuevos programas de forma automática. De hecho, la compilación y la carga en la raíz de la tarjeta microSD del fichero `kernel8.img` (que contiene el programa a ejecutar de forma directa)

es un proceso no automatizable a la hora de poder efectuar miles de ejecuciones de diferentes programas y variar la ejecución de los mismos en tiempo real. Puede ser interesante explorar esta opción de todos modos a fin de establecer una comparación entre la ejecución de los programas con y sin sistema operativo.

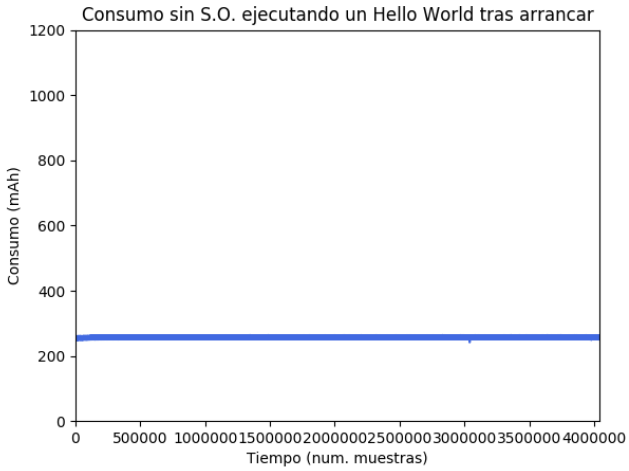

Figura 32: Medición de 60 minutos del consumo de la ExperiPi sin sistema operativo, ejecutando un "Hola Mundo" tras el arranque.

5.2. Validez de las mediciones conseguidas

Seguido de las comparativas de consumo, se desea comprobar si las mediciones realizadas se producen en las condiciones que se desean. Se harán una serie de pruebas para comprobarlo, descritas a continuación.

5.2.1. Comparativa entre los distintos métodos de espera

A lo largo de las mediciones, previo a la ejecución de una iteración del programa, se produce una ligera espera para asegurar la correcta sincronización entre los dispositivos de medición y experimentación. Del mismo modo, si se desease, podría añadirse un tiempo adicional previo a esta espera para asegurar que las iteraciones sean independientes unas de las otras, aislándolas en el tiempo con una espera suficientemente larga.

Estas esperas, sin embargo, de manera ideal no tienen efecto en la ejecución que se produce a continuación. De este modo, se desea comprobar si se produce o no un efecto utilizando distintos métodos. Estos métodos de espera planteados son:

- Función “sleep” de la librería “time”, estándar de Python: La documentación oficial de referencia detalla que suspende al hilo que llama a esta función durante un tiempo recibido, que puede tener parte decimal y parte entera. La duración de la espera puede ser menor al esperado si se recibe alguna señal. Además, puede ser arbitrariamente más largo si el sistema requiere más tiempo para organizar otras tareas.
- Función “wait” de la clase “Event”, a su vez de librería “threading”, estándar de Python: Se detalla que la clase “Event” modela un evento asíncrono. La función “wait” de esta clase únicamente detalla que espera durante un tiempo determinado. Al mirar la implementación de su función en la versión 3.6, vemos como para implementar la espera hace uso de un cierre de exclusión mutua, también conocido como candado, que “adquiere” el cerrojo durante un tiempo igual al especificado.
- Bucle de espera ocupada: Se implementa un bucle que espera hasta que pase un determinado tiempo. Para ello, se utiliza la función “time” de la librería “time” de la librería estándar de Python, que nos devuelve la cantidad de segundos sucedidos desde una fecha fijada por igual en todo computador. Es similar al conocido como tiempo Unix, o Unix timestamp, de igual funcionamiento. Sin embargo, esta función nos da suficientes decimales como para obtener el tiempo en el orden de los microsegundos.

Para ello, realizamos un experimento en el que se mide una espera de un segundo, un benchmark, otra espera de la misma duración, el benchmark de nuevo y finalmente una espera adicional. Los resultados se muestran en la figura 33.

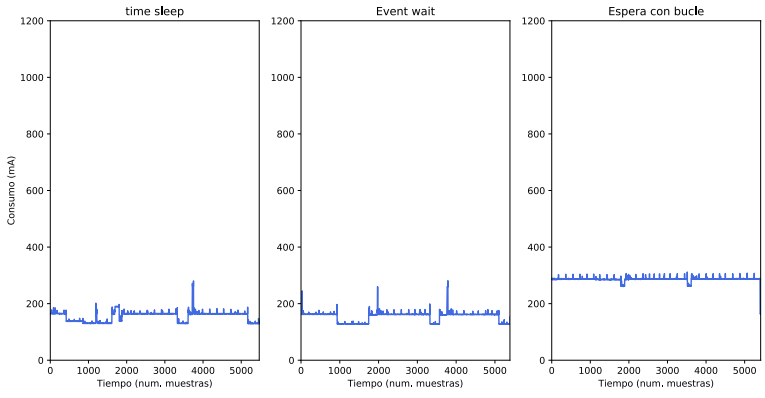

Figura 33: Comprobación del consumo del experimento propuesto usando los tres métodos diferentes de espera planteados.

Los resultados de los experimentos hechos con los tres métodos muestran un tiempo, tanto en el dispositivo de medición como el de experimentación, de 3.22s

aproximadamente. 3 de estos segundos corresponden a las 3 esperas realizadas, mientras que los 0.22 segundos corresponden a la ejecución del benchmark, que tarda 0.11 segundos aproximadamente.

Como se puede observar de la figura, usando la función de sleep no se puede discernir claramente la diferencia entre las funciones sleep y el benchmark. Usando la función wait de la clase Event se obtiene un resultado más diferenciado, pero sin poder asignar un rango de valores a qué corresponde el benchmark y qué a la espera, ya que los datos tras subidas y bajas de flanco no tienen una longitud fija. Sin embargo, usando la espera ocupada podemos observar como hay tres rectas estables y dos ligeras variaciones, correspondientes a las esperas y ejecuciones de los benchmarks respectivamente.

Una vez realizadas estas observaciones, podemos afirmar que **se escoge el método de espera mediante el bucle de espera ocupada**.

