Un Entorno de Análisis del Consumo de Aplicaciones Paralelas

Maria Barreda, Manuel F. Dolz, Rafael Mayo y Enrique S. Quintana-Ortí[1]

Resumen— **Entender el uso energético que supone la ejecución de cargas paralelas es fundamental para implementar aplicaciones eficientes que se ejecutarán en las futuras plataformas Exaescala. Con este objetivo, el artículo presenta un entorno de trabajo para el perfilado, la monitorización y el análisis de la potencia y energía consumida por aplicaciones paralelas, tanto de memoria compartida como de distribuida. Este entorno de trabajo, compuesto por un conjunto de dispositivos de medida de energía y una sencilla interfaz de medición, es capaz de recolectar muestras de potencia/energía consumida por aplicaciones paralelas. Dicha interfaz de medición se emplea de forma combinada con las herramientas de instrumentación y visualización Extrae y Paraver, respectivamente. El resultado proporciona una visión ampliada en forma de traza de ejecución y de potencia que permite al programador relacionar y detectar las posibles causas de ineficiencia energética en el código de la aplicación.**

Palabras clave— **Potencia, energía, computación de altas prestaciones, procesadores multinúcleo.**

I. Introducción

En el camino hacia los sistemas Exascala, el consumo de energía es un aspecto fundamental al que la computación de alto rendimiento (HPC) debe hacer frente [1], [2]. Un cálculo simple muestra que una plataforma exaescala construida con la tecnología hardware actual disiparía 100 MWatts, haciéndola económicamente inviable [3], [4], [5], [6], [7]. En otras palabras, incluso con la tasa de mejora en la eficiencia energética disfrutada por las supercomputadoras en los últimos años [8], el consumo de energía objetivo de 20 MWatts para el equipo Exascala todavía se superará en un factor de 5× en el 2018-2020. Por tanto, si se tiene que superar la barrera de la energía, se necesita un enfoque global de potencia/energía, en particular uno que tenga como objetivo el desarrollo de hardware más eficiente en cuanto a consumo, y sea consciente del ahorro de energía en el software del sistema, las bibliotecas de comunicación y de cálculo, y las aplicaciones de tareas paralelas. En este artículo se ahonda este desafío, proponiendo un entorno de trabajo de herramientas sencillas y escalables para el análisis de la disipación de potencia y el consumo de energía en aplicaciones paralelas MPI y/o aplicaciones multi-thread científicas. Estas herramientas se componen de los siguientes componentes hardware y software:

- Un conjunto de medidores internos de corriente continua cuyo diseño está basado en un microcontrolador que muestrea la potencia disipada por la placa base del sistema a tasas que oscilan entre 25 y 100 Hz.
- Una API (*Application Programming Interface*) simple para interactuar con distintos dispositivos de medición de potencia, incluyendo medidores de corriente alterna comerciales y externos, tales como *WattsUp? Pro .Net*, así como nuestros medidores de potencia internos.
- La biblioteca `pm` asociada y los dispositivos que permiten capturar la potencia disipada durante la ejecución de una aplicación en un sistema separado.
- Una integración de estas herramientas con los paquetes `Extrae+Paraver` [9] que permite un análisis interactivo de una traza gráfica que muestre la disipación de potencia por nodo/núcleo con información de la actividad del núcleo.

En conjunto, estas herramientas proporcionan un medio útil para los administradores de sistemas, desarrolladores de bibliotecas y científicos con el fin de identificar puntos críticos en el hardware (a nivel de núcleo) y sumideros de energía en el software del sistema, en el código de la biblioteca y en las aplicaciones.

El resto del artículo está estructurado de la siguiente forma. En la siguiente subsección se resume brevemente el trabajo previo. En la Sección II se presentan las herramientas de análisis de rendimiento y medición de potencia, incluyendo la arquitectura del sistema, los componentes software y las interfaces software-hardware. En la Sección III se ilustra el uso y los beneficios de estas herramientas mediante algunos ejemplos representativos de álgebra lineal. Por último, en la Sección IV se ofrece un resumen del trabajo y algunas conclusiones.

