Consumo Energético y Velocidad en Plataformas CPU-GPU de Algoritmos Paralelos Multi-objetivo para Selección de Características de EEGs

Juan José Escobar [1], Julio Ortega [1], Antonio F. Díaz [1], Jesús González [1] y Miguel Damas [1]

Resumen— **Muchas aplicaciones de minería de datos y aprendizaje automático incluyen tareas con tipos distintos de paralelismo, pudiendo beneficiarse enormemente de plataformas heterogéneas que incluyen microprocesadores con núcleos superescalares y Graphic Processing Units (GPU), y que denominaremos plataformas CPU-GPU. Las mejoras que se pueden conseguir no solo se refieren al tiempo de ejecución de la aplicación, sino también a su consumo energético. En este artículo se propone una implementación maestro-trabajador, en una plataforma CPU-GPU, de un procedimiento evolutivo multi-objetivo con subpoblaciones para la selección de características en problemas de clasificación de electroencefalogramas (EEG) e interfaces cerebro-computador (BCI). El procedimiento distribuye dinámicamente las subpoblaciones entre los núcleos de la CPU y la GPU, aprovechando el paralelismo de datos de la GPU en las subpoblaciones asignadas a ella. Se ha analizado el comportamiento del procedimiento no sólo en cuanto a la velocidad, sino también en relación con su consumo energético, comparándose además con un procedimiento basado en la distribución de individuos.**

Palabras clave— **Algoritmos Evolutivos Paralelos con Subpoblaciones, BCI, Clasificación de EEG, Computación Sensible a la Energía (Energy-aware Computing), Optimización Multi-objetivo, Distribución Dinámica, Selección de Características**

I. Introducción

MUCHAS aplicaciones de minería de datos y aprendizaje automático implican resolver problemas de clasificación y/o *Clustering* de patrones de dimensionalidad muy elevada, y problemas de optimización complejos que se abordan a través de algoritmos evolutivos. Estos algoritmos se basan en la mejora, iteración tras iteración (generación tras generación), de una población de soluciones cuya calidad (*fitness*) debe evaluarse en cada iteración para seleccionar las que, aplicando distintos operadores genéticos (selección, mutación y cruce), permiten obtener la población de la nueva iteración. Así, el coste computacional de un algoritmo evolutivo proviene de evaluar la calidad de las soluciones de la población y de aplicar los operadores genéticos. Por tanto, crece con el tamaño de la población. Además, en los problemas de minería de datos y aprendizaje automático que consideramos en este trabajo, el coste de la evaluación de la calidad de las soluciones no solo está relacionado con que haya que evaluarlo para cada una de las soluciones de la población, sino que, además, el propio cómputo de la calidad para una solución dada requiere un tiempo considerable. Por ello, en una aproximación evolutiva a este tipo de problemas, el paralelismo no solo debe aprovecharse para evaluar simultáneamente la calidad de cada una de las soluciones de la población, sino también para acelerar la obtención de la calidad de cada solución. De esa forma, el aprovechamiento de plataformas heterogéneas desde el punto de vista del paralelismo que implementan constituye una línea de trabajo que puede resultar muy fructífera, no solo por la aceleración de las aplicaciones, sino también en el ámbito del desarrollo de procedimientos que tengan en cuenta el consumo energético de la aplicación.

Este artículo, por un lado, proporciona un algoritmo evolutivo paralelo basado en subpoblaciones que se distribuyen dinámicamente entre los núcleos de una CPU multi-núcleo y de una GPU. Además, en la evaluación de la calidad de las soluciones de las subpoblaciones asignadas a la GPU se aprovecha el paralelismo de datos que ésta implementa. Precisamente, la aplicación de clasificación de EEG que se aborda aquí, implica patrones (cada patrón es un EEG) cuyo número es mucho menor que el de sus dimensiones (número de componentes o características de cada EEG), y las técnicas de selección de características que deben aplicarse, cuando se abordan a través de algoritmos evolutivos, implican procedimientos muy costosos de evaluación de la calidad de las soluciones de la población (selecciones de características). En trabajos previos [1], [2], [3], hemos descrito implementaciones paralelas maestro-trabajador que aprovechan el paralelismo funcional de datos de una GPU para acelerar la clasificación de EEG en tareas de BCI [4], optimizando el uso de la memoria para aprovechar la coalescencia de memoria y reduciendo los conflictos de acceso a los bancos de memoria local de la GPU [2], [3]. El procedimiento que proponemos aquí distribuye subpoblaciones de soluciones tanto entre los núcleos de la CPU como en la GPU. Estas subpoblaciones evolucionan de forma independiente durante una serie de generaciones, tras las que se comunican e intercambian información para redistribuirse nuevamente.