5.2.2. Tiempos medidos en ambos dispositivos usados

La comprobación más fundamental a hacer es la de los tiempos medidos. Es decir, si para un benchmark de duración n segundos, conseguimos medir los n segundos que corresponden a su ejecución. Para ello, se diseña el experimento de ejecutar un benchmark durante una hora y almacenar el tiempo en ambos dispositivos.

Las diferencias calculadas entre los tiempos medidos por ambos dispositivos se muestra en la figura 34.

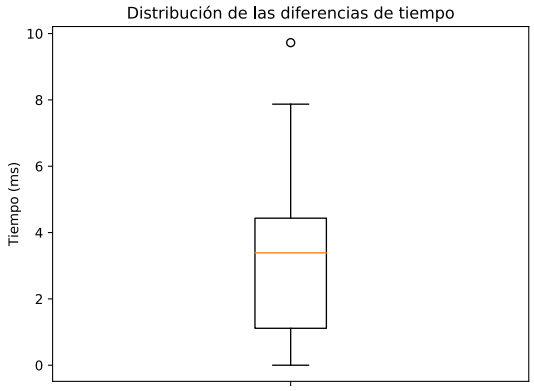

Figura 34: Comprobación de las distribuciones de tiempos conseguidos en ambos dispositivos al medir un benchmark de una hora.

Los estadísticos referentes a las diferencias de tiempo presentadas se encuentran en la tabla 3.

Diff t. ejec. y t. medido	Media	Mediana	Máximo	Mínimo
Tiempo (ms)	2.9550	3.3886	9.7258	0.0004

Cuadro 3: Estadísticos de las diferencias entre el tiempo de ejecución del dispositivo de experimentación y el medido.

A priori, la diferencia que se aprecia en la distribución se podría decir que no es significativa, dada la unidad del eje vertical. Estas diferencias, en benchmarks con tiempo de ejecución de segundos, pueden no presentar gran importancia, pero existen benchmarks con los que no nos situemos en este caso.

Esta variabilidad se presenta debido a que, para que el dispositivo de medición compruebe que el de experimentación ha comenzado, se realiza un polling de los pines GPIO. Este polling tarda en realizarse un milisegundo, tiempo de espera para evitar los posibles rebotes del cambio de valor del pin, más el tiempo que tarde el sistema en realizar varias operaciones. Se ha probado a reducir el número de operaciones entre cada polling, o incluso reduciendo el tiempo de espera de los rebotes, obteniendo un resultado similar.

Se concluye, finalmente, que se debe a limitaciones de la implementación realizada, el equipo y, en mayor medida, el lenguaje de programación escogido. Los tiempos de espera son implementados, como se explicó previamente, mediante una espera ocupada donde se hace uso del bucle "while", así como de la función "time" de la librería estándar "time" de Python. La combinación de ambos, sumado a las operaciones necesarias hechas entre comprobaciones, resultan en esta diferencia de tiempo.

De esta manera, se recomienda el uso de benchmarks de una significativa duración, mayor a varios milisegundos. No solo por el problema presentado, si no también por la resolución que se tiene actualmente a la hora de obtener mediciones. Se estima, de manera empírica, que es de aproximadamente 1670 muestras por segundo, por lo que, en benchmarks de escasos milisegundos no se obtendría un número suficiente de muestras.

6. Conclusiones

En un inicio, se pretendía definir una suite de benchmarks para la optimización y comparación de energía, además de un diseño hardware y software con el que poder realizar mediciones de consumo. Tras todos los pasos descritos en la metodología y discusión, podemos decir que se han realizado las siguientes contribuciones:

- A partir de unos benchmarks ya definidos, crear una suite propia.
- Crear un método de caracterización de benchmarks relativo al consumo energético.
- Diseño hardware de alta adaptabilidad de un equipo para la medición del consumo energético de un software.
- Diseño software de fácil uso, con un variado rango de configuraciones a escoger para distintas funcionalidades, y alta escalabilidad a nuevas necesidades del usuario.
- Manual de uso para usuarios destinado a facilitar y comprender cómo usar el software.
- Estudio de las decisiones pertinentes del diseño, tales como la elección del sistema operativo, optimización al reducir servicios, así como decisiones de la implementación del software.
- Validez en varios aspectos de los resultados conseguidos.
- Varias opciones exploradas de alto interés al objetivo definido.

Dadas estas contribuciones, se puede afirmar que, a priori, se ha establecido una base para continuar con el desarrollo e investigación de la reducción del consumo energético. Además, se han observado un amplio margen de estudio de cara al consumo desde múltiples perspectivas, principalmente desde el punto de vista de la toma de valores. De esta manera, se evidencia la necesidad de investigación en este campo, así como proseguir en la reducción de consumo del software.

6.1. Líneas futuras

A partir del desarrollo realizado, se han visto los siguientes puntos como posibles líneas futuras de investigación:

- Reducción del consumo energético, al disponer de las técnicas vistas en este estudio para la medición de consumo.
- Estudio de un nuevo método de estudio de la ejecución de la suite, donde bien se incluyen:

    - Medición de nuestra suite al completo, sin suponer que el consumo total se puede calcular a partir de los consumos individuales de benchmarks. Este paso únicamente sería necesario si este procede, tras hacer pruebas de si el consumo total se puede aproximar a partir de ese cálculo.
    - Ejecución de la suite en distinto orden del establecido en un inicio.
    - Combinación de este método junto a otro que, dado un benchmark, intente reducir su consumo energético, de forma que se obtendría una suite de menor consumo conforme progrese el método.

- Estudio de posibles nuevas optimizaciones a realizar al sistema operativo, desde el punto de vista de los servicios.
- Estudio de uso de nuevos sistemas operativos no contemplados en este estudio, de cara a conseguir una estabilidad mayor a la ya conseguida.
- Establecimiento de una nueva metodología para hacer mediciones de consumo sin el uso de un sistema operativo, como se vio previamente, consiguiendo así un resultado de gran estabilidad.
- Estudio del uso de nuevos módulos que permitan obtener lecturas, así como posibles problemas derivados de uso, como puede ser la ausencia de un driver que permita su manejo.