A. Trabajo previo

Dado el interés que está generando el consumo de energía, no es de extrañar que existan una serie de proyectos de hardware y software que persigan fines similares a los que se abordan en este trabajo. A continuación se describen algunos de ellos. En [10] se presenta un estudio sobre el hardware, el software y las herramientas de perfiles de energía híbridas. Entre estos proyectos, Powermon 2 [11] es un dispositivo que, introducido entre una fuente de potencia y la placa base del computador, mide el consumo de energía de las líneas de corriente continua, y ofrece una interfaz de software muy básica.

PowerPack [12] utiliza un medidor de corriente continua comercial conectado a las líneas que salen de la fuente de alimentación. Además, este paquete

[1]Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, e-mails: `mvaya@guest.uji.es`, `{dolzm,mayo,quintana}@icc.uji.es`

realiza una serie de tests con el fin de identificar las líneas de alimentación de los diferentes componentes del computador como discos, memoria, tarjetas de red y procesadores, de modo que esta información se pueda aprovechar para identificar cómo y dónde consumen energía las aplicaciones HPC. PowerPack, por lo tanto, proporciona una interfaz más amigable que PowerMon 2, pero sólo se refiere a las aplicaciones científicas que se ejecutan en plataformas de un solo nodo.

HDTrace [13] es un paquete de software que permite tracear y simular el comportamiento de programas MPI en un clúster. Se incluye explícitamente el soporte de trazado interno de MPICH2 y de los archivos PVFS paralelos del sistema. En [14] esta herramienta se ha ampliado para proporcionar información que ayude a identificar los puntos calientes de potencia en estas aplicaciones, utilizando la información de medidores comerciales de corriente alterna.

Nuestro trabajo proporciona una solución con componentes hardware y software, especialmente integrada con los dispositivos de medición y bibliotecas para análisis de rendimiento y traceado de potencia. Además, se aprovecha el potencial del paquete de instrumentación `Extrae` para permitir un control escalable de aplicaciones que se ejecutan en clústers con procesadores multinúcleo, tanto desde la perspectiva de rendimiento como desde la de potencia, y las posibilidades de `Paraver` para ofrecer una interfaz gráfica de usuario que posibilita un rápido análisis de los resultados.

```
#define A_ref(i,j) A[((j)-1)*Alda+((i)-1)]

void dgeqrf( int n, int m, int nb, double *A,
             int Alda, double * tau,
             double * work, int lwork,
             int *info ){
  // Declaration of variables ...

  pm_start_counter(&pm_ctr);
  Extrae_init();
  for (k=1; k<=n; k+=nb) {
    // Factor current diagonal block
    Extrae_event(500000001,1);
    dgeqr2(n-k+1, nb, &A_ref(k,k),
           Alda, tau[k], work, info );
    Extrae_event(500000001,0);

    if( k+nb <= n ) {
      // Triangular factor
      Extrae_event(500000001,2);
      dlarft( "F", "C", n-k+1, nb,
           &A_ref( k, k ), Alda,
           tau[k], work, ldwork );
      Extrae_event(500000001,0);

      // Update trailing submatrix
      Extrae_event(500000001,3);
      dlarfb( "L", "T", "F", "C", n-k+1,
           n-k-nb+1, nb, &A_ref( k, k ), Alda,
           &work, &ldwork, &A_ref( k, k+nb ),
           Alda, &work[ nb+1 ], &ldwork );
      Extrae_event(500000001,0);
    }
  }
  Extrae_fini();
  pm_stop_counter(&pm_ctr);
}
```

TABLA I

ALGORITMO DE LA FACTORIZACIÓN QR CON PIVOTAMIENTO PARCIAL ANOTADO CON LAS APIS DE EXTRAE Y MEDIDAS DE POTENCIA.