Por otro lado, la evaluación y comparación de las ejecuciones en las opciones implementadas se ha realizado, no solo en términos de la aceleración que

[1]Departamento de Arquitectura y Tecnología de Computadores, CITIC, Universidad de Granada. E-mails: {jjescobar, jortega, afdiaz, jesusgonzalez, mdamas}@ugr.es

las distintas alternativas paralelas proporcionan, sino también en cuanto a su consumo energético. El desarrollo de técnicas de programación paralela para el ahorro energético (*Energy-aware Computing*) constituye una línea de trabajo de enorme interés por razones económicas y ambientales. Desarrollar procedimientos paralelos energéticamente eficientes debe ser un objetivo igual de importante que el de optimizar la ganancia de velocidad, de forma que consumo energético y tiempo de procesamiento deberían tenerse en cuenta a la par a la hora de distribuir la carga de trabajo en un programa paralelo. Hasta donde sabemos, no existen artículos donde se comparen los consumos energéticos de los algoritmos paralelos evolutivos en plataformas heterogéneas. En [5] se analiza el consumo energético en distintas plataformas (no heterogéneas) de un algoritmo evolutivo secuencial para un problema específico (el problema del multiplexor) que no requiere una complejidad elevada para el cálculo de la calidad de las soluciones. El mencionado artículo se centra más en la eficiencia energética de las distintas plataformas que en estudiar la eficiencia energética de algoritmos distintos, pero muestra la necesidad de tener en cuenta criterios de consumo energético en el diseño de algoritmos genéticos.

A continuación, la Sección II describe nuestra aproximación multi-objetivo a la selección de características en problemas de clasificación. La Sección III describe el procedimiento paralelo basado en subpoblaciones que se propone para su implementación en plataformas heterogéneas CPU-GPU, mientras que la Sección IV muestra los resultados experimentales obtenidos. Finalmente, las conclusiones del trabajo se dan en la Sección V.

II. Selección Multi-objetivo de Características

Los procedimientos paralelos que se proponen se aplican a problemas de clasificación de EEG en tareas de BCI. Para caracterizar un EEG suele utilizarse un número muy elevado de características (componentes) debido a la baja relación señal-ruido, la necesidad de representar la evolución temporal de las señales de los electrodos y el carácter no estacionario de las señales. Por otra parte, dado que el número de EEG disponibles para ajustar el clasificador es usualmente menor que el de características (generalmente debido al coste experimental asociado a su obtención), tendremos un problema de clasificación con pocos patrones de dimensión elevada y nos enfrentamos a la denominada "maldición de la dimensionalidad" (*Curse of Dimensionality*). La selección de un conjunto de características que no sean ruidosas, redundantes o irrelevantes permitiría evitar dicho problema y disminuir el coste asociado al entrenamiento del clasificador.

La Figura 1 muestra el esquema del procedimiento multi-objetivo de selección de características, de tipo *wrapper*, que se ha paralelizado en este trabajo. Un procedimiento de optimización multi-objetivo busca

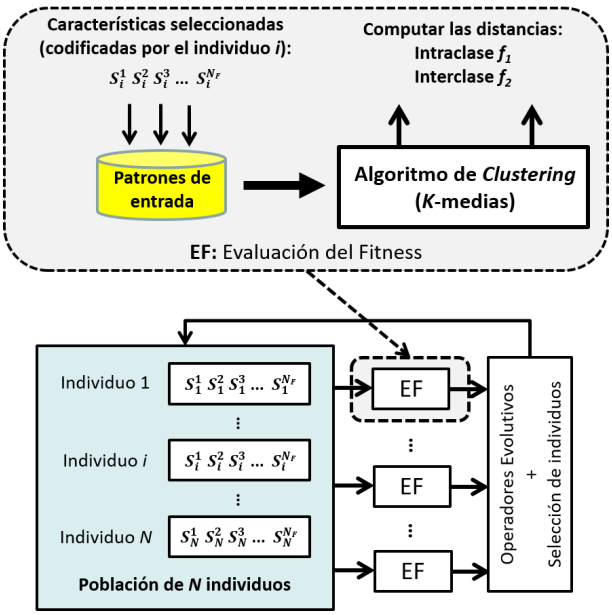

Fig. 1. Procedimiento de tipo *wrapper* para selección multi-objetivo de características

el vector de variables de decisión $S \in \Re^N$, $S = (s_1, s_2, ..., s_N)$, que satisface un conjunto de restricciones, $g(S) \leq 0, h(S) = 0$, y optimiza el vector $F(S) = (f_1(S), f_2(S), ..., f_M(S))$ cuyos componentes representan las funciones objetivo a optimizar. Dado que los objetivos están usualmente en conflicto, en lugar de obtener una única solución, la optimización multi-objetivo busca un conjunto de soluciones conocidas con el nombre de soluciones óptimas de *Pareto*, entre las que el usuario puede escoger la más adecuada en cada caso. Las soluciones óptimas de Pareto definen el denominado frente de Pareto, en el que no existe ninguna solución peor que otra cuando se tienen en cuenta todos los objetivos: las denominadas soluciones *no-dominadas*.

En la aplicación de clasificación de EEG considerada, los objetivos se definen a partir de las prestaciones del *Clustering* obtenido con la correspondiente selección de características (esto es, un individuo de la población del algoritmo evolutivo). Así, a partir de los N_P patrones de N_F características de la base de datos, DS, se construye el conjunto de patrones de entrenamiento del algoritmo de *Clustering* tomando como componentes de cada patrón aquellas de las N_F características que indica la selección de características considerada. El algoritmo de *Clustering* que utilizamos en este artículo es el conocido K-medias, que permite determinar, para el conjunto de patrones $P = (p_i^1, ..., p_i^{N_F}); \forall i = 1, ..., N_P$, los centroides $K^t = (k_1, ..., k_j); \forall j = 1, ..., W$ clusters en que se agrupan los patrones (este número es conocido en nuestro problema y es igual al número de clases). Una vez se han determinado los centroides, cada patrón se asigna al *cluster* de su centroide más cercano y se obtienen los valores de las dos funciones objetivo utilizadas, f_1 y f_2, a partir de, respectivamente, la distancia intraclase y la distancia interclase del *Clustering* obtenido (tal y como se describe