7. Anexo

7.1. Guía de uso

7.1.1. Requisitos hardware y software:

- Requisitos mínimos:

  - Dispositivo de micro-arquitectura hardware con módulo INA219, pines GPIO y disponibilidad protocolo SSH a través del estándar Ethernet o WiFi. A este se le conoce como **dispositivo de medición** o ***MediPi***.
  - Dispositivo de micro-arquitectura hardware con pines GPIO y disponibilidad protocolo SSH a través del estándar Ethernet o WiFi. Este dispositivo tiene el nombre de **dispositivo de experimentación** o ***ExperiPi***.
  - Ambos dispositivos deben de tener una instalación de una microdistribución de Linux que permita la interpretación de órdenes recibidas por el protocolo SSH.
  - Además, En estos dispositivos se ha de tener a disposición los siguientes paquetes instalados:
    - La infraestructura de compilación *LLVM* en su versión 9.
    - *Python* en su versión 3, junto a su correspondiente `pip` (`python3` y `python3-pip`).
    - Aplicación `rsync`.
    - Aplicación `scp`.
    - Aplicación `rclone`.
    - Paquete `i2c-tools`.
    - Los paquetes: `libopenjp2-7` y `ttf-dejavu` para el uso de la pantalla en los dispositivos.
  - Python en su versión 3 hace uso de las siguientes librerías, que necesitan ser instaladas:
    - `smbus`.
    - `jMetalPy`.
    - `Spidev`, `Pillow`, `libatlas-base-dev`, `adafruit-circuitpython-rgb-display` para el uso de la pantalla en los dispositivos.

  De esta manera, la instalación de las paquetes y librerías queda reflejada en el siguiente comando:

```
# Instalar paquetes esenciales
$ sudo apt install -y tmux build-essential llvm-9
python3 python3-pip python3-numpy python3-pil python3-
spidev  python3-smbus i2c-tools libatlas-base-dev
libopenjp2-7  ttf-dejavu rclone
```

```
3
4       # Instalar paquetes de Python
5       $ python3 -m pip install --upgrade pip
6       $ python3 -m pip install --upgrade Pillow
7       $ python3 -m pip install --upgrade adafruit-
        circuitpython-rgb-display
8       $ python3 -m pip install --upgrade adafruit-
        circuitpython-debouncer
9
```

- Requisitos recomendados:
  - Raspberry Pi modelo 3B+ y 3B, como dispositivos con y sin sensor INA219, respectivamente.
  - Raspberry Pi OS Lite instalado en ambos.
  - Conexión de ambos mediante WiFi, facilitando la maniobrabilidad al permitir el acceso remoto a estos dispositivos. Del mismo modo, se tiene un acceso más fácil a los archivos generados mediante el uso del software que se plantea, estando los dispositivos preparados para la subida de archivos al servicio Google Drive.
  - La instalación del paquete `ipython3` mediante apt, debido a su gran utilidad para la depuración de código python.

7.1.2. Esquema de conexiones

A continuación, se muestra en la figura 35 las conexiones necesarias entre los dispositivos utilizados. Debido a limitaciones del software usado, en esta aparecen baterías, las cuales en realidad deben de ser una conexión a corriente.

7.1.3. Configuración de la red y sus dispositivos

Los sistemas MediPi y ExperiPi intercambian archivos ejecutables y resultados de experimentación mediante una red basada en el protocolo TCP/IP. La interconexión se puede hacer de manera local con un *switch* a través de la interfaz Ethernet o mediante un *router* WiFi que además proporcione acceso desde una red externa de área ancha.

El usuario final podrá hacer uso de cualquier tipo de red siempre que adapte los *scripts* de comunicación. En este caso por convenio se ha adoptado la subred `192.168.3.0` con máscara de red `255.255.255.0` para la red de área local, reservando la dirección IP `192.168.3.1` para el *router* o puerta de enlace. La dirección `192.168.3.2` está reservada para que pueda ser asignada de forma manual al ordenador de un usuario supervisor de las mediciones en caso de que no haya un servidor DHCP activo. A partir de aquí las direcciones IP **múltiplos de tres** se reservan para los sistemas MediPi, esto es, `192.168.3.3`, `192.168.3.6`, `192.168.3.9` y así sucesivamente. A cada MediPi se le han reservado las dos siguientes direcciones IP para las ExperiPis que es capaz de medir,

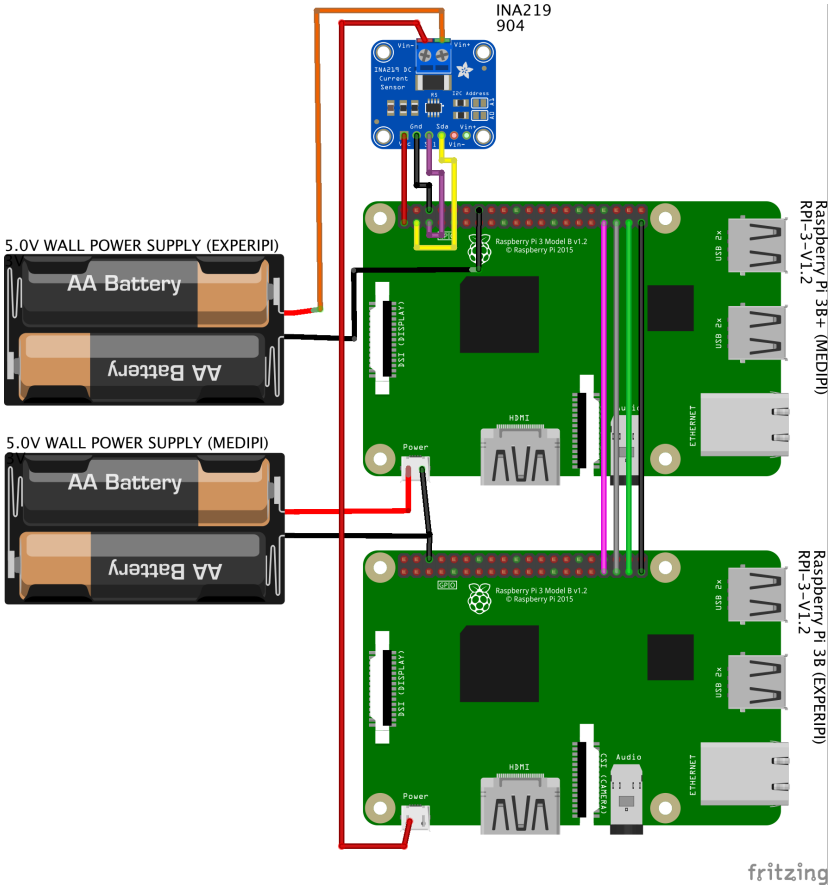

Figura 35: Diagrama de conexiones de los dispositivos.

pudiendo realizar mediciones en tupla o en trío. Así, la MediPi `192.168.3.3` puede tener asociadas las ExperiPis `192.168.3.4` y `192.168.3.5`.

