Técnicas de ahorro de energía (Green Computing)

Christian Guzman Ruiz

Resumen— Los computadores ofrecen cada vez más prestaciones en términos de potencia de cómputo, rapidez, almacenamiento de datos, etcétera. Éste incremento de la potencia de cómputo tiene como consecuencia y limitación un incremento significativo del consumo energético. Para resolver este problema, han aparecido recientemente diversos proyectos que buscan un mayor control y una mejor utilización de la energía en los computadores, sobretodo en el ámbito de los supercomputadores. En este trabajo queremos investigar las diversas técnicas y herramientas desarrolladas para el ahorro de consumo energético. Se identifican y catalogan las herramientas encontradas actualmente, y se describen cada una de ellas desde el enfoque de la capacidad de medir y gestionar el consumo energético. Además se muestran y analizan los resultados obtenidos para cada una de ellas, para desarrollar una evaluación crítica de cada una de las herramientas consideradas y una reflexión general sobre los retos aun abiertos en este ámbito.

Palabras clave— Herramienta, consumo energético, librería, configuración

Abstract— Computers offer more and more features in terms of computing power, speed, data storage, and so on. This increase in computing power has as a consequence and limitation a significant increase in energy consumption. To solve this problem, several projects have recently appeared that seek greater control and better use of energy in computers, especially in the field of supercomputers. In this work we want to investigate the different techniques and tools developed to save energy consumption. The currently found tools are identified and cataloged, and each of them is described from the perspective of the capacity to measure and manage energy consumption. In addition, the results obtained for each of them are shown and analyzed, to develop a critical evaluation of each of the tools considered and a general reflection on the challenges still open in this area.

Index Terms—Tool, energy consumption, library, configuration

1 Introducción

La computación de altas prestaciones cada vez apunta a realizar más cálculos por segundo [1]. El objetivo actual es llegar al exascale [2], es decir, sistemas de computación capaces de realizar un mínimo de un exaFlops (10^{18} cálculos por segundo).

Para realizar estos sistemas de alto rendimiento es necesario utilizar una gran cantidad de ordenadores organizados en forma de clúster. Un clúster es un conjunto de ordenadores unidos entre sí por una red que se comportan como si fuesen una única computadora. En un clúster cada ordenador ejecuta en paralelo una parte del código de una aplicación, o bien una copia del mismo código sobre diferentes conjuntos de datos. En la actualidad, puede realizarse trabajo en paralelo también en un único ordenador si este dispone de varias unidades o núcleos de procesamiento. Las unidades de procesamiento utilizadas pueden corresponder al procesador (CPU) o a un acelerador, como por ejemplo la tarjeta gráfica (GPU).

- *E-mail de contacto: christian.guzman@e-campus.uab.cat*
- *Mención realizada: Ingeniería de Computadores*
- *Trabajo tutorizado por: Eduardo Cesar Galobardes (Departamento CAOS)*
- *Curso 2017/18.*

Además de aumentar la capacidad de cómputo, utilizar múltiples ordenadores también aumenta el consumo energético. El clúster con mayor capacidad de cómputo en la actualidad consume 15.3 MW, alcanzando una capacidad de cómputo de 93 TeraFlops (10^{12} cálculos por segundo) [3]. Se calcula que para alcanzar el exascale sería necesario un consumo energético de 200 MW, el cual es demasiado costoso de mantener. El objetivo actual de consumo energético para el exascale es de 20-40 MW [4]. Para lograr este objetivo, es necesario desarrollar tanto *hardware* como *software* muy eficiente.

El principal causante del consumo energético es la frecuencia de reloj del procesador o la GPU en cuestión. Una manera de reducir el consumo energético es reducir esta frecuencia durante la ejecución de operaciones de traspaso de memoria. Existen proyectos enfocados a este cometido como READEX (*Runtime Exploitation of Application Dynamism for Energy-efficient*) [5] y GEOPM (*Global Extensive Open Power Management*) [6]. No obstante, también existen otras técnicas como el balanceo de carga en clúster [7], o la suspensión de nodos dinámica [8].

El exuberante consumo energético es un tema aún no resuelto y con mucho trabajo por delante. Por esta razón

el objetivo de este trabajo es investigar sobre los métodos disponibles, centrándonos en los métodos basados en variar la frecuencia de reloj, para mostrar los problemas existentes y sentar las bases para plantear posibles nuevas soluciones o mejoras de las existentes.

Para realizar este objetivo, este informe incluye descripciones de diversas herramientas actuales enfocadas en el consumo energético. Estas descripciones están enfocadas en la funcionalidad de monitorización y regulación del consumo energético. Además se incluyen resultados obtenidos tras probar la funcionalidad de las herramientas. Posteriormente se analiza la utilidad de las herramientas mediante los resultados obtenidos, tanto en monitorización y ahorro energético, como dificultad de configuración y uso.

La estructura de este informe es la siguiente: La sección 2 detalla la metodología utilizada para realizar una correcta evaluación de las herramientas definidas en la sección anterior. La sección 3 introduce el código utilizado para analizar las herramientas. La sección 4 define los tipos de proyectos y herramientas actuales. Las secciones 5, 6 y 7 describen más detalladamente las herramientas y sus opciones de análisis y optimización de la energía. La sección 8 muestra los resultados obtenidos de las distintas herramientas y configuraciones estudiadas. Finalmente la sección 9 contiene las conclusiones finales.

2 METODOLOGÍA

Para lograr el objetivo principal, el trabajo ha seguido una metodología de investigación [9]. Las tareas correspondientes a la obtención de los datos se han planteado ordenadas según su dificultad de ejecución, comenzando por la más sencilla, para así poder tener más flexibilidad en la planificación en caso de no obtener el funcionamiento o los resultados en el tiempo esperado.

Las tareas que componen la metodología son las siguientes:

1. Búsqueda de técnicas, trabajos y herramientas que compartan nuestra finalidad. Esta tarea se ha realizado tanto al inicio como durante todo el proyecto, ya que al inicio se seleccionó probar las más relevantes, pero al configurarlas, algunas no eran válidas para nuestros ordenadores y se vio conveniente seguir investigando.

2. Configurar un código base en distintas versiones de lenguajes paralelos (OpenMP [10], MPI [11], CUDA [12]) sobre el que realizar las mediciones. Cada uno de estos lenguajes está enfocado a explotar el paralelismo en diferentes niveles (OpenMP para paralelismo del procesador, CUDA para la GPU y MPI para múltiples nodos), lo cual nos permite estudiar el problema en cada uno de dichos niveles. Además se seleccionó un código base con suficiente complejidad para realizar las pruebas.