en [2]). Así, la evaluación de las funciones de coste de cada individuo requiere un tiempo de cómputo considerable. Por ejemplo, en el código secuencial de nuestra aplicación, la evaluación de las funciones de coste representa un 99,93% del tiempo total para una población de 120 individuos, y un 97,36% para una población de 30.000 individuos. Las ventajas de una formulación multi-objetivo del problema de selección de características varían según si la clasificación es supervisada o no supervisada [6], [7]. En una clasificación supervisada, usualmente se intenta maximizar las prestaciones del clasificador junto con la reducción del número de características, dado que a medida que dicho número aumenta son más probables los problemas de sobreajuste (*overfitting*) y se reduce la capacidad de generalización. En los problemas de clasificación no supervisada, como el que abordamos aquí, las medidas de calidad del clustering de patrones que se alcanzan suelen estar sesgadas bien hacia la maximización, o bien hacia la minimización de características, una formulación multi-objetivo puede contrarrestar este efecto.

III. Procedimiento Paralelo Basado en Subpoblaciones para Plataformas CPU-GPU

La mayoría de los trabajos sobre implementaciones paralelas de algoritmos evolutivos en plataformas heterogéneas CPU-GPU se centran en la aceleración que proporciona una GPU [8], [9], [10], [11], [12] con respecto a uno de los núcleos de la CPU. Además, no existen muchos artículos que analicen implementaciones completas de aplicaciones con funciones de *fitness* costosas en cómputo [13]. El Algoritmo 1 describe el procedimiento de distribución de subpoblaciones que proponemos aquí, denominado `D2S_NSGAII` y elaborado en *OpenMP-OpenCL*, que se incluye dentro de la implementación paralela maestro-trabajador del procedimiento de selección multi-objetivo de la Figura 1. En el Algoritmo 1, las hebras creadas por la directiva *parallel* de *OpenMP* lanzan *kernels OpenCL* a los dispositivos CPU y GPU disponibles en la plataforma para evaluar las dos funciones objetivo de cada individuo de la subpoblación correspondiente.

El procedimiento `D2S_NSGAII` toma el conjunto inicial de N_{Spop} subpoblaciones, $Sp_i; \forall i = 1, ..., N_{Spop}$, y las distribuye entre los N_D dispositivos *OpenCL* distintos. Como cada subpoblación incluye M individuos (cada individuo es una selección de características), el número N de individuos en la población total es $N = M \times N_{Spop}$. Las subpoblaciones evolucionan de forma independiente durante un número de generaciones hasta que se produce la migración de soluciones entre subpoblaciones.

En cada generación, se aplican los operadores genéticos a la subpoblación (`UniformCrossover` en la línea 5 del Algoritmo 1) y se llaman a los procedimientos *OpenCL* de evaluación de la solución según se haya asignado la subpoblación a un dispositivo de tipo CPU (`evaluationsCPU` en la línea 7 del Algoritmo 1) o GPU (`evaluationsGPU` en la línea 9). Una descripción más detallada de la implementación de la evaluación de las soluciones en la GPU y de las optimizaciones del uso de la memoria de la GPU se puede encontrar en [2]. Una vez se han evaluado todas las soluciones de una subpoblación, se lleva a cabo el proceso de reemplazo para construir una nueva subpoblación (líneas 11-13 del Algoritmo 1), incluyendo la unión de la subpoblación existente y las nuevas soluciones generadas por los operadores genéticos (línea 11), la ordenación, según la relación de dominancia de las soluciones del nuevo conjunto (`nonDominatedSorting` en la línea 12), y la selección de las nuevas M soluciones que constituyen la nueva subpoblación (línea 13).

Una vez se completa la evolución independiente de cada una de las subpoblaciones durante un número establecido de generaciones, se produce la migración de soluciones entre las distintas subpoblaciones (línea 16 del Algoritmo 1). La migración implica la construcción de un nuevo conjunto de N_{Spop} subpoblaciones, Sp. Para construir una nueva subpoblación, Sp_i, se añaden a las soluciones que constituyen dicha subpoblación la mitad de las soluciones del frente de Pareto que hayan alcanzado el resto de subpoblaciones $Sp_k; \forall k = 1, ..., N_{Spop}, k \neq i$ como mucho. Al conjunto de soluciones obtenido se le aplica el mismo proceso de reemplazo utilizado en las genera-