En esta red se podría albergar como máximo hasta 84 sistemas MediPi con sus correspondientes 168 sistemas ExperiPi, cantidad muy sobredimensionada para una pequeña o mediana experimentación. El usuario del sistema siempre será libre de utilizar y configurar otro rango de direcciones IP diferente según sean sus necesidades.

7.1.3.1 Configuración del router

En este apartado se detalla la configuración básica que se debe realizar en el *router* para crear la red y redirigir los puertos desde una red externa a la *intranet* donde se encuentran los sistemas MediPi/ExperiPi. En este manual se describen los pasos a realizar para el *router* TP-Link Archer C9, deberá seguir pasos similares para la configuración del *router* del que el usuario disponga.

En primer lugar, diríjase al apartado LAN de su *router* y establezca la dirección IP de puerta de enlace como `192.168.3.1` y máscara de red `255.255.255.0` como se detalla en la figura 36.

Figura 36: Configuración de la red LAN.

A continuación será recomendable configurar un servidor DHCP. Su utilidad radica en que por defecto los sistemas Raspberry Pi utilizan un direccionamiento IP dinámico, y si se emplea un sistema operativo sin interfaz gráfica con conexión remota SSH y una conexión Ethernet le interesará que el sistema asigne una IP temporal a fin de poder localizar este nuevo sistema en la red mediante un escáner de red y conectarse a él para configurar una IP estática. También resulta útil para que se asignen direcciones temporales a los equipos de supervisión del usuario a fin de no tener que realizar una configuración manual que pueda interferir con una dirección ya existente.

En la figura 37 se ha configurado un servidor DHCP para que reparta direcciones IP de forma automática en el rango `192.168.3.100 - 192.168.3.199`, en total 100 direcciones temporales. Este rango deberá adaptarse según las necesidades del usuario ya que reduce el número máximo de sistemas MediPi a 32

y el de ExperiPi a 64. Opcionalmente se han especificado los DNS de Google `8.8.8.8` e IBM `9.9.9.9` por su popularidad y facilidad de uso.

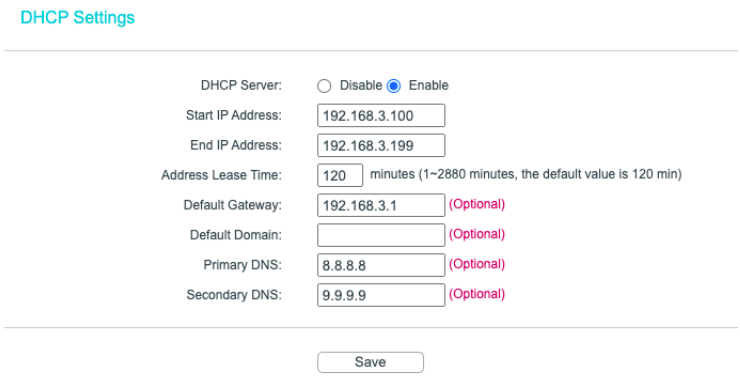

Figura 37: Configuración de un servidor DHCP.

Finalmente, si hace uso de un *router*, es altamente recomendable configurar el re-direccionamiento de puertos a veces conocido como *Port Forwarding* o *Virtual Servers* dentro de los apartados de configuración. Esto le permitirá realizar conexiones SSH desde fuera de la red hacia la red interna donde se encuentre el sistema de experimentación. Como el puerto SSH utiliza el puerto 22 y los sistemas MediPi-ExperiPi tienen subfijos de red del 3 al 98 en esta configuración deberá programarse en el router una asociación del puerto 22xx al puerto 22 de la dirección `192.168.3.xx`, siendo xx cualquier número entre 3 y 99. Para las direcciones IP de la 3 al 9 se rellenan con ceros. En la imagen 38 se está configurando la redirección del puerto 2218 al puerto 22 de la dirección IP `192.168.3.18` con los protocolos TCP y UDP (opción *All*). La opción *Status* se establece a *Enabled* para habilitar la asociación. Tras guardar la configuración volverá a una pantalla como la de la figura 39 donde aparecerán todas las redirecciones que se encuentran configuradas. Esta asignación suele realizarse de forma completamente manual mediante la interfaz gráfica de usuario del *router*.

7.1.3.2 Configuración del direccionamiento estático en sistemas Raspberry Pi

Es necesario hacer uso de un direccionamiento IP estático para saber fácilmente qué equipo es un MediPi (tienen subfijos de red múltiplos de tres) y cuáles sus ExperiPis asociadas (los dos subfijos siguientes a una dirección múltiplo de tres). Para configurar una IP estática en un sistema Raspberry Pi basta con iniciar sesión con el comando `ssh` y modificar el fichero `/etc/dhcpcd.conf`. Por ejemplo, para configurar desde su equipo supervisor, el direccionamiento estático del sistema al que se le ha asignado una dirección IP dinámica `192.168.3.102`, si se encuentra en la misma red, introduzca el comando `ssh pi@192.168.3.102` o

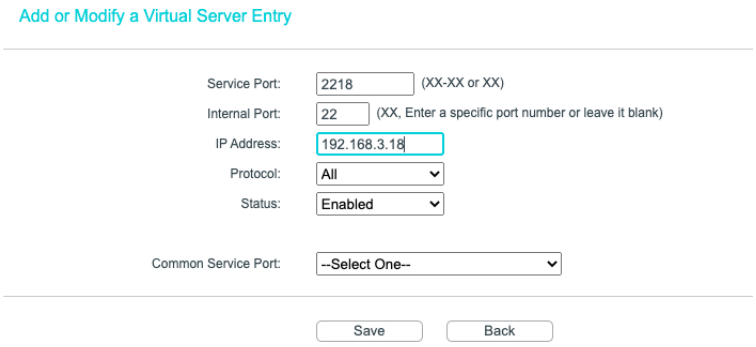

Figura 38: Configuración de la redirección de puertos SSH.