3. Configurar y utilizar las herramientas seleccionadas en un único ordenador con varias unidades de procesamiento. El objetivo de esta tarea es comprobar de manera sencilla el funcionamiento de las mismas en un entorno en el que los privilegios del usuario no representaran una limitación. Primero se trabajó con herramientas de escritorio y a continuación con herramientas basadas en librerías.

4. Probar las herramientas viables en un sistema HPC (*High Performance Computing*) [13] como READEX.

5. Finalmente, analizar los resultados obtenidos y plantear las conclusiones de este estudio.

3 CÓDIGO

Para analizar las distintas herramientas, hemos partido del código base mencionado en la metodología.

Cabe mencionar que la herramienta READEX actualmente no funciona correctamente con cualquier código, y para probar sus funcionalidades se utilizó un código distinto proporcionado por los desarrolladores de READEX.

A continuación se explican ambos códigos.

3.1 Código base

El código base es una adaptación del código de una red neuronal [14] realizada por John Bullinaria [15]. La red neuronal se basa en a partir de unas imágenes detalladas de letras del abecedario, identificar posteriormente imágenes no tan detalladas de estas letras. Esta adaptación se divide en dos partes:

- Fase de entrenamiento: En esta fase se reconocen el conjunto de imágenes detalladas o patrones y se prepara la red neuronal para identificar imágenes menos detalladas. La mayoría de tiempo de ejecución de la aplicación recae en esta fase.

- Fase de trabajo: En esta fase la red neuronal recibe varias imágenes similares a los patrones y las identifica. Esta fase requiere poco tiempo de ejecución de la aplicación.

Sobre este código se configuraron versiones en los lenguajes paralelos OpenMP, MPI y CUDA. En estas versiones se paraleliza y distribuye el trabajo de la fase de entrenamiento entre los distintos núcleos de cómputo. En el caso de OpenMP entre los núcleos de la CPU, en el caso de MPI, entre los computadores del clúster, y en el caso de CUDA, los núcleos de la GPU.

Estas versiones reducen el tiempo de cómputo sobre la versión original. Mediante el uso de las herramientas explicadas en las secciones posteriores, se analizará también su efecto en el consumo energético.

3.2 Código de ejemplo READEX

El código utilizado para analizar READEX es una versión simple y paralela de dinámica molecular [16] llamada miniMD y desarrollada en el proyecto Mantevo [17] por el Laboratorio Nacional de Sandia [18]. Esta versión además está instrumentada [19] en varias partes para facilitar el correcto funcionamiento de READEX.

El código incluye el lenguaje OpenMP y MPI, paralelizando tanto a nivel de núcleos del procesador como de nodos disponibles.

4 Tipos de herramientas

Después de hacer una revisión de las herramientas existentes, relacionadas a la medición y consumo de energía en computadores, hemos creído conveniente clasificarlas de la siguiente manera:

- Herramientas de escritorio: Son herramientas de fácil instalación y uso. Pueden servir para establecer una indicación inicial, a veces muy inexacta, del consumo energético de nuestra aplicación. Las herramientas analizadas en esta categoría son Powertop [20], Powerstat [21] y Likwid [22].

- Herramientas basadas en librerías: Permiten medir el consumo energético de la aplicación y además permiten realizar diversas configuraciones sobre parámetros *hardware* que podrían conseguir reducir el consumo energético. Los ejemplos analizados son las librerías *Mammut* [23] y Nvidia Management Library (NVML) [24].

- Herramientas de sintonización automática: Son las herramientas más complejas, pero también las más prometedoras en el ámbito de los supercomputadores. Contienen las características de las herramientas anteriores, pero estas herramientas buscan obtener la configuración óptima de los parámetros *hardware* de manera automática. El ejemplo analizado en esta categoría es READEX.

Existen más herramientas no analizadas en este trabajo como la herramienta de sintonización GEOPM, o la herramienta basada en librerías *libadapt* [25]. Ambas necesitan un largo y complicado trabajo de configuración. Hay que tener en cuenta que la herramienta *libadapt* es un proyecto anterior a READEX desarrollado por el mismo centro [26], y que requiere *software* actualmente obsoleto como es el caso de VampirTrace [27]. La herramienta GEOPM no se analizó debido a que la configuración de esta herramienta y READEX requiere demasiada carga de trabajo, y ambas herramientas son similares en metodología usada. Se escogió READEX al disponer de más documentación.

5 Herramientas para escritorio

5.1 Powertop

Powertop es una herramienta disponible en sistemas Linux [28] para habilitar varios modos de ahorro de energía en el espacio de usuarios, el núcleo y el *hardware*. Es posible monitorizar los procesos del sistema, lo que permite identificar aplicaciones de elevado consumo energético. La herramienta obtiene los datos a partir del consumo energético de la batería en ordenadores portátiles, funcionando solamente para estos ordenadores con batería extraíble. Por tanto esta herramienta solo es para un uso particular de usuario. En la figura 1 puede verse un ejemplo del uso de Powertop.

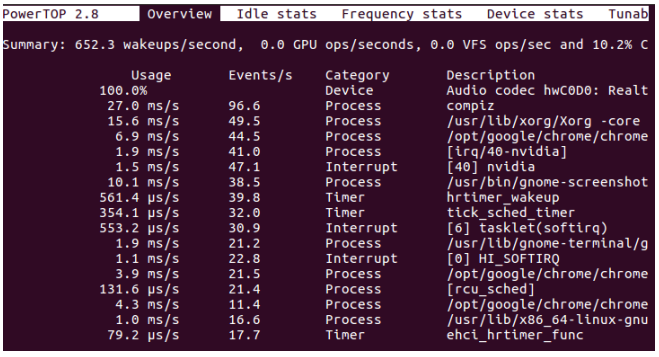

Fig. 1. Datos obtenidos en Powertop en un ordenador no portátil

5.1 Powerstat

Powerstat es un monitor de consumo energético para Ubuntu [29] destinado a portátiles, similar a Powertop. En la salida muestra además información sobre procesos, paginación, etcétera, de manera similar a la utilidad vmstat [30]. Powerstat recoge varias muestras cada 10 segundos, calcula el promedio de los datos de cada muestra y presenta el resultado por pantalla. Se basa en la descarga de la batería del portátil, funcionando solamente para estos ordenadores con batería extraíble. Como Powertop, esta herramienta está destinada solo a uso particular. En la figura 2 puede verse un ejemplo del uso de Powerstat.

