Optimización con 100 millones de variables reales sobre múltiples unidades de procesamiento gráfico

Alberto Cano[1] and Carlos García-Martínez[2]

[1] Department of Computer Science, Virginia Commonwealth University, USA
acano@vcu.edu
[2] Dept. de Informática y Análisis Numérico. Universidad de Córdoba, España
cgarcia@uco.es

Resumen Muchos problemas de optimización se modelan en base a un gran número de variables de decisión. Las metaheurísticas clásicas consiguen resultados excelentes en problemas de dimensión acotada, pero suelen perder eficacia cuando la dimensión de los problemas crece. Esto se conoce como la *maldición de la dimensionalidad*. Por otro lado, las unidades de procesamiento gráfico actuales se han vuelto bastante atractivas en los últimos años por sus altos niveles de computación paralela. En este trabajo estudiamos el uso de una arquitectura computacional escalable y distribuida con múltiples unidades de procesamiento gráfico para la optimización de funciones de dimensionalidad extrema. En particular, presentamos un modelo de escalado de las funciones del Congreso IEEE de Computación Evolutiva 2013 y resultados con hasta 10^8 variables.

Keywords: Optimización con dimensionalidad extrema, unidades de procesamiento gráfico, escalabilidad

1. Introducción

Las elevadas capacidades computacionales de las unidades de procesamiento gráfico actuales (*GPUs*), y la aparición de entornos de computación masivamente paralela como CUDA [1] han atraído la atención de la comunidad científica y fomentado el tratamiento de problemas cada vez mas grandes [2,3,4,5,6]. Las tarjetas GPU ofrecen un conjunto de multi-procesadores, con muchos núcleos, memoria local, global y compartida. Así, se pueden ejecutar miles de hebras en paralelo sobre diferentes conjuntos de datos. Este tipo de arquitectura es especialmente apropiada para abordar problemas con altos niveles de paralelización de datos.

Recientemente, Lastra y otros [7] extendieron la dimensionalidad de algunas funciones de optimización con parámetros reales hasta considerar 3 millones de variables, lo que representó un incremento de tres órdenes de magnitud con respecto a las funciones consideradas en las competiciones de los congresos IEEE de Computación Evolutiva (CEC) [8,9]. Utilizaron una única GPU para ejecutar

en paralelo las operaciones de dichas funciones, lo cual era posible por ser aditivamente descomponibles; es decir, éstas se pueden expresar como la suma de otras funciones. La limitación del estudio vendría por la memoria relativamente reducida de la GPU, que sólo podía albergar soluciones candidatas de dicha dimensión.

El objetivo de este trabajo es explorar el uso de más de una GPU para la optimización de funciones aditivamente descomponibles de muy elevada dimensionalidad. Para ello, primero presentamos un modelo de escalado para este tipo de funciones sobre varias GPUs, particularizado a las funciones de la competición del CEC 2013. Además, presentaremos una extensión de un algoritmo memético para explotar la disponibilidad de dichas GPUs a la hora de abordar estas funciones. Posteriormente, presentaremos los resultados de aplicar el algoritmo memético sobre las funciones del CEC 2013 con una dimensionalidad de 10^8 variables, lo cual representa un incremento de dos órdenes de magnitud frente al máximo, a nuestro conocimiento, considerado en la literatura [7].

Este trabajo se organiza de la siguiente forma. Primero describimos la propuesta de escalado de funciones aditivamente descomponibles sobre varias GPUs en la Sección 2. Presentamos un algoritmo memético que explota el poder computacional de varias GPUs en la Sección 3. La Sección 4 muestra los resultados de aplicar el algoritmo a funciones con 10^8 variables de decisión. Finalmente, la Sección 5 presenta las conclusiones el trabajo.