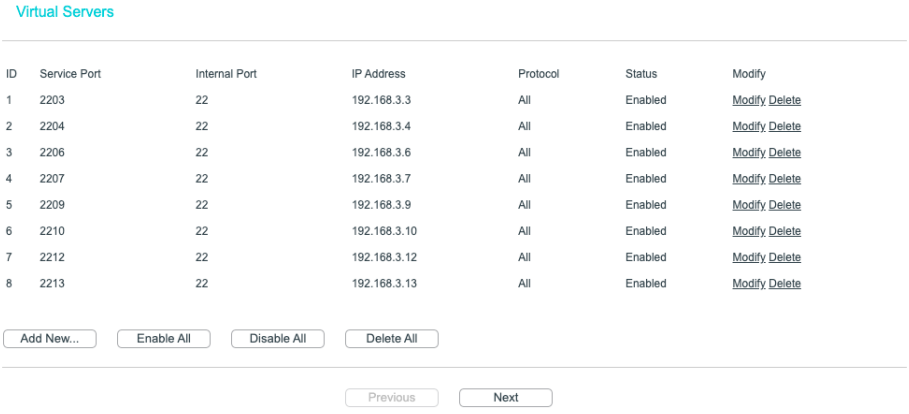

Figura 39: Visión global de la redirección de puertos SSH.

introduzca `ssh -p 22102 pi@10.10.10.27` si se conecta remotamente a través de otra red cuya dirección pública del *router* es la `10.10.10.27` y se encuentra activo el re-direccionamiento de puertos SSH del 22102 al 22 de la dirección local `192.168.3.102`. Deberá saber previamente con seguridad estos datos.

Una vez dentro del sistema abra y edite el fichero de configuración con `sudo nano /etc/dhcpcd.conf` y añada al final del mismo la siguiente configuración para establecer la dirección IP estática `192.168.3.3` en la interfaz de red Ethernet (`eth0`) y en la interfaz de red WiFi (`wlan0`) si hace uso de ella. Puede consultar el nombre de sus interfaces introduciendo el comando `ip address show` en una ventana de terminal.

```
1    interface eth0
2    static ip_address=192.168.3.3/24
3    static routers=192.168.3.1
4
5    interface wlan0
6    static ip_address=192.168.3.3/24
7    static routers=192.168.3.1
8
```

7.1.3.3 Configuración de la subida de archivos a Google Drive

El software desarrollado está preparado para subir los archivos con los datos de las mediciones al servicio Google Drive si se dispone de una conexión a Internet activa al finalizar la medición. Para ello, se ha de configurar previamente el dispositivo de experimentación para que el servicio esté disponible. Los pasos a dar son los siguientes:

- Para comenzar, necesitamos abrir la configuración de la aplicación `rclone`, introduciendo el comando `rclone config` en una terminal.
- Se desplegará un menú con varias opciones. La que debemos de seleccionar es `n) New remote`. Con esta, configuramos un nuevo dispositivo de conexión remota, el cual nos brinda acceso al servicio, introduciendo `n`.
- Se nos pedirá un nombre del dispositivo remoto, mostrando `name)` por terminal. Este debe ser `gdrive`.
- El siguiente dato que se nos pedirá es el almacenamiento que usa este dispositivo. Se muestra una lista con las diferentes opciones, donde la que nos interesa es Google Drive. De esta manera, introducimos la letra que corresponda a `Google Drive`. En función de la versión de `rclone` instalada, el número puede cambiar, de modo que tendrá que consultarse.
- Los dos siguientes campos que nos solicitarán son `client_id)` y `client_secret)`. En estos no introduciremos dato alguno, ya que están reservados para la creación especifica de una aplicación de Google que haga las veces de `rclone` en entornos empresariales.

- A continuación, nos pedirá si queremos usar la auto-configuración del dispositivo remoto, mostrando `auto config: y/n>` en la terminal. Debemos introducir `n` para no usar esta configuración.

- Al no usar esta auto-configuración, se generará un enlace que debemos abrir para seleccionar a qué cuenta de Google Drive tendrá acceso nuestro dispositivo remoto. Una vez iniciamos sesión, debemos brindar a nuestro dispositivo los distintos permisos que solicita.

- Tras darle los permisos necesarios, se genera un código que hemos de introducir en la terminal. Tras introducirlo, podemos observar toda la configuración realizada hasta ahora. Finalmente, se nos pregunta si la configuración mostrada es correcta. Si es así, podemos introducir el parámetro `y` seguido del parámetro `q` para salir.

Esta serie de pasos descritos puede modificar debido a los posibles cambios de versión de `rclone`.

7.1.4. Uso del software de medición y experimentación disponible

El software está preparado para, desde un ordenador supervisor bajo el control del usuario, poder enviar instrucciones a nuestro dispositivo de medición que orqueste unas operaciones u otras, en función del fin que se desee.

Para ello, se dispone del archivo `pc_launch.py`, desde el que poder realizar todas las operaciones que se desee. Este debe de ser configurado, dando valor a una serie de variables, situadas en las líneas 289 a 296. A continuación, se listan estas, así como los valores que deben de tener:

- `id_host_medipi`: Identificador de la MediPi o dispositivo de medición, que bien puede corresponder a un número de puerto o al último octeto de la dirección IP. El formato de este puede estar escrito como un entero, o como una cadena de caracteres. Ejemplos: '12', 12.

- `id_host_exprpi`: Identificador de la ExperiPi o dispositivo de experimentación, que bien puede corresponder a un número de puerto o al último octeto de la dirección IP. El formato de este puede estar escrito como un entero, o como una cadena de caracteres. Ejemplos: '13', 13.

- `ip_red_wan`: Cadena de caracteres que representa la dirección IP del *router* a la que se debe de conectar nuestro equipo para, a través de WiFi, realizar las operaciones que sean necesarias. Ejemplo: '10.142.155.18'.

- `id_red_lan`: Cadena de caracteres que representa los tres primeros octetos que tienen como dirección IP a nivel de red local, teniendo el último octeto en 0. Ejemplo: '192.168.3.0'.

- `str_benchmarks_suites_directory`: Cadena de caracteres que indican la ruta, relativa o absoluta, en el equipo al directorio de suites de benchmarks. Ejemplo: './/EEMBC_benchmark_suites_redefinido'.