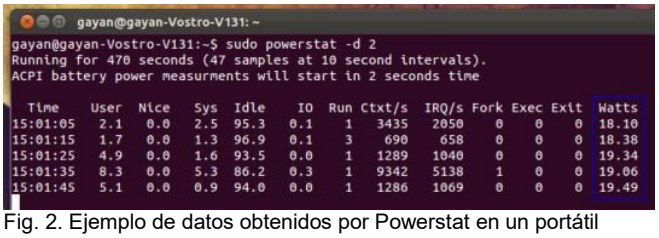

Fig. 2. Ejemplo de datos obtenidos por Powerstat en un portátil

5.1 Likwid

Likwid es una herramienta fácil de instalar y utilizar mediante su interfaz por línea de comandos. Está destinada a programadores orientados al rendimiento de las aplicaciones, con módulos de monitorización y sintonización como la topología de la caché o *threads* [31], control de la frecuencia de la CPU o lector del consumo energético.

En especial Likwid ofrece un comando para medir el consumo energético durante el tiempo deseado. Obtiene los datos de consumo energético mediante el registro MSR (*model-specific registers*) [32] el cual tiene que estar activo. En la figura 3 puede verse un ejemplo de su uso.

```
guz@guz-System-Product-Name:~$ sudo likwid-powermeter -s 4
--------------------------------------------------------------------------------
CPU name:       Intel Core IvyBridge processor
CPU clock:      3.41 GHz
--------------------------------------------------------------------------------
Measure on sockets: 0
Runtime: 4 second
--------------------------------------------------------------------------------
Socket 0 (Measured on CPU 0)
Domain: PKG
Energy consumed: 27.7886 Joules
Power consumed: 6.94714 Watts
Domain: PP0
Energy consumed: 6.23524 Joules
Power consumed: 1.55881 Watts
Domain: PP1
Energy consumed: 0 Joules
Power consumed: 0 Watts
```

Fig. 3. Consumo energético obtenido mediante Likwid

Es una buena herramienta, el problema es que hay que indicar el tiempo de medición, y para una lectura correcta habría que ejecutarla al mismo tiempo que nuestra aplicación indicando el tiempo estimado de ejecución. Por esta razón podríamos obtener resultados imprecisos.

6 HERRAMIENTAS BASADAS EN LIBRERÍAS

6.1 *Mammut*

Mammut es un *framework* [33] de código abierto orientado a objetos para sistemas Linux locales y remotos. Mediante el uso de su librería en nuestro código permite tanto monitorizar la energía consumida como modificar características *hardware* (frecuencia del reloj de la CPU o limitar el uso de la CPU) en partes específicas del código.

Se instaló la última versión de *Mammut*, la 1.1. La librería contiene códigos de ejemplo para comprobar y utilizar sus funciones. Se comprobó que mediante los registros MSR, la librería proporcionaba información sobre el consumo energético, topología y frecuencia de la CPU. Además permite sintonizar diversos parámetros relativos al consumo energético como la frecuencia de la CPU o el estado de sus núcleos.

Se probó en un ordenador la medición de energía mediante *Mammut*, la energía consumida al maximizar el uso de los núcleos de la CPU (mediante una función de la API), y el consumo al cambiar la frecuencia de la CPU. Estos pasos fueron sencillos de realizar integrando las funciones de los ejemplos en nuestro código, excepto por el cambio de frecuencia.

Cabe mencionar que para su ejecución en un clúster es necesario permisos de administrador del sistema, por tanto los usuarios del sistema no pueden ejecutar *Mammut*.

6.1.1 Variación de la frecuencia de la CPU

Para cambiar la frecuencia de la CPU primero es necesario comprobar los modos de funcionamiento disponibles de la CPU (*governors*). Por ejemplo, un procesador Intel Core i5-3570K tiene dos modos de funcionamiento, "powersave" y "performance", con frecuencias disponibles de 1.6GHz y 3.8Ghz. La frecuencia más alta y el modo performance es el configurado por defecto.

Para cambiar de governor, *Mammut* necesita activar el controlador acpi-cpufreq [34] según su documentación [35], aumentando la dificultad de configuración. Una vez activado, es posible modificar el código y ajustar la frecuencia de la CPU. No obstante, según la arquitectura, es necesario realizar ajustes de configuración adicionales, que pueden estar muy alejados al conocimiento de un usuario estándar (reconfigurar el *kernel* [36] y modificar la BIOS [37]).

Es posible identificar la necesidad de realizar ajustes de configuración adicionales mediante las herramientas cpufreqd [38] y cpupower [39], las cuales se basan en el controlador acpi-cpufreq y son sencillas de configurar y utilizar.

Ambas aplicaciones muestran las frecuencias disponibles y la frecuencia actual. En el caso de cpufreqd en la figura 4 es posible ver que la CPU puede variar la frecuencia en un límite de 1.6GHz a 3.8GHz, pero que el *governor* "powersave" no permite variar la frecuencia de su configuración por defecto a 3.8GHz.

```
guz@guz-System-Product-Name:~$ cpufreq-info
cpufrequtils 008: cpufreq-info (C) Dominik Brodowski 2004-2009
Report errors and bugs to cpufreq@vger.kernel.org, please.
analyzing CPU 0:
  driver: intel_pstate
  CPUs which run at the same hardware frequency: 0
  CPUs which need to have their frequency coordinated by software: 0
  maximum transition latency: 0.97 ms.
  hardware limits: 1.60 GHz - 3.80 GHz
  available cpufreq governors: performance, powersave
  current policy: frequency should be within 3.80 GHz and 3.80 GHz.
                  The governor "powersave" may decide which speed to use
                  within this range.
  current CPU frequency is 2.62 GHz.
```

Fig. 4. Información de la CPU obtenida mediante cpufreq

En el caso de cpupower, en la figura 5 se puede observar que el estado de impulso (*boost*) está activo, el cual ofrece una frecuencia cercana a la máxima. Este estado se activa al ejecutar el código no permitiendo bajar la frecuencia.

```
guz@guz-System-Product-Name:~/cap_practicas$ sudo cpupower frequency-info
analyzing CPU 0:
  driver: intel_pstate
  CPUs which run at the same hardware frequency: 0
  CPUs which need to have their frequency coordinated by software: 0
  maximum transition latency: 0.97 ms.
  hardware limits: 1.60 GHz - 3.80 GHz
  available cpufreq governors: performance, powersave
  current policy: frequency should be within 1.60 GHz and 3.80 GHz.
                  The governor "powersave" may decide which speed to use
                  within this range.
  current CPU frequency is 1.63 GHz (asserted by call to hardware).
  boost state support:
    Supported: yes
    Active: yes
    3600 MHz max turbo 4 active cores
    3700 MHz max turbo 3 active cores
    3800 MHz max turbo 2 active cores
    3800 MHz max turbo 1 active cores
```

Fig. 5. Información de la CPU obtenida mediante cpupower

6.2 NVML

Nvidia Management Library (NVML) es una API [40] para monitorizar y administrar diversos estados de los dispositivos GPU de Nvidia. Esta herramienta viene instalada por defecto en los controladores de Nvidia y es sencilla de utilizar. En este trabajo se investigó sobre la versión 384.81.