2. Modelo de Distribución y Escalado de Funciones Aditivamente Descomponibles sobre Varias GPUs

Se dice que una función es aditivamente descomponible si puede definirse como la suma de funciones componentes tal que cada una depende de un subconjunto de las variables de entrada [10]. Muchos de los problemas que aparecen en la literatura son aditivamente descomponibles, como los problemas NK-landscape, algunas instancias del problema de la asignación cuadrática (QAP), o las funciones de la competición del CEC 2013 [9]. En esta sección presentamos un modelo de distribución de tareas, representación de soluciones candidatas y evaluación de estas soluciones para funciones aditivamente descomponibles que explota la disponibilidad de múltiples tarjetas GPU. Primero describimos cómo se llevan a cabo estos procesos sobre una única GPU para la evaluación de P soluciones candidatas (Sección 2.1). En la Sección 2.2 comentamos cómo extender el modelo a múltiples GPUs, las posibilidades y desventajas. Finalmente, en la Sección 2.3 proponemos un modelo de escalado de las funciones del CEC 2013 mediante la descripción de un ejemplo particular. Debemos indicar que de forma similar propusimos un modelo que explotaba la propiedad descomponible de estas funciones sobre el paradigma Map-Reduce en [11].

2.1. Distribución de Tareas sobre una GPU

En la mayoría de los casos, una solución se representa como un vector de D variables. Para la evaluación de las P soluciones de la población de un algoritmo

evolutivo podemos cargar sus valores en una estructura de vectores para optimizar el rendimiento y el patrón de acceso a las variables. El acceso a la variable j de la solución i se calcula como un desplazamiento sobre el inicio de la estructura de $i \cdot D + j$. Este patrón de acceso permite la coalescencia de los accesos a memoria y maximiza el ancho de banda de la transferencia de memoria. El principal problema es que representar P soluciones de D variables con representación de punto flotante y doble precisión requiere $P \cdot D \cdot 8$ bytes. Consecuentemente, la dimensionalidad de la función está limitada a unos cuantos millones de variables si se usa la memoria de una única GPU. Además, hay que tener en cuenta que la función a evaluar puede requerir la carga de información adicional para ser evaluada, como matrices de rotación o definición de subfunciones componente.

En la GPU, los procesos de generación de la población inicial y la aplicación de los operadores genéticos pueden beneficiarse de dos niveles de paralelización: a nivel de población y a nivel de genes. En el primero, las subtareas, como la generación de los valores iniciales o de los valores de los genes de los descendientes, se aplican de forma concurrente para cada solución de la población P. En el segundo, si dichos procesos no establecen relaciones entre las variables de una misma solución, como suele ocurrir en la inicialización u operadores de cruce para soluciones con representación en punto flotante, las subtareas pueden ejecutarse concurrentemente para cada variable D de las soluciones. Por tanto, los mencionados procesos pueden llevarse a cabo por hasta $P \cdot D$ hebras que se ejecutan potencialmente en paralelo sobre los núcleos de una GPU.

Por otro lado, la evaluación de las soluciones puede requerir un tiempo de ejecución desmesurado cuando crece la dimensionalidad del problema. Para reducir este tiempo, cuando se consideran funciones aditivamente descomponibles con $NComp$ componentes, se divide el proceso en los siguientes pasos:

1. La evaluación se descompone en $NComp$ tareas. Cada tarea se asocia a cada componente de la función, que hace referencia a un subconjunto de las variables D_c, según la definición del problema. Las diferentes componentes podrían requerir la ejecución de diferentes códigos. En el caso de tratar con una función con una única componente, entonces sólo habrá una tarea que involucra a todas las variables de decisión, desde la primera a la última (D).
2. Se programan un conjunto de hebras para calcular colaborativamente el valor de la función. Cada hebra se encarga de un conjunto de tareas, equilibrando la carga de trabajo, y almacena el valor de salida de las tareas que ha realizado en la memoria compartida. Las hebras se ejecutan en paralelo usando flujos, lo que permite de forma concurrente la transferencia de datos y la ejecución de código, maximizando la carga y rendimiento de la GPU.
3. Se agregan los resultados de la diferentes componentes para calcular el valor objetivo de la solución completa. Estas fases se ejecutan en paralelo para cada solución de la población.

