UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Universitat Politècnica de València

Reducción del Consumo Energético en Sistemas Heterogéneos Mediante Balanceo de Carga

TRABAJO FIN DE MÁSTER

Máster Universitario en Computación en la Nube y de Altas Prestaciones

Autor: Juan González Caminero

Tutor: Arturo González Escribano
Pedro Alonso Jordá

Curso 2020-2021

Resum

El problema de balancejar la càrrega computacional d'una aplicació paral·lela que s'executa sobre un conjunt de màquines és interessant, i existeixen una gran quantitat d'aproximacions a aquest. Tradicionalment l'objectiu principal de les solucions de balanceig de càrrega ha sigut el d'optimitzar les execucions reduint el temps emprat. Donat l'actual interés per l'eficiència energètica, en aquest projecte es parteix de la pregunta de si és possible elaborar una solució de balanceig de càrrega orientada a reduir el consum energètic d'una aplicació.

Per a contestar a aquesta pregunta, s'empra la llibreria HitMap desenvolupada pel grup Trasgo, que inclou entre la seua funcionalitat un sistema de balanceig de càrrega. Així mateix, s'investiga sobre les eines disponibles per a mesurar el consum energètic d'una aplicació i se seleccionen les interfícies *RAPL disponibles en processadors Intel.

Es duu a terme una fase d'experimentació prèvia on es documenta el comportament quant a consum energètic d'una aplicació usada sovint pel grup d'investigació en la qual és interessant l'ús de balanceig de càrrega. També es documenten els diferents perfils de consum en les màquines utilitzades.

Els resultats d'aquesta experimentació prèvia indiquen que una solució de balanceig de càrrega tradicional pot reduir el consum energètic, però el comportament dels propis processadors, així com d'eines utilitzades àmpliament com els sistemes Linux o la llibreria MPI introdueixen ineficiències quant a consum.

En vista dels resultats observats es conclou que la idea inicial de basar el balanceig de càrrega en el consum energètic de l'aplicació en lloc d'en el seu temps d'execució probablement no pot dur-se a terme. No obstant això, a partir d'aquesta experimentació prèvia es presenta com a prova de concepte un mètode per a reduir el consum energètic modificant les freqüències dels processadors utilitzats.

Paraules clau: Balanceig de Càrrega, Sistemes Heterogenis, Eficiència Energètica

---

Resumen

El problema de balancear la carga computacional de una aplicación paralela que se ejecuta sobre un conjunto de máquinas es interesante, y existen una gran cantidad de aproximaciones al mismo. Tradicionalmente el objetivo principal de las soluciones de balanceo de carga ha sido el de optimizar las ejecuciones reduciendo el tiempo empleado. Dado el actual interés por la eficiencia energética, en este proyecto se parte de la pregunta de si es posible elaborar una solución de balanceo de carga orientada a reducir el consumo energético de una aplicación.

Para contestar a esta pregunta, se emplea la librería HitMap desarrollada por el grupo Trasgo, que incluye entre su funcionalidad un sistema de balanceo de carga. Asimismo, se investiga acerca de las herramientas disponibles para medir el consumo energético de una aplicación y se seleccionan las interfaces RAPL disponibles en procesadores Intel.

Se lleva a cabo una fase de experimentación previa donde se documenta el comportamiento en cuanto a consumo energético de una aplicación usada frecuentemente por el grupo de investigación en la cual es interesante el uso de balanceo de carga. También se documentan los distintos perfiles de consumo en las máquinas utilizadas.

Los resultados de esta experimentación previa indican que una solución de balanceo de carga tradicional puede reducir el consumo energético, pero el comportamiento de los propios procesadores, así como de herramientas utilizadas ampliamente como los sistemas Linux o la librería MPI introducen ineficiencias en cuanto a consumo.

En vista de los resultados observados se concluye que la idea inicial de basar el balanceo de carga en el consumo energético de la aplicación en lugar de en su tiempo de ejecución probablemente no pueda llevarse a cabo. Sin embargo, a partir de esta experimentación previa se presenta como prueba de concepto un método para reducir el consumo energético modificando las frecuencias de los procesadores utilizados.

Palabras clave: Balanceo de Carga, Sistemas Heterogéneos, Eficiencia Energética

Abstract

The problem of balancing the computational load of a parallel application running on a set of computers is interesting, and there are a large number of approaches to it. Traditionally, the main objective of load balancing solutions has been to optimize executions by reducing the execution time. Given the current interest in energy efficiency, this project starts from the question of whether it is possible to develop a load balancing solution aimed at reducing the energy consumption of an application.

To answer this question, we use the HitMap library developed by the Trasgo group, which includes a load balancing system among its functionality. Likewise, the tools available to measure the energy consumption of an application are investigated and the RAPL interfaces available in Intel processors are selected.

A preliminary experimentation phase is carried out where we document the behavior in terms of energy consumption of an application frequently used by the research group in which the use of load balancing is interesting. The different consumption profiles of the machines used are also reported.

The results of this preliminary experimentation indicate that a traditional load balancing solution can reduce energy consumption, but the behavior of the processors themselves, as well as widely used tools such as Linux systems or the MPI library, introduce inefficiencies in terms of consumption.

In view of the observed results, it is concluded that the initial idea of basing load balancing on the energy consumption of the application instead of its execution time probably cannot be carried out. However, based on this preliminary experimentation, a method to reduce energy consumption by modifying the frequencies of the processors used is presented as a proof of concept.

Key words: Load Balancing, Heterogeneous Systems, Energy Efficiency

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

1.1 Contexto

El problema de repartir la carga computacional de un programa paralelo sobre un conjunto de máquinas o procesadores es complejo. Los desequilibrios que pueden aparecer entre los distintos elementos de cómputo llevan a un aumento en el tiempo de ejecución del programa, ya que por lo general se tendrá que esperar a que finalice el más lento.

Si el sistema es homogéneo, es decir, todas las máquinas utilizadas son iguales, y dividimos la carga a partes iguales entre las máquinas o procesadores, nos encontramos con que pueden aparecer desequilibrios a lo largo de la ejecución debidos a múltiples factores, como diferencias en la latencia de acceso a datos, interferencias por parte de otros programas, diferencias de temperatura, etc. En sistemas heterogéneos, donde las máquinas no tienen el mismo hardware, además de estos problemas nos encontramos con diferencias de base entre los componentes.

Se ha propuesto una gran cantidad de aproximaciones a estos problemas, centradas en distintos tipos de algoritmos paralelos. Algunas pretenden repartir iteraciones de bucles paralelizados [2, 3, 4], otras dividen el algoritmo en tareas que pueden ser ejecutadas por cualquiera de las máquinas [6, 7, 8], existen algoritmos orientados a problemas concretos [9, 10]...

1.2 Motivación

En la actualidad hay un gran interés en la eficiencia energética en todos los ámbitos, en el caso de la computación de altas prestaciones tenemos, por ejemplo, la lista Green500 [16] que ordena los supercomputadores de la lista Top500 [17] en base a su eficiencia energética.

El balanceo de carga tradicional se centra en minimizar el tiempo de ejecución de los programas. Con este proyecto, se plantea la siguiente pregunta:

> **¿Es posible elaborar una solución de balanceo de carga orientada a reducir el consumo energético de un programa?**

Además de una disminución en el consumo energético, una solución de este tipo podría proporcionar efectos beneficiosos para el tiempo de ejecución de los programas. Es posible que exista una correlación entre el consumo energético de una máquina y

ralentizaciones en la ejecución debidas a factores como el aumento de temperatura, con la consiguiente reducción de frecuencias en la mayoría de procesadores modernos.

1.3 Trabajo relacionado

En el balanceo de carga tradicional el objetivo es la reducción del tiempo de ejecución de los programas, aquí se presentan algunas de las aproximaciones que se han tomado en este ámbito.

De manera general, los métodos de balanceo de carga se pueden dividir en métodos estáticos y dinámicos, donde los estáticos parten de información a priori de las máquinas sobre las que se ejecutan, mientras que los dinámicos son capaces de adaptarse a los cambios que se produzcan durante la ejecución del programa [1].

Una de las formas tradicionales de realizar el reparto de carga de manera estática es a través de modelos del rendimiento de los procesadores que se van a utilizar, obtenidos mediante pruebas previas a la ejecución del programa. Estos modelos asumen que la velocidad de los procesadores es constante independientemente de la carga que se les asigne, lo cual no se cumple en muchos casos de uso reales [1].

Debido a lo relativamente simple de los métodos estáticos, la mayor parte de la bibliografía consultada busca definir métodos dinámicos, que puedan adaptarse a los desequilibrios que aparecen durante la ejecución de los programas.

Existen varias técnicas de reparto de carga orientadas a repartir iteraciones de un bucle paralelizado entre las distintas máquinas:

- Static Chunking (SC) [2]: El método más sencillo, consiste en repartir las iteraciones a partes iguales entre las máquinas.
- Self-Scheduling (SS) [2, 3]: Se reparten las iteraciones de una en una, cuando una máquina termina de ejecutar una iteración se le asigna otra. Consigue que todos los procesadores terminen casi al mismo tiempo, pero implica un sobrecoste muy grande de planificación.
- Guided Self-Scheduling (GSS) [2, 3, 11]: De manera similar a Self-Scheduling, los procesadores van recibiendo trabajo a medida que terminan con lo que se les había asignado. Sin embargo en este caso, en lugar de asignarse iteración a iteración, se asignan bloques de iteraciones de tamaño proporcional al número de iteraciones restante, lo cual resulta en un sobrecoste menor.
- Factoring [2, 3]: Este método también asigna bloques de iteraciones a los procesadores que terminan su trabajo, pero en este caso el tamaño de bloque no varía con cada bloque asignado. Se planifican "lotes" de bloques de iteraciones, donde todos los bloques de cada lote son del mismo tamaño, una vez se terminan los bloques del lote, se elige un nuevo tamaño de bloque proporcional al número de iteraciones restante, y se crea un nuevo lote. Esto evita situaciones que se pueden producir con GSS, donde se asigna demasiado trabajo inicialmente, y no se reserva suficiente trabajo para repartir hacia el final de la ejecución y disminuir las diferencias en el tiempo de ejecución de cada procesador.
- Weighted Factoring [5]: Una ampliación del método Factoring para adaptarlo a la ejecución en sistemas heterogéneos. Parte de una serie de pesos asignados a cada una de las máquinas que intervienen en la ejecución del algoritmo, que utiliza para determinar el número de iteraciones que recibe cada procesador en función del

tamaño de los bloques que se estén repartiendo en cada momento. Así, si, por ejemplo, el tamaño de bloque en un lote es de 64 iteraciones, un procesador con peso 0.5 recibiría 32.

- Adaptative Factoring [3]: Este método pretende adaptar Factoring al hecho de que la duración de las iteraciones puede variar a lo largo de la ejecución. Calcula dinámicamente la media y la varianza de los tiempos de ejecución de las iteraciones para tomar mejores decisiones en cuanto a los tamaños de chunk.

Otra aproximación utilizada por muchas publicaciones es la de dividir los algoritmos en tareas, a ejecutar de manera independiente por las máquinas implicadas:

- En [6] se propone un runtime que sea capaz de soportar la migración tanto de tareas como de datos e hilos entre distintas máquinas, proporcionado distintos grados de ajuste en función de la magnitud del desequilibrio. Estas migraciones se producirían mediante *work-stealing*, donde las máquinas que terminen su trabajo cogen parte del de otros nodos.

- En [12] se describe un método en el que se miden una serie de parámetros de las máquinas (Utilización de la CPU, Estado de la memoria, etc.), y se utilizan para definir una serie de reglas basadas en umbrales, que determinarán cuándo se considera que una máquina está sobrecargada. Cuando ocurre esto, parte de las tareas de esa máquina migran a otras con menor carga.

- En [13] se miden cuatro parámetros: tiempo de cómputo, coste de las comunicaciones, coste del uso de recursos, y coste del tiempo de espera de los procesadores sin trabajo. A partir de estos valores se define una función objetivo, la distribución óptima de acuerdo a esta función se obtiene hallando el mínimo.

- En [14] se utilizan reglas basadas en umbrales sobre el tiempo de ejecución de las tareas en cada máquina. Cuando una máquina supera el umbral para el tiempo de ejecución de una sección de código definida por el programador, parte de las tareas asignadas a la misma se reparten entre las demás.

1.4 Objetivos

Se plantean los siguientes objetivos para responder a la pregunta principal del proyecto:

- Determinar la mejor forma de obtener medidas sobre el consumo energético de una aplicación vía software.

- Documentar cómo se comportan algunas aplicaciones reales en cuanto a consumo energético.

- Documentar el efecto del balanceo de carga tradicional, orientado a reducir el tiempo de ejecución, en el consumo energético.

- Determinar si es factible balancear la carga en base a marcadores de consumo energético.

1.5 Estructura de la memoria

La estructura de la memoria es la siguiente: En la Sección 2 se habla de las herramientas empleadas en el proyecto. En la Sección 3 se describe el prototipado realizado con algunas de esas herramientas, y en la Sección 4 se presenta una fase de experimentación previa. La Sección 5 presenta una propuesta de reducción del consumo energético en base a los resultados obtenidos previamente, y por último la Sección 6 expone las conclusiones del proyecto y da una propuesta de trabajo futuro.

CAPÍTULO 2
Entorno de Trabajo

2.1 Herramientas para medir el consumo energético

A continuación se detallan las opciones consideradas para obtener medidas del consumo energético de una máquina, con las ventajas y desventajas identificadas. Se han analizado opciones genéricas que puedan ser usadas en máquinas que no dispongan de un hardware especial de monitorización, así como las herramientas específicas de la Nvidia Jetson AGX Xavier.

2.1.1. PowerTop

PowerTop [18] es una herramienta desarrollada por Intel, cuyo principal objetivo es ayudar a diagnosticar y resolver problemas en el consumo energético de una máquina. Powertop también proporciona mediciones del consumo energético de los programas en ejecución en la máquina.

Los informes generados por PowerTop incluyen, para el periodo de tiempo durante el que se ha realizado la medición:

- El porcentaje de tiempo que cada Core pasa en cada uno de sus C-States [19] (Modos de ahorro de energía para los períodos en los que el procesador no tiene instrucciones que procesar).
- Frecuencia media de cada Core.
- Listado de los procesos en ejecución con su tiempo de uso de CPU y consumo energético estimado entre otros datos.
- Estimación de la energía consumida por la CPU y otros dispositivos, si es posible obtener la estimación.
- Ajustes software que se pueden modificar para mejorar la eficiencia energética.
- Ajustes óptimos.

Los principales puntos positivos de esta herramienta para este proyecto son:

- Facilidad para medir el consumo energético de cada proceso en un programa MPI, ya que MPI proporciona una llamada para obtener el PID de cada uno de los procesos lanzados.

- Las mediciones sobre el consumo energético son independientes de la CPU concreta en la que se ejecute el proceso, con lo que el programa no tiene que tener en cuenta dónde se está ejecutando.

Por otro lado, vemos varios inconvenientes:

- PowerTop necesita recoger datos durante un período de tiempo antes de calcular la estimación del consumo energético. Tomar mediciones durante periodos largos ralentizará la ejecución de los programas, mientras que usar periodos más cortos podría afectar a la precisión de las estimaciones.
- El programa no proporciona una API, es necesario ejecutarlo y buscar manualmente en el informe generado los resultados que nos interesan.
- Los informes que se generan son muy extensos e incluyen una gran cantidad de datos que no nos interesan para este caso de uso.
- Es necesario ejecutar el programa como root.

2.1.2. RAPL

RAPL [20, 21] es una herramienta constituida por una serie de interfaces para procesadores Intel que permiten establecer límites en el consumo energético del procesador. Estas interfaces son accesibles a través de registros MSR [24] del procesador (Accesibles en Linux a través de `/dev/cpu/[número de cpu]/msr`), o a través de los archivos en `/sys/class/powercap/intel-rapl/intel-rapl:0`.

De acuerdo con [21], RAPL nos proporciona interfaces para limitar el consumo de energía, y obtener estimaciones de consumo en cuatro dominios distintos:

- El paquete completo (La energía consumida por todos los componentes del procesador).
- Los cores del procesador (El total de energía consumida por el conjunto de cores del procesador).
- Un dispositivo uncore no especificado, sólo en CPUs para consumidores. En las CPUs en las que se ha probado este dispositivo es la GPU integrada del procesador.
- La DRAM del procesador, sólo en CPUs para servidores.

Según [20] estas medidas no se obtienen a través de un dispositivo de medición físico, sino mediante un modelo software del consumo de energía.

Las principales ventajas de usar RAPL en el proyecto son las siguientes:

- El acceso a los datos es rápido y sencillo.
- No es necesario un periodo de análisis para obtener las mediciones, según [21], el registro MSR que nos proporciona el consumo de energía estimado, en Julios, desde la última vez que se inicializó el registro. Se actualiza cada milisegundo.

Y las mayores desventajas:

- No se proporciona información sobre el consumo de energía de cada core por separado. Lo ideal sería tener acceso a estos datos para el balanceo de carga dentro de una sola máquina.

- Es necesario tener acceso root o bien modificar los permisos de acceso de los ficheros que nos dan acceso a los registros MSR o de los ficheros en `/sys/class/powercap/intel-rapl`.

2.1.3. Turbostat

Turbostat [25] es una herramienta desarrollada por Intel que genera un informe en base a los valores proporcionados por RAPL. La principal ventaja de este programa es que ofrece una forma sencilla de acceder a la vez a toda la información que ofrece RAPL, y controlar el formato de salida, por ejemplo, cambiando las unidades en las que se proporcionan los datos, o seleccionando los datos que queremos. También ofrece la posibilidad de actualizar los resultados regularmente con el período que indique el usuario.

2.1.4. NVIDIA Management Library (NVML)

NVML [28] es una API de Nvidia que permite acceder desde un programa en C a una serie de parámetros relacionados con las GPUs instaladas en el sistema (Nombre del dispositivo, Número de serie, Temperatura, Frecuencia, Procesos en ejecución, Uso de la memoria... La lista es muy extensa).

Entre estos parámetros podemos encontrar la última lectura de potencia para la GPU. Según la documentación, estas lecturas se realizan mediante sensores INA, y se refieren a la potencia consumida por el dispositivo entero excepto en dispositivos que no dispongan de estos sensores, en cuyo caso el valor que obtenemos es la potencia consumida por la GPU (En este caso no se incluiría la memoria y otros componentes).

Una forma altermativa de acceder a estos valores es a través del comando **nvidia-smi** [29], que nos ofrece una forma más sencilla de usar NVML.

En este caso, tenemos dos puntos positivos claros, ya que el acceso se puede realizar a través de una API, y las lecturas proporcionadas se obtienen mediante sensores hardware. No se han visto factores que puedan ser negativos para el proyecto.

2.1.5. Monitores Hardware de la Nvidia Jetson AGX Xavier

La Nvidia Jetson AGX Xavier incluye dos circuitos de monitorización INA3221 [26] cuya información puede ser extraída a través de una serie de nodos de sysfs [27].

En concreto, se proporciona la energia en tres "Railes" distintos por cada sensor INA3221, en el primero:

- Energía consumida por la CPU.
- Energía consumida por la GPU.
- Energía consumida por el sistema completo.

Las mediciones que nos proporciona el segundo sensor no son interesantes para el uso que se plantea en este proyecto.

Con estos datos tenemos el mismo inconveniente que con RAPL, al no tener información específica de cada core por separado. Por otro lado, dos grandes ventajas de este sistema son que obtenemos la medida a través de un contador analógico y no con una estimación software, y que podemos medir de forma directa el consumo de la GPU de la tarjeta.

2.2 Herramientas de paralelización

Se emplean principalmente dos herramientas a la hora de paralelizar las aplicaciones, OpenMP y MPI.

2.2.1. OpenMP

OpenMP [35] es un modelo de programación que nos permite paralelizar una aplicación en base a directivas de compilación. Podemos obtener una aplicación paralela a partir de una versión secuencial añadiendo anotaciones al código, las cuales indican cómo se compartirá el trabajo entre hilos, y permiten al compilador producir una versión paralela de la aplicación.