Por medio de línea de comandos (utilizando el comando "nvidia-smi"), esta librería permite consultar diversas características de la GPU como temperatura, velocidad del ventilador, porcentaje de uso, frecuencias de relojes y administración de energía. Está dirigida a los productos de Nvidia TeslaTM, GRIDTM, QuadroTM y Titan X, aunque también tiene un soporte limitado para otras GPUS de Nvidia.

Estas otras GPU, como por ejemplo el modelo GTX 680, no disponen por ejemplo de la característica de administración de energía, la cual muestra datos del consumo energético y permite establecer un límite de consumo máximo.

En cambio, en una GPU de la arquitectura Tesla como la GTX1080 sí que dispone de esta utilidad. Además, teniendo permisos de administrador es posible regular la frecuencia de reloj de la GPU para disminuir el consumo según la aplicación. En la figura 6 puede verse un ejemplo de la información obtenida con NVML sobre las GPU mencionadas.

```
cguzman@aolin21:~$ nvidia-smi
Thu Jan  4 15:24:09 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.81                 Driver Version: 384.
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 680     Off  | 00000000:02:00.0 N/A |                  N/A |
| 39%   40C    P0    N/A /  N/A |      0MiB /  1996MiB |     N/A      Default |
+-------------------------------+----------------------+----------------------+
|   1  GeForce GTX 108...  Off  | 00000000:03:00.0 Off |                  N/A |
|  0%   41C    P5    28W / 250W |      0MiB / 11172MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

Fig. 6.Información general de la GPU obtenida por NVML

7 HERRAMIENTAS DE SINTONIZACIÓN AUTOMÁTICA: READEX

Si bien es posible utilizar herramientas como *Mammut* y NVML para implementar técnicas con el fin de reducir el consumo de energía de aplicaciones individuales, el ajuste manual requerido en estos sistemas es una tarea compleja y, en consecuencia, a menudo descuidada.

Por tanto, el siguiente paso lógico consiste en intentar desarrollar herramientas automáticas de ajuste de consumo. En este sentido, posiblemente los dos esfuerzos más relevantes en la actualidad son GEOPM y READEX. Este trabajo se enfoca en READEX por lo mencionado anteriormente, concretamente en su versión alfa.

El proyecto READEX busca automatizar este ajuste (o sintonización) mediante el desarrollo de diversas herramientas. Estas herramientas combinan la metodología de los sistemas integrados y del campo de HPC. En la figura 7 se aprecia un esquema de los componentes de READEX y sus relaciones.

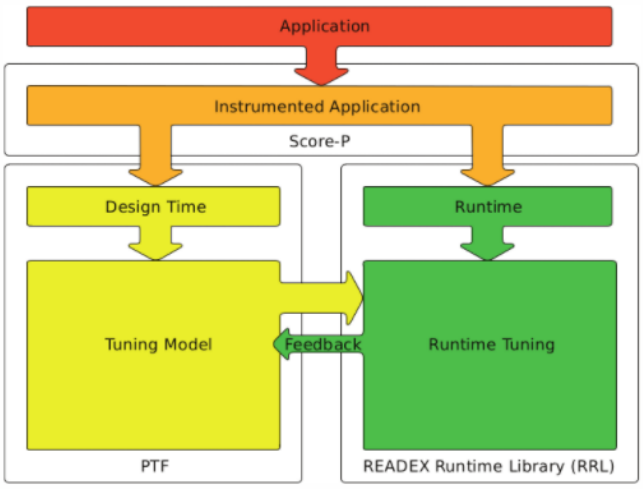

Fig. 7 Esquema de los componentes de READEX y sus relaciones

El primer componente en el que se divide READEX se basa en el *software* Score-P, que a partir de la aplicación que queremos tratar, obtenemos una versión instrumentada del código. A continuación esta versión es tratada por dos componentes diferentes: el *software Periscope Tuning Framework* (PTF) [41] y la librería READEX *Runtime Library* (RRL). PTF se encarga de obtener los parámetros de sintonización (variación de frecuencia de la CPU, frecuencia máxima, mínima, etcétera) más óptimos según nuestras preferencias (por ejemplo preferencia en ahorro energético o en rendimiento). RRL se encarga de aplicar estos parámetros en tiempo de ejecución de la aplicación.

Estos componentes utilizan además herramientas de compilación como el compilador GCC [42] y el *software* de medición de energía hdeem [43]. Es posible preparar todas estas herramientas mediante la ejecución de dos comandos que incluyen todos los componentes. El comando "module load readex/beta_gcc6.3" carga elementos como el compilador gcc6.3. El segundo comando "module load scorep-hdeem/sync-xmpi-gcc6.3" carga el módulo encargado de añadir las mediciones de energía al código.

A continuación se explican con más detalle los diversos componentes y los pasos a realizar para sintonizar nuestra aplicación automáticamente.

7.1 Instrumentación

En primer lugar, se utiliza la herramienta de instrumentación Score-p [44] al compilar el código añadiendo la palabra "scorep" antes del comando para compilar, en este caso primero se decidió probar con la versión Openmp. Esto genera un perfil con la instrumentación de todo el código. La instrumentación recoge todo tipo de mediciones sobre el código.

Al instrumentar todo el código también se instrumentan partes que consumen poco tiempo de ejecución, y la instrumentación de estas partes aumenta considerablemente el tiempo de ejecución cuando la información que pueden dar es despreciable. Este exceso en el tiempo de ejecución se le llama sobrecoste u *overhead*.

Para reducir el sobrecoste, READEX proporciona la utilidad scorep-autofilter que genera un archivo con las regiones a no tener en cuenta para los siguientes pasos. Para seleccionar (filtrar) estas regiones seleccionamos un límite de tiempo de 100ms (las que tardan menos son filtradas). Es recomendable pero no necesario repetir esta operación de filtrado varias veces, ya que en el primer filtrado puede que esas regiones de código tuviesen un tiempo de ejecución mayor debido al sobrecoste producido por regiones anidadas. Para ello, al comando se le pasa por parámetro el último archivo filtro producido. En la figura 8 se observa como filtra correctamente varias regiones, dejando instrumentada la región trainN() (correspondiente a la fase de entrenamiento) que es la que compone el bucle principal del código.