II. HERRAMIENTAS Y APIS PARA EL TRACEO DE RENDIMIENTO Y DE POTENCIA

Como punto de partida para exponer nuestro objetivo consideramos un código para una aplicación científica, al que llamaremos `app.c`, paralelo MPI, multihebra (OpenMP, POSIX threads, runtime de tareas paralelas como SMPSs [15], etc.), o una combinación de ambos. En un primer momento el usuario anota el código con un número mínimo de llamadas a las APIs de análisis de rendimiento y potencia (que se describen a continuación), obteniendo como resultado una versión modificada del programa. En la etapa siguiente de este proceso inicial, se obtiene el ejecutable `app.x` usando un compilador estándar y enlazando con las bibliotecas necesarias, en particular, las de análisis de rendimiento y de potencia.

A. *El entorno de trabajo* `Extrae`+`Paraver`

`Extrae` es un paquete de instrumentación y medición desarrollado al *Centro de Supercomputación de Barcelona (BSC)*. Esta herramienta intercepta llamadas a MPI, OpenMP y POSIX threads, registrando información relevante (p.e. eventos, marcas de tiempo, valores de contadores hardware, etc.) en diferentes ficheros de traza que posteriormente serán combinados para producir un solo fichero que contenga el perfil detallado de la actividad de una aplicación paralela. La herramienta complementaria `Paraver`, también desarrollada por el BSC, puede usarse para analizar la información contenida en el fichero de traza a través de una interfaz gráfica. Esta herramienta facilita enormemente la inspección del paralelismo y la escalabilidad de la aplicación, así como la visualización de un número de métricas que caracterizan el código y el rendimiento durante la ejecución.

Con el fin de introducir las herramientas de rendimiento y de potencia (bibliotecas), APIs, vistas gráficas, etc., en lo sucesivo, se empleará la factorización QR [16] como un ejemplo de guía. La Tabla I muestra el código la factorización QR de una matriz `A` de tamaño $\mathtt{n} \times \mathtt{n}$ que se factoriza en paneles de `b` columnas, imitando el procedimiento de la rutina `dgeqrf` de LAPACK [17]. (Para simplificar, en lo sucesivo, se asume que el tamaño de la matriz, `n`, es un entero múltiplo del tamaño de bloque, `nb`.) En la práctica, el factor `R` triangular resultante sobreescribe la parte triangular superior de la matriz `A`, mientras que la matriz ortogonal `Q` no se construye implícitamente, sino que se almacena como una colección de vectores de Householder utilizando las entradas en la parte triangular inferior de `A` y un vector de orden `n`. Por otra parte, el algoritmo requiere $4\mathtt{n}^3/3$ flops. La mayor parte de estos cálculos vienen dados en términos de productos de matrices, por lo que el cómputo del algoritmo tiene un buen rendimiento y proporciona un considerable nivel de

paralelismo en implementaciones para plataformas multinúcleo. Este paralelismo se extrae mediante la invocación de implementaciones multihilo de los *kernels* numéricos `dgetf2`, `dlarfb`, y `dlarft`, tradicionalmente ofrecidas por los fabricantes de hardware (p.e., Intel MKL, AMD ACML or IBM ESSL).

La interacción con el paquete de instrumentación de rendimiento es sencilla: las rutinas `Extrae_init` y `Extrae_fini`, de la API de `Extrae`, inicializan y finalizan la herramienta de traceado. La rutina `Extrae_event` registra un evento en el fichero de traza que hace referencia al código fuente y, al termino de la instrumentación, el sistema registra todos los datos de rendimiento en los archivos del disco.

B. El entorno de trabajo de medición (`pm`*)*

La biblioteca `pm` es un paquete de software creado por la *Universidad Jaume I* (UJI) para interactuar con los dispositivos de medida de potencia. La implementación actual de este paquete proporciona una interfaz para utilizar nuestros diseños internos de medición de corriente continua, aunque también es compatible con medidores comerciales externos de corriente alterna como *WattsUp? Pro .Net*, *APC Power Distribution Units*, etc. y otros internos, como tarjetas de adquisición de datos de *National Instruments*.

La medición de potencia se inicia y termina desde la aplicación a través de las rutinas `pm_start_counter` y `pm_stop_counter`, respectivamente, de la API `pm`. La Tabla I ilustra el uso de estas rutinas para capturar el consumo de potencia de la rutina que realiza la factorización QR. A pesar de que las bibliotecas de traceado y extracción de perfil de potencia son independientes, hay que tener en cuenta que no supondría una tarea complicada integrar las llamadas a `pm_start_counter`/`pm_stop_counter` dentro de la inicialización/finalización de `Extrae`, activando el modo de traceado a través de un archivo de configuración. Por razones de compatibilidad con otras bibliotecas de traceado, tales como TAU, se ha decidido mantener la interfaz `pm` desligada de éstas.