Una vez definidos los valores de estas variables, se tienen tres funcionalidades disponibles a realizar:

7.1.4.1 Medición del consumo basal

Trata de determinar el consumo del dispositivo de experimentación inactivo, sin realizar tarea alguna más allá de las del sistema operativo, tras tumbar todos los servicios.

Para ello, se ejecuta la función `lanzar_basal`. Los parámetros de esta función son:

- `tupla`: Objeto de clase `Measure_tuple`, donde se pasan los parámetros definidos al inicio del script `pc_launch.py`. En el script ya se define un objeto de esta clase, de modo que se define tupla = tupla.
- `tiempo_ejecucion`: Número, entero o con coma decimal, que indica durante cuánto tiempo se mide el consumo. Medido en segundos.
- `tumbarServicios`: Booleano que indica si se tumban o no los servicios del dispositivo de experimentación. Ejemplo: True, False.
- `subir_a_drive`: Booleano que indica si se sube o no a Google Drive el archivo, según la configuración hecha en el dispositivo de experimentación.

Como resultado de la medición, se genera un archivo llamado `consumption-basal-`, seguido del nombre de usuario dado al dispositivo de medición y la fecha y hora actual, así como otro llamado `mean-std-and-time-seconds-basal-`, seguido de los mismos elementos. El primero contiene las mediciones en una fila, y el segundo información relevante sobre la medición basal realizada. El contenido de este se especifica en el apartado siguiente.

En caso de que se subiese a Google Drive, la carpeta generada tendría el nombre del año, mes y fecha del día actual. Dentro de esta se encontraría los archivos ya mencionados.

7.1.4.2 Medición del consumo de suites de benchmarks

Mide el consumo de la ejecución de los benchmarks de cada una de las suites que se especifiquen. Cada benchmark puede ser medido durante la longitud que se desee, guardando los resultados individualmente. Esta longitud puede ser definida en un rango de iteraciones a ejecutar y/o en tiempo de ejecución.

Para ello, se ejecuta la función `lanzar_medicion`. Los parámetros de esta función son:

- `tupla`: Objeto de clase `Measure_tuple`, donde se pasan los parámetros definidos al inicio del script `pc_launch.py`. En el script ya se define un objeto de esta clase, de modo que se define tupla = tupla.

- `iteraciones_ini`: Número entero de iteraciones que se empieza a ejecutar los benchmarks. Este número va aumentando, siendo multiplicado por dos cada vez que se ejecuta $x \cdot 2^i$ veces.

- `iteraciones_fin`: Número entero de iteraciones que se ejecutan por última vez.

  Ejemplo de funcionamiento de estos dos parámetros: Si ajustamos los parámetros a `iteraciones_ini=5`, `iteraciones_fin=40`, nuestros benchmarks se ejecutarán primero 5 veces, luego 10, más tarde 20, y finalmente 40.

- `tiempo`: Número entero o flotante, que indica una cantidad de tiempo en segundos. Si el número es menor o igual que 0, el comportamiento será el anteriormente descrito. En caso contrario, el funcionamiento será el siguiente:

  Se itera el programa tantas veces como sea necesario hasta que su tiempo de ejecución sea igual o superior al definido en esta variable. Este proceso se repetirá tantas veces como vengan dado por el número de iteraciones actual, en el rango definido por `iteraciones_ini` e `iteraciones_fin`.

  Ejemplo: Si se define `tiempo=1`, `iteraciones_ini=5`, `iteraciones_fin=40`, el comportamiento será medir un benchmark tantas iteraciones como sea necesario hasta llegar a que el tiempo de ejecución total de las iteraciones sea mayor o igual a 1 segundo, y repetir esto 5 veces. Más tarde se repite con 10, 20, y finalmente 40.

- `tiempo_de_espera`: Número entero o flotante que indica durante cuántos segundos se espera tras una ejecución para realizar la siguiente.

- `subir_a_drive`: Booleano que indica si se sube o no a Google Drive el archivo, según la configuración hecha en el dispositivo de experimentación.

Las mediciones generan unos archivos de nombre `consumption`, seguido del nombre del binario medido, el nombre del dispositivo de medición (nombre de usuario dado), y o bien `-iters-` seguido del número de iteraciones si el valor

de `tiempo` es menor o igual que 0, o el valor de la variable tiempo, seguido de `s-total-iter-`, el total de ejecuciones actuales a medir, y por qué ejecución de esas totales va. En cada uno de estos archivos se encuentran las mediciones de consumo hechas, ordenada por filas, donde cada fila representa a una iteración.

A continuación, se muestran un par de ejemplos:

- `consumption-a2time01-MediPi-12-3600s-total-iter-15-iter-num-1.csv`: Un archivo de consumo donde se ha medido el benchmark `a2time01` en un dispositivo llamado "MediPi-12". Este benchmark se pretende ejecutar 15 veces, y cada una de estas medir el consumo a lo largo de una hora. El archivo representa que va por la segunda de esas 15, por el número 1 (ya que se empieza a contar en 0).
- `consumption-bezier-MediPi-03-iters-20`: Un archivo de consumo donde se midió el benchmark `bezier` en un dispositivo de nombre "MediPi-03" durante 20 iteraciones.

Además, se encuentran otros archivos de nombre `mean-std-and-time-seconds-`, seguido de la misma estructura de nombre, los cuales almacenan información relevante de cada una de las iteraciones, como bien es, de forma ordenada:

- Media de las medidas obtenidas, en unidades de miliamperios.
- Desviación típica de las medidas obtenidas.
- Tiempo de ejecución de la iteración, en segundos.
- Suma de todas las mediciones medidas, en unidad de mA.
- Número de medidas tomadas.
- Energía consumida en esa iteración, en unidad de mWs.

En caso de que se subiese a Google Drive, la carpeta generada tendría el nombre del año, mes y fecha del día actual. Dentro de esta se encontraría los archivos ya mencionados.

7.1.4.3 Ejecución de algoritmo genético para buscar la ejecución óptima de las suites

Se lanza un algoritmo genético que calcula cuánto es el número de iteraciones de cada benchmark, y construye un programa que implementa esta ejecución calculada. Para ello, se intenta maximizar o minimizar el consumo, en función de lo que se desee. La ejecución de las suites debe de aproximarse a un tiempo que se especifica.