```
SCOREP_REGION_NAMES_BEGIN
EXCLUDE
char* readImg(char*)
char** loadTrainingSet(int, char**)
double rando()
void freeDeltaWeights(double**, double**)
void freeFList(int, char**)
void freeTSet(int, char**)
void freeWeights(double**, double**)
void initFileList(char**)
void initRunFileList(char**)
void printRecognized(int, double*)
void runN()
SCOREP_REGION_NAMES_END
```

Fig. 8. Regiones excluidas tras varias ejecuciones de scorep-autofilter

También es posible realizar manualmente la instrumentación, seleccionando el usuario en el código solo la región más interesante: trainN().

No obstante, esta opción requiere un mayor conocimiento de la aplicación por parte del usuario y también un mayor dominio de los detalles de Score-p.

7.2 Obtención de los parámetros de sintonización

El siguiente paso es obtener los parámetros de sintonización detectando y analizando el dinamismo de la aplicación utilizando readex-dyn-detect. La herramienta identifica automáticamente las regiones sujetas a la metodología de sintonización READEX y genera un reporte sobre el dinamismo potencial de estas regiones.

Teniendo el perfil de Score-p obtenido después de varios filtrados o mediante la instrumentación manual, pasamos como parámetro este fichero al comando readex-dyn-detect. Además, seleccionamos en el comando un límite de tiempo de 100ms y la región a analizar (trainN()). Existen otros parámetros como la mínima desviación estándar del cómputo en regiones significantes, pero los dejamos al valor por defecto ya que son filtros que limitan el análisis, y de momento no son de interés para las primeras pruebas. Esto genera un fichero "readex_config.xml" que contiene un resumen de la sintonización potencial a utilizar.

El siguiente paso es modificar el fichero "xml" generado, ya que actualmente solo contiene la fase, granularidad, etc. especificados anteriormente, pero ofrece también parámetros interesantes a probar como la mínima y la máxima frecuencia o la prioridad de la sintonización (por ejemplo priorizar energía, tiempo o energía de la CPU). Para las primeras pruebas completamos el "xml" con los parámetros descritos en el ejemplo de READEX.

El último paso es aplicar la herramienta PTF para obtener diversos escenarios posibles según las sintonizaciones seleccionadas. La herramienta muestra las propiedades de estos escenarios, los mejores y peores escenarios para la fase seleccionada, etc. Además genera un fichero de extensión ".json" con la mejor configuración para cada escenario.

Este paso es más complicado de configurar, y su correcto funcionamiento depende de nuestra aplicación y de las regiones encontradas mediante readex-dyn-detect. Por ejemplo este paso no funciona correctamente con el código de la red neuronal.

7.3 Sintonización en tiempo de ejecución

Obtenido el modelo de sintonización ".json" mediante PTF, el único paso restante es indicar como variable del sistema la librería a utilizar, llamada RRL y el fichero ".json". En el ejemplo que nos ofrecen, tenemos un script con dos ejecuciones del código de ejemplo miniMD, una sin indicar ninguna librería a utilizar (ejecución original del código) y otra indicando la librería RRL (y por tanto la sintonización obtenida mediante todas las herramientas de READEX descritas). Ambas ejecuciones utilizan la herramienta hdeem para obtener datos del consumo energético. De esta manera se obtiene el consumo energético y el tiempo total de ejecución de ambas ejecuciones.

8 RESULTADOS

Cabe mencionar que el consumo energético puede medirse de dos maneras. Una es según el consumo total de la aplicación (*Joules*) o el consumo energético por segundo (*Watts*). Para el objetivo del exascale, la medición más apropiada son los Watts, ya que un supercomputador va a estar ejecutando trabajos continuamente. No obstante, puede ser que una versión de código tenga un consumo por segundo elevado, pero su tiempo de ejecución sea mucho menor, finalizando la aplicación con un consumo total menor. En el análisis de resultados se mostraran tanto el consumo total para alcanzar el objetivo teórico del exascale, como el consumo por segundo para una aplicación práctica.

8.1 Entorno de pruebas

En esta sección se define el entorno en el que se utilizaron las herramientas, referido a sistema operativo y *hardware* utilizado.

- Sistema operativo: Para las pruebas en un único ordenador de las herramientas basadas en librerías se utilizó el sistema operativo Ubuntu, mientras que para las herramientas de sintonización automática (READEX) se trabajó sobre Red Hat Enterprise Linux Server [45].

- *Hardware*: Para las pruebas en el sistema operativo Ubuntu el *hardware* utilizado fue una CPU Intel Core i5-3570K de 3.4Ghz. En cambio, sobre Red Hat se trabajó sobre CPUs Intel Xeon E5-2680 v3 (12 cores) de 2.4GHz [46].