Para evitar la interferencia entre las pruebas, las muestras de los medidores externos y/o internos se recogen en sistemas separados. La Figura 1 muestra la conexión de los elementos que se encargan de medir la potencia: el nodo de la aplicación, el hardware del medidor y el servidor encargado de recoger las medidas de potencia. Hay que destacar que los dispositivos de medición internos muestrean la potencia disipada por las líneas que conectan la fuente de alimentación con la placa base. Por otra parte, los dispositivos de medición externos miden la potencia disipada por el nodo completo, por lo que incluyen, además de la potencia de la placa base, la consumida por los discos, tarjetas de red, procesadores gráficos, etc. Tanto los dispositivos de medición internos como externos se comunican con el servidor `pm` a través de los puertos USB, RS232 y Ethernet.

Las rutinas principales de la biblioteca `pm` se comu-

`int pm_set_server( char *svrip, int port, server_t *svr )`	
Uso:	Incializa el servidor con la dirección IP y puerto para la comunicación con el servidor PM.
svrip:	Dirección IP del servidor PM.
port:	Puerto de comunicación del servidor PM.
svr:	Especificación del servidor con los parámetros inicializados.
`int pm_get_devices( server_t *svr, char** ldev, int *ndev )`	
Uso:	Lista los dispositivos actualmente conectados al servidor.
svr:	Especificación del servidor.
ldev:	Lista de los dispositivos conectados.
ndev:	Número de dispositivos conectados.
`int pm_get_device_info( server_t *svr, char* devn, device_t * dev )`	
Uso:	Devuelve la frecuencia de muestreo y el número de lineas del dispositivo.
svr:	Especificación del servidor.
ndev:	Nombre del dispositivo.
dev:	Especificación del dispositivo (nombre, frecuencia máxima y número de líneas).
`int pm_create_counter( char *devn, mask_t lin, int aggr, int freq, server_t svr, counter_t *pm_ctr)`	
Uso:	Envía una petición al servidor PM para la creación de un nuevo contador.
devn:	Nombre del dispositivo.
lin:	Máscara con las líneas seleccionadas para la medición.
aggr:	Boleano que establece si se requiere realizar una medición por líneas de forma separada o agregada.
freq:	Frecuencia de medición.
svr:	Especificación del servidor.
pm_ctr:	Especificación del servidor del contador como parámetro de retorno.
`int pm_start_counter( counter_t *pm_ctr )`	
Uso:	Empieza la medición del contador.
pm_ctr:	Especificación del contador.
`int pm_continue_counter( counter_t *pm_ctr )`	
Uso:	Continúa la medición consevando datos previos.
pm_ctr:	Especificación del contador.
`int pm_stop_counter( counter_t *pm_ctr )`	
Uso:	Detiene la medición del contador.
pm_ctr:	Especificación del contador.
`int pm_get_counter_data( counter_t *pm_ctr )`	
Uso:	Vuelca los datos en memoria.
pm_ctr:	Especificación del contador.
`int pm_print_data_stdout( counter_t *pm_ctr )`	
Uso:	Imprime los datos por la salida estándar.
pm_ctr:	Especificación del contador.
`int pm_print_data_paraver( char *file, counter_t *pm_ctr, char *unit )`	
Uso:	Imprime los datos por la salida estándar.
file:	Nombre del fichero de salida.
pm_ctr:	Especificación del contador.
unit:	Unidad de tiempo.
`int pm_finalize_counter( counter_t *pm_ctr )`	
Uso:	Finaliza el contador en el servidor.
pm_ctr:	Especificación del contador.

TABLA II
RUTINAS BÁSICAS DE LA API `pm`.