Para ello, se ejecuta la función `lanzar_genetico`. Los parámetros de esta función son:

- `tupla`: Objeto de clase `Measure_tuple`, donde se pasan los parámetros definidos al inicio del script `pc_launch.py`. En el script ya se define un objeto de esta clase, de modo que se define tupla = tupla.

- `tiempo_de_espera`: Número entero o flotante que indica durante cuántos segundos se espera tras una ejecución para realizar la siguiente.

- `subir_a_drive`: Booleano que indica si se sube o no a Google Drive el archivo, según la configuración hecha en el dispositivo de experimentación.

En el script `genetico_ejecucion_suite_v1_3.py` dentro del directorio `scriptsMediPi` se encuentran una serie de parámetros adicionales relacionados con el algoritmo genético. Si se tiene conocimientos básicos del funcionamiento de este, se recomienda encarecidamente cambiar los valores de las variables. Estas son:

- `population_size`: Número entero que indica el tamaño de la población.
- `prob_mutation`: Número con coma decimal, entre 0 y 1, que indica la probabilidad de la mutación.
- `prob_cross`: Número con coma decimal, entre 0 y 1, que indica la probabilidad de cruce.
- `n_iters`: Número de iteraciones que se realiza el algoritmo genético.
- `bool_minimize`: Booleano que indica si se desea que se minimice o maximice el consumo.
- `iters_to_measure_consumption`: Número entero que indica el número de ejecuciones de cada benchmark para calcular cuál es su consumo y tiempo de ejecución. A mayor valor, mayor precisión pero más tardará el algoritmo genético.
- `seconds_time_objective`: Número, entero o con coma decimal, que indica el tiempo que tiene como referencia el algoritmo genético. El objetivo del algoritmo genético es que la ejecución de todas las suites se aproxime a este tiempo, mientras que maximiza o minimiza el consumo.
- `flags`: Lista de cadena de caracteres que indica las transformaciones aplicadas a la hora de compilar el programa que ejecuta todas las suites.

Es importante destacar que, para el **correcto funcionamiento del algoritmo genético, es imprescindible que no haya funciones o variables globales definidas con el mismo nombre en varios benchmark**. Esto provoca un error llamado redefinición durante el proceso de compilación. Para ello, **se han incluído dos versiones de las suites de benchmarks**. En una

de ellas, los elementos que tenían nombres repetidos han sido renombrados, cuyo nombre del directorio es `EEMBC_Benchmark_suites_redefinidas`. En la otra versión, este renombrado no se ha producido, de manera que el código fuente puede ser más claro. Se recomienda, pues, usar la primera.

De igual forma, se dispone de una operación llamada `save_and_fix_multiple_definitions`, cuyo parámetro `route_bench` es una cadena de caracteres que indica la ruta al directorio de la suites de benchmarks, cuya finalidad es la de arreglar estas redefiniciones en las suites de la ruta especificada. Esta operación puede ser utilizada desde `pc_launch.py`.

Los archivos generados por esta operación son múltiples de consumo y de datos, `consumption` y `mean-std-and-time-seconds-` debido a que se realizan múltiples mediciones de consumo. Además, se genera otro archivo, de nombre `population-ga.csv`, que representa la población del algoritmo genético a lo largo de las iteraciones calculadas, así como la evaluación o *fitness* de cada una de las poblaciones.

En caso de que se subiese a Google Drive, la carpeta generada tendría el nombre "*genetic*", seguido del año, mes y fecha del día actual. Dentro de esta se encontraría los archivos ya mencionados, además del archivo `all.c` de código fuente, `all.bc` de código intermedio y `all`, el binario de la solución encontrada por el algoritmo.

7.1.5. Nomenclatura, estructurado de directorios: adición de nuevos benchmarks o suites

Para el correcto uso de este software, se ha de tener clara la nomenclatura usada con respecto de los directorios y nombres de archivos.

Para comenzar, dentro del comprimido que se entrega, se encuentran dos directorios de nombre `EEMBC_Benchmark_suites_redefinidas` y `EEMBC_Benchmark_suites`. Dentro de cada uno de estos, se encuentran a su vez otros directorios que contienen *suites de benchmark*, o colecciones de *benchmarks*. Los nombres de los directorios de las suites no pueden contener ni "-" (guiones) ni "." (puntos), y es recomendable que sus nombres sean `benchmarks_` seguido del nombre de la suite. Asimismo, los nombres de las suites deben de ser únicos.

Del mismo modo, dentro de cada directorio de suite se encuentran los directorios de benchmarks. Cada uno de estos contiene todos los archivos necesarios para ejecutar el benchmark un benchmark en sí. De nuevo, como con los directorios de suites, no pueden contener ni "-" (guiones) ni "." (puntos), aunque esta vez su nombre es de libre elección, aunque manteniéndose que deben de ser únicos. Estos deben ser del lenguaje de programación `C`, y su función `main` de los benchmarks debe de estar incluida en un fichero `.c` cuyo nombre puede ser el mismo que el del *benchmark*, o bien `bmark_lite`.

De igual modo, a través del *script* `pc_launch.py`, se puede lanzar la operación `check_correctness_files_and_folders`, cuyo parámetro `str_directory` es una cadena de caracteres que indica la ruta al directorio de la suites de *benchmarks*, que comprueba sustituye las apariciones de "-" (guiones), "." (puntos) y las posibles repeticiones de nombres entre *benchmarks*, pero no de suites.

El motivo por el que se encuentran dos directorios con dos *suites de benchmarks* puede ser consultado en 7.1.4.3.

De este modo, si se desease crear un directorio de suites, únicamente debería de crearse una carpeta vacía con cualquier nombre. Su contenido únicamente debería de ser los directorios de la suites que se deseen tener, y los *scripts* que aparecen en `EEMBC_Benchmark_suites_redefinidas` o `EEMBC_Benchmark_suites`.

Si, más adelante, se quisiese añadir una nueva suite, se debería de crear un directorio vacío cualquier nombre, aunque es recomendable que este empiece por `benchmarks_`. Esta suites debe de procurar tener, de nuevo, un nombre no repetido con respecto del resto de suites, si las hubiese, y no contener punto ni guión.