OpenMP emplea un modelo de ejecución Fork-Join. Disponemos de una serie de directivas para crear hilos y dividir el trabajo. Inicialmente un único hilo "maestro" ejecuta el programa, cuando la ejecución llega a una de estas regiones paralelas cada hilo toma una parte del trabajo, y cuando todos terminan el hilo maestro continúa la ejecución. Los hilos se crean con la primera región paralela que aparece, y se reutilizan en las siguientes.

En cuanto al modelo de memoria, se emplea una modelo de memoria compartida. Cada hilo dispone de su propio contexto de ejecución, incluyendo la pila, y además se tiene acceso a una zona de memoria común, la cual se puede emplear en comunicaciones.

2.2.2. MPI

MPI [36] es la especificación de una librería de paso de mensajes, centrada en el uso en arquitecturas paralelas. La programación con MPI se realiza mediante llamadas a funciones, que implementan las diversas operaciones definidas en el estándar.

Se dispone de distintos tipos de operaciones, como las de comunicación punto a punto, donde se intercambian datos entre dos procesos, comunicaciones colectivas, entre grupos de procesos, operaciones de gestión de datos, que permiten definir tipos de datos especiales para las comunicaciones, operaciones que nos permiten gestionar grupos de procesos, topología de los mismos, etc., entre otras.

En cuanto al modelo de ejecución, al lanzar una aplicación MPI se crean simultáneamente los procesos necesarios, que pueden estar en la misma o en distintas máquinas. A lo largo de la ejecución, estos procesos, agrupados en "Comunicadores", ejecutan la aplicación intercambiando datos mediante llamadas a las funciones MPI.

2.3 La librería HitMap

HitMap [30] es una librería desarrollada por el grupo de investigación Trasgo [31] sobre MPI [36] para facilitar la redistribución de datos entre distintos procesos. HitMap se basa en la división de los datos del programa en "Tiles", permitiendo la creación de una jerarquía de Tiles para permitir un control de grano más fino o más grueso en función de la situación. Por ejemplo, por lo general querremos distribuir Tiles más grandes entre distintas máquinas, y subtiles entre los cores de un procesador.

Tenemos tres tipos de funciones:

- Funciones de Tiling: Usadas para la creación de Tiles.

- Funciones de Mapping: Calculan, en función de una topología virtual seleccionada por el programador y los datos a distribuir, las Tiles que es necesario crear y qué Tiles se asignan a cada procesador.
- Funciones de Comunicación: Nos permiten mover Tiles entre procesadores, ocultando al programador la estructura interna de las Tiles que se están moviendo. Se proporcionan abstracciones para distintos patrones de comunicación, como punto a punto, comunicaciones colectivas, etc.

Este proyecto se implementa sobre HitMap, por una parte, por el balanceo de carga ya implementado en la librería, por otro lado, porque utilizar esta librería nos facilita el organizar los datos en estructuras de distinto tamaño, y nos proporciona abstracciones para moverlos entre procesadores de forma sencilla, lo que nos permite centrarnos en el método que queremos implementar y no en cómo realizar las comunicaciones.

En [15] se describe la implementación de un sistema de balanceo de carga sobre la librería HitMap que aprovecha sus funcionalidades. Este sistema es interesante para el proyecto por dos motivos, uno es que nos permite estudiar cómo se comporta el consumo energético cuando usamos una solución de este tipo, además, puede servir como base para una implementación de un sistema que priorice reducir el consumo.

En este sistema de balanceo, la carga se reparte redistribuyendo los datos a procesar entre las máquinas implicadas en la ejecución. Para ello, entendiendo una aplicación que organiza su ejecución en una serie de iteraciones, donde cada iteración hace alguna operación con los datos, el proceso es el siguiente:

1. El programador mide el tiempo de ejecución de la sección de código donde se procesan estos datos, y le da esa información a la librería.
2. La librería almacena estos tiempos de ejecución para las máquinas implicadas durante un número de iteraciones determinado.
3. Una vez se dispone de los datos suficientes, se calcula una métrica para decidir cómo hacer la redistribución. La librería implementa distintas métricas por defecto:
   - Media Móvil Simple
   - Media Móvil Ponderada
   - Media Móvil Exponencial
4. Finalmente, se mueven los datos entre máquinas en función de la decisión tomada a partir de las métricas computadas.

La librería ofrece distintas opciones para hacer las redistribuciones, ya que en función del sistema y aplicación con que se utilicen tendrán un rendimiento distinto. Las principales son redistribuciones en intervalos fijos, y redistribuciones en intervalos progresivos.

Los intervalos de redistribución fijos son interesantes en sistemas en los que continuamente pueden aparecer nuevos desequilibrios, con lo que conviene controlar periódicamente que los tiempos de ejecución en cada máquina no difieran mucho.

Los intervalos progresivos son interesantes cuando tras las primeras redistribuciones de carga no aparecen nuevas variaciones, y los tiempos de ejecución se mantienen similares. En estos casos realizar redistribuciones frecuentemente puede llegar a ser perjudicial.

CAPÍTULO 3
Prototipado

Con el objetivo de determinar cuál de las herramientas de medición del consumo energético analizadas en la Sección 2.1 es más conveniente utilizar en este proyecto, se crean sendos prototipos con las dos más prometedoras teniendo en cuenta el escenario experimental disponible, descrito a continuación: PowerTop y las interfaces RAPL.

3.1 Escenario experimental

Las pruebas se ejecutan sobre tres de las máquinas del cluster del grupo Trasgo, cuyas características son las siguientes:

- Manticore: Consta de dos nodos NUMA, cada uno con una CPU Intel Xeon Platinum 8160 con un TDP de 150W, y 128GB de memoria RAM. En total, la máquina tiene 48 cores físicos y 96 hilos.
- Hydra: Consta de dos nodos NUMA, cada uno con una CPU Intel Xeon E5-2609 v3 con un TDP de 85W, y 31GB de memoria RAM. En total, la máquina tiene 12 cores físicos y 12 hilos.
- Medusa: Consta de dos nodos NUMA, cada uno con una CPU Intel Xeon Silver 4208 con un TDP de 85W, y 31GB de memoria RAM. En total, la máquina tiene 16 cores físicos y 32 hilos.

3.2 Prototipos implementados

3.2.1. Prototipo con PowerTop

PowerTop es una herramienta interesante ya que nos permite obtener estimaciones del consumo de cada proceso por separado, esta información es más detallada de la que podemos obtener mediante otras herramientas.

Por otro lado, PowerTop es un programa externo, y no nos permite obtener estas estimaciones directamente desde un programa en C, sino que es necesario ejecutar la herramienta, almacenar su salida, y extraer los datos necesarios de la misma.

Debido a esta limitación, el prototipo desarrollado utiliza un script externo para monitorizar el consumo de energía, el cual se puede ver en la Figura 3.1. Este script simplemente llama a `powertop` periódicamente y almacena sus informes en archivos separados, hasta que se le indica que pare mediante una bandera almacenada en un archivo externo.

```
#!/bin/bash

i=0
stop=0

while [ $stop -ne 1 ]
do
  nohup powertop --csv=powertop_logs/${i} -t 1 &
  i=$(($i+1))
  stop=$(cat stop_monitoring)
  sleep 1
done
```

Figura 3.1: Script de monitorización del consumo de energía mediante PowerTop.

Prueba del prototipo

Para probar este prototipo, se utiliza un programa en C que utiliza MPI para lanzar varios procesos, los cuales, en función de su rango (Similar al PID pero utilizado internamente por MPI), pasan un tiempo suspendidos, seguido de la ejecución de un bucle vacío durante diez segundos y otro período de suspensión, como se muestra en la Figura 3.2. El objetivo de este código es demostrar que PowerTop nos permite ver cómo los procesos aumentan y disminuyen su consumo de energía por separado.

Como paso previo a la ejecución de estos bucles, el programa primero obtiene los PIDs de cada uno de los procesos en ejecución, los envía al proceso raíz, y los almacena en un archivo. Esto es necesario para poder extraer la potencia consumida por los procesos de los informes de PowerTop.

```
sleep(rank * 5);
time(&start);time(&current);
while(current < start+10)
{
  time(&current);
}
sleep(((size-1)-rank) * 5);
```

Figura 3.2: Código MPI utilizado para probar la monitorización del consumo de energía.

Las pruebas realizadas consisten en lo siguiente:

1. Se inicia el script de monitorización del consumo de energía.
2. Se lanza el programa MPI de prueba con la opción `-bind-to core`, de manera que cada proceso se ejecutará en uno de los cores de la máquina.
3. Una vez terminada la ejecución, se para el bucle de monitorización.
4. Se extrae el consumo de energía de los procesos de los informes generados, y se crea una gráfica.

Se realizan las pruebas sobre una máquina con cuatro cores, lanzando PowerTop cada segundo. En la Figura 3.3 se puede ver el resultado de una de estas ejecuciones. Podemos

ver cómo se puede distinguir claramente el período donde cada uno de los procesos ha estado activo.

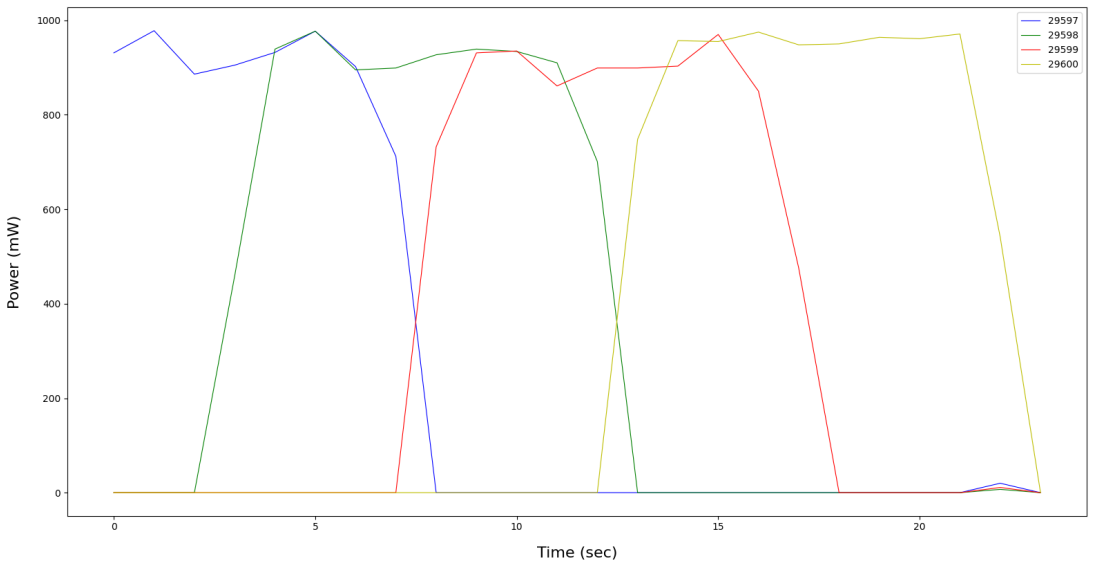

Figura 3.3: Gráfica del consumo de energía generada con datos obtenidos mediante PowerTop.

3.2.2. Prototipo con RAPL

Una ventaja de utilizar las interfaces de RAPL frente a PowerTop es que podemos acceder a las estimaciones de consumo sin depender de un programa externo. Además de eso, otro inconveniente de PowerTop es que tiene un periodo de inicialización tras ejecutarlo, seguido de un período de monitorización, y finalmente produce el informe. Mediante RAPL podemos obtener las mediciones de forma casi instantánea, ya que sólo tenemos que leer del MSR el último valor registrado.

El prototipo desarrollado consta de:

- Una serie de funciones para leer los MSR del procesador y extraer los valores de los distintos campos.
- Una serie de funciones para interpretar los valores de los MSR relacionados con RAPL y calcular la potencia del procesador.

RAPL proporciona, en primer lugar, las precisiones con las que el procesador mide la potencia, la energía, y el tiempo, a través del MSR MSR_RAPL_POWER_UNIT. Este MSR nos proporciona tres valores: PU, ESU, y TU. Calculamos los incrementos que mide el procesador como:

- Potencia: $1/2^{PU}$ Vatios.
- Energía: $1/2^{ESU}$ Julios.
- Tiempo: $1/2^{TU}$ Segundos.

Así, por ejemplo, si ESU toma el valor 10000b (16 en decimal), sabemos que el procesador nos proporciona la energía en incrementos de $1/2^{16} = 0{,}00001525878$ Julios. Tendremos

que multiplicar las medidas de energía que obtengamos por 0.00001525878 para obtener el valor en Julios.

Esto se calcula una vez mediante una función de inicialización y se guardan los valores obtenidos para el resto de la ejecución.

A través del MSR MSR_[Dominio]_ENERGY_STATUS (Donde dominio puede ser PKG, PP0, PP1, o DRAM), RAPL nos proporciona la energía consumida por ese dominio desde la última vez que se vació el registro, en las unidades de energía explicadas previamente.

Las funciones implementadas utilizan este valor para devolver la potencia consumida en Vatios (1 Julio/Segundo). Para ello, cada vez que se llama a la función se almacena el timestamp en el que se ha hecho la llamada y la energía consumida hasta ese momento en Julios. Se utiliza esta información para calcular la energía consumida desde la última llamada y la diferencia de tiempo en segundos, después se divide la energía entre el tiempo para obtener el valor en Vatios.

De esta manera obtenemos la potencia media durante el período entre llamadas. Este valor es más util que obtener la energía consumida en Julios, por un lado podemos comparar distintos períodos aunque su duración no sea la misma, y especialmente, nos permite comparar ese valor con la potencia máxima del procesador según su especificación, que se proporciona en Vatios.

TDP de un procesador

El TDP (Thermal Design Power) [32] de una CPU Intel nos indica el consumo en Vatios que tendría el procesador bajo su mayor carga computacional teórica. Nos resulta útil para obtener el consumo de un procesador como un porcentaje de su máximo teórico, lo cual nos permite comparar el consumo de máquinas con CPUs distintas.

Podemos obtener el TDP de la CPU a través del MSR MSR_PKG_POWER_INFO (según [21]). En este prototipo se utiliza el valor obtenido de esta forma para proporcionar los resultados como un porcentaje del TDP de cada máquina.

Máquinas con varios procesadores

En el caso de máquinas con varios procesadores, se tiene en cuenta que cada procesador mantiene sus propios registros, por lo que es necesario calcular la energía consumida por cada uno y posteriormente combinar esa información.

Respecto al TDP de la máquina en este caso, consideramos la suma del TDP de cada procesador de la misma.

Prueba del prototipo

Una opción para probar este sistema sería la de usar un programa de monitorización, como en el caso anterior con PowerTop. Sin embargo, ya que en este caso vamos a ejecutar cada instancia del programa en una máquina distinta en lugar de en distintos cores de un procesador, tendríamos el problema añadido de buscar una forma de obtener medidas simultáneamente en todas las máquinas. Por otra parte, en este caso no es necesario un monitor externo, y no es la forma en la que se pretende que se use el sistema.

Por tanto, se ha implementado una aplicación de prueba distinta a la anterior, que se acerca más al uso que veríamos en un programa real. En esta aplicación, tenemos varios períodos en los que simulamos una carga computacional distinta para cada procesador,

con sincronizaciones entre máquinas al acabar cada período, antes de las cuales medimos el consumo de energía durante el período. Se va variando la carga computacional de los procesadores en los distintos períodos, con el objetivo de mostrar que el programa desarrollado detecta correctamente estos cambios en el consumo de energía, y que nos permite identificar las máquinas bajo una mayor carga computacional.

El código utilizado se muestra en la Figura 3.4. El mecanismo utilizado para generar carga computacional es un bucle en el cual se incluyen operaciones de distinto tipo, con el objetivo de utilizar todo el procesador, en concreto se incluyen sumas, multiplicaciones, y dos llamadas a funciones matemáticas, en concreto una raíz cuadrada y una potenciación. Para variar la carga computacional entre períodos se hace que el programa pase parte del período durmiendo y parte en este bucle. En concreto, estamos utilizando bucles de 4 segundos de duración, lo que nos facilita hacer que cada procesador duerma un 0 %, 25 %, 50 %, 75 % o 100 % del período. Para asignar un porcentaje de carga distinto a cada procesador en cada período, toman un offset en función de su rango MPI. Por ejemplo, los cinco primeros períodos de la máquina con rango 0 corresponderían a un 0-25-50-75-100 % de carga, mientras que los de la máquina con rango 1 corresponderían a un 25-50-75-100-0 % de carga.

Se utiliza OpenMP para poder ejecutar el programa tanto en un solo hilo como en todos los hilos de la máquina simultáneamente, y determinar si hay diferencias en el comportamiento de las máquinas. Se puede ver la parte relevante del código utilizado en la Figura 3.4.

Resultados

Las Figuras 3.5 y 3.6 muestran las gráficas generadas mediante las aplicaciones de prueba descritas anteriormente.

Vemos que en ambos casos se detectan correctamente los cambios en el consumo de energía de cada máquina, así como las diferencias en el consumo de las máquinas. Esta sección se centra en mostrar el funcionamiento de la librería de medición del consumo de energía mediante RAPL, pero se elabora más sobre estos resultados en el Capítulo 4.

3.2.3. Conclusiones y herramienta elegida

Los dos prototipos implementados obtienen los resultados esperados, y nos permiten distinguir correctamente procesos distintos en base a su consumo de energía. Mediante PowerTop podemos distinguir entre hilos dentro de una máquina, mientras que con RAPL tenemos una menor resolución, y tenemos el consumo de todo el procesador, con lo que distinguimos entre hilos en máquinas distintas.

A pesar de que nos ofrece menos información, se ha observado que es mucho más sencillo y fiable medir el consumo de energía mediante RAPL en máquinas distintas. Por ejemplo, no se ha conseguido que PowerTop produzca estimaciones del consumo de energía en algunas de las máquinas del cluster. Además de eso, RAPL produce medidas de forma mucho más rápida, sin tener que llamar a un programa externo con un período de inicialización y otro de medición. Por estos motivos se ha decidido medir el consumo de energía mediante RAPL para este proyecto.

```
#pragma omp parallel private(thread_num, num_threads)
{
  thread_num = omp_get_thread_num();
  num_threads = omp_get_num_threads();
  for(int i=0; i<20; i++){
    if(thread_num == 0){
    clock_gettime(CLOCK_REALTIME, &start);
    clock_gettime(CLOCK_REALTIME, &current);
    end_loop = 0;
  }
  temp = 9; loop_time = 4;
  #pragma omp barrier
  //4 second simulated load period
  sleep(4 - ((i + rank)%5));
  while(!end_loop){
    temp = temp * sqrt(temp) + pow(temp, 2);
    if(thread_num == 0){
      clock_gettime(CLOCK_REALTIME, &current);
      if((double)current.tv_sec + ((double)current.tv_nsec / 1E9) >=
        ((double)start.tv_sec + ((double)start.tv_nsec / 1E9)) + loop_time)
    {
        end_loop = 1;
      }
    }
  }
    if(thread_num == 0){
      //Obtain this period's average power consumption
      period_consumption = get_package_power();
      //Store the results for visualization
      fprintf(results, "%f\n", period_consumption/tdp*100);
      //Synchronization
      MPI_Barrier(MPI_COMM_WORLD);
    }
    #pragma omp barrier
  }
}
```

Figura 3.4: Código MPI utilizado para generar carga de forma artificial.