Algoritmo 1: Pseudocódigo de distribución de subpoblaciones. La evaluación de subpoblaciones se distribuye entre los dispositivos *OpenCL* asignados a hebras *OpenMP*

```
1  Función D2S_NSGAII(Sp, N_D, D, N_Spop, M, DS, K, DS^t)
       Entrada: Las subpoblaciones, Sp_i; ∀i = 1, ..., N_Spop
       Entrada: Número de dispositivos OpenCL, N_D
       Entrada: Dispositivos OpenCL, D_j; ∀j = 1, ..., N_D
       Entrada: Número de subpoblaciones N_Spop
       Entrada: Individuos en cada subpoblación, M
       Entrada: DS: N_P patrones de N_F características
       Entrada: Conjunto K con los centroides
       Entrada: DS^t: Conjunto DS ordenado por columnas
       Salida : S, la nueva solución para el problema
2      repetir
           // Sección OpenMP paralela (N_D dispositivos)
3          repetir
               // Proceso evolutivo. H tiene a los hijos
4              repetir
5                  H ← UniformCrossover(Sp_i)
6                  si D_j es la CPU entonces
7                      H ← evaluationsCPU(H, M, DS, K)
8                  en otro caso
9                      H ← evaluationsGPU(H, M, DS, K, DS^t)
10                 fin
                   // Reemplazo. A es un array Auxiliar
11                 A ← Unir Sp_i y H en un único array
12                 A ← nonDominatedSorting(A, M + N_H)
13                 Sp_i ← Primeros M individuos de A
14             hasta que se supera el número de generaciones de subpoblaciones;
15         hasta que las N_Spop subpoblaciones son evaluadas;
16         Sp ← migration(Sp, N_Spop, M)
17     hasta que se realizan las migraciones deseadas;
       // Proceso de recombinación
18     Sp ← nonDominatedSorting(Sp, N_Spop × M)
19     S ← Primeros M individuos de Sp
20     devolver S
21 End
```

ciones de evolución independiente de las subpoblaciones (líneas 11 y 12).

Hay que tener en cuenta que, en esta implementación maestro-trabajador, tanto los operadores genéticos como las migraciones son implementados en uno de los núcleos de CPU. Los *kernels* que se ejecutan en la GPU precisan que los patrones necesarios para la evaluación de los individuos sean transferidos desde la memoria del *host* a la memoria de la GPU. Esto se realiza al comienzo del procedimiento. Además, en cada generación, deben comunicarse, entre CPU y *host*, los individuos de las subpoblaciones y los valores de sus funciones objetivo (éstos desde la GPU al *host*). Estas transferencias de información, a través de un bus con peor latencia y ancho de banda que los buses del *host*, constituyen un aspecto a tener en cuenta en la mejora de prestaciones que aporta la GPU. En cualquier caso, dado que las subpoblaciones se asignan dinámicamente a los dispositivos disponibles en la plataforma, en muchos casos se consigue un solapamiento del tiempo de transferencia de información entre *host* y GPU.

IV. Resultados Experimentales

En esta sección analizamos las prestaciones de nuestros códigos *OpenMP-OpenCL* en uno de los nodos de un computador NUMA con red Gigabit Ethernet, y Linux CentOS 6.7. El nodo utilizado dispone de dos procesadores Intel Xeon E5-2640 v4 a 2,1 GHz, ocho núcleos por procesador con dos hebras por núcleo, y 32 GBytes de memoria DDR3. También incluye una GPU Tesla K40m con 12 GB de memoria global, 288 GBytes/s de ancho de banda de memoria máximo y 2.880 núcleos CUDA distribuidos en 15 SMXs (*Stream Multiprocessors*) a 745 MHz. Para medir la potencia y el consumo de energía de cada uno de los nodos, hemos desarrollado un sistema de adquisición basado en *Arduino Mega* (Figura 2) que proporciona, en tiempo real, cuatro medidas por segundo de la potencia y la energía consumida por cada uno de los nodos.

Los conjuntos de datos que se han utilizado pertenecen al laboratorio de BCI de la Universidad de Essex, cuya descripción puede encontrarse en [14]. En concreto hemos utilizado conjuntos de entrenamiento y test de 178 patrones (178 EEGs) con 3.600 características por patrón (12 características por segmento temporal y electrodo, con 20 segmentos y 15 electrodos por EEG). Aunque se han utilizado datos pertenecientes a tres sujetos designados como 104, 107 y 110, los resultados que se muestran en este artículo corresponden al sujeto 110 dado que el comportamiento observado en los otros sujetos es análogo y deriva en las mismas conclusiones.

El algoritmo evolutivo que se ha utilizado es *NSGA-II* [15], uno de los más referenciados y usados para abordar aplicaciones con una aproximación multi-objetivo. En cualquier caso, los procedimientos paralelos que se proponen también podrían implementarse en otros algoritmos que evolucionen una población de soluciones. El algoritmo aplica un operador de cruce en dos puntos con una probabilidad de 0,9, mutación mediante cambio del bit seleccionado con una probabilidad de 0,1, y selección mediante torneo binario. Dado que los valores mínimos para las funciones objetivo f_1 y f_2 son respectivamente 0 y -1, y puesto que se ha tomado como punto de referencia el (1,1), el valor máximo que puede alcanzar el hipervolumen es 2. Cada uno de los experimentos se ha repetido 10 veces para analizar si las diferencias observadas son estadísticamente significativas, mediante los test de Kolmogorov-Smirnov y Kruskal-Wallis. En este artículo solo nos centraremos en los tiempos de ejecución y los consumos energéticos de los procedimientos paralelos implementados. En cuanto a la calidad de las soluciones obtenidas, baste decir que los frentes de Pareto obtenidos corresponden a hipervolúmenes promedio de entre 1,893 y 1,972, con desviaciones típicas entre 0,001 y 0,01. Dado que se trata de valores de hipervolumen similares a los obtenidos por el algoritmo secuencial con una única población, podemos concluir que, a pesar de que los algoritmos paralelos basados en subpoblaciones implementados no tienen las mismas características que el algoritmo secuencial, obtienen soluciones de calidad similar.