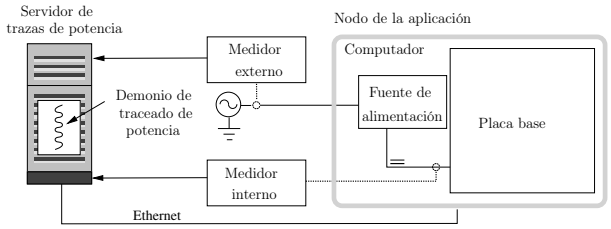

Fig. 1. Sistema de aplicación de un solo nodo y puntos de muestreo para medidores externos e internos.

nican con el servidor como sigue. Invocando la rutina `pm_start_counter` desde el código de la aplicación se indica al del servidor `pm` (Figura 1) el comienzo del registro de mediciones obtenidas en el medidor de potencia. Por el contrario, la rutina `pm_stop_counter` indica al servidor que termine con el registro de datos para que, posteriormente, éstos puedan ser volcados en un fichero con formato compatible con `Paraver`. La interacción/sincronización entre la aplicación y el servidor `pm` de recogida de muestras de potencia se lleva a cabo a través de distintas rutinas de la API `pm` resumidas en la Tabla II. Los mensajes enviados por la biblioteca `pm` entre el nodo de aplicación y el servidor de potencia son transferidos a través de un concentrador de Fast Ethernet y un protocolo específico sobre TCP.

C. Dispositivos de medición

En esta sección se enumeran los dispositivos de medición internos y externos disponibles para los cuales la biblioteca `pm` es capaz de registrar y comunicarse para extraer medidas de potencia consumida por los nodos conectados a ellos.

C.1 Dispositivos internos

Los dispositivos de medición internos que actualmente soporta el servidor `pm` son los siguientes:

- Dispositivo de interno basado en ASIC: Como ya se ha mencionado anteriormente, se han diseñado y construido una serie de dispositivos de medición de potencia disipada por los componentes. Estos dispositivos están compuestos por una serie de transductores de corriente LEM HXS 20-NP y un microcontrolador. Cada transductor es capaz de recoger la potencia disipada por cada línea con una frecuencia que varía entre los 25 y 100 Hz, dependiendo de las especificaciones del diseño; el microcontrolador digitaliza el resultado, y envía los datos al través del puerto de comunicaciones, en este caso RS232.
- Dispositivo de medición basado en tarjeta de adquisición de datos: En este caso se emplea una tarjeta comercial de adquisición de datos de National Instruments que, conectada a una serie de transductores adjuntos a las líneas conectadas entre la fuente de alimentación y la placa base, es capaz de recoger hasta 7000 muestras por segundo. Las medidas recogidas por los transductores reflejan la potencia disipada por las líneas conectadas entre la fuente de alimentación y la placa base. Los datos recogidos por la tajeta de adquisición de datos se leen desde el servidor a través del puerto USB.

C.2 Dispositivos externos

Los medidores externos que actualmente soporta el servidor `pm` son los siguientes:

- Unidades de distribución de potencia: Las unidades de distribución de potencia son frecuentemente utilizadas en racks y se emplean para alimentar a los nodos que se encuentran montados en él. Normalmente, estas unidades disponen de un puerto Ethernet y una interfaz `ssh` en la que se puede consultar la potencia medida. El modelo utilizado en nuestro sistema es la unidad APC8653, compuesta por 24 tomas, que se comunica con el servidor `pm` y envía muestras a 1 Hz.
- Dispositivos de medición dedicados: Tienen cómo único propósito la medición de la potencia disipada por los nodos. Éstos se conectan entre la toma de corriente y el nodo. Entre ellos, se ha empleado el dispositivo *WattsUp? Pro .Net*, capaz de recoger muestras a 1 Hz y enviarlas por los puertos Ethernet y USB.

El servidor `pm` se ha diseñado de forma que es capaz de interactuar con todos estos dispositivos y extraer la potencia por los nodos conectados a ellos cuando el usuario utiliza las funciones asociadas a la parte cliente de la biblioteca `pm`.