8.2 Herramientas basadas en librerías: *Mammut* y NVLM

8.2.1 Medición de energía

Ejecutamos la versión 1.1 de *Mammut* (Mammut-1.1) con las funciones de medición de energía, obteniendo el consumo energético del ordenador durante la ejecución del bucle principal del código. Este consumo también incluye el de otros procesos del ordenador aparte de nuestro código. Para obtener solamente el del código, ejecutamos un código con solamente las funciones de medición de *Mammut* y un "sleep()"[47] que mantiene en espera el código, obteniendo el consumo de los procesos del ordenador. Así, al consumo total medido por *Mammut* se le resto el consumo del resto de procesos, procurando tener los mismos procesos.

La versión OpenMP como vemos en la figura 9 consume más energía por segundo (*Watts*). Esto es lógico ya que la versión original utiliza solamente un núcleo de procesamiento de la CPU, mientras que la versión paralela utiliza varios núcleos de procesamiento (según la herramienta perf [48], el uso de los núcleos disponibles de la CPU en el código es equivalente a 2.5 núcleos). Estos núcleos son denominados por la herramienta perf como CPUs, pero también pueden denominarse *threads*.

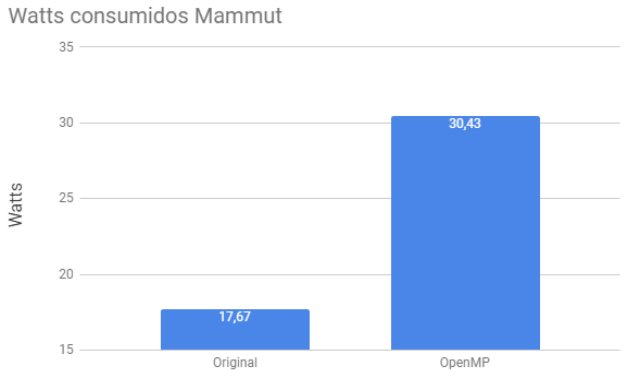

Fig. 9. Gráfico de Watts consumidos según Mammut

Además la versión OpenMP distribuye el cómputo realizado por la aplicación en estos *threads* de manera que el tiempo de ejecución disminuye frente a la versión original. Al disminuir el tiempo de ejecución también afecta al consumo total. En la figura 10 podemos ver como el consumo total disminuye ligeramente (cerca de un 13%), haciendo la versión OpenMP más eficiente en tiempo y en consumo energético total.

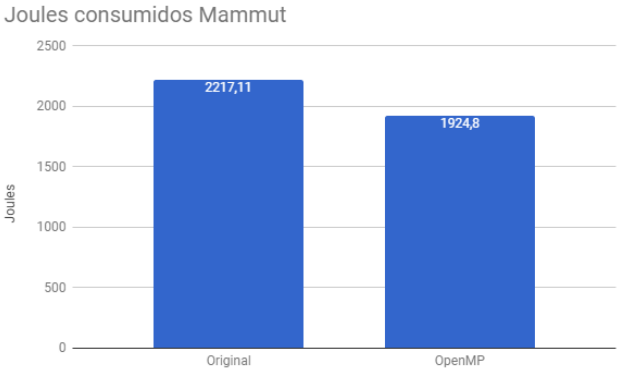

Fig. 10. Gráfico de *Joules* consumidos según *Mammut*

8.2.2 Medición de energía: Núcleos al máximo uso

La librería de Mammut-1.1 contiene una función para cambiar el uso de los núcleos de la CPU al máximo posible. Este aumento del uso es forzado y por tanto innecesario para la ejecución del código, por lo que podemos ver que el tiempo de ejecución no se reduce y si aumenta el consumo energético, pero se decidió comprobar en qué medida afectaba. Se añadió esta función junto con las mediciones de energía en nuestro bucle principal, maximizando el uso de un núcleo. Cada núcleo de nuestra CPU contiene 2 *threads*, es decir 2 CPUs. En la figura 11 puede verse los resultados obtenidos.

En el caso de la versión original las CPU utilizadas aumentaron de 1 a 2, y el consumo aumentó un 42%, al no ser proporcional, se puede apreciar que hay más factores que el uso de la CPU en el consumo energético (accesos a memoria por ejemplo). En cambio en OpenMP las CPU aumentan un 27% y el consumo en un 33%. Estas proporciones son las mismas para el consumo total, ya que el tiempo de ejecución no varía.

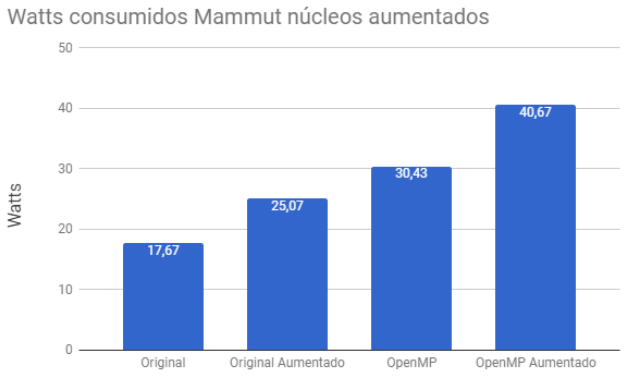

Fig. 11. Gráfico de *Watts* consumidos con un núcleo al 100% de uso según *Mammut*

8.2.2 Medición de energía: *Mammut* y NVLM

A continuación, en la figura 12, se comparan los resultados de la ejecución por CPU con Mammut vistos anteriormente, y la ejecución de la versión CUDA del código en una tarjeta gráfica NVIDIA GTX1080.

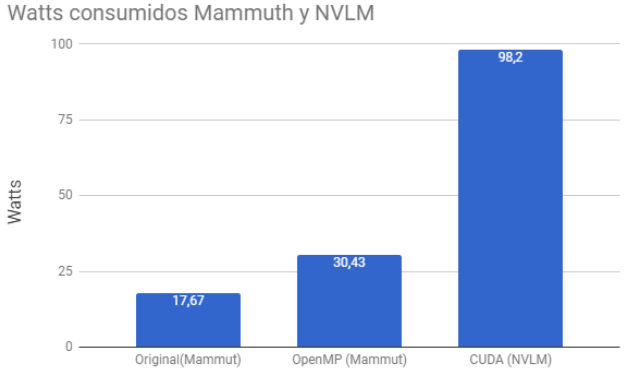

Fig. 12. Gráfico de *Watts* consumidos según *Mammut* y NVLM

El consumo de la versión CUDA del código es notablemente más elevado que el resto de versiones. No obstante, hay que recalcar que el tiempo de ejecución de la versión CUDA baja considerablemente, reduciendo también el consumo total como se observa en la figura 13. Esto hace la versión CUDA muy eficiente en términos prácticos.

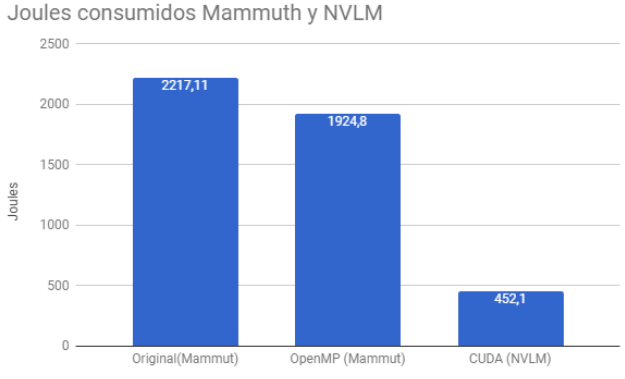

Fig. 13. Gráfico de Watts consumidos según mammut y NVLM

8.3 Herramientas de sintonización automática: READEX

Ejecutamos el script de ejemplo de miniMD que contiene las dos ejecuciones y conseguimos ver algunos resultados.

Para analizar estos resultados primero debemos revisamos el contenido del fichero ".json". Observamos que selecciona de 2 ficheros diferentes una fase a sintonizar. Decidimos realizar una observación rápida del contenido.

En la fase "force_lj_comp_half" hay dos bucles anidados con el bucle interior vectorizado. En la fase "integrate_run_loop" hay una paralelización del código mediante openmp, con diversas sincronizaciones mediante "pragma omp barrier" y partes de la fase que solamente utiliza ejecuta un *thread*, siendo una fase compleja para sintonizar correctamente.

Al ejecutar el script se muestra también diversos parámetros configurados (se pueden observar en la figura 14), el más interesante en nuestro caso es el uso de un solo *thread* y un solo proceso MPI. Por tanto los resultados son sobre la versión base del ejemplo miniMD.