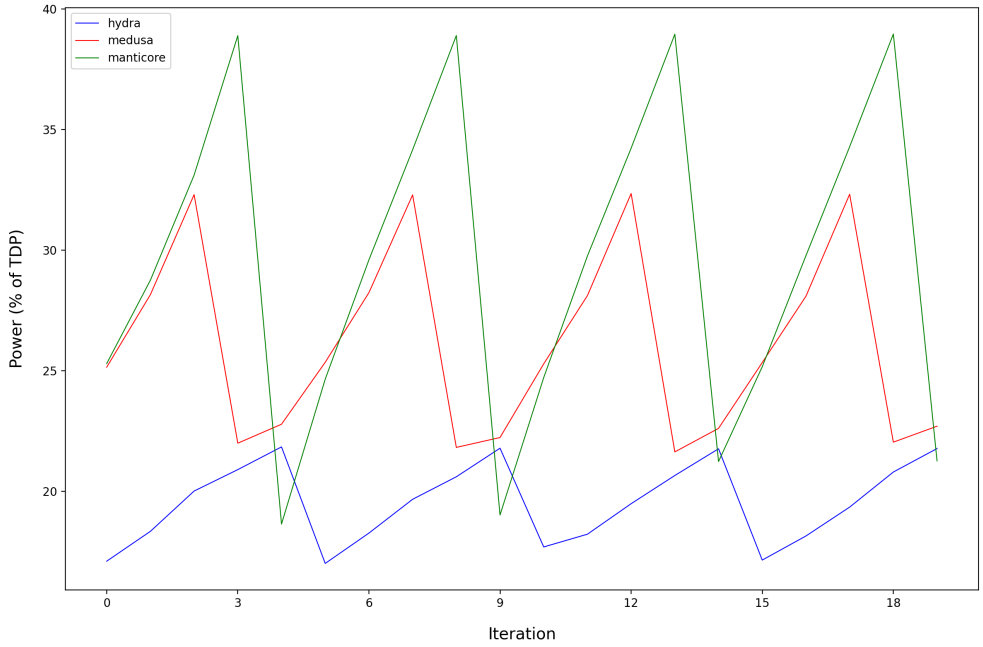

Figura 3.5: Resultados con carga artificial sobre un hilo.

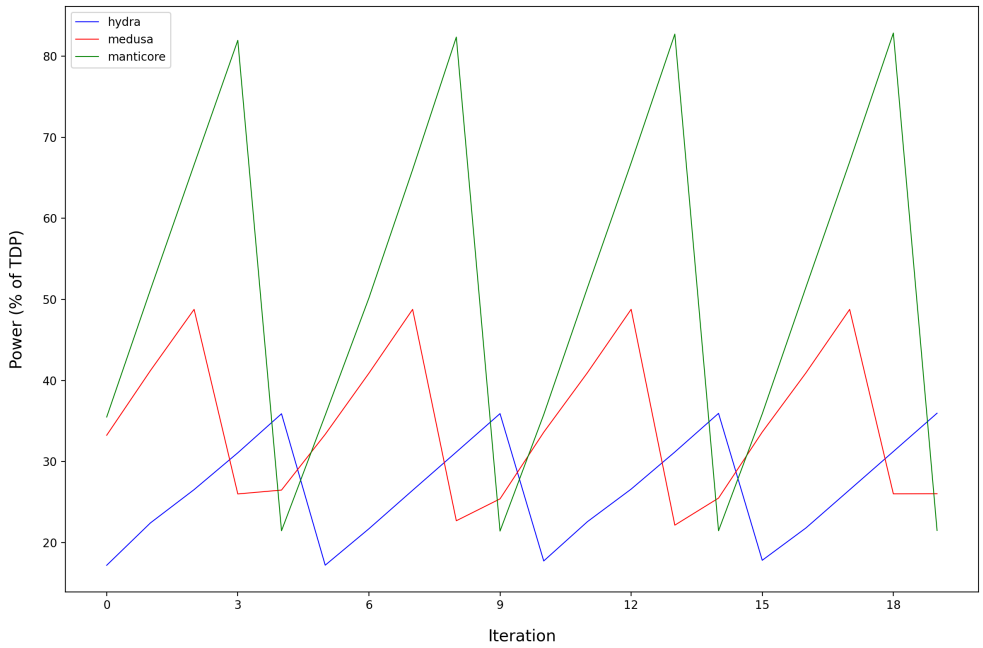

Figura 3.6: Resultados con carga artificial sobre todos los hilos de cada máquina.

CAPÍTULO 4
Experimentación previa

Una vez elegida la herramienta que se utilizará para medir el consumo energético, se llevan a cabo una serie de pruebas con el objetivo de determinar la viabilidad de la propuesta así como la mejor estrategia a seguir para reducir el consumo energético de una aplicación.

En esta sección se trata lo siguiente:

- **Aplicaciones principales empleadas, Stencil y Matmul.**
- **Estudio del comportamiento de Stencil y el programa de carga artificial.**

  Se comienza estudiando el comportamiento que podemos esperar con Stencil, principalmente en cuanto al consumo energético. También se habla sobre los resultados del programa de carga artificial presentados en la sección anterior.
- **Estudio del efecto del balanceo de carga en el consumo energético.**

  Se determina si una solución de balanceo de carga tradicional, orientada a reducir el tiempo de ejecución de un programa puede afectar al consumo energético. Esto podría ir asociado a la propia reducción del tiempo de ejecución, pero también se determina si un cambio en la carga computacional de una máquina afecta a su consumo.
- **Estudio del perfil de consumo energético de las máquinas utilizadas.**

  En vista de los resultados obtenidos, se estudia la modificación de las frecuencias de los procesadores como una vía para alterar el consumo de energía de las máquinas en función de su carga de trabajo.

4.1 Aplicaciones Empleadas

4.1.1. Stencil

Esta es la principal aplicación empleada, fue la aplicación utilizada para probar la solución de balanceo de carga de Hitmap, desarrollada en [15].

El programa calcula el punto de estabilidad de una Ecuación Parcial Diferencial, en este caso la ecuación de Poisson de difusión del calor, mediante un método iterativo de Jacobi sobre un espacio discreto de 2 dimensiones, representado mediante una matriz.

La aplicación está implementada como un método de Stencil de 4 puntos, que ejecuta el número especificado de iteraciones sobre toda la matriz. Durante cada iteración, se calcula un nuevo valor para cada celda de la matriz a partir de las cuatro vecinas.

Paralelizando esta aplicación, cada hilo se encarga de actualizar parte de la matriz, lo que implica realizar sincronizaciones entre vecinos al finalizar cada iteración.

Para implementar balanceo de carga en esta aplicación mediante HitMap, se mide el tiempo empleado en la fase de actualización de cada iteración. Una vez se dispone de suficientes medidas, en caso de que haya que realizar una redistribución de carga, se envía parte de los datos correspondientes a los hilos más lentos a los más rápidos.

Esta aplicación se elige debido a sus características principales: Una ejecución dividida en iteraciones, en la que la carga computacional se puede distribuir entre distintas máquinas, realizando comunicaciones al final de cada iteración en las cuales se envían a las otras máquinas parte de los resultados, necesarios para realizar la siguiente iteración.

Debido a esta necesidad de comunicar resultados entre máquinas, nos encontramos ante uno de los tipos de aplicación en los que es más interesante aplicar balanceo de carga. Para obtener el máximo rendimiento, es necesario que el cómputo de cada iteración tenga una duración lo más similar posible en cada máquina.

4.1.2. Multiplicación de Matrices

Esta aplicación es una multiplicación de matrices clásica, se calcula el producto de dos matrices cuadradas distintas, almacenando el resultado en una tercera: C = A * B.

El cómputo de cada celda de la matriz resultado es independiente del resto, sin embargo, hay elementos de A y B que se utilizan para calcular distintas celdas de C, y se pueden realizar optimizaciones centradas en reutilizar estos datos. Otro tipo de optimización común es la de explotar los patrones de acceso a las matrices.

En este caso, utilizamos el programa únicamente como una forma de generar carga computacional, por lo que no nos preocupamos en exceso por este tipo de optimizaciones. La implementación utilizada es sencilla, el cómputo se divide en bloques, y está paralelizado mediante OpenMP de modo que el procesamiento de estos bloques se divida entre los hilos de la máquina.

4.2 Estudio del comportamiento de la aplicación de Stencil

Se lleva a cabo un estudio para determinar el comportamiento de la aplicación de Stencil descrita previamente en cuanto al tiempo de ejecución y consumo energético.

También se estudian los resultados obtenidos por el programa utilizado para probar el prototipo de medición del consumo energético mediante RAPL.

4.2.1. Resultados con el programa de carga artificial

En las Figuras 3.5 y 3.6 vemos los resultados en cuanto al consumo de energía de la aplicación descrita en la Sección 3.2.2. Vemos que, como esperábamos, se observan claramente los cambios en la potencia usada por las máquinas entre los distintos períodos, también vemos el cambio entre usar un único hilo y usar todos los hilos de la máquina.

Es muy destacable que, pese a que se está ejecutando el mismo programa en las tres máquinas, la potencia media consumida como porcentaje del TDP de la máquina es muy distinta. Vemos, por ejemplo, que con un sólo hilo, el consumo máximo de la máquina hydra está alrededor de un 25 % de su TDP (170W), mientras que el de manticore se acerca a un 40 % del TDP (300W).

Vemos también una gran diferencia en el comportamiento cuando pasamos a usar todos los hilos de la máquina. De nuevo comparando los consumos máximos de hydra y manticore, el de hydra está sobre un 38 % del TDP, es decir, aumenta en un factor de 1.52 respecto a usar un sólo hilo, mientras que el de manticore se acerca al 80 % del TDP, un factor de 2 respecto al consumo con un único hilo.

Estos resultados son problemáticos hasta cierto punto ya que implican que comparar dos máquinas distintas en cuanto a la potencia que utilizan no es sencillo, y probablemente requiera un conocimiento previo del perfil de consumo de las mismas.

4.2.2. Resultados con el programa de Stencil

En esta sección se obtienen datos del programa de Stencil descrito en la Sección 4.1.1, instrumentado mediante RAPL para obtener mediciones de consumo energético. La experimentación se realiza en las máquinas descritas en Sección 3.1. Se emplean todos los hilos disponibles en cada máquina.

Experimentación realizada

Se realiza una ejecución del programa en la cual se emplean todos los hilos disponibles en cada máquina con los siguientes parámetros:

- Tamaño de la matriz: 32768×32768
- Número de iteraciones: 1024

Se registran los siguientes datos:

- Tiempo medio de iteración, en segundos.
- Tiempo medio de iteración, sin incluir el tiempo empleado en comunicaciones.
- Potencia media por iteración, en Vatios.
- Energía consumida por iteración, en Julios.
- Tiempo total de ejecución, en segundos.
- Energía total consumida, en Julios.

Resultados en cuanto a tiempo de ejecución

La Figura 4.1 muestra los tiempos de iteración medios en cada máquina durante las 1024 iteraciones realizadas para esta prueba.

Vemos que durante la primera mitad de la ejecución los tiempos de iteración son relativamente constantes en todas las máquinas. Sin embargo, llega un punto en el que estos tiempos comienzan a aumentar progresivamente en las tres.

El hecho de que el aumento se produzca de manera simultánea en todas las máquinas es que, al haber sincronizaciones entre máquinas al final de cada iteración, una ralentización en una máquina afectará a las demás de forma similar.

Sin embargo, resulta difícil explicar por qué se da este comportamiento. Como se menciona en la introducción de este trabajo, en ocasiones aparecen factores externos durante la ejecución de un programa que están fuera del alcance del programador. Precisamente este tipo de cambios durante la ejecución es lo que motiva muchas soluciones de balanceo de carga.

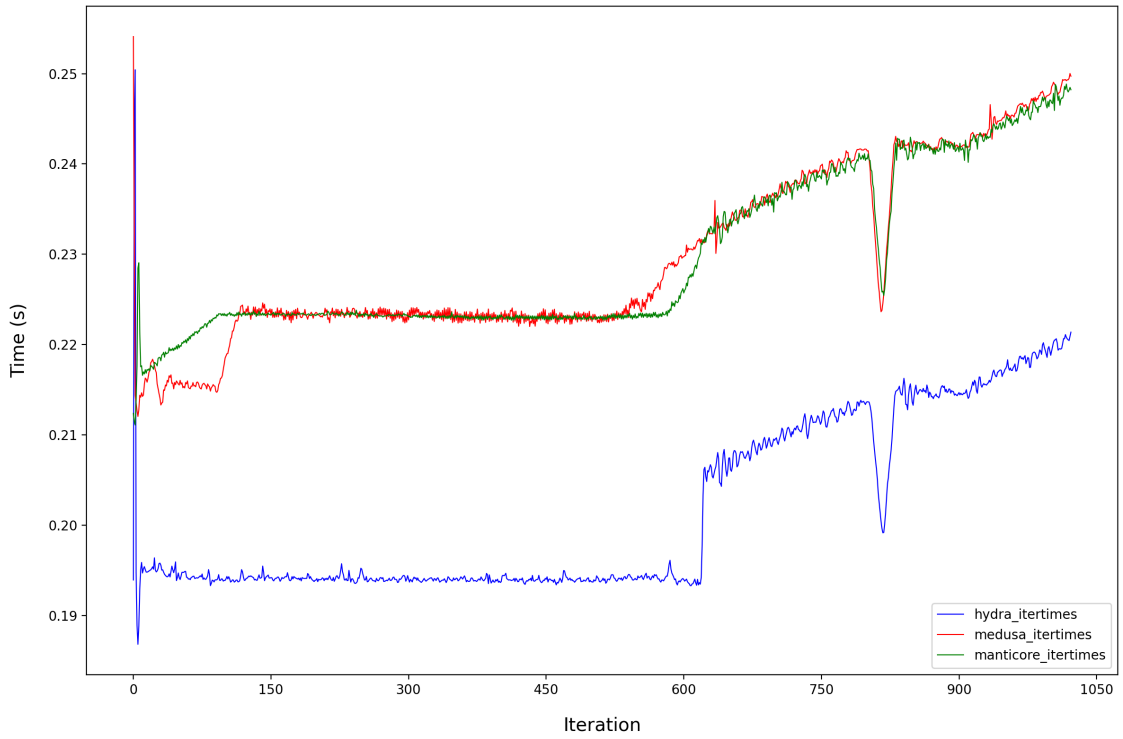

Figura 4.1: Tiempo medio de iteración del método de Stencil.

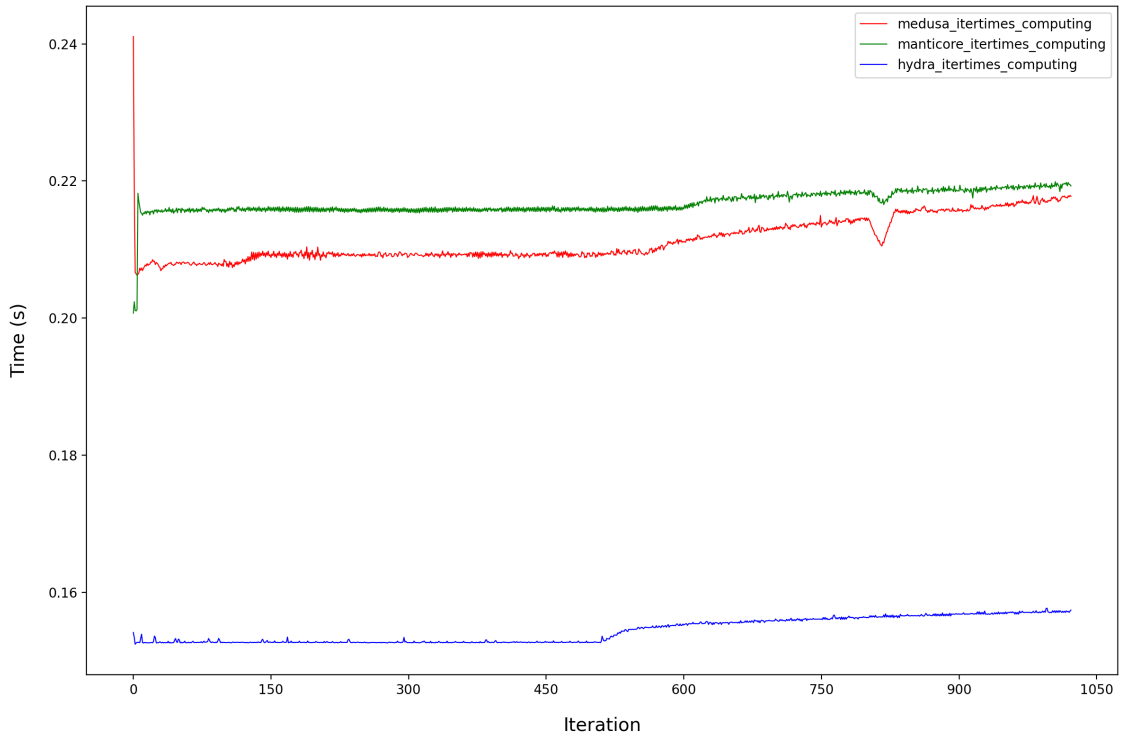

Figura 4.2: Tiempo medio de iteración del método de Stencil, sin comunicaciones.

Resultados en cuanto a consumo energético

La Figura 4.3 muestra la potencia media consumida por iteración en cada una de las máquinas.

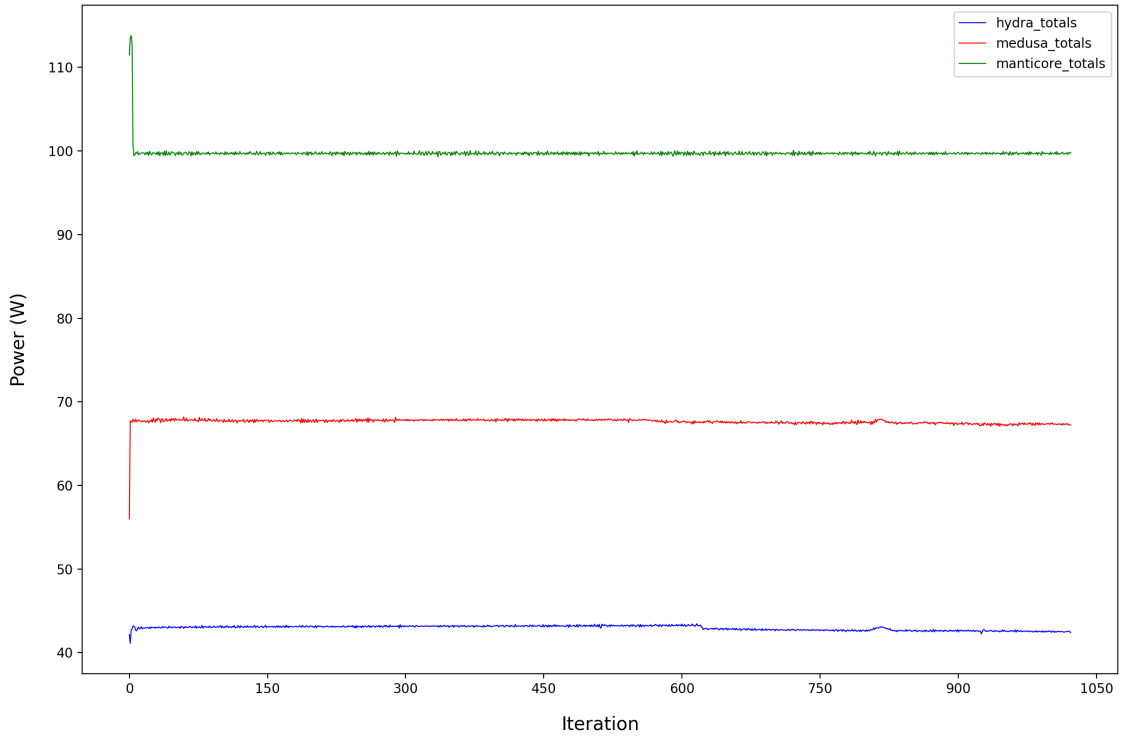

Figura 4.3: Potencia media por iteración del método de Stencil.

Vemos, en primer lugar, que el consumo energético en las tres máquinas es muy estable a lo largo de la ejecución. Además de esto, al igual que se explica en la sección anterior, vemos que el consumo de cada máquina como porcentaje de su TDP es muy distinto.