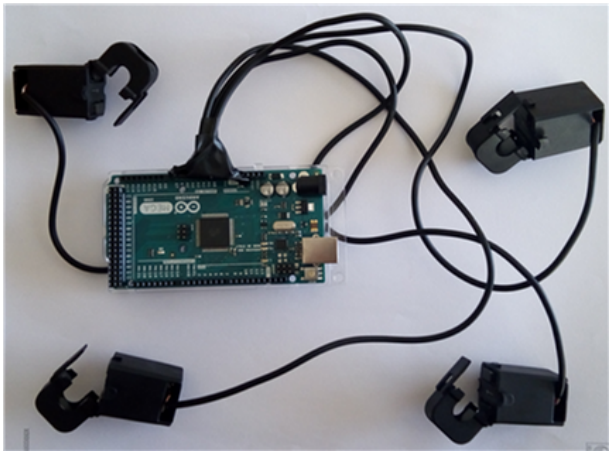

Fig. 2. Sistema basado en *Arduino Mega* para la medida de consumo de energía en los nodos

La Figura 3 muestra las ganancias de velocidad respecto a la ejecución secuencial del procedimiento correspondiente en un único núcleo de uno de los microprocesadores (núcleo de CPU). Se ha utilizado una población de 480 individuos distribuida en 1, 2, 4, 8 y 16 subpoblaciones, de 480, 240, 120, 60 y 30 individuos, respectivamente, y se han ejecutado 60 generaciones. También se han considerado ejecuciones del procedimiento paralelo de subpoblaciones con 1, 2, 3, 4 y 5 migraciones (respectivamente, cada 60, 30, 20, 15 y 12 generaciones de evolución independiente de las subpoblaciones). La Figura 3.a muestra las ganancias cuando las subpoblaciones se distribuyen únicamente entre los SMXs de la GPU, la Figura 3.b corresponde al uso únicamente de los núcleos de la CPU, y la 3.c a la distribución de subpoblaciones tanto entre los núcleos de la CPU como entre los SMXs de la GPU. En los tres casos se observa una

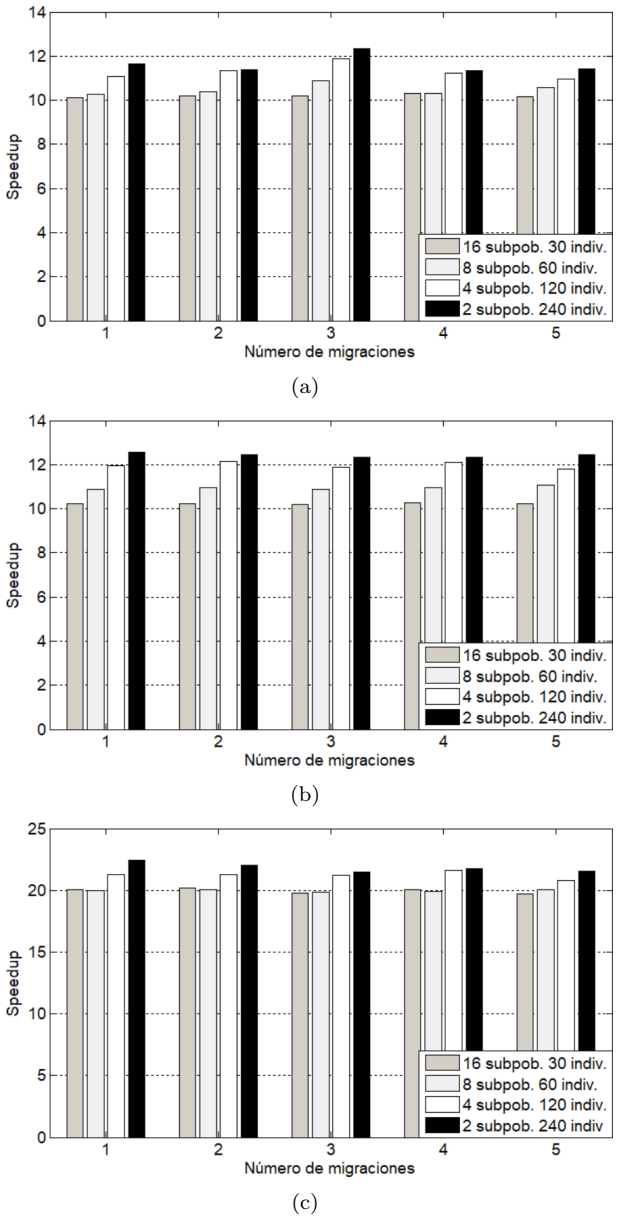

Fig. 3. Ganancias de velocidad para 480 soluciones distribuidas en 2, 4, 8 y 16 subpoblaciones, con distintas configuraciones de plataformas heterogéneas: (a) GPU (15 SMXs); (b) CPU (32 núcleos); (c) CPU + GPU