Si dentro de una suite se quisiese añadir un *benchmark*, se crearía un nuevo directorio dentro de la suite, procurando que el nombre no se repitiese. Como se especificó antes, se debe de localizar la función `main` en un fichero con el nombre del directorio o en `bmark_lite.c`.

7.1.6. Usos avanzados del software

Además de todas las funcionalidades descritas, el software implementado puede ser utilizado para añadir las funcionalidades que se crean necesarias. Para ello, existe un *script* llamado `Measurer.py` dentro del directorio `scriptsMediPi` que contiene una clase `Measurer`, la cual, mediante la herencia y sobre-escritura de los tres métodos que necesita, puede realizar las funcionalidades que se desee. Estos métodos son los más troncales de la ejecución del *script* como tal, encargándose de ejecutar *benchmarks* en el dispositivo de experimentación y compilar. Varios ejemplos de clases que heredan de `Measurer` pueden ser encontrados en `benchmark_measurer.py`, donde se define la clase que usamos para medir las *suites de benchmarks*, y `measure_basal`, donde se define la clase usada para medir el consumo basal del dispositivo.

En una última instancia, si se desease desear la clase `Measurer.py` o cualquier otra, existe documentación interna explicando la funcionalidad de cada método, sus parámetros de entrada, los pasos que realizan estos métodos, la funcionalidad de los atributos de las clases, así como variables globales.

Asimismo, la funcionalidad de la ExperiPi puede ser modificada, cambiando el *script* `main_experipi.py` de la carpeta `scriptsExperiPi`.

Si se desea medir el consumo de los *benchmarks* utilizando la opción `lanzar_medicion`, pero compilando los *benchmarks* con unos *flags* de compilación LLVM concretos, también conocidos como *passes*, se puede realizar. Para ello, hay que configurar el valor de la variable `flags` dentro de `benchmark_measurer`. El valor debe de ser una lista de cadena de caracteres, donde cada cadena equivalga a un *flag*. Al igual que este parámetro, existen otros que puede ser de interés cambiar su valor, como bien puede ser el tamaño del *buffer*, dirección I2c del sensor *INA* a usar, entre otros.

Finalmente, es importante señalar que se puede prescindir de `pc_launch.py` como *script* que lanza las operaciones a realizar, y trabajar directamente desde MediPi. Importando las clases necesarias, utilizando herencia, y modificando o creando archivos, se pueden realizar los comportamientos que se desee de igual modo.

Para más detalle de esta información comentada, se recomienda encarecidamente leer la documentación interna de los *scripts*.

8. Bibliografía

[1] Allinea, Concertim, Embecosm, and STFC Daresbury Hartree Centre. Tsero webpage. http://tsero.org/. Disponible en http://tsero.org/, última vez accedido 4 de marzo de 2022.

[2] Megan Bray. Review of computer energy consumption and potential savings. *Dragon Systems Software Limited (DssW)*, 12 2006.

[3] EEMBC. Página de productos eembc. https://https://www.eembc.org/products/. Disponible en https://www.eembc.org/products/, última vez accedido 4 de marzo de 2022.

[4] EEMBC. Página principal de eembc. https://https://www.eembc.org/. Disponible en https://www.eembc.org/, última vez accedido 4 de marzo de 2022.

[5] Embecosm. Embecosm webpage. https://www.embecosm.com. Disponible en https://www.embecosm.com, última vez accedido 4 de marzo de 2022.

[6] Eva García-Martín, Crefeda Faviola Rodrigues, Graham Riley, and Håkan Grahn. Estimation of energy consumption in machine learning. *Journal of Parallel and Distributed Computing*, 134:75–88, 2019.

[7] Embench group. Página principal de embench. https://www.embench.org. Disponible en https://www.embench.org/, última vez accedido 4 de marzo de 2022.

[8] Embench group. Página web de embench, novedades. https://www.embench.org, 1 2021. Disponible en https://www.embench.org/news.html, última vez accedido 4 de marzo de 2022.

[9] Departamento de computación de la universidad de Bristol Innovate UK, Embecosm. Mageec webpage. http://mageec.org. Disponible en http://mageec.org, última vez accedido 4 de marzo de 2022.

[10] IRNAS. Api de nordic semiconductor power profiler i. https://github.com/IRNAS/ppk2apipython. Disponible en https://github.com/IRNAS/ppk2apipython, última vez accedido 4 de marzo de 2022.

[11] Daniel Molka, Daniel Hackenberg, Robert Schöne, and Matthias S Müller. Characterizing the energy consumption of data transfers and arithmetic operations on x86- 64 processors. In *International conference on green computing*, pages 123–133. IEEE, 2010.

[12] James Pallister, Simon Hollis, and Jeremy Bennett. Beebs: Open benchmarks for energy measurements on embedded platforms. *arXiv preprint arXiv:1308.5174*, 2013.

[13] Jason A Poovey, Thomas M Conte, Markus Levy, and Shay Gal-On. A benchmark characterization of the eembc benchmark suite. *IEEE micro*, 29(5):18–29, 2009.

[14] s matyukevich. Learning operating system development using linux kernel and raspberry pi. github.com, 4 2021. Disponible en https://github.com/s-matyukevich/raspberry-pi-os, última vez accedido 4 de marzo de 2022.

[15] Nordic Semiconductor. Nordic semiconductor power profiler i webpage. https://www.nordicsemi.com/Products/Development-hardware/Power-Profiler-Kit. Disponible en https://www.nordicsemi.com/Products/Development-hardware/Power-Profiler-Kit, última vez accedido 4 de marzo de 2022.

[16] Nordic Semiconductor. Nordic semiconductor power profiler ii webpage. https://www.nordicsemi.com/Products/Development-hardware/Power-Profiler-Kit-2. Disponible en https://www.nordicsemi.com/Products/Development-hardware/Power-Profiler-Kit-2, última vez accedido 4 de marzo de 2022.

[17] RF Szydlowski and WD Chvala Jr. Energy consumption of personal computer workstations. Technical report, Pacific Northwest Lab., Richland, WA (United States), 1994.

[18] wlgrd. Api de nordic semiconductor power profiler ii. https://github.com/wlgrd/ppk_api. Disponible en https://github.com/wlgrd/ppk_api, última vez accedido 4 de marzo de 2022.