III. Interacción entre rendimiento y potencia

Siguiendo con nuestro ejemplo, el ejecutable de la aplicación `app.x`, enlazado con las bibliotecas de traceado de potencia y rendimiento, produce los correspondientes archivos de traza (por ejemplo `power.prv` y `performance.prv`). `Paraver` puede usarse para mostrar y analizar la traza de potencia de la aplicación y relacionarla con las actividades computacionales a nivel de núcleo/nodo; ver Figura 2. Las diferentes opciones del entorno de visualización se pueden personalizar a través de los archivos de configuración correspondientes (extensiones `app.pcf` y `app.row`).

A continuación se muestran algunos de los beneficios que puede tener la inspección visual entre el rendimiento y potencia utilizando como ejemplo la factorización QR para matrices densas. Durante la realización de los experimentos, el código fue enlazado con Intel MKL (v10.3.9), `Extrae` (v2.2.0), `Paraver` (v4.1.0), `pm` (v2.0); y ejecutado en una plataforma con cuatro procesadores AMD Opteron 6172 (doce núcleos a 2.1 GHz) y 256 GB de RAM. Se ha evaluado la implementación en `LAPACK` de la factorización QR empleando un tamaño de n=10.240 y sólo un procesador (12 cores). La rutina `dgeqrf` está disponible en `http://www.netlib.org` y explota el paralelismo en las llamadas a las rutinas de

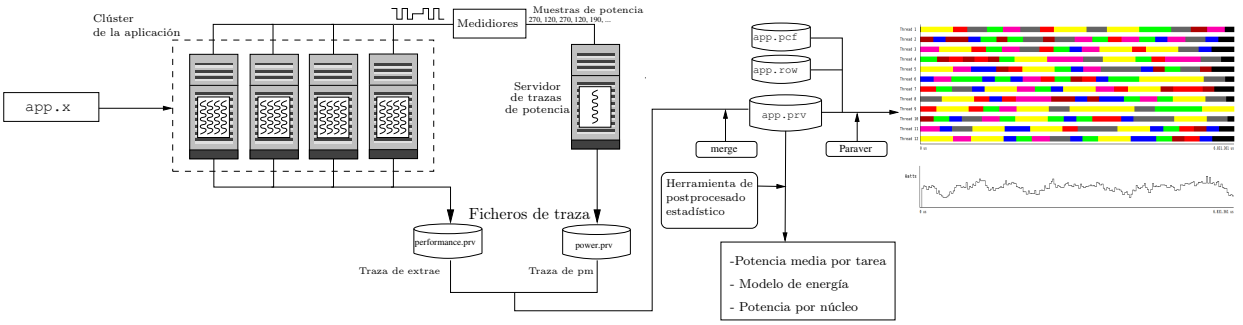

Fig. 2. Recogida de muestras de potencia en tiempo de ejecución y visualización del perfil de potencia.

	Factorización QR LAPACK
T (s)	43.70
GFLOPS	32.76
P_{min} (W)	266.96
P_{avg} (W)	326.63
P_{max} (W)	376.65
E_{tot} (J)	14,276

TABLA III
PRESTACIONES, POTENCIA Y ENERGÍA PARA LAS DIFERENTES IMPLEMENTACIÓNES DE LA FACTORIZACIÓN QR.

Intel (multihebra) MKL BLAS. El tamaño de bloque fue establecido b=128 para el tamaño de problema n=10.240, pues es el valor óptimo para la arquitectura y BLAS empleado en nuestros experimentos.

La Figura 3 muestra las trazas de ejecución y perfil de potencia consumida obtenidas en la ejecución de la rutina de LAPACK `dgeqrf`, que calcula la factorización QR. De arriba a abajo, las dos primeras figuras corresponden a las trazas de la factorización completa (llamadas a los núcleos computacionales y potencia, respectivamente), mientras que las dos siguientes amplian las dos primeras iteraciones del procedimiento (tercera y cuarta figura, para llamadas a los núcleos computacionales y potencia). Los resultados ilustran que, en esta plataforma, la rutina de LAPACK enlazada con la implementación multihebra de Intel MKL BLAS intercala fases secuenciales y paralelas durante la factorización QR del panel actual y la parte paralela dedicada a aplicar las transformaciones ortogonales. Desde el punto de vista del consumo, las etapas secuenciales y paralelas en el algoritmo producen fases de menor y mayor consumo respectivamente. Concretamente la potencia oscila entre los 266,9 vatios en las fases secuenciales y llega hasta los 376,6 vatios en las fases paralelas, donde se emplean los 12 cores.