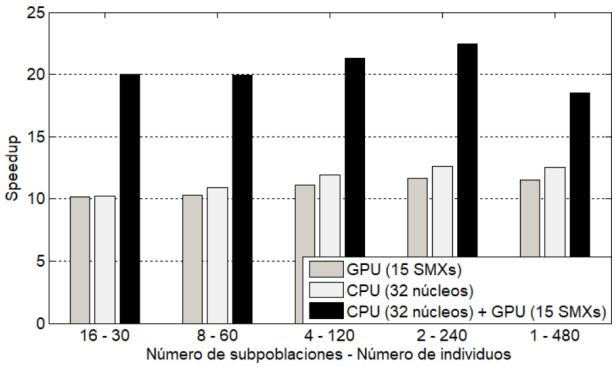

Fig. 4. Comparación de las ganancias de velocidad para distintas subpoblaciones, individuos y configuraciones

cierta mejora de dichas ganancias (*speedups*) para subpoblaciones con un mayor número de individuos y, por lo tanto, menos subpoblaciones. En cuanto al efecto de las migraciones, se observa que, dado un número de subpoblaciones, el *speedup* se mantiene aproximadamente constante al cambiar el número de migraciones. Hay que tener en cuenta que cada migración implica el envío de la mitad de los individuos del frente de Pareto obtenido por cada subpoblación a las demás subpoblaciones. Por tanto, el coste de cada migración aumentaría con el número de individuos en la subpoblación y con el número de subpoblaciones. No obstante, hay que tener en cuenta que, dado que estamos comparando ejecuciones que corresponden al mismo número de individuos, si aumenta el número de subpoblaciones disminuye el de individuos en cada población. Por otra parte, en realidad las comunicaciones se realizan a través de accesos a la memoria compartida que contiene la información correspondiente a individuos de las subpoblaciones. Además, hay que recordar que al aumentar el número de migraciones disminuye el número de generaciones durante las que cada subpoblación evoluciona independientemente para que el número total de generaciones de todos los algoritmos sea el mismo. Esto explicaría que, como muestra la Figura 3, para un número fijo de subpoblaciones no se observen cambios significativos en las ganancias obtenidas con distinto número de migraciones.

El efecto dominante desde el punto de vista de los cambios observados en los *speedups* proviene del número se subpoblaciones y del tamaño de ellas dado que estamos considerando constante la población total (si se aumenta el número de subpoblaciones disminuye el de individuos en la subpoblación). Cuanto mayor sea el número de subpoblaciones mayor es el número de entradas al *kernel* CPU o GPU, donde la subpoblación es evaluada. Además, dado que el dispositivo se encarga de distribuir los individuos de la subpoblación entre los núcleos de la CPU o de la GPU, si las subpoblaciones tienen menos individuos es posible que haya mayor desequilibrio en la carga de trabajo de la subpoblación. Es decir, cuanto mayores sean las subpoblaciones (teniendo en cuenta que el tamaño de la población total es constante) más fácilmente se distribuirán sus individuos entre los núcleos de la CPU o de la GPU.

Por otro lado, comparando las Figuras 3.a, 3.b y 3.c se observa que los *speedups*, cuando se usan los 15 SMXs de las GPU son muy similares a los obtenidos cuando se utilizan los 32 núcleos de la CPU. Los 15 SMXs consiguen una aceleración casi igual a la que se alcanza con los 32 núcleos de la CPU debido a que en la GPU se está aprovechando el paralelismo de datos asociado al cálculo de las funciones de coste gracias a los núcleos *CUDA*. El efecto del uso conjunto de los núcleos de la CPU y de los SMXs de la GPU se pone de manifiesto en la Figura 3.c y también en la Figura 4, donde se muestran, para el caso de una migración, los *speedups* para las distintas configuraciones de subpoblaciones. Como se puede ver, el incremento de *speedup* es considerable con respecto al uso de solo núcleos CPU o solo de GPU, casos para los que la Figura 4 muestra niveles de ganancia

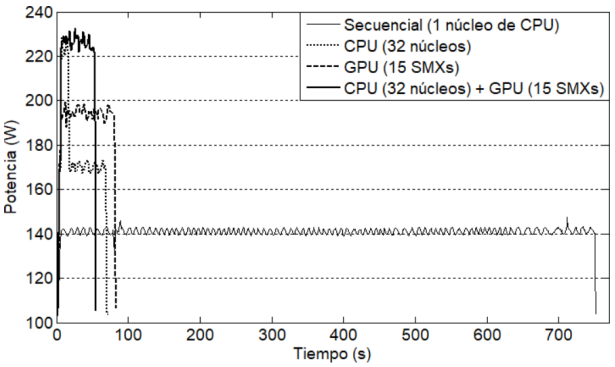

Fig. 5. Consumo de potencia instantánea durante la ejecución en cuatro configuraciones diferentes y 600 generaciones: Secuencial (1 núcleo); CPU (32 núcleos); GPU (15 SMXs); CPU + GPU

de velocidad parecidos. Cuando se tiene una única población, cada vez que entra el *kernel* de CPU y el de GPU toman, respectivamente 32 y 15 individuos (en este caso individuos de toda la población) con lo que se producirían entre 15 y 32 entradas al *kernel* correspondiente ($480/32 = 15$ y $480/15 = 32$). Por esta razón, tal y como muestra la Figura 4, la mejora de velocidad crece a medida que disminuye el número de subpoblaciones salvo para el caso de una única población. El efecto es mayor cuando se tienen las dos hebras que asignan carga al *kernel* de CPU y GPU respectivamente, dado que además dichas hebras deben competir por el núcleo donde se ejecutan.