Esto presenta un problema para el objetivo de este proyecto de equilibrar el consumo energético variando la carga asignada a cada nodo. Sabemos que si se parte de un reparto de carga uniforme, como es el caso, esta aplicación no está bien balanceada y tenemos hilos que pasan parte del tiempo esperando por los resultados del resto. Sin embargo, nada en estas gráficas nos indica que el consumo de alguna máquina se ve afectado por esto (Por ejemplo, esperaríamos ver un consumo reducido en una máquina que pase parte de su tiempo esperando los resultados del resto).

Lo que es más, cuando nos centramos en los tiempos de iteración vemos que estos varían a lo largo de la ejecución, sin embargo, no parece que el consumo se vea afectado por esto.

Por último, la Figura 4.4 muestra la energía consumida por iteración, en Julios. Como la potencia se mantiene constante en todas las máquinas, vemos que la energía consumida se comporta de manera muy similar a los tiempos de iteración.

4.3 Estudio del efecto del balanceo de carga en el consumo energético

Se estudia el efecto que tiene aplicar una solución de balanceo de carga tradicional en el consumo energético de una aplicación. Algo expecialmente interesante en este caso es

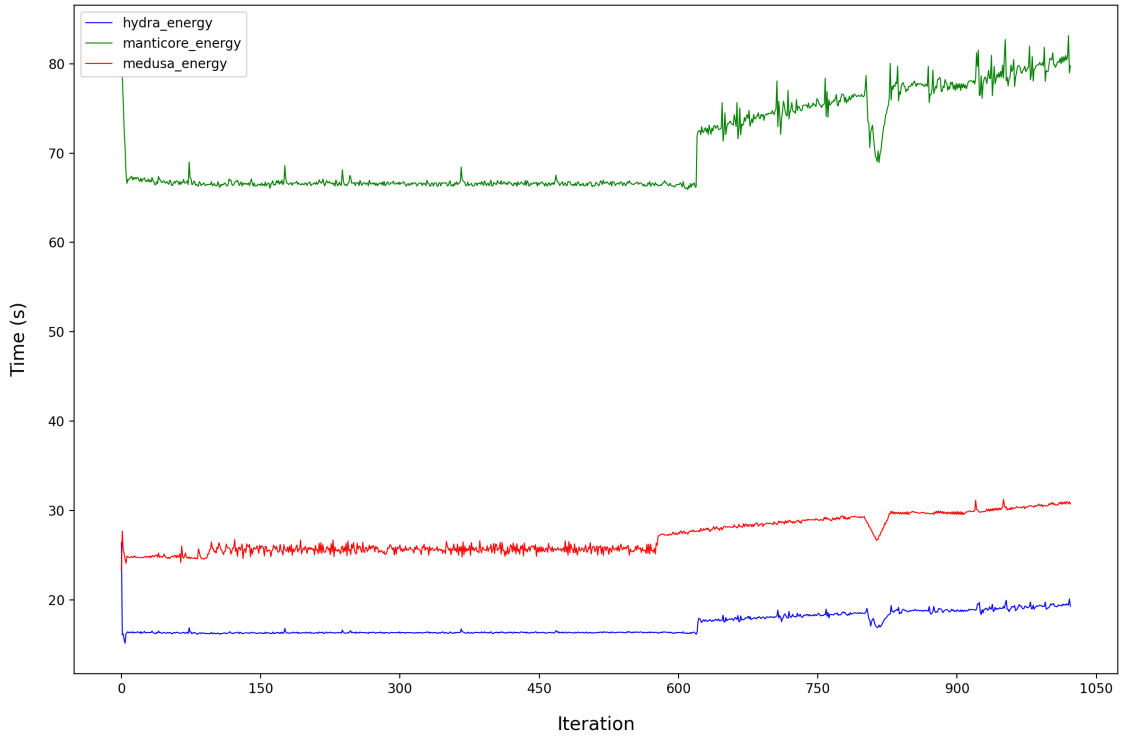

Figura 4.4: Energía consumida en Julios por iteración del método de Stencil.

estudiar si aumentar o reducir la cantidad de datos que procesan las máquinas durante una iteración afecta al consumo.

4.3.1. Resultados con el programa de Stencil

Experimentación realizada

Se ejecuta el mismo experimento que en el caso sin balanceo de carga, es decir, usamos los siguientes parámetros:

- Tamaño de la matriz: 32768×32768
- Número de iteraciones: 1024

En este caso, activamos el mecanismo de balanceo de carga descrito en la Sección 4.1.1, aplicando la estrategia en la que el período entre redistribuciones aumenta progresivamente.

Resultados en cuanto a tiempo de ejecución

La Figura 4.5 muestra los tiempos medios de iteración en este caso. Podemos apreciar tres picos en la gráfica, que se corresponden con las redistribuciones realizadas.

Para observar mejor el efecto del balanceo de carga, la Figura 4.6 muestra exclusivamente el tiempo empleado en la parte de la iteración dedicada a cómputo, sin tener en cuenta el tiempo relacionado con las comunicaciones. En contraste con la ejecución sin balanceo de carga, vista en la Sección 4.2, se aprecia que tras la primera redistribución realizada los tiempos de iteración pasan a ser más similares entre todas las máquinas. Tras la segunda redistribución el reparto mejora en cuanto a que los tiempos de iteración

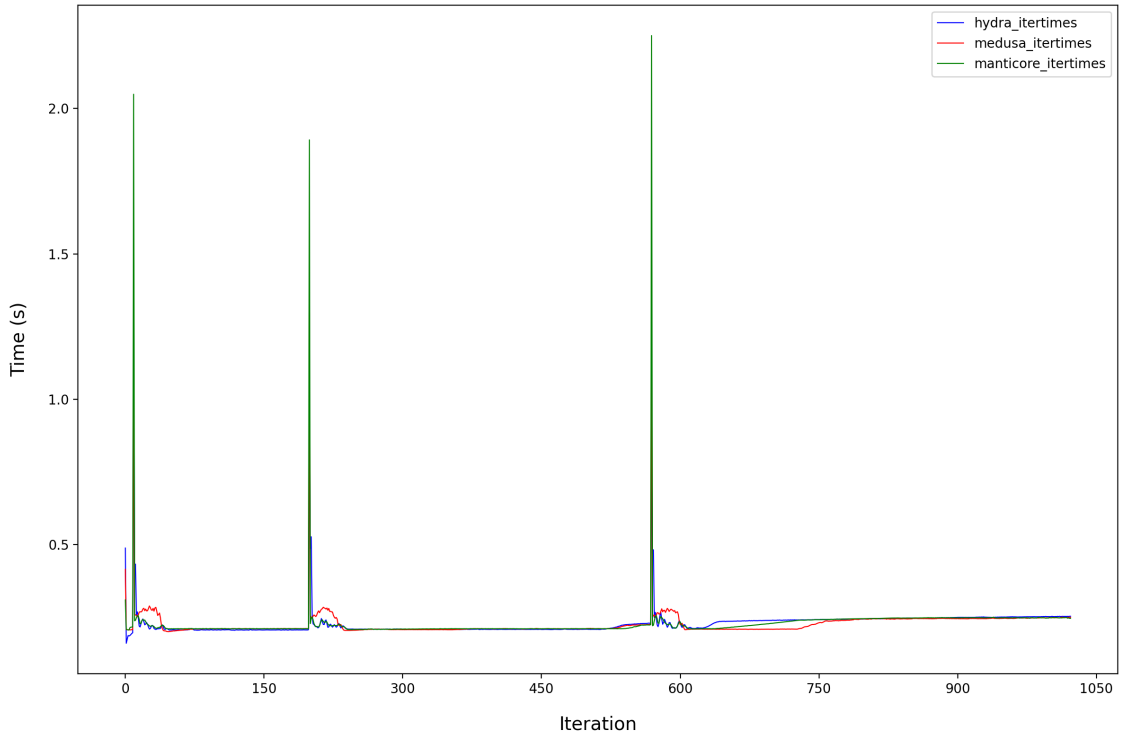

Figura 4.5: Tiempo medio de iteración del método de Stencil con balanceo de carga.

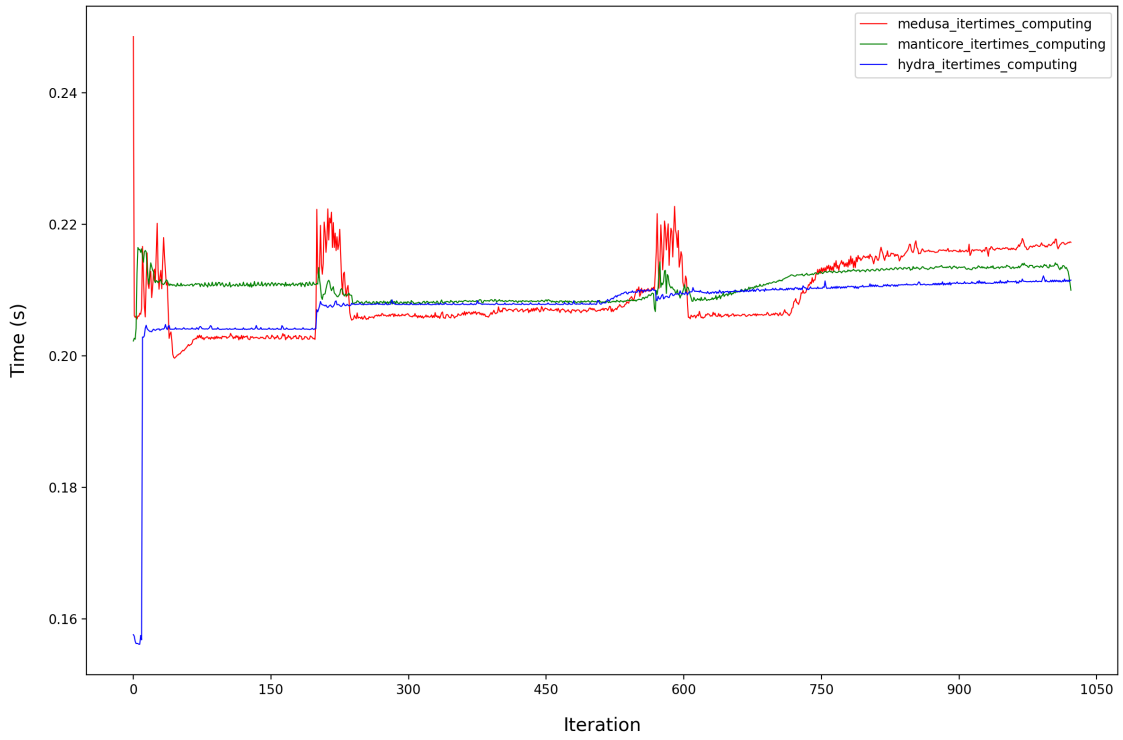

Figura 4.6: Tiempo medio de iteración del método de Stencil con balanceo de carga, sin comunicaciones.

se acercan más. Vemos que aparece el mismo comportamiento que observábamos en la ejecución sin balanceo en la que los tiempos de iteración empiezan a aumentar a partir de un punto determinado. El último balanceo de carga realizado lo contrarresta en cierta medida.

Resultados en cuanto a consumo energético

La Figura 4.7 muestra la potencia media por iteración de esta ejecución.

Se pueden destacar las pequeñas variaciones en la potencia que coinciden con las redistribuciones observadas en la Figura 4.5 con los tiempos de ejecución. En general, sin embargo, estos resultados son muy similares a los obtenidos cuando no usamos balanceo de carga, las diferencias en el tiempo de iteración y por tanto en la carga computacional de las máquinas no parecen afectar al consumo de las mismas, ya que este sigue siendo muy constante.

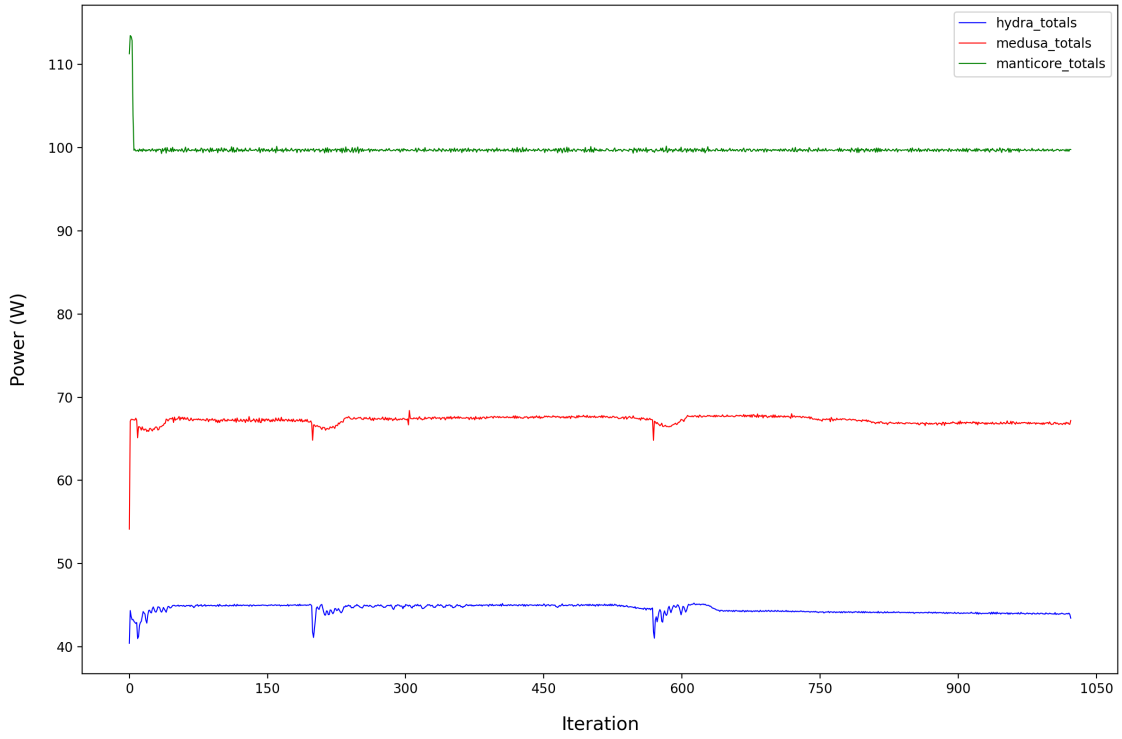

Figura 4.7: Potencia media por iteración del método de Stencil con balanceo de carga.

En la Figura 4.8 vemos la energía consumida por iteración en este caso. Dado que la potencia es constante, al igual que cuando no usamos balanceo de carga la energía consumida aumenta cuando el tiempo de iteración aumenta. Además de eso, en este caso destacan mucho los picos de consumo que aparecen durante las redistribuciones realizadas.

4.3.2. Comparativa con y sin balanceo de carga

Una vez se ha estudiado el comportamiento de la aplicación de Stencil cuando aplicamos balanceo de carga, se realiza una comparativa con la aplicación que no lo emplea, centrándonos en las diferencias en el consumo total de energía. El objetivo es determinar si, a pesar de que parece no afectar a la potencia consumida, puede proporcionar una reducción en el consumo derivada de una reducción en el tiempo de ejecución.

En este caso, en función de la duración de la ejecución el balanceo de carga tiene un impacto mayor o menor, por tanto, se amplía la experimentación realizando ejecuciones con distinto número de iteraciones.

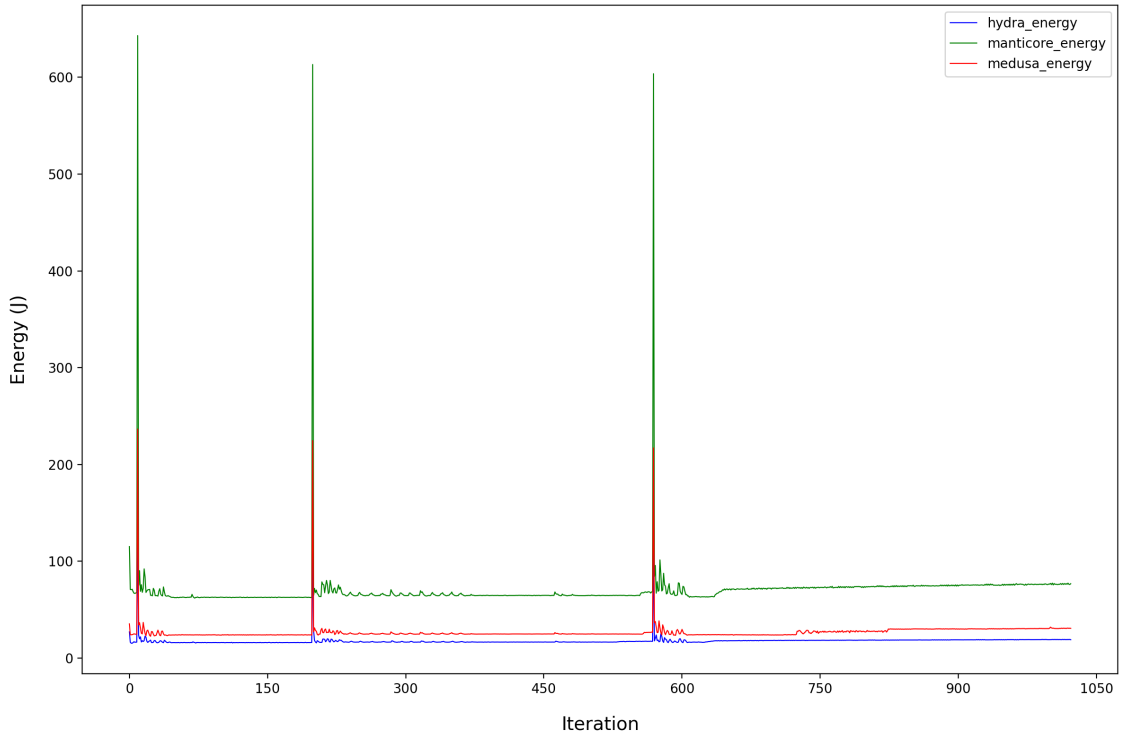

Figura 4.8: Energía consumida en Julios por iteración del método de Stencil con balanceo de carga.

Tabla 4.1: Comparativa con y sin balanceo de carga.

		Sin Balanceo		Con Balanceo		Cambio	
Tamaño	**Iter**	**Tiempo**	**Energía**	**Tiempo**	**Energía**	**Tiempo**	**Energía**
32768	512	113	55335	113	55412	0.00 %	0.14 %
32768	1024	241	117662	239	117114	-0.83 %	-0.47 %
32768	2048	531	258467	514	251402	-3.20 %	-2.73 %
32768	4096	1189	576531	1086	530039	-8.66 %	-8.06 %

Experimentación realizada

Se realizan las siguientes ejecuciones:

- Matriz de 32768 × 32768, 512 iteraciones.
- Matriz de 32768 × 32768, 1024 iteraciones.
- Matriz de 32768 × 32768, 2048 iteraciones.
- Matriz de 32768 × 32768, 4096 iteraciones.

4.3.3. Resultados

La Tabla 4.1 muestra los resultados obtenidos por cada aplicación en cuanto a tiempo de ejecución y consumo energético total, así como la diferencia entre ambas.

Vemos que a medida que las ejecuciones se hacen más largas, el aplicar balanceo de carga proporciona una mejora en el tiempo de ejecución. Dado que, como se ha visto en la

sección anterior, la potencia de las máquinas cambia muy poco a lo largo de la ejecución, podemos decir que esta reducción en el tiempo de ejecución es la que provoca que el consumo también se reduzca.

Dado que la potencia es relativamente constante para ambas aplicaciones, es esperable que la diferencia porcentual en tiempo de ejecución sea muy similar a la diferencia en cuanto a consumo. Vemos que la diferencia de consumo es algo menor que la de tiempo, es posible que esto se deba a los picos de consumo que observábamos cuando se producen las redistribuciones.

4.4 Problemas detectados y posibles explicaciones

Mediante esta experimentación previa se identifican dos problemas principales:

- Existen grandes diferencias en el consumo de cada máquina, incluso cuando realizamos la comparación en base al porcentaje del TDP utilizado.
- Aunque la carga computacional de las máquinas varíe a lo largo de la ejecución, el consumo permanece uniforme.