Finalmente, la Tabla III evalúa la implementación de la factorización QR desde el punto de vista del tiempo (T, en segundos); GLOPS (Gigaflops por segundo); potencia mínima, media y máxima (P_{min}, P_{avg} y P_{max}, respectivamente, en Vatios); y energía total (E_{tot}, en Julios).

IV. Conclusiones

En este artículo se describe y emplea un marco de trabajo capaz de analizar la interacción entre potencia consumida y rendimiento de aplicaciones científicas paralelas MPI y/o multihebra. En este caso, empleando la factorización QR como ejemplo de operación representativa en álgebra lineal densa, se muestran las herramientas hardware y software de este entorno de trabajo. Concretamente, este entorno permite determinar la potencia consumida con un alto grado de granularidad, tanto a nivel de funciones como de fragmentos de código.

En la parte experimental se analiza la implementación usando LAPACK de la factorización QR en plataformas multinúcleo. En este caso, los resultados demuestran la relación directa entre tiempo de ejecución y energía para operaciones de cómputo intensivo en algoritmos de álgebra lineal densa. A consecuencia de esto y, como principio general, una forma directa de optimizar la energía consumida es ajustar la rutina para obtener un máximo redimiento.

Agradecimientos

Los autores de este artículo están apoyados por el proyecto CICYT TIN2011-23283 del *Ministerio de Economía y Competitividad* y FEDER.

Referencias

[1] Shekhar Borkar and Andrew A. Chien, "The future of microprocessors," *Communications of the ACM*, vol. 54, no. 5, pp. 67–77, 2011.

[2] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger, "Dark silicon and the end of multicore scaling," in *Proceedings of the 38th Annual International Symposium on Computer architecture*, 2011, ISCA'11, pp. 365–376.

[3] J. Dongarra et al, "The international ExaScale software project roadmap," *Int. J. of High Performance Computing & Applications*, vol. 25, no. 1, 2011.

[4] M. Duranton et al., "The HiPEAC vision," 2010, Available from `http://www.hipeac.net/roadmap`.

[5] Wu-chun Feng, Xizhou Feng, and Rong Ge, "Green supercomputing comes of age," *IT Professional*, vol. 10, no. 1, pp. 17 –23, 2008.

[6] Ralf Gruber and Vincent Keller, "One Joule per GFlop for BLAS2 Now!," in *AIP Conf. Proceedings*, Simos Theodore E., Psihoyios George, and Tsitouras Ch, Eds. 2010, vol. 1281, pp. 1321–1324, American Institute of Physics.

[7] Thomas Ludwig, "Editorial for the First International Conference on Energy-Aware High Performance Computing," *Computer Science - Research and Development*, vol. 25, no. 3, pp. 123–124, 2010.