La Figura 5 ilustra las medidas de potencia instantánea realizadas durante la evolución, para el caso de 2 subpoblaciones de 240 individuos, del procedimiento secuencial y de los procedimientos paralelos en cada una de las alternativas de uso de la plataforma considerada (solo núcleos de CPU, solo GPU, y núcleos de CPU y GPU conjuntamente). Se observa menor consumo de potencia en la ejecución secuencial y en la de GPU, frente al uso de los núcleos de la CPU, aunque hay que tener en cuenta que el número de núcleos de CPU usados es igual a 32. La Figura 5 también pone de manifiesto los tiempos de comienzo y final de cada una de las alternativas. En cualquier caso, el consumo energético depende tanto de la potencia consumida como del tiempo durante el cual se está consumiendo dicha potencia.

La Tabla I muestra, para las distintas subpoblaciones y configuraciones del hardware disponible, las medidas de tiempo y energía consumida durante la ejecución junto con los porcentajes del tiempo de ejecución y de la energía consumida con respecto a las del programa secuencial. En las medidas de energía no se ha tenido en cuenta la energía que el sistema consumiría en estado de reposo o *idle*. Como se puede observar, salvo en el caso de utilizar una única población, los mejores resultados para el consumo energético corresponden a la ejecución que utiliza núcleos de la CPU y la GPU. Aunque los niveles de potencia instantánea son mayores, la ganancia conseguida hace que el balance final respecto a la energía sea positivo. Hay que tener en cuenta que en las medidas de consumo proporcionadas se incluyen

TABLA I
MEDIDAS DE TIEMPO DE EJECUCIÓN T(S) Y ENERGÍA CONSUMIDA $E(W \times h)$ PARA DISTINTAS EJECUCIONES DEL PROCEDIMIENTO PARALELO PROPUESTO

N_{Spop}	M	Modo	T	E	% Tiempo Secuencial	% Energía Secuencial
1	480	Seq.	83,08	0,8	-	-
		Tesla	7,52	0,36	9,07	**45,66**
		Xeon	6,55	0,46	7,89	56,21
		Het.	4,97	0,41	**5,99**	50,34
2	240	Seq.	89,36	0,91	-	-
		Tesla	7,64	0,37	8,56	40,88
		Xeon	6,53	0,42	7,31	45,88
		Het.	4,12	0,34	**4,62**	**40,26**
4	120	Seq.	83,13	0,85	-	-
		Tesla	7,92	0,38	9,53	46,14
		Xeon	6,86	0,42	8,25	49,61
		Het.	4,19	0,34	**5,04**	**39,93**
8	60	Seq.	82,85	0,86	-	-
		Tesla	8,51	0,42	10,27	48,74
		Xeon	7,42	0,41	8,96	47,76
		Het.	4,58	0,36	**5,53**	**42,62**
16	30	Seq.	79,61	0,84	-	-
		Tesla	8,59	0,43	10,78	51,74
		Xeon	8,12	0,44	10,2	52,8
		Het.	4,45	0,37	**5,58**	**44,44**

todos los elementos del nodo, en particular el consumo de las memorias y los buses que permiten la conexión con la GPU. A pesar de ello, son aparentes las diferencias que se observan según el tipo de arquitectura de procesamiento que se esté utilizando. Al comparar las medidas de ganancia de velocidad de las distintas configuraciones podría argumentarse que la comparación directa no es muy justa dado que el número de núcleos utilizado en cada caso es muy diferente: 32 núcleos de CPU, 15 SMXs en la GPU, y 32 núcleos de CPU más 15 SMXs de la GPU en la configuración heterogénea. Sin embargo, si dividimos la ganancia de velocidad entre la energía necesaria en cada plataforma para obtener dicha ganancia, sí tendríamos un criterio equitativo para comparar las distintas opciones. Así, la Tabla I también proporciona el cociente entre la ganancia de velocidad que se alcanza con cada configuración y la energía consumida para alcanzar la misma. Esta medida de eficacia energética manifiesta claramente los beneficios de utilizar la plataforma heterogénea.

V. Conclusiones

En este artículo se propone y evalúa una implementación heterogénea paralela, basada en subpoblaciones, de un procedimiento multi-objetivo de selección de características. El procedimiento paralelo aprovecha tanto la CPU multi-núcleo como las GPUs disponibles en un nodo de computador NUMA. La aplicación que se considera, como muchas otras típicas de la minería de datos y el aprendizaje automático, se basa en un algoritmo de optimización (en este caso de optimización evolutiva multi-objetivo) que consume la mayor parte de su tiempo en el cálculo de las funciones objetivo (*fitness*) de los individuos de la población de soluciones que evoluciona. De hecho, se ha comprobado experimentalmente que más del 90% del tiempo de cómputo corresponde a esta etapa. Esto implica que,

más que detenerse en la paralelización de los operadores evolutivos, debe buscarse aprovechar el paralelismo disponible en el cálculo de las funciones objetivo. Para ello, en nuestra implementación no solo se distribuyen las subpoblaciones entre los núcleos de la CPU y los SMXs de la GPU, sino que también se aprovecha el paralelismo de datos que ofrece la GPU para acelerar la aplicación del algoritmo K-medias (en el que se basan las funciones objetivo en nuestro caso) a cada uno de los individuos de cada subpoblación. Las alternativas de ejecución del procedimiento propuesto (considerando distinto número de subpoblaciones y número de migraciones entre subpoblaciones) se han evaluado desde el punto de vista de la ganancia de velocidad y del consumo de energía que ocasionan con respecto a la ejecución en un solo núcleo, y se han comparado con una implementación que únicamente distribuye los individuos de una única población.