4.4.1. Diferencias en el consumo de cada máquina

Una explicación sencilla, aunque probablemente muy complicada de demostrar, es que el TDP proporcionado por Intel esté por encima del consumo real de algunos de sus procesadores. En este caso, el TDP del procesador de manticore estaría bien ajustado al consumo real, ya que en las gráficas aparece siempre al 100 %, pero en hydra y medusa el consumo máximo real sería en torno a un 45 y 65 % de lo indicado por Intel, respectivamente.

4.4.2. Consumo uniforme independientemente de la carga

En este caso consideramos que hay dos factores principales que explican los resultados obtenidos.

El primero está relacionado con la implementación del programa mediante MPI. Una iteración de la aplicación tiene dos fases principales: cómputo y comunicación, donde la fase de comunicación tiene un envío y una recepción. Inicialmente se esperaba que la fase de cómputo generara la mayor parte del gasto energético, mientras que durante la comunicación el procesador estaría en descanso. Sin embargo, para hacer las comunicaciones lo más rápidas posibles las esperas en MPI son activas, es decir, el procesador está en un bucle comprobando constantemente si, en este caso, se han recibido los datos necesarios y se puede continuar con la ejecución. Esto provoca que la actividad del procesador sea esencialmente la misma durante toda la iteración.

El segundo factor que provoca esto es la gestión que el sistema operativo hace del consumo energético del procesador. En concreto, el estándar ACPI [22] define los *P-States* como una serie de niveles de rendimiento del procesador mientras está activo (Ejecutando instrucciones), este sistema se implementa en procesadores Intel como una serie de interfaces software que permiten controlar la frecuencia y el voltaje del procesador [23].

Estos estados comienzan en P0, el estado de máximo rendimiento (Y, por tanto, de máximo consumo), y los estados siguientes, P1, P2, etc., reducen el consumo, y por tanto también el rendimiento, mediante la reducción de frecuencia y voltaje del procesador.

El comportamiento que se ha observado tanto en las máquinas del cluster del grupo Trasgo como en la máquina personal del autor, todas ellas ejecutando sistemas Linux, es que cuando se empieza a ejecutar una aplicación con cómputo contínuo, como es el programa de Stencil, la frecuencia del procesador aumenta y se mantiene constante, lo que parece indicar que el sistema pone el procesador en el mismo estado, P0, durante toda la ejecución.

Esto es una buena aproximación en general, ya que si estamos ejecutando un programa normalmente querremos que obtenga resultados lo más rápido posible. Sin embargo, podemos encontrarnos casos como este, donde el desarrollador sabe que en ocasiones aunque una máquina termine sus cálculos rápidamente, va a tener que esperar a los datos de otras máquinas, y consumirá energía innecesariamente. En estas situaciones, ralentizar una máquina para que termine su trabajo de manera más sincronizada con las otras puede reducir el consumo energético sin que aumente el tiempo de ejecución global.

4.5 Conclusiones

A partir de estas pruebas previas, se extraen las siguientes conclusiones principales:

- En el escenario experimental utilizado aparecen efectos de ejecución a priori fuera del control del desarrollador.

- Variaciones significativas en algunos de los parámetros monitorizados, como el tiempo de ejecución, no parecen afectar a la potencia utilizada por las máquinas.

  Si el hecho de variar la carga computacional afectase de forma significativa a la potencia consumida por las máquinas, quizá sería posible plantear una solución de balanceo de carga que tuviese en cuenta la potencia en lugar del tiempo de ejecución, y tratase de mantener el consumo del cluster en un mínimo.

- El consumo real en relación al TDP especificado varía significativamente entre distintas máquinas.

- El uso de una solución de balanceo de carga orientada a reducir el tiempo de ejecución puede proporcionar mejoras en la energía consumida.

  La solución de balanceo de carga utilizada no tiene en cuenta el consumo de las máquinas, e introduce picos de consumo al realizar redistribuciones, sin embargo, la reducción en el tiempo de ejecución compensa este efecto.

- Las estrategias de gestión energética del sistema operativo puede introducir ineficiencias en cuanto a consumo en algunos programas.

  Incluso si se consigue reducir el consumo utilizando balanceo de carga, siguen existiendo ineficiencias en cuanto a consumo relacionadas en parte con las estrategias de escalado de frecuencias del sistema operativo.

- MPI aplica estrategias que buscan obtener el máximo rendimiento de las aplicaciones, sin embargo, esto no siempre es óptimo en cuanto a consumo energético.

4.6 Estudio del perfil de consumo energético en función de la frecuencia del procesador

En vista de las conclusiones anteriores, se decide estudiar el comportamiento de las máquinas disponibles cuando variamos la frecuencia de la CPU, con el objetivo de contestar a las siguientes preguntas:

- ¿Es posible reducir la energía empleada en realizar un trabajo reduciendo la frecuencia del procesador?
- ¿La relación entre frecuencia de ejecución y consumo es lineal?

4.6.1. Modificación de la frecuencia de la CPU en sistemas Linux

En sistemas Linux, la frecuencia de la CPU se gestiona a través del sistema `cpufreq` [33]. Este sistema permite escalar la frecuencia del procesador para ahorrar energía, ya sea de forma automática, o por parte del usuario.

El sistema consta de dos componentes principales, los "Drivers" y los "Governors":

- Los Drivers son módulos del kernel que se comunican con la CPU para establecer su frecuencia y voltaje mediante el sistema de P-States descrito en la Seeción 4.4.2. En función de la CPU se utilizará un driver distinto, pero los más comunes son intel_pstate, utilizado en las CPUs más modernas de Intel, intel_cpufreq, en CPUs de Intel de 5ª generación o más antiguas, y por último acpi-cpufreq.
- Los Governors [34] son programas que deciden la frecuencia de ejecución de la CPU en base a una política determinada y a unas frecuencias mínima y máximas especificadas por el usuario, y la aplican mediante los Drivers de `cpufreq`. En Linux existen los siguientes Governors por defecto:

  Powersave: Pone la CPU en la frecuencia mínima especificada para el governor.

  Performance: Pone la CPU en la máxima frecuencia especificada para el governor.

  Userspace: Pone la CPU a la frecuencia especificada por el usuario.

  Ondemand: Establece la frecuencia de la CPU dinámicamente en función de la carga del sistema.

  Conservative: Similar a Ondemand, pero los cambios de frecuencia ocurren de forma más gradual, mientras que con Ondemand cuando la carga computacional aumenta la frecuencia rápidamente salta al máximo.

  Schedutil: Trata de estimar la carga y la frecuencia necesaria mediante la información proporcionada por el Scheduler de procesos del kernel.

Por otro lado, en el caso de estar usando el driver intel_pstate, los Governors compatibles son los llamados "Performance" y "Powersave", de Intel, cuyo comportamiento sería análogo a los Governors estándar Schedutil y Ondemand, respectivamente.

La interfaz preferida [33] para interactuar con el sistema `cpufreq` por parte del usuario son los nodos `sysfs` definidos para cada core de la cpu en `/sys/devices/system/cpu/cpu#/cpufreq/`. Las funcionalidades más interesantes para este trabajo son las siguientes:

- Acceder a las frecuencias mínima y máximas para la CPU.
- Establecer las frecuencias mínima y máximas que el Governor puede establecer.

- Acceder a la frecuencia actual de la CPU.

Para realizar esta experimentación, y dado que las máquinas utilizadas ejecutan por defecto el Governor Powersave de Intel, se decide que la forma de variar las frecuencias para estas pruebas será variar la frecuencia máxima a la que el Governor puede poner la CPU. Esta estrategia funciona bien con este Governor y produce el comportamiento esperado ya que el comportamiento del Governor es saltar rápidamente a la frecuencia máxima cuando hay carga en la CPU, y mantener esa frecuencia hasta que la carga desaparece.

La notación utilizada para hablar sobre la frecuencia de las máquinas es la siguiente: Definimos la frecuencia mínima del Hardware como el 0 %, y la máxima como el 100 %. En función de la CPU que estemos probando estos porcentajes representarán frecuencias distintas, pero el número concreto no es especialmente relevante, y hablar en estos términos nos permite comparar el comportamiento de las máquinas.

4.6.2. Experimentación realizada

Se realizan pruebas con dos aplicaciones diferentes, el método de Stencil ya utilizado previamente, y un programa de multiplicación de matrices por bloques. El objetivo de incluir estas dos aplicaciones es determinar si los resultados obtenidos varían en función de las características de la aplicación.

El objetivo principal es establecer cómo varía el consumo de las máquinas disponibles, todas diferentes entre sí, a medida que modificamos la frecuencia de la CPU entre el mínimo y el máximo posibles.

Para ambas aplicaciones, se ejecutan las pruebas descritas tomando 100 valores distintos para la frecuencia, distribuidos uniformemente entre el mínimo y el máximo.

Además de esto, se realiza una prueba a mayores con el programa de carga artificial descrito en la Sección 3.2.2.

Stencil

Con Stencil, ejecutamos 10 iteraciones para cada valor de la frecuencia, y medimos lo siguiente:

- Tiempo de ejecución en segundos.
- Potencia media consumida en Vatios.
- Energía consumida en Julios.

Utilizamos en estas ejecuciones una matriz de tamaño 32768×32768. A diferencia de la experimentación anterior, estas ejecuciones se realizan utilizando una única máquina en cada caso.

Matmul

En el caso de la multiplicación de matrices, multiplicamos para cada valor de la frecuencia dos matrices de tamaño 2048×2048. Se miden los mismos parámetros que en el caso de Stencil.

4.6.3. Resultados en función de la variación de la frecuencia

Stencil

Las Figuras 4.9, 4.10 y 4.11 muestran los tiempos de ejecución respecto a la frecuencia.

Vemos que el aumento en el tiempo de ejecución a medida que disminuye la frecuencia no es lineal. Se aprecia un comportamiento similar en todos los casos, donde inicialmente la frecuencia se reduce pero no aumenta el tiempo de ejecución, y a continuación el tiempo aumenta a medida que se reduce la frecuencia. En Hydra este aumento es mucho más brusco que en las otras dos máquinas.

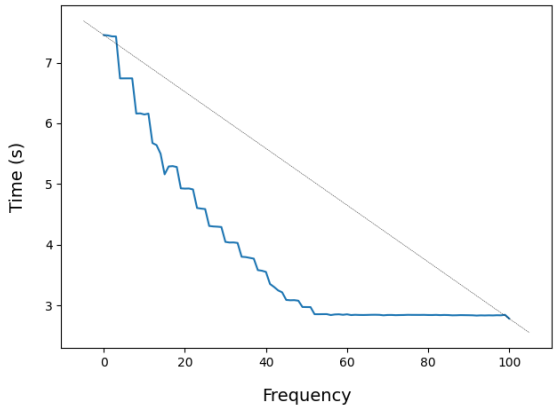

Figura 4.9: Tiempo de ejecución vs Frecuencia, Manticore. Stencil

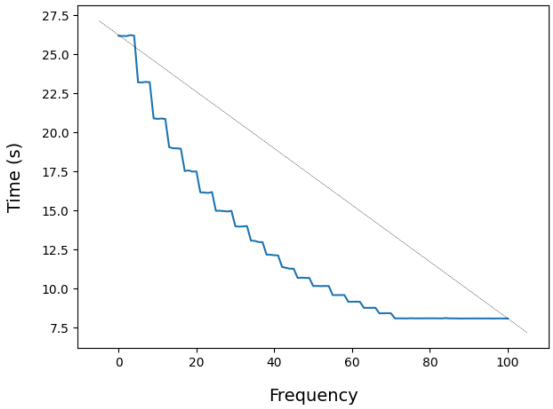

Figura 4.10: Tiempo de ejecución vs Frecuencia, Medusa. Stencil

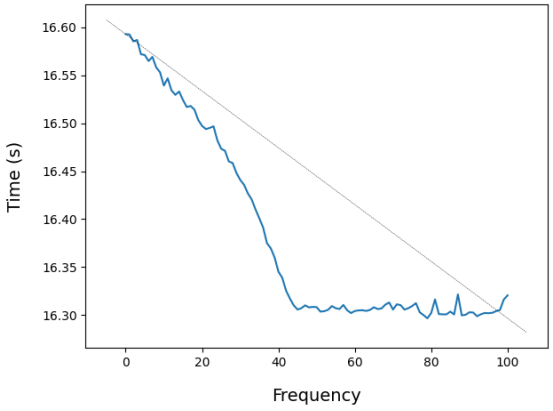

Figura 4.11: Tiempo de ejecución vs Frecuencia, Hydra. Stencil

Las Figuras 4.12, 4.13 y 4.14 muestran la potencia media respecto de la frecuencia.

Aunque el caso de Hydra es algo distinto, y la potencia consumida parece aumentar ligeramente hasta cierto punto a medida que disminuye la frecuencia, los comportamientos de Manticore y Medusa están más en línea con los resultados que se podrían esperar intuitivamente.

Vemos que hay un cierto desajuste respecto a las gráficas de tiempo de ejecución, donde, sobre todo en los casos de Medusa y Manticore, tenemos ciertos rangos de frecuencias en los que el tiempo de ejecución aumenta mientras que la potencia se mantiene constante. Esto indica rangos de frecuencias en los que la máquina es especialmente ineficiente.

Por otro lado, vemos que en cierto momento la potencia consumida varía rápidamente, antes de empezar a descender de forma gradual. Como se ve a continuación en las

gráficas de consumo total, estos saltos, en combinación con el aumento mucho más gradual de los tiempos de ejecución, indican puntos en los que la máquina es más eficiente en cuanto a trabajo realizado respecto al consumo. Por último, las gráficas de las Figu-

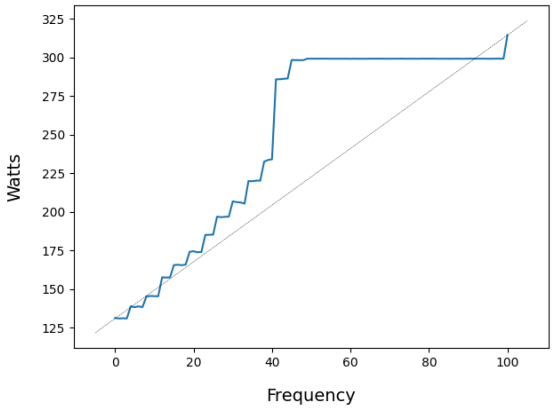

Figura 4.12: Potencia vs Frecuencia, Manticore. Stencil

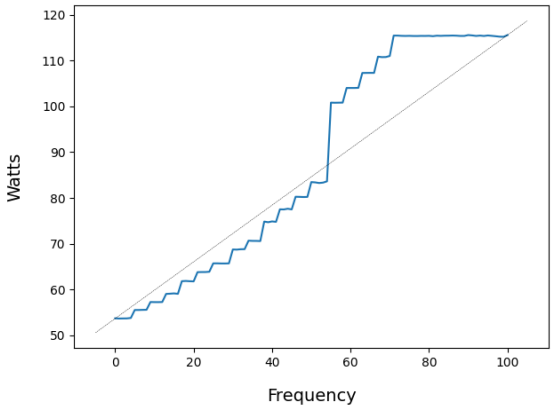

Figura 4.13: Potencia vs Frecuencia, Medusa. Stencil

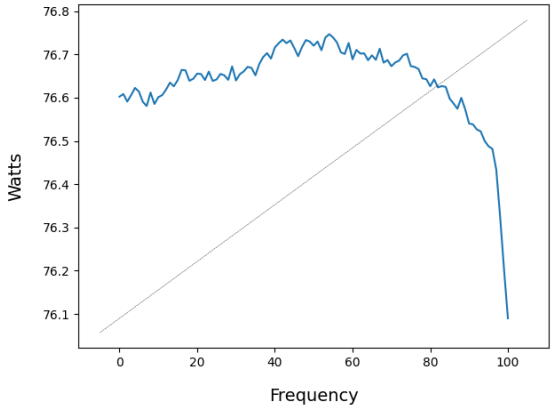

Figura 4.14: Potencia vs Frecuencia, Hydra. Stencil

ras 4.15, 4.16 y 4.17 muestran la energía consumida, en Julios, respecto a la frecuencia. Los resultados en Manticore y Medusa son especialmente interesantes. Muestran cómo, para realizar la misma cantidad de trabajo, tenemos ciertos rangos de frecuencia en los que la energía consumida es mínima, a pesar de que como hemos visto anteriormente el tiempo de ejecución sea mayor. En Manticore esto se produce alrededor de un 40 % de frecuencia, mientras que en Medusa sucede alrededor de un 55 %. En el caso de Hydra, el punto de menor consumo parece ser también el de máxima frecuencia.

Matmul

En el caso de la multiplicación de matrices vemos algunas diferencias. Las gráficas de las Figuras 4.18, 4.19 y 4.20 muestran los tiempos de ejecución, en este caso de cada multiplicación de matrices en cada frecuencia determinada. Vemos que en Manticore y Medusa las tendencias son muy similares al caso anterior, sin embargo en Hydra hay una diferencia considerable, y los tiempos de ejecución no parecen tener correlación con la frecuencia.

En las Figuras 4.21, 4.22 y 4.23 vemos la potencia frente a la frecuencia, de nuevo hay algunas diferencias respecto a Stencil. En el caso de Medusa los resultados son prácti-

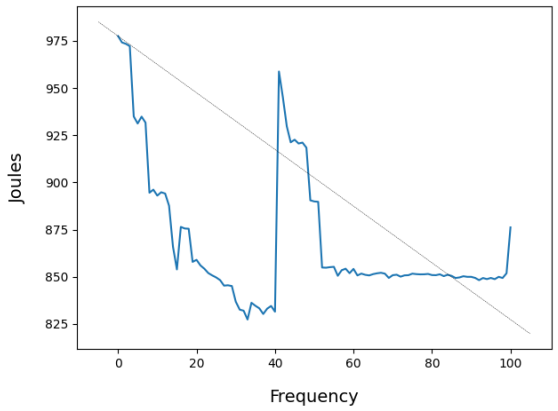

Figura 4.15: Energía vs Frecuencia, Manticore. Stencil

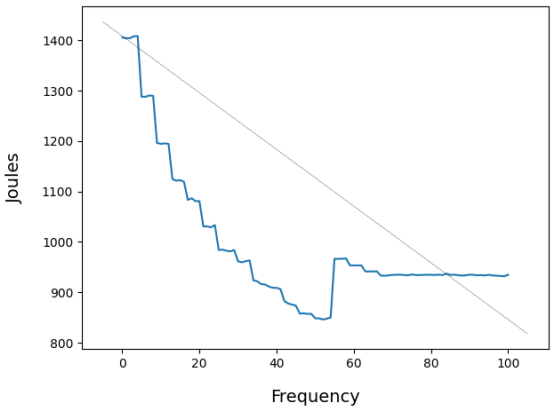

Figura 4.16: Energía vs Frecuencia, Medusa. Stencil

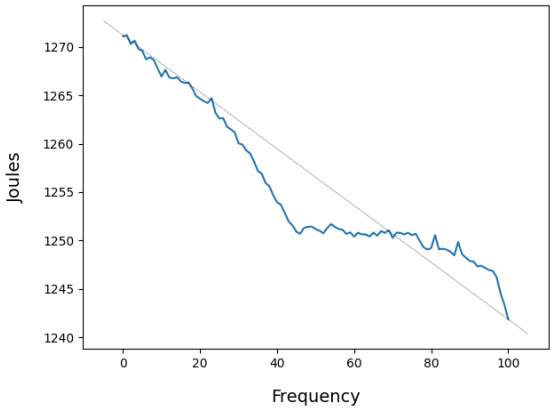

Figura 4.17: Energía vs Frecuencia, Hydra. Stencil

camente idénticos, sin embargo, vemos en Manticore que la potencia es mucho menos estable en las potencias más altas (Aunque la forma general de la gráfica sigue siendo muy similar). Por último, en Hydra vemos la misma situación que observamos con el tiempo, no parece que haya correlación entre potencia y frecuencia.