Un detalle interesante es que algunas de las variables pueden requerir más de una lectura, si algunas componentes se definen sobre conjuntos de variables que se solapan. En cualquier caso, esto se puede llevar a cabo en paralelo sin

ninguna sobrecarga adicional, pues la memoria de la GPU puede leerse por todos sus procesadores y núcleos en el mismo instante.

2.2. Escalado en Múltiples GPUs

En el caso de disponer de múltiples GPUs, el esquema anterior puede extenderse de alguna de las siguientes dos formas:

- *Reducción de tiempo*: Si $NComp$ es suficientemente elevado, la carga de trabajo se podría reducir distribuyendo el cálculo de las componentes entre los procesadores de las diferentes GPUs. En este caso, la presencia de componentes con grupos de variables con solapamiento requeriría que éstas se duplicasen en las diferentes memorias de las diferentes GPUs, conllevando consigo cierta sobrecarga por comunicación.
- *Incremento de la dimensionalidad*: La memoria combinada de las GPUs se podría utilizar para gestionar un mayor número de variables de decisión. En este caso, hay que evaluar la complejidad espacial y temporal de la función, dado que el número de núcleos y cantidad de memoria sólo crece linealmente con el número de GPUs. Por ejemplo, si una función utilizase una matriz de rotación $D \times D$, entonces su complejidad espacial es cuadrática, y requeriría un incremente elevado del número de GPUs para un pequeño incremento en el número de variables de decisión. Por suerte, las funciones del CEC 2013 consideran matrices de rotación de 100×100 como máximo.

En este trabajo hemos escalado de la dimensionalidad de la función, aplicando la idea introducida por Lastra y otros [7] y Cano y otros [11]. Ésta consiste en considerar la función como una caja negra de D variables, y replicarla tantas veces como se requiera. En el caso de las funciones del CEC 2013, las funciones tienen 1,000 variables, y nos referiremos al conjunto de variables de cada réplica como un bloque de D variables. Las ventajas de esta estrategia son que se evita la comunicación entre GPUs, pues los cálculos en cada una de ellas son independientes de los otros, y que los requerimientos de memoria crecen linealmente con el número de réplicas. Por el contrario, dicha estrategia tiene la desventaja de que las propiedades de las funciones, como su no-separabilidad, se restringen a cada grupo de D variables.

Tendríamos entonces dos modelos para distribuir la población de P soluciones, de alta dimensionalidad D, sobre $NGPUs$ tarjetas gráficas. Una sería dividir la población en diferentes subpoblaciones y asignar cada una a una GPU, por lo que cada una manejaría a $P/NGPUs$ soluciones con D variables. De esta forma, cada GPU almacena y procesa la información de $P/NGPUs$ soluciones y todas las D variables de cada solución. En el segundo modelo, dividimos las soluciones en $NGPUs$ partes, de forma que cada GPU almacena y procesa $D/NGPUs$ variables de cada individuo de la población (como aparece más adelante en la Figura 1).

El principal problema del primer modelo es que requiere bastante comunicación entre las diferentes GPUs para llevar a cabo las distintas operaciones del

algoritmo de optimización. Estas transferencias, de tamaño D a través del bus PCI tendrían un impacto negativo en el rendimiento del algoritmo. Por otro lado, el segundo modelo produce un mapeo eficiente de las variables y permite un escalado a un número de variables mucho mayor, simplemente añadiendo más tarjetas. Los operadores genéticos no se verían afectados por esta distribución, dado que sus operadores se realizan localmente a nivel de gen, evitando por ello las comunicaciones entre diferentes GPUs. Sin embargo, la evaluación de la calidad de cada individuo requeriría la evaluación del subconjunto de variables que gestiona cada tarjeta en paralelo por parte de las GPUs, y después sincronizar y combinar los valores de fitness parciales, por parte del proceso principal, para producir un valor de fitness de la solución, que se transferirá después a cada una de las tarjetas. Afortunadamente, esta comunicación sólo transmite P valores de punto flotante y por tanto, su impacto es