Los resultados ponen de manifiesto las ventajas que supone el aprovechamiento de las arquitecturas heterogéneas CPU-GPU, no solo desde el punto de vista de la aceleración que proporcionan al involucrar más núcleos de cómputo (de un tipo u otro) sino también desde el punto de vista de su consumo energético. De hecho, esta plataforma también proporciona la mejor relación entre ganancia de velocidad y energía consumida para alcanzarla.

Entre las líneas a explorar en nuestros trabajos futuros está el desarrollo de una implementación de paso de mensajes que permita aprovechar los recursos de las CPUs y GPUs de todos los nodos del computador NUMA y su análisis, no solo desde el punto de vista de la aceleración que proporcione sino también de su eficiencia energética. También se considerará la implementación de procedimientos coevolutivos y funciones objetivo asociadas a nuevas alternativas de clasificación, entre ellas las correspondientes a redes neuronales (*deep learning*).

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad y los fondos ERDF a través del proyecto TIN2015-67020-P. También agradecemos al profesor John Q. Gan el acceso a las bases de datos del laboratorio de BCI de la Universidad de Essex.

Referencias

[1] J.J. Escobar, J. Ortega, J. González, and M. Damas, "Assessing parallel heterogeneous computer architectures for multiobjective feature selection on eeg classification," in *Proceedings of the 4th International Conference on Bioinformatics and Biomedical Engineering*, F. Ortuño and I. Rojas, Eds., Granada, Spain, April 2016, IWBBIO'2016, pp. 277–289, Springer.

[2] J.J. Escobar, J. Ortega, J. González, and M. Damas, "Improving memory accesses for heterogeneous parallel multi-objective feature selection on eeg classification," in *Proceedings of the 4th International Workshop on Parallelism in Bioinformatics*, Grenoble, France, August 2016, PBIO'2016, pp. 372–383, Springer.

[3] J.J. Escobar, J. Ortega, J. González, M. Damas, and B. Prieto, "Issues on gpu parallel implementation of evolutionary high-dimensional multi-objective feature selection," in *Proceedings of the 20th European Conference on Applications of Evolutionary Computation, Part I*, Amsterdam, The Netherlands, April 2017, EVOSTAR'2017, pp. 773–788, Springer.

[4] R. Rupp, S.C. Kleih, R. Leeb, J. del R. Millan, A. Kübler, and G.R. Müller-Putz, "Brain-computer interfaces and assistive technology," in *Brain-Computer-Interfaces in their Ethical, Social and Cultural Contexts*, G. Grübler and E. Hildt, Eds., The International Library of Ethics, Law and Technology, pp. 7–38. Springer, 2014.

[5] F. Fernández-de-Vega, F. Chávez, J. Díaz, J.A. García, P.A. Castillo, J.J. Merelo, and C. Cotta, "A cross-platform assessment of energy consumption in evolutionary algorithms," in *Proceedings of the 14th International Conference on Parallel Problem Solving from Nature*, Edinburgh, UK, September 2016, PPSN'2016, pp. 548–557, Springer.

[6] D. Kimovski, J. Ortega, A. Ortiz, and R. Baños, "Leveraging cooperation for parallel multi-objective feature selection in high-dimensional eeg data," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 18, pp. 5476–5499, 2015.

[7] J. Handl and J. Knowles, "Feature subset selection in unsupervised learning via multiobjective optimization," *International Journal of Computational Intelligence Research*, vol. 2, no. 3, pp. 217–238, 2006.

[8] P. Collet, "Why gpgpus for evolutionary computation?," in *Massively Parallel Evolutionary Computation on GPGPUs*, S. Tsutsui and P. Collet, Eds., Natural Computing Series, pp. 3–14. Springer, 2013.

[9] T.V. Luong, N. Melab, and E-G. Talbi, "Gpu-based island model for evolutionary algorithms," in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, Portland, OR, USA, July 2010, GECCO'2010, pp. 1089–1096, ACM.

[10] P. Jähne, "Overview of the current state of research on parallelisation of evolutionary algorithms on graphic cards," in *GI-Jahrestagung*, Bonn, Germany, September 2016, INFORMATIK'2016, pp. 2163–2174, LNI.

[11] S. Cavuoti, M. Garofalo, M. Brescia, A. Pescape, G. Longo, and G. Ventre, "Genetic algorithm modeling with gpu parallel computing technology," in *Neural Nets and Surroundings: 22nd Italian Workshop on Neural Nets*, Vietri sul Mare, Salerno, Italy, May 2012, WIRN'2012, pp. 29–39, Springer.

[12] D. Robilliard, V. Marion-Poty, and C. Fonlupt, "Population parallel gp on the g80 gpu," in *Genetic Programming: 11th European Conference*, Naples, Italy, March 2008, EUROGP'2008, pp. 98–109, Springer.

[13] S. Debattisti, N. Marlat, L. Mussi, and S. Cagnoni, "Implementation of simple genetic algorithm within the cuda architecture," in *In GPUs for Genetic and Evolutionary Computation*, Montreal, CAN, USA, July 2008, GECCO'2009, ACM.

[14] J. Asensio-Cubero, J.Q. Gan, and R. Palaniappan, "Multiresolution analysis over simple graphs for brain computer interfaces," *Journal of Neural Engineering*, vol. 10, no. 4, 2013.

[15] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii," in *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, Paris, France, September 2000, PPSN VI, pp. 849–858, Springer.