Por último, tenemos las gráficas de la energía total consumida para realizar el trabajo en las Figuras 4.24, 4.25 y 4.26. Vemos que Medusa se comporta de forma prácticamente idéntica con la multiplicación de matrices que con Stencil, Manticore muestra más diferencias, pero el comportamiento es similar, y en Hydra, de nuevo, no hay correlación.

4.6.4. Conclusiones

Extraemos las siguientes conclusiones de la experimentación con los cambios de frecuencias:

- Aunque las pruebas realizadas son limitadas, sólo dos aplicaciones diferentes, parece que en dos de las máquinas utilizadas los resultados son relativamente consistentes, mientras que en la última hay grandes diferencias.
- En algunas máquinas existen rangos de frecuencia en los que son especialmente eficientes, realizando la misma cantidad de trabajo con un consumo menor que cuando trabajan en otras frecuencias.

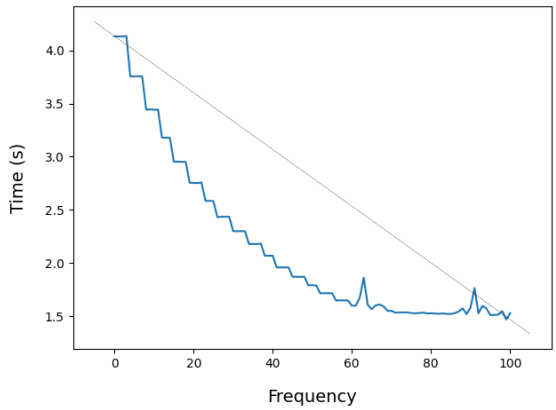

Figura 4.18: Tiempo de ejecución vs Frecuencia, Manticore. Matmul

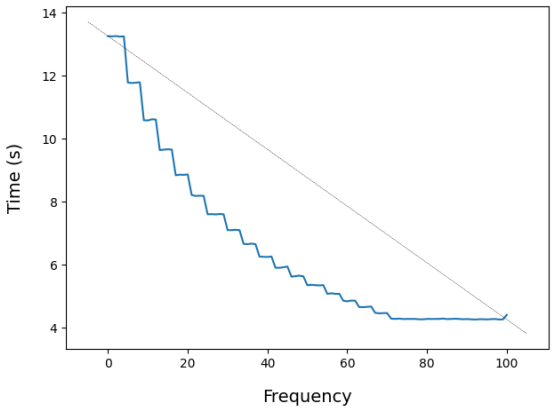

Figura 4.19: Tiempo de ejecución vs Frecuencia, Medusa. Matmul

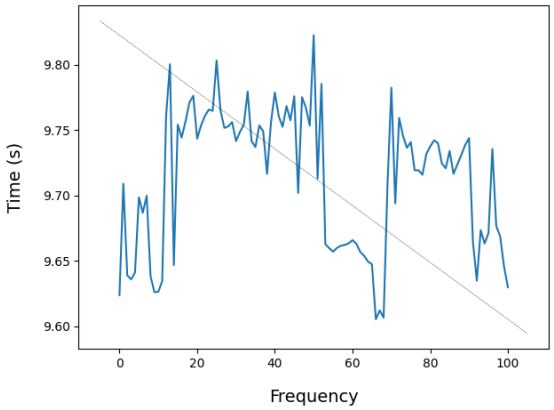

Figura 4.20: Tiempo de ejecución vs Frecuencia, Hydra. Matmul

Esto es particularmente interesante cuando este punto no es la máxima frecuencia de la máquina, sino un punto intermedio. Los governors de escalado de frecuencias en Linux priorizan el rendimiento, sin embargo, estos datos indican que un menor tiempo de ejecución no siempre implica un consumo óptimo.

- No parece haber una relación lineal entre la frecuencia de las máquinas y ninguno de los parámetros medidos, probablemente debido al funcionamiento interno de los mecanismos de cambio de frecuencias de cada CPU y su diseño particular.

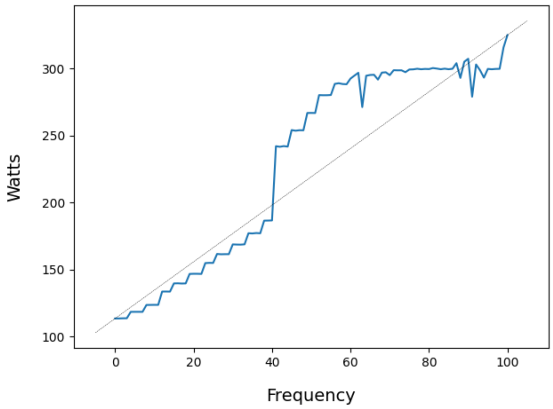

Figura 4.21: Potencia vs Frecuencia, Manticore. Matmul

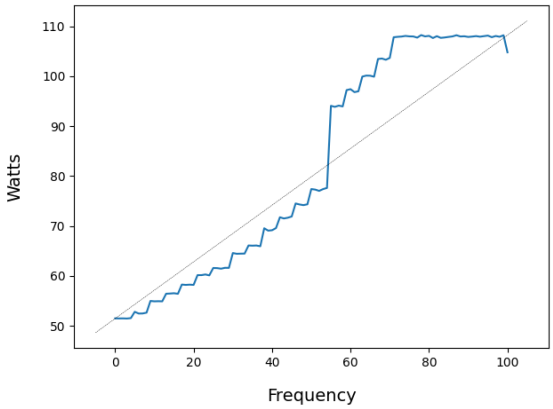

Figura 4.22: Potencia vs Frecuencia, Medusa. Matmul

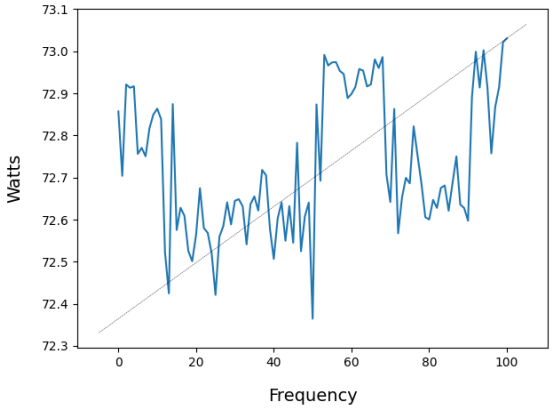

Figura 4.23: Potencia vs Frecuencia, Hydra. Matmul

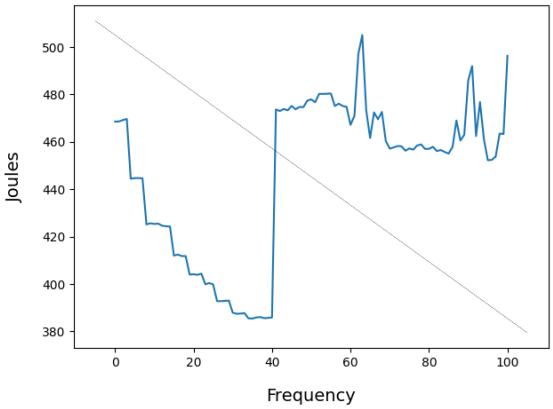

Figura 4.24: Energía vs Frecuencia, Manticore. Matmul

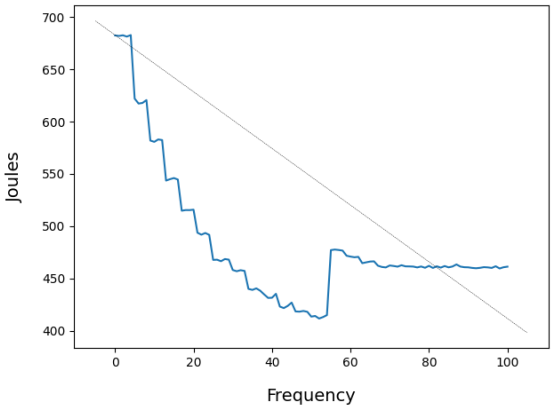

Figura 4.25: Energía vs Frecuencia, Medusa. Matmul

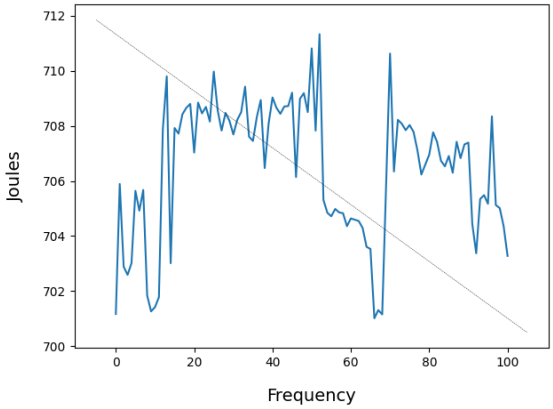

Figura 4.26: Energía vs Frecuencia, Hydra. Matmul

CAPÍTULO 5

Minimización Ad-Hoc del consumo

En este capítulo se llevan a cabo una serie de pruebas de minimización del consumo en base a los resultados obtenidos en el capítulo anterior.

5.1 Minimización Ad-Hoc

En base a los datos obtenidos mediante la experimentación previa, se decide implementar una versión del programa Stencil en el cual las frecuencias se establezcan manualmente en los puntos de máximo rendimiento detectados. Toda la ejecución se realizará con las frecuencias modificadas, sin tener en cuenta la sección que se esté ejecutando.

Esto nos permitirá establecer la reducción en el consumo y la penalización en cuanto a tiempo de ejecución que podemos esperar con la configuración a priori más óptima en cuanto a consumo energético en cada máquina.

Se implementa tanto en la versión sin balanceo de carga como en la que sí lo emplea.

Los valores de frecuencia que se establecen para cada máquina son:

- Manticore: 40 % de frecuencia, o 2080000 Hercios.
- Medusa: 50 % de frecuencia, o 2000000 Hercios.
- Hydra: 100 % de frecuencia, o 1900000 Hercios.

En la Sección 5.2 se muestran los resultados obtenidos mediante esta aplicación.

5.2 Experimentación

5.2.1. Experimentación realizada

Se realizan múltiples ejecuciones de la aplicación tanto con balanceo de carga como sin él, variando el número de iteraciones del método con el objetivo de tener ejemplos de diferente duración. Las pruebas realizadas son:

- Matriz de 32768×32768, 512 iteraciones.
- Matriz de 32768×32768, 1024 iteraciones.
- Matriz de 32768×32768, 2048 iteraciones.

- Matriz de 32768×32768, 4096 iteraciones.

Para cada prueba, los resultados en cuanto a tiempo de ejecución y consumo totales se obtienen como la media de 5 ejecuciones.

Todas las pruebas se realizan sobre las máquinas descritas en la Sección 3.1.

5.2.2. Sin balanceo de carga

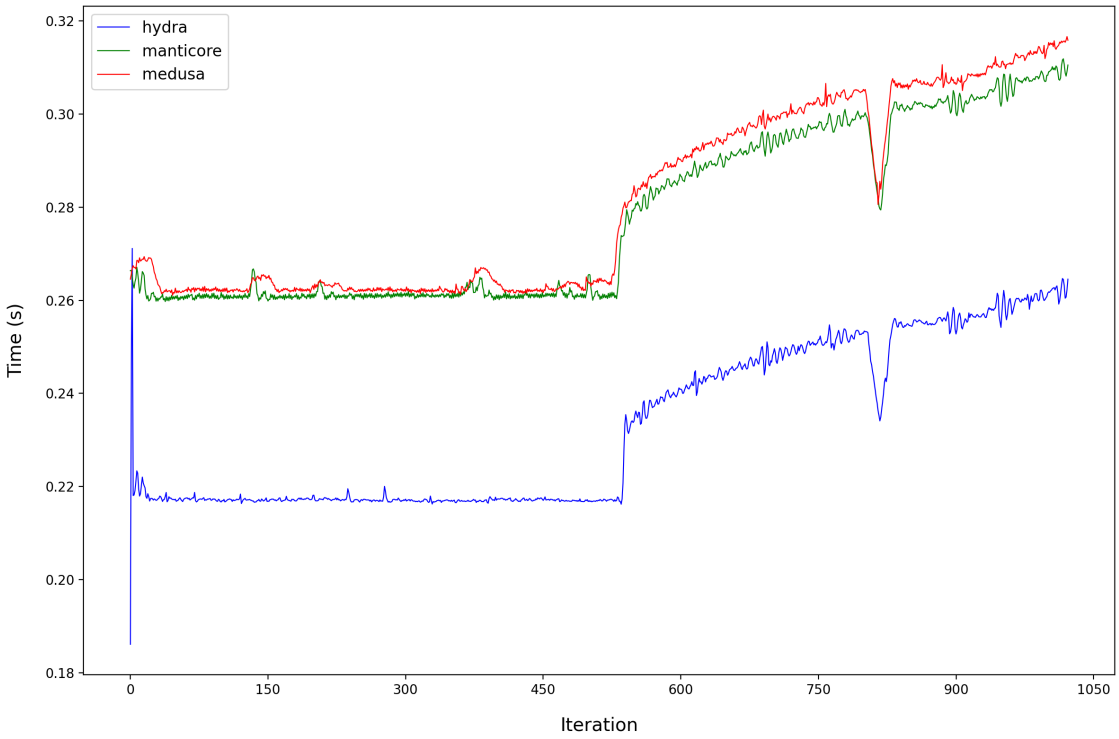

Figura 5.1: Tiempo medio de iteración del método de Stencil.

Estas gráficas muestran los resultados obtenidos para la ejecución con 1024 iteraciones, para facilitar la comparativa directa con lo visto en la experimentación previa, en la Sección 4.2.2.

En cuanto al tiempo y consumo energético, en las Figuras 5.1 y 5.2, los resultados son muy similares a cuando utilizamos las máquinas al máximo de frecuencia, aunque se aprecia el aumento de tiempo de ejecución por iteración y la ligera reducción en la energía consumida por iteración en este caso, en la Figura 5.4. Principalmente, como Hydra permanece al 100 % de frecuencia mientras que en las otras máquinas disminuye, vemos que la diferencia entre ellas aumenta en cuanto a tiempo de ejecución y se reduce en cuanto a consumo.

Probablemente el resultado más destacable es el de la potencia, en la Figura 5.3. Por un lado, vemos cómo en Manticore y Medusa se reduce la potencia, de un 100 % del TDP a alrededor de un 80 %, y de cerca de un 70 % a alrededor del 50 % del TDP, respectivamente en ambas máquinas. Por otro lado, vemos cómo el aumento del tiempo de iteración parece ir asociado a una cierta reducción en la potencia consumida por las máquinas.

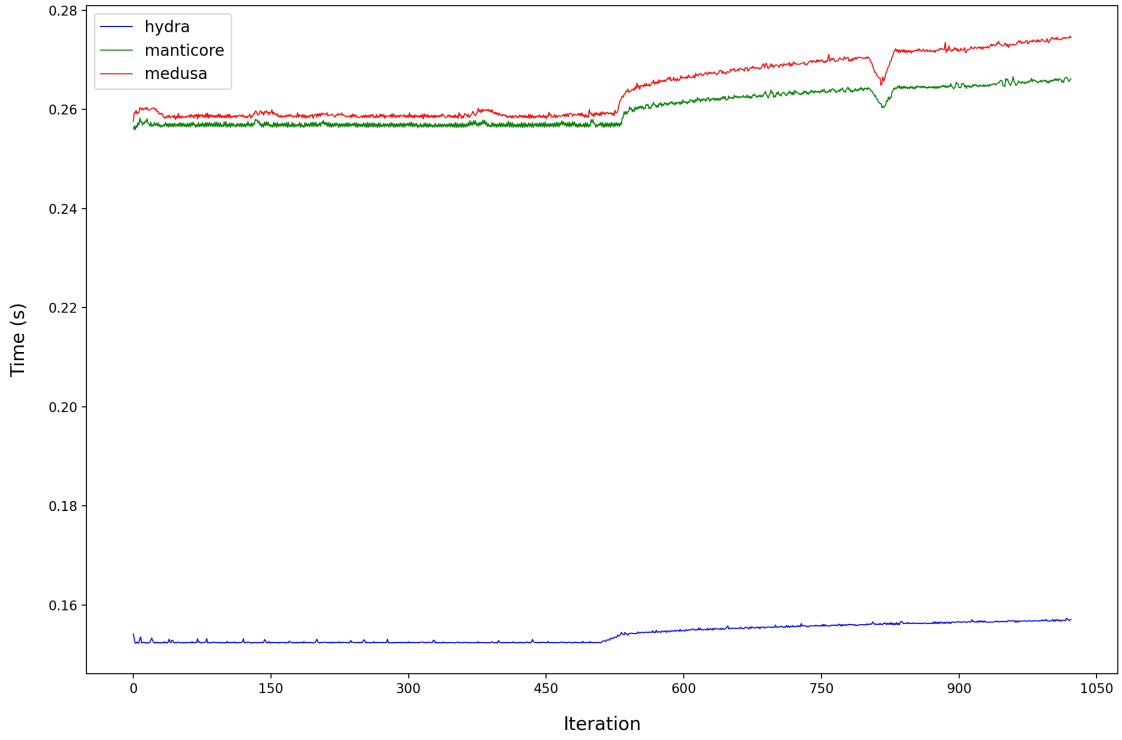

Figura 5.2: Tiempo medio de iteración del método de Stencil, sin comunicaciones.

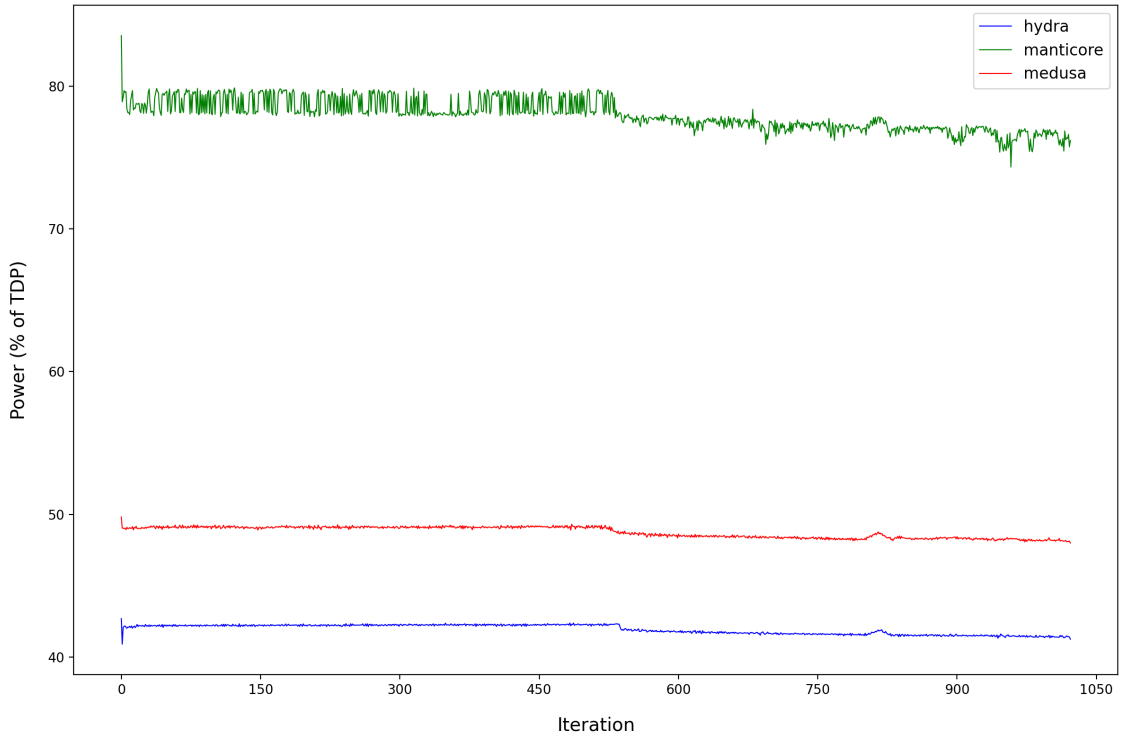

Figura 5.3: Potencia media por iteración del método de Stencil.