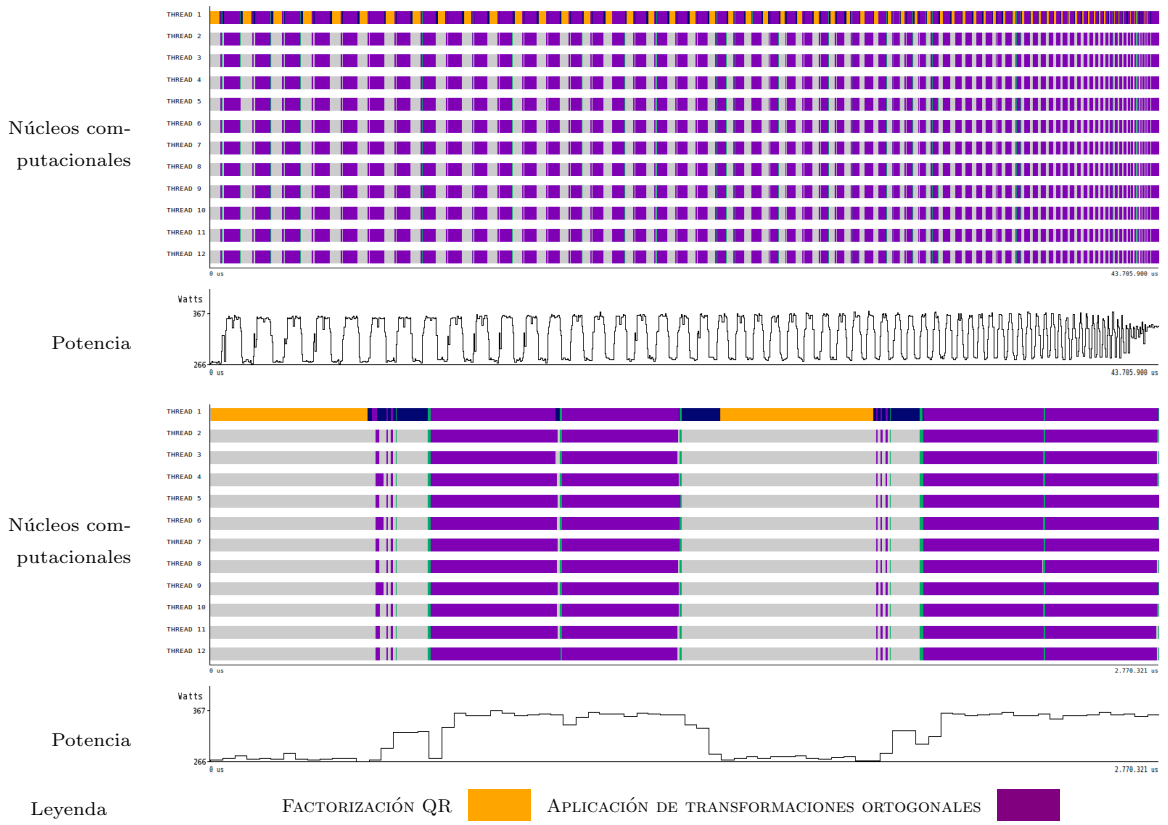

Fig. 3. Traza de LAPACK `dgeqrf`. De arriba a abajo: núcleos computacionales y potencia de la factorización completa; núcleos computacionales y potencia de la factorización de las dos primeras iteraciones del bucle k.

[8] "The Green500 list," 2010, Available at `http://www.green500.org`.

[9] "Paraver project," `http://www.bsc.es/plantillaA.php?cat_id=485`.

[10] S. Wang, H. Chen, and W. Shi, "SPAN: A software power analyzer for multicore computer systems," *Sustainable Computing: Informatics and Systems*, vol. 1, no. 1, pp. 23–34, 2011.

[11] Daniel Bedard, Allan Porterfield, Rob Fowler, and Min Yeol Lim, "PowerMon 2: Fine-grained, integrated power measurement," Tech. Rep. TR-09-04, RENCI, North Carolina, 2009.

[12] Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and Kirk W. Cameron, "PowerPack: Energy profiling and analysis of high-performance systems and applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 99, pp. 658–671, 2009.

[13] Julian Kunkel, "HDTrace - a tracing and simulation environment of application and system interaction," Tech. Rep. 2, Department of Informatics, Scientific Computing. Universität Hamburg, 2011.

[14] Timo Minartz, Daniel Molka, Julian Kunkel, Michael Knobloch, Michael Kuhn, and Thomas Ludwig, *Tool environments to measure power consumption and computational performance*, Chapman and Hall/CRC Press Taylor and Francis Group LLC, 2011.

[15] "SMP superscalar project," `http://www.bsc.es/plantillaG.php?cat_id=385`.

[16] Gene H. Golub and Charles F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 3rd edition, 1996.

[17] E. Anderson, Z. Bai, J. Demmel, J. E. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. E. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide*, SIAM, Philadelphia, 1992.

[18] Peter Strazdins, "A comparison of lookahead and algorithmic blocking techniques for parallel matrix factorization," Tech. Rep. TR-CS-98-07, Department of Computer Science, The Australian National University, Canberra 0200 ACT, Australia, 1998.