```
# Systemparameters:
        # MPI processes: 1
        # OpenMP threads: 1
        # Inputfile: in3.data
        # Datafile: None
        # ForceStyle: LJ
        # Units: LJ
        # Atoms: 108000
        # System size: 50.39 50.39 50.39 (unit cells: 30 30 30)
        # Density: 0.844200
        # Force cutoff: 2.500000
        # Neigh cutoff: 2.800000
        # Half neighborlists: 1
        # Neighbor bins: 25 25 25
        # Neighbor frequency: 10
        # Timestep size: 0.005000
        # Thermo frequency: 100
        # Ghost Newton: 1
        # Use SSE intrinsics: 0
        # Do safe exchange: 0
        # Size of float: 8
```

Fig. 14. Parámetros de configuración del ejemplo miniMD

Apreciamos en la figura 15 que el consumo por segundo se ha reducido considerablemente, en un 20%. Como dato a apreciar, el tiempo de ejecución entre la versión sin sintonizar y la sintonizada son prácticamente iguales, obteniendo la misma proporción sobre el consumo total. Además el aumento de nodos y procesos utilizados en la aplicación prácticamente no afecta a la proporción del 20%.

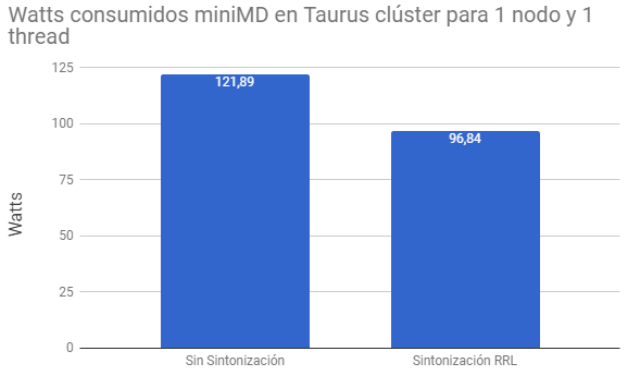

Fig. 15. Gráfico de *Watts* consumidos miniMD en Taurus clúster para 1 nodo y 1 *thread*

Al aumentar el número de nodos y *threads* utilizados en el proceso, el tiempo de ejecución disminuye (como efecto de la paralelización), disminuye el consumo total energético y los *Watts* consumidos se mantienen en valores similares al ejemplo anterior. En la figura 16 se puede observar un ejemplo del consumo total con 2 nodos y 8 *threads*.

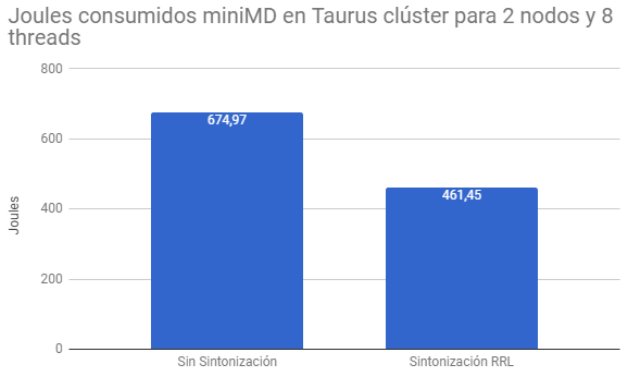

Fig. 16. Gráfico de *Joules* consumidos miniMD en Taurus clúster para 2 nodos y 8 *threads*

Para este ejemplo concluimos que la sintonización ofrece un considerable ahorro energético y de tiempo.

9 Conclusión

Realizar optimizaciones en paralelo en el código como OpenMP o sobretodo CUDA es beneficioso para el tiempo de ejecución y el consumo energético total. Pueden parecer negativas para alcanzar el exascale, ya que la medición para este objetiva se realiza en *Watts*, pero en la práctica al observar que su consumo energético total es menor, vemos que son beneficiosas.

Las herramientas de librerías y escritorio son en general sencillas de configurar y utilizar pero solamente pueden utilizarse en un único computador (no funcionan para supercomputadores). Profundizar en la configuración óptima puede ser complicado, pero probar las configuraciones más sencillas, como rebajar la frecuencia de reloj, pueden ayudar al consumo energético sin necesidad de complicarse. No obstante, se recomienda utilizar las últimas arquitecturas de *hardware*, sobretodo en el caso de la GPU, para poder utilizar completamente estas herramientas y recibir posibles actualizaciones futuras.

READEX permite monitorizar y controlar el consumo energético con muchas opciones de configuración, tanto en computadoras únicas como en clúster. Es efectiva reduciendo el consumo energético en el código de la dinámica molecular, y tiene potencial para extrapolarse a más aplicaciones. La dificultad de aprendizaje de uso para un usuario no habituado a modificar parámetros *hardware* es elevada, pero respecto a realizar la configuración manualmente se reduce el trabajo y complejidad. Por tanto concluimos que es una buena herramienta para obtener una medición del consumo energético, y tiene potencial para convertirse además en una herramienta sencilla y de ahorro notable de consumo energético.

Se consiguió gran parte del objetivo de investigar los métodos disponibles para el ahorro energético, ofreciendo una visión general de una buena cantidad de herramientas *software*. No obstante, no se ha podido analizar todas las herramientas actuales (como es el caso de GEOPM) ni con toda la profundidad que contienen (todas las opciones de configuración que disponen las herramientas analizada). En el futuro, además del análisis general que aporta este trabajo, se podría profundizar sobre herramientas con gran potencial como READEX, utilizando varias versiones de código y el resto de sus opciones de configuración.

Agraïments

Agradecimientos a Eduardo Cesar Galobardes por los consejos y correcciones recibidas. Agradecimientos también a Venkatesh Kannan por el acceso y soporte a la herramienta READEX.

Bibliografía

[1] "Performance Development | TOP500 Supercomputer Sites." https://www.top500.org/statistics/perfdevel/. Se consultó el 7 feb.. 2018.

[2] "Exascale computing - Wikipedia." https://en.wikipedia.org/wiki/Exascale_computing. Se consultó el 7 feb.. 2018.

[3] "Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C ... - Top500." https://www.top500.org/system/178764. Se consultó el 7 feb.. 2018.

[4] "Exascale Challenges - DOE Office of Science - Department of Energy." 5 mar.. 2016, https://science.energy.gov/ascr/research/scidac/exascale-challenges/. Se consultó el 7 feb.. 2018.

[5] "The READEX Project – The READEX Project." http://www.readex.eu/index.php/the-readex-project

[6] "GitHub - geopm/geopm: Global Extensible Open Power Manager." 17 jun.. 2017, https://github.com/geopm/geopm. Se consultó el 30 sept.. 2017.

[7] "Reducing Cluster Energy Consumption through ... - EECS Berkeley."

https://www2.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-108.pdf. Se consultó el 7 feb.. 2018.

[8] "Reducing Cluster Power Consumption by Dynamically Suspending ...." http://digitalcommons.calpoly.edu/theses/305/. Se consultó el 7 feb.. 2018.