5.2.3. Con balanceo de carga

Respecto a lo observado en la experimentación previa (Sección 4.3.1), vemos de nuevo comportamientos similares en las tres variables medidas, tiempo de ejecución en las Figuras 5.5 y 5.6, potencia en la Figura 5.7, y energía en la Figura 5.8. Se aprecia, al igual

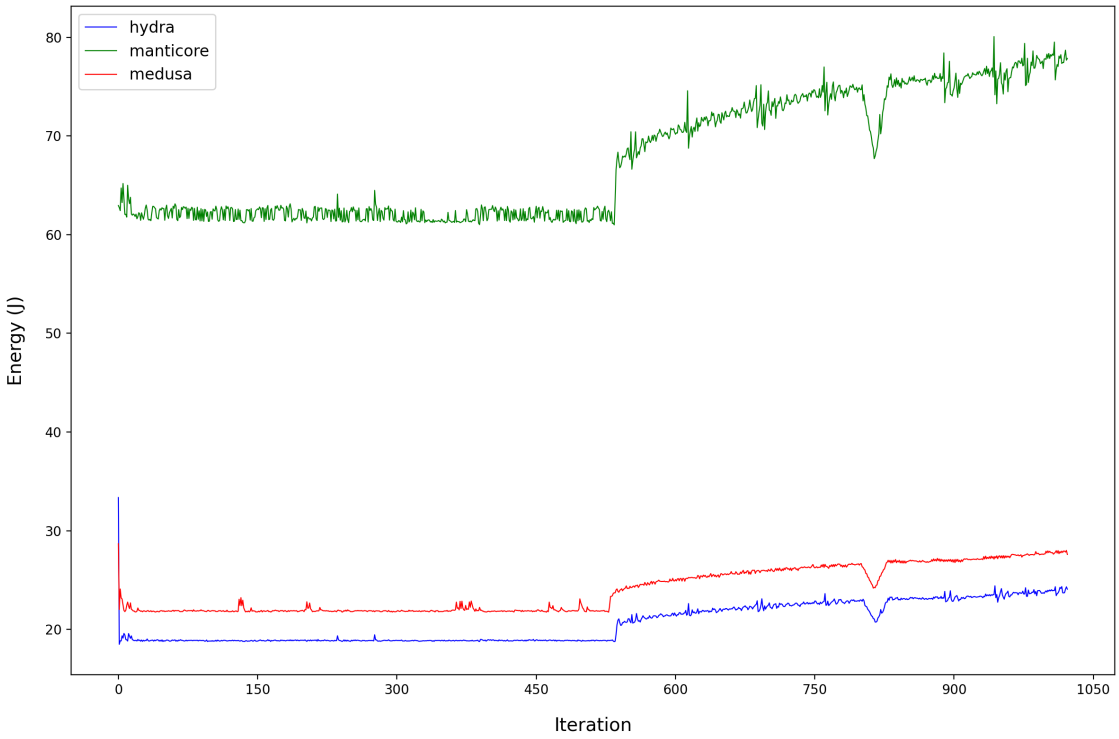

Figura 5.4: Energía consumida en Julios por iteración del método de Stencil.

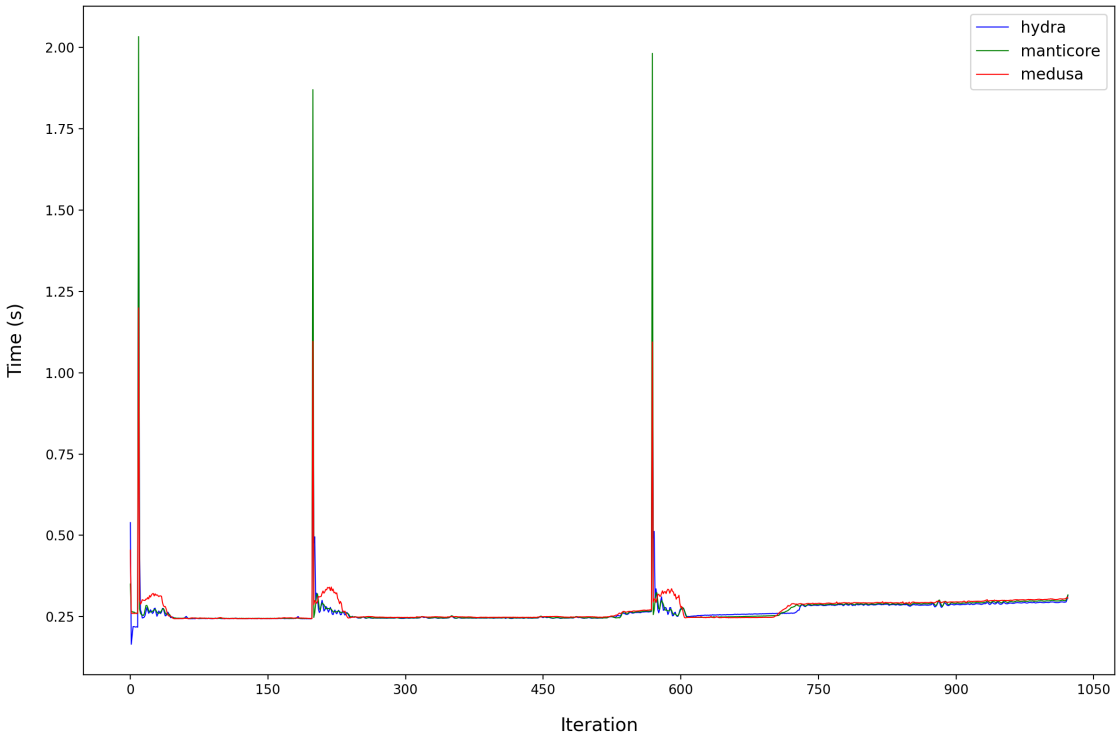

Figura 5.5: Tiempo medio de iteración del método de Stencil con balanceo de carga.

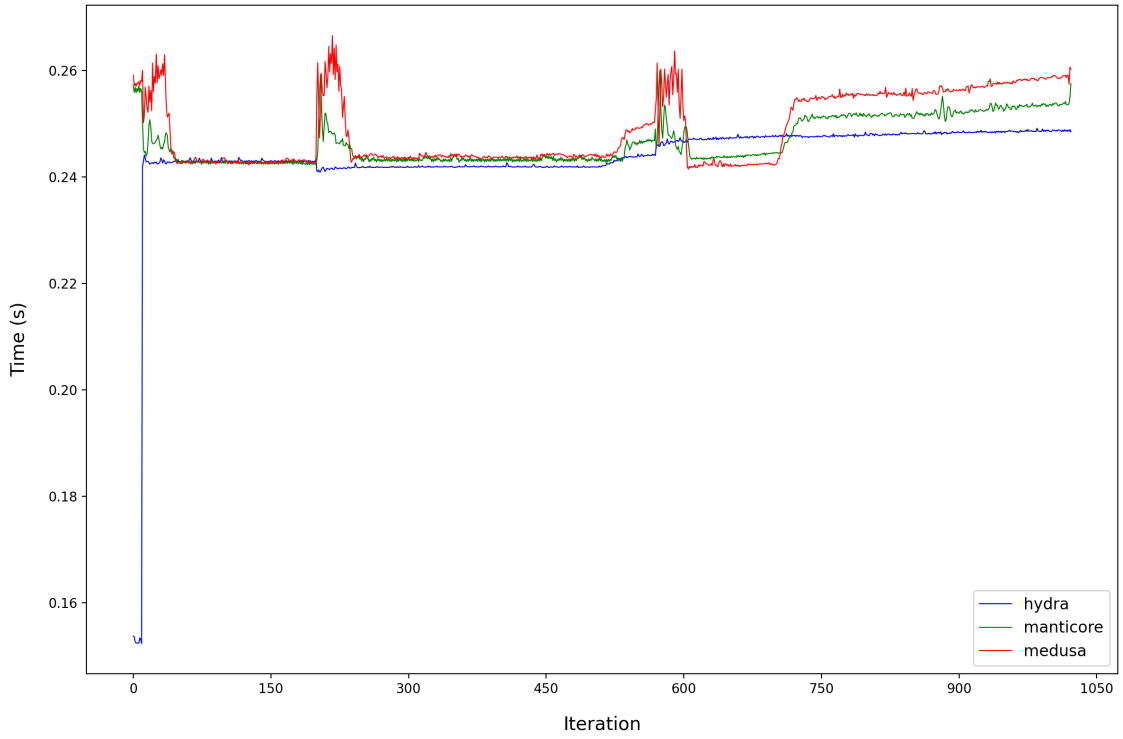

Figura 5.6: Tiempo medio de iteración del método de Stencil con balanceo de carga, sin comunicaciones.

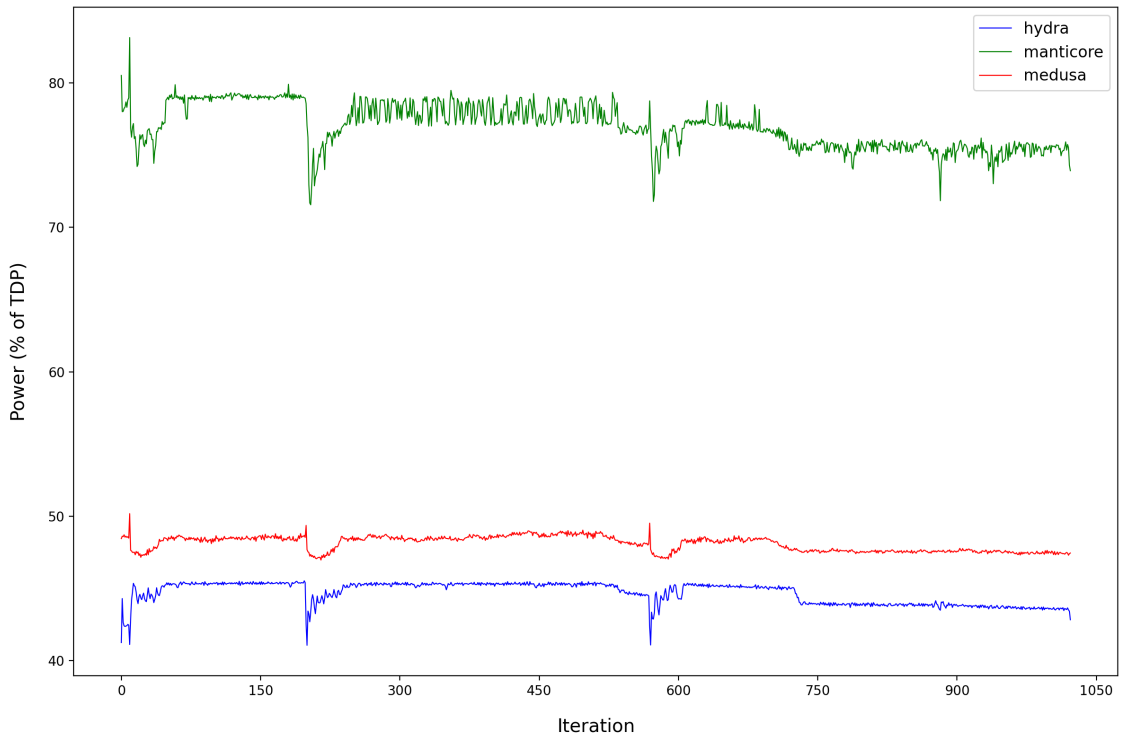

Figura 5.7: Potencia media por iteración del método de Stencil con balanceo de carga.

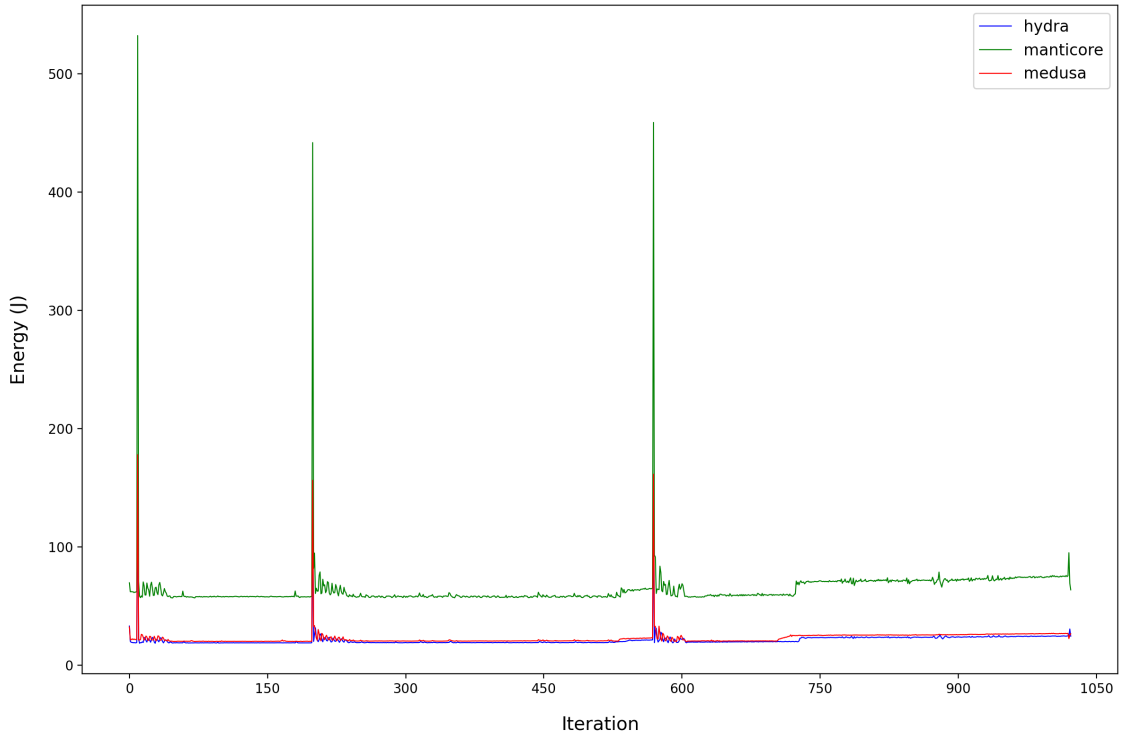

Figura 5.8: Energía consumida en Julios por iteración del método de Stencil con balanceo de carga.

que en la sección anterior, que la potencia de las máquinas, parece reducirse cuando el tiempo de iteración en estas aumenta.

Un resultado destacable es que el balanceo de carga consigue igualar los tiempos de ejecución entre máquinas a pesar de la mayor diferencia entre Hydra y las otras.

5.2.4. Comparativa con y sin balanceo de carga

Comparamos el efecto del balanceo de carga en función de las frecuencias de las máquinas. Con las frecuencias máximas y con las adaptadas.

Tabla 5.1: Comparativa con y sin balanceo, frecuencias máximas.

Frecuencias Máximas

		Sin Balanceo		Con Balanceo		Cambio	
Tamaño	**Iter**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**
32768	**512**	112	54751	113	55487	0.84 %	1.34 %
32768	**1024**	241	117120	239	116620	-0.92 %	-0.43 %
32768	**2048**	533	258398	514	250873	-3.52 %	-2.91 %
32768	**4096**	1191	576108	1080	526429	-9.31 %	-8.62 %

Esta es la tabla que ya veíamos en la Sección 4.1. Vemos cómo la tendencia a medida que aumenta la duración de las ejecuciones es que el balanceo de carga proporcione una mayor mejora en el tiempo de ejecución, con una mejora porcentual algo inferior en cuanto a consumo energético.

En la Tabla 5.2 vemos la comparativa entre usar y no usar balanceo de carga cuando ejecutamos todo el programa con las frecuencias óptimas en cuanto a consumo. En este

Tabla 5.2: Comparativa con y sin balanceo, frecuencias adaptadas.

Frecuencias Adaptadas							
		Sin Balanceo		Con Balanceo		Cambio	
Tamaño	**Iter**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**
32768	**512**	135	52127	133	52138	-1.61 %	0.02 %
32768	**1024**	297	113757	286	110892	-3.67 %	-2.52 %
32768	**2048**	667	253011	634	242948	-4.95 %	-3.98 %
32768	**4096**	1497	561255	1354	512030	-9.57 %	-8.77 %

caso, la mejora porcentual parece algo mayor cuando usamos balanceo de carga, probablemente debido a que en este caso la diferencia en el tiempo de ejecución entre máquinas es más marcada (Al permanecer hydra, la máquina con menor tiempo de iteración el 100 % de frecuencia), y el balanceo de carga compensa este efecto.

5.2.5. Comparativa con ejecutar con las frecuencias máximas

A continuación, se compara el efecto que tiene la modificación de frecuencias cuando usamos balanceo de carga y cuando no. En primer lugar, se observa con un mayor detalle el efecto que tiene en cada máquina en la ejecución de 1024 iteraciones, ya que nos permite entender mejor los resultados. Después vemos la comparativa de resultados globales con cada ejecución.

Tabla 5.3: Comparativa detallada frecuencias máximas y adaptadas, sin balanceo.

Comparativa detallada, Sin Balanceo de Carga						
	Estándar		Minimizada		Cambio	
Máquina	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**
Manticore	241	72251	296	69378	22.82 %	-3.98 %
Medusa	241	27884	296	24601	22.82 %	-11.77 %
Hydra	241	17526	296	21151	22.82 %	20.68 %

Tabla 5.4: Comparativa detallada frecuencias máximas y adaptadas, con balanceo.

Comparativa detallada, Con Balanceo de Carga						
	Estándar		Minimizada		Cambio	
Máquina	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**
Manticore	239	71624	287	66227	20.08 %	-7.54 %
Medusa	239	27288	287	23473	20.08 %	-13.98 %
Hydra	239	18201	287	21755	20.08 %	19.53 %

En las Tabla 5.3 y 5.4 vemos la comparativa detallada para la ejecución de 1024 iteraciones. Los resultados son similares para ambas ejecuciones, aunque cuando aplicamos balanceo de carga el tiempo es algo mejor, y se obtiene una mayor mejora en el consumo.

El resultado más interesante a destacar es que, aunque con Manticore y Medusa mejoramos el consumo energético, en el caso de Hydra el consumo se incrementa. Esto es debido a que Hydra sigue trabajando a la frecuencia máxima ya que ese es su punto de máxima eficiencia, por tanto, al aumentar el tiempo de ejecución con la misma potencia, el consumo es mayor.

Tabla 5.5: Comparativa frecuencias máximas y adaptadas, sin balanceo.

Sin Balanceo de Carga							
		Estándar		Minimizada		Cambio	
Tamaño	**Iter**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**
32768	**512**	112	54751	135	52127	20.41 %	-4.79 %
32768	**1024**	241	117120	297	113757	23.27 %	-2.87 %
32768	**2048**	533	258398	667	253011	25.16 %	-2.08 %
32768	**4096**	1191	576108	1497	561255	25.67 %	-2.58 %

La Tabla 5.5 muestra el efecto de adaptar las frecuencias cuando no usamos balanceo de carga. Por un lado vemos que el tiempo de ejecución aumenta, hasta un 25 % en el caso de la ejecución más larga, y que el consumo energético, aunque mejora, parece ser mayor en las ejecuciones más cortas. La mejora en la ejecución más larga es de tan sólo un 2 %.

Tabla 5.6: Comparativa frecuencias máximas y adaptadas, con balanceo.

Con Balanceo de Carga							
		Estándar		Minimizada		Cambio	
Tamaño	**Iter**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**
32768	**512**	113	55487	133	52138	17.48 %	-6.04 %
32768	**1024**	239	116620	286	110892	19.85 %	-4.91 %
32768	**2048**	514	250873	634	242948	23.31 %	-3.16 %
32768	**4096**	1080	526429	1354	512030	25.32 %	-2.74 %

En la Tabla 5.6 vemos el efecto de las frecuencias adaptadas cuando usamos balanceo de carga. Los resultados son muy similares a los de la sección anterior, aunque parece que en este caso la mejora en cuanto a consumo es algo mayor.