[9] "Etapas de la metodología de la investigación | Webscolar." http://www.webscolar.com/etapas-de-la-metodologia-de-la-investigacion. Se consultó el 10 feb.. 2018.

[10] "OpenMP - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/OpenMP. Se consultó el 10 feb.. 2018.

[11] "Message Passing Interface (MPI ...." 19 jun.. 2017, https://computing.llnl.gov/tutorials/mpi/. Se consultó el 10 feb.. 2018.

[12] "Procesamiento paralelo CUDA | Qué es CUDA |NVIDIA." http://www.nvidia.es/object/cuda-parallel-computing-es.html. Se consultó el 10 feb.. 2018.

[13] "Supercomputer - Wikipedia." https://en.wikipedia.org/wiki/Supercomputer. Se consultó el 10 feb.. 2018.

[14] "Red neuronal artificial - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/Red_neuronal_artificial. Se consultó el 10 feb.. 2018.

[15] "Implementing a Neural Network in C." 14 oct.. 2009, http://www.cs.bham.ac.uk/~jxb/INC/nn.html. Se consultó el 10 feb.. 2018.

[16] "Dinámica molecular - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/Din%C3%A1mica_molecular. Se consultó el 10 feb.. 2018.

[17] "Mantevo.org – Home of the Mantevo Project." https://mantevo.org/. Se consultó el 10 feb.. 2018.

[18] "Sandia National Laboratories: Exceptional Service in the National ...." http://www.sandia.gov/. Se consultó el 10 feb.. 2018.

[19] "Instrumentation (computer programming) - Wikipedia." https://en.wikipedia.org/wiki/Instrumentation_(computer_programming). Se consultó el 10 feb.. 2018.

[20] "Powertop - ArchWiki." https://wiki.archlinux.org/index.php/powertop. Se consultó el 4 nov.. 2017.

[21] "Ubuntu Manpage: powerstat - a tool to measure laptop power ...." http://manpages.ubuntu.com/manpages/trusty/man8/powerstat.8.html. Se consultó el 15 nov.. 2017.

[22] "GitHub - RRZE-HPC/likwid: Performance monitoring and ...." https://github.com/RRZE-HPC/likwid. Se consultó el 4 nov.. 2017

[23] "Mammut: High-level management of system knobs and sensors." 18 jul.. 2017, http://www.sciencedirect.com/science/article/pii/S2352711017300225. Se consultó el 30 sept.. 2017.

[24] "NVIDIA Management Library (NVML) | NVIDIA Developer." https://developer.nvidia.com/nvidia-management-library-nvml. Se consultó el 10 feb.. 2018.

[25] "GitHub - tud-zih-energy/libadapt: A library to change system ...." https://github.com/tud-zih-energy/libadapt. Se consultó el 10 feb.. 2018

[26] "READEX - Runtime Exploitation of Application ... - TU Dresden." 2 oct.. 2017, https://tu-dresden.de/zih/forschung/projekte/readex. Se consultó el 10 feb.. 2018

[27] "Getting Started - Vampir." https://www.vampir.eu/tutorial/manual/getting_started. Se consultó el 10 feb.. 2018.

[28] "GNU/Linux - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/GNU/Linux. Se consultó el 10 feb.. 2018.

[29] "Ubuntu." https://www.ubuntu.com/. Se consultó el 10 feb.. 2018.

[30] "vmstat(8): Report virtual memory statistics - Linux man page." https://linux.die.net/man/8/vmstat. Se consultó el 10 feb.. 2018.

[31] "Thread (computing) - Wikipedia." https://en.wikipedia.org/wiki/Thread_(computing). Se consultó el 10 feb.. 2018.

[32] "msr(4) - Linux manual page - man7.org." http://man7.org/linux/man-pages/man4/msr.4.html. Se consultó el 4 nov.. 2017.

[33] "Framework - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/Framework. Se consultó el 10 feb.. 2018.

[34] "CONFIG_X86_ACPI_CPUFREQ - cateee.net Homepage." https://cateee.net/lkddb/web-lkddb/X86_ACPI_CPUFREQ.html. Se consultó el 24 dic.. 2017.

[35] "Mammut - GitHub Pages." http://danieledesensi.github.io/mammut/manual.html. Se consultó el 24 dic.. 2017.

[36] "Cómo compilar el kernel Linux en Ubuntu, Fedora, y otras » MuyLinux." 10 nov.. 2010, https://www.muylinux.com/2010/11/10/como-compilar-el-kernel-linux-en-ubuntu-fedora-y-otras/. Se consultó el 10 feb.. 2018.

[37] "BIOS - Wikipedia." https://es.wikipedia.org/wiki/BIOS. Se consultó el 10 feb.. 2018.

[38] "Ubuntu Manpage: cpufreqd - intelligently monitor and manipulate CPU ...." http://manpages.ubuntu.com/manpages/precise/man8/cpufreqd.8.html. Se consultó el 24 dic.. 2017.

[39] "CPU frequency scaling - Arch Wiki." https://wiki.archlinux.org/index.php/CPU_frequency_scaling. Se consultó el 24 dic.. 2017.

[40] "Interfaz de programación de aplicaciones - Wikipedia, la enciclopedia ...."

https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones. Se consultó el 10 feb.. 2018.

[41] "Periscope Tuning Framework." http://periscope.in.tum.de/. Se consultó el 24 dic.. 2017.

[42] "GCC, the GNU Compiler Collection - GNU Project - Free Software ...." https://gcc.gnu.org/. Se consultó el 10 feb.. 2018.

[43] "HDEEM: High Definition Energy Efficiency Monitoring — Zentrum für ...." https://tu-dresden.de/zih/forschung/projekte/hdeem. Se consultó el 10 feb.. 2018.

[44] "VI-HPS :: Projects :: Score-P." http://www.vi-hps.org/projects/score-p/. Se consultó el 10 feb.. 2018.

[45] "Red Hat Enterprise Linux." https://www.redhat.com/es/technologies/linux-platforms/enterprise-linux. Se consultó el 10 feb.. 2018.

[46] "HardwareTaurus - Compendium - Foswiki - TU Dresden." 8 ago.. 2017, https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/HardwareTaurus. Se consultó el 10 feb.. 2018.

[47] "C Program to show Sleep() function example. - C Program Examples." 12 oct.. 2011, http://c-program-example.com/2011/10/c-program-to-show-sleep-function-example.html. Se consultó el 10 feb.. 2018.

[48] "Perf Wiki." 28 sept.. 2015, https://perf.wiki.kernel.org/index.php/Main_Page. Se consultó el 10 feb.. 2018.