5.2.6. Comparativa mejor y peor caso en cuanto a consumo

Por último, se realiza una comparativa entre los que son, a priori, el mejor y peor casos en cuanto a consumo energético. El primero sería el caso estándar, en el que no aplicamos balanceo de carga y las frecuencias son máximas en todas las máquinas, a continuación tenemos el caso con balanceo de carga y las frecuencias óptimas.

En la Tabla 5.7 vemos que la aplicación modificada obtiene una mejora en el consumo que aumenta con el tiempo de ejecución, hasta alrededor de un 11 % menos de consumo en la ejecución más larga. Por otro lado, aunque el tiempo de ejecución aumenta en las tres primeras ejecuciones, vemos que en la más larga vuelve a disminuir, estando alrededor de un 14 %.

5.2.7. Utilizando sólo las máquinas óptimas

Por último, en vista de estos resultados surge la pregunta de qué cambios ocurrirían si todas las máquinas utilizadas redujeran la energía consumida al reducir las frecuencias. En el caso anterior aparece la máquina Hydra, que debido a que tiene el punto óptimo en su frecuencia máxima consume más energía cuando bajamos la frecuencia en las otras dos máquinas.

Tabla 5.7: Comparativa entre frecuencias máximas sin balanceo y adaptadas con balanceo.

Peor y mejor caso							
		Estándar sin LB		Minimizada con LB		Cambio	
Tamaño	**Iter**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**
32768	**512**	112	54751	133	52138	18.47 %	-4.77 %
32768	**1024**	241	117120	286	110892	18.75 %	-5.32 %
32768	**2048**	533	258398	634	242948	18.96 %	-5.98 %
32768	**4096**	1191	576108	1354	512030	13.64 %	-11.12 %

Tabla 5.8: Comparativa con y sin balanceo, frecuencias máximas, dos máquinas.

Frecuencias Máximas							
		Sin Balanceo		Con Balanceo		Cambio	
Tamaño	**Iter**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**
32768	**512**	122	50347	127	52587	4.50 %	4.45 %
32768	**1024**	261	107803	260	107874	-0.19 %	0.07 %
32768	**2048**	574	237264	550	227416	-4.26 %	-4.15 %
32768	**4096**	1276	526453	1236	511334	-3.14 %	-2.87 %

Tabla 5.9: Comparativa con y sin balanceo, frecuencias adaptadas, dos máquinas.

Frecuencias Adaptadas							
		Sin Balanceo		Con Balanceo		Cambio	
Tamaño	**Iter**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**
32768	**512**	147	46419	154	48393	5.08 %	4.25 %
32768	**1024**	323	100956	316	98834	-1.97 %	-2.10 %
32768	**2048**	719	222249	684	212024	-4.80 %	-4.60 %
32768	**4096**	1607	491302	1535	467741	-4.52 %	-4.80 %

Aunque es de esperar que se den este tipo de escenarios en sistemas heterogéneos, es interesante ver qué sucede si no se tiene una máquina en la que se contrarresta hasta cierto punto la mejora obtenida por las otras. El escenario experimental del que se dispone es limitado, pero una forma de probar esto es realizar ejecuciones mediante las máquinas Medusa y Manticore, únicamente. Este escenario es menos interesante de cara a emplear balanceo de carga, al tratarse únicamente de dos máquinas.

En las Tablas 5.8 y 5.9 vemos que en este caso el balanceo de carga parece tener un efecto algo menor en cuanto a disminución de tiempos y energía consumida. Este resultado es esperable debido a que sólo se usan dos máquinas, lo cual puede implicar que haya menos diferencias en los tiempos obtenidos con la distribución inicial de datos, con lo que la mejora sería menor.

En las Tablas 5.10 y 5.11 vemos el efecto de la reducción de frecuencias en ambos casos. En comparación con utilizar las tres máquinas, la mejora de energía consumida es muy estable en los 4 casos probados, siendo además considerablemente mayor. De una mejora de un 2 % pasamos a una de un 6.68 % cuando no se usa balanceo de carga, y cuando usamos balanceo de carga se pasa de un 3.29 % de mejora a un 8.53 %. Estos resultados dan una pista de que si todas las máquinas utilizadas responden bien al ajuste de frecuencias, los resultados obtenidos pueden mejorar en gran medida.

Por otra parte, vemos que en estas ejecuciones, al disponer de menor potencia de cómputo, el tiempo necesario para finalizar es mayor.

Tabla 5.10: Comparativa frecuencias máximas y adaptadas, sin balanceo, dos máquinas.

Sin Balanceo de Carga

		Estándar		Minimizada		Cambio	
Tamaño	**Iter**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**
32768	**512**	122	50347	147	46419	20.63 %	-7.80 %
32768	**1024**	261	107803	323	100956	23.79 %	-6.35 %
32768	**2048**	574	237264	719	222249	25.10 %	-6.33 %
32768	**4096**	1276	526453	1607	491302	25.94 %	-6.68 %

Tabla 5.11: Comparativa frecuencias máximas y adaptadas, con balanceo, dos máquinas.

Con Balanceo de Carga

		Estándar		Minimizada		Cambio	
Tamaño	**Iter**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**
32768	**512**	127	52587	154	48393	21.30 %	-7.98 %
32768	**1024**	260	107874	316	98834	21.58 %	-8.38 %
32768	**2048**	550	227416	684	212024	24.39 %	-6.77 %
32768	**4096**	1236	511334	1535	467741	24.14 %	-8.53 %

Tabla 5.12: Comparativa entre frecuencias máximas sin balanceo y adaptadas con balanceo, dos máquinas.

Peor y mejor caso

		Estándar sin LB		Minimizada con LB		Cambio	
Tamaño	**Iter**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**
32768	**512**	122	50347	154	48393	26.76 %	-3.88 %
32768	**1024**	261	107803	316	98834	21.35 %	-8.32 %
32768	**2048**	574	237264	684	212024	19.09 %	-10.64 %
32768	**4096**	1276	526453	1535	467741	20.25 %	-11.15 %

En la Tabla 5.12 tenemos la comparativa de mejor y peor caso que teníamos también en la sección anterior. Vemos que, pese a que sobre todo en las dos ejecuciones intermedias vemos una mejora respecto a lo anterior, la mejora en cuanto a energía en la ejecución más larga es muy similar a cuando utilizamos tres máquinas, con una diferencia de tiempo peor. Estos resultados pueden deberse al menor efecto del balanceo de carga en este caso, pese a que la mejora en cuanto a energía al adaptar las frecuencias sea mayor.

Por último, como mera curiosidad, tenemos una comparativa entre el que sería el peor caso analizado en cuanto a consumo, no utilizar balanceo de carga ni modificar las frecuencias, utilizando todas las máquinas, y el mejor, con frecuencias adaptadas, balanceo de carga y las dos mejores máquinas. Vemos que las mejoras totales obtenidas son considerablemente mayores en todos los casos, y que en la ejecución más larga se llega casi a un 19 % de ahorro energético.

Esta situación es muy concreta y depende mucho del escenario experimental que se esté usando, pero puede indicar que en función de lo que se quiera priorizar, tiempo

Tabla 5.13: Comparativa entre frecuencias máximas sin balanceo y adaptadas con balanceo, dos máquinas frente a tres.

Peor caso con tres máquinas, y mejor caso con dos

		Estándar sin LB		Minimizada con LB		Cambio	
Tamaño	**Iter**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**	**Tiempo (s)**	**Energía (J)**
32768	**512**	112	54751	154	48393	37.19 %	-11.61 %
32768	**1024**	241	117120	316	98834	31.27 %	-15.61 %
32768	**2048**	533	258398	684	212024	28.36 %	-17.95 %
32768	**4096**	1191	576108	1535	467741	28.82 %	-18.81 %

de ejecución o consumo energético, puede llegar a ser conveniente hacer un análisis en detalle del sistema utilizado y decidir si utilizar o no ciertas máquinas.

CAPÍTULO 6
Conclusiones y Trabajo Futuro

6.1 Conclusiones

Este proyecto parte de la pregunta:

> **¿Es posible elaborar una solución de balanceo de carga orientada a reducir el consumo energético de un programa?**

Los objetivos planteados, y el resultado de los mismos son:

- **Determinar la mejor forma de obtener medidas sobre el consumo energético de una aplicación vía software.**

  Se han encontrado varias herramientas que ofrecen este tipo de medidas. Por lo general, cada plataforma tiene una herramienta específica para tomar medidas. Se eligen las interfaces RAPL de Intel debido a las máquinas disponibles para el proyecto y a la funcionalidad que ofrecen.

- **Documentar cómo se comportan las aplicaciones empleadas en cuanto a consumo energético.**

  Se realiza un estudio centrado principalmente en la aplicación de Stencil. Los resultados muestran que los marcadores de consumo energético utilizados no varían significativamente pese a grandes variaciones de la carga computacional, que si tienen un efecto en el tiempo de ejecución. Además, al tratarse de máquinas con grandes diferencias en el hardware, los datos de consumo no parecen ser directamente comparables.

  Por otro lado, estos resultados permiten detectar ciertas ineficiencias, en este caso relacionadas por un lado con las políticas de consumo energético de Linux y con el funcionamiento de la librería MPI. Estas implementaciones son óptimas cuando queremos que las aplicaciones se ejecuten lo más rápido posible, pero no cuando intentamos minimizar el consumo.

  Estos resultados llevan a estudiar el consumo de las máquinas al variar la frecuencia de los procesadores. Esta experimentación permite detectar que en algunas máquinas existen puntos óptimos en cuanto a consumo que no son las frecuencias máximas.

- **Documentar el efecto del balanceo de carga tradicional, orientado a reducir el tiempo de ejecución, en el consumo energético.**

Como parte del estudio anterior, vemos el efecto que tiene en el consumo aplicar balanceo de carga, en este caso mediante la solución de balanceo de carga implementada en la librería HitMap. Los resultados muestran, por un lado, que las redistribuciones de carga suponen picos de consumo, pero a pesar de esto la reducción en el tiempo de ejecución de las aplicaciones conlleva reducciones en el consumo.

- **Determinar si es factible balancear la carga en base a marcadores de consumo energético.**

  Debido al poco efecto que tienen las redistribuciones de carga en los marcadores de consumo energético, parece complicado construir una solución de balanceo de carga en torno a estos, al contrario que lo que sucedería con otros parámetros como el tiempo de ejecución de cada iteración.

A pesar de que no parece haber una forma clara de balancear la carga en torno al consumo de las máquinas por factores como las diferencias en el hardware, que hacen complicado comparar los datos obtenidos con cada máquina, o el hecho de que variar la carga no parece afectar en gran medida al consumo, los resultados obtenidos sí indican una forma de reducir el consumo para la misma cantidad de trabajo.

La solución propuesta es una prueba de concepto basada en la experimentación con las máquinas disponibles. Bajando las frecuencias hasta los puntos óptimos detectados obtenemos pequeñas mejoras de un 2 y un 3 % en función de si aplicamos o no balanceo de carga, cuando utilizamos las tres máquinas disponibles. Si utilizamos los resultados de la experimentación para excluir la máquina en la que no podemos obtener una mejora variando las frecuencias, las mejoras obtenidas suben hasta un 6.5 y 8.5 %. En la comparativa entre el peor caso absoluto, y la mejor configuración que podemos obtener a partir de los resultados del análisis previo, llegamos a una mejora de casi un 19 % por la misma cantidad de trabajo, con una penalización de un 29 % en el tiempo de ejecución.

Aunque esta aproximación a la reducción del consumo es simple y no tiene en cuenta factores como las diferencias en el trabajo que se hace a lo largo de una ejecución (Por ejemplo fases de cómputo, fases de comunicación, etc.), o diferencias en el tiempo de ejecución en cada máquina, parece una forma consistente de obtener mejoras, ya que al menos en las máquinas disponibles en este caso tenemos puntos claros en los que la máquina puede realizar la misma cantidad de trabajo empleando menos energía.

6.2 Trabajo Futuro

6.2.1. Automatizar la detección de puntos óptimos

La fase de experimentación previa realizada en este proyecto ha permitido identificar puntos de máxima eficiencia en las máquinas utilizadas, y se ha comprobado mediante la aplicación Stencil que si se realiza la ejecución con las máquinas en estos puntos óptimos podemos reducir la energía total consumida para realizar el trabajo.

Es una forma relativamente sencilla de reducir la energía consumida en una ejecución en casos en los que el tiempo de ejecución no sea crítico. En vista de los resultados obtenidos, parece que sería interesante diseñar un sistema que permita realizar de forma automática un análisis previo de las máquinas utilizadas, y detectar estos puntos óptimos en caso de que existan.

Una de las principales dificultades a la hora de hacer esto sería el determinar la mejor forma de hallar el perfil de consumo de las máquinas. De acuerdo con la experimentación realizada, hemos visto una máquina que tiene un comportamiento prácticamente

idéntico en dos aplicaciones distintas, otra en la que el perfil de consumo varía en mayor medida, aunque mantiene el mismo punto óptimo, y una última en la que los resultados son muy distintos.

Por tanto, por un lado sería necesario ampliar la experimentación a un mayor número de máquinas distintas, por otro lado, parece interesante determinar el por qué de estas diferencias al utilizar distintas aplicaciones, y si los puntos óptimos tienden a estar en las mismas posiciones independientemente de otras diferencias en el perfil de consumo.

Bibliografía

[1] Clarke, David & Lastovetsky, Alexey & Rychkov, Vladimir. (2011). Dynamic Load Balancing of Parallel Computational Iterative Routines on Highly Heterogeneous HPC Platforms.. Parallel Processing Letters. 21. 195-217. 10.1142/S0129626411000163.

[2] S. F. Hummel, E. Schonberg and L. E. Flynn, "Factoring: a practical and robust method for scheduling parallel loops," Supercomputing '91:Proceedings of the 1991 ACM/IEEE Conference on Supercomputing, 1991, pp. 610-632, doi: 10.1145/125826.126137.

[3] I. Banicescu and V. Velusamy, "Load balancing highly irregular computations with the adaptive factoring," Proceedings 16th International Parallel and Distributed Processing Symposium, 2002, pp. 12 pp-, doi: 10.1109/IPDPS.2002.1015661.

[4] Cariño, Ricolindo & Banicescu, Ioana. (2008). Dynamic load balancing with adaptive factoring methods in scientific applications. The Journal of Supercomputing. 44. 41-63. 10.1007/s11227-007-0148-y.

[5] Susan Flynn Hummel, Jeanette Schmidt, R. N. Uma, and Joel Wein. 1996. Load-sharing in heterogeneous systems via weighted factoring. In Proceedings of the eighth annual ACM symposium on Parallel Algorithms and Architectures (SPAA '96). Association for Computing Machinery, New York, NY, USA, 318–328. DOI:https://doi.org/10.1145/237502.237576

[6] S. H. Russ, I. Banicescu, S. Ghafoor, B. Janapareddi, J. Robinson and Rong Lu, "Hectiling: an integration of fine and coarse-grained load-balancing strategies," Proceedings. The Seventh International Symposium on High Performance Distributed Computing (Cat. No.98TB100244), 1998, pp. 106-113, doi: 10.1109/HPDC.1998.709962.

[7] E. Haddad, "Dynamic optimization of load distribution in heterogeneous systems," Proceedings Heterogeneous Computing Workshop, 1994, pp. 29-34, doi: 10.1109/HCW.1994.324965.

[8] Samfass, Philipp & Weinzierl, Tobias & Charrier, Dominic Etienne & Bader, Michael. (2020). Lightweight task offloading exploiting MPI wait times for parallel adaptive mesh refinement. Concurrency and Computation: Practice and Experience. 32. 10.1002/cpe.5916.

[9] I. Banicescu and S. F. Hummel, "Balancing Processor Loads and Exploiting Data Locality in N-Body Simulations," Supercomputing '95:Proceedings of the 1995 ACM/IEEE Conference on Supercomputing, 1995, pp. 43-43, doi: 10.1109/SUPERC.1995.241497.

[10] Rus, P., Stok, B., & Mole, N. (2003). Parallel computing with load balancing on heterogeneous distributed systems. Advances in Engineering Software, 34, 185-201.

[11] C. D. Polychronopoulos and D. J. Kuck, "Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers," in IEEE Transactions on Computers, vol. C-36, no. 12, pp. 1425-1439, Dec. 1987, doi: 10.1109/TC.1987.5009495.

[12] C. Du, S. Ghosh, S. Shankar and Xian-He Sun, "Runtime system for autonomic rescheduling of MPI programs," International Conference on Parallel Processing, 2004. ICPP 2004., 2004, pp. 4-11 vol.1, doi: 10.1109/ICPP.2004.1327898.

[13] E. Haddad, "Dynamic optimization of load distribution in heterogeneous systems," Proceedings Heterogeneous Computing Workshop, 1994, pp. 29-34, doi: 10.1109/HCW.1994.324965.

[14] Galindo, J. (2008). Dynamic Load Balancing on Dedicated Heterogeneous Systems . In Recent Advances in Parallel Virtual Machine and Message Passing Interface (pp. 64–74). Springer Berlin Heidelberg.

[15] María Sánchez Girón. Distribución dinámica de carga y redistribuciones de datos en aplicaciones paralelas. UVaDOC.

[16] Top500. Lista Green500. https://www.top500.org/lists/green500/2020/11/. Último acceso Junio de 2021.

[17] Top500. Top 500 main page. https://www.top500.org. Último acceso Junio de 2021.

[18] 01. Powertop. https://01.org/powertop. Último acceso Junio de 2021.

[19] Intel User Guide. C-State. Último acceso Junio de 2021. https://software.intel.com/content/www/us/en/develop/documentation/energy-analysis-user-guide/top/energy-analysis-metrics-reference/c-state.html. Último acceso Junio de 2021.

[20] 01. RAPL. https://01.org/blogs/2014/running-average-power-limit-%E2%80%93-rapl. Último acceso Junio de 2021.

[21] Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3, Chapter 14, Section 10

[22] UEFI. ACPI. https://uefi.org/acpi. Último acceso Marzo de 2022.

[23] Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3, Chapter 14, Section 2

[24] Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 4, Chapter 2

[25] GitHub. Turbostat.
https://github.com/torvalds/linux/tree/master/tools/power/x86/turbostat.
Último acceso Junio de 2021.

[26] Texas Instruments. INA3221. https://www.ti.com/product/INA3221. Último acceso Junio de 2021.

[27] Nvidia Jetson Linux Developer Guide. Software-Based Power Consumption Modeling. https://docs.nvidia.com/jetson/l4t/#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/power_management_jetson_xavier.html#wwpID0E0AG0HA.
Último acceso Junio de 2021.

[28] Nvidia Developer. NVML. https://developer.nvidia.com/nvidia-management-library-nvml. Último acceso Junio de 2021.

[29] Nvidia Developer. Nvidia SMI. https://developer.nvidia.com/nvidia-system-management-interface. Último acceso Junio de 2021.

[30] A. Gonzalez-Escribano, Y. Torres, J. Fresno and D. R. Llanos, "An Extensible System for Multilevel Automatic Data Partition and Mapping," in IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 5, pp. 1145-1154, May 2014, doi: 10.1109/TPDS.2013.83.

[31] Grupo Trasgo. Página principal. https://trasgo.infor.uva.es/. Último acceso Junio de 2021.

[32] Intel. Soporte, TDP. https://www.intel.com/content/www/us/en/support/articles/000055611/processors.html. Último acceso Junio de 2021.

[33] Linux Kernel. Cpufreq user guide. https://www.kernel.org/doc/Documentation/cpufreq/user-guide.txt

[34] Linux Kernel. Cpufreq governors. https://www.kernel.org/doc/Documentation/cpufreq/governors.txt

[35] OpenMP. OpenMP main page. https://www.openmp.org/. Último acceso Marzo de 2021.

[36] MPI. MPI Forum. https://www.mpi-forum.org/. Último acceso Marzo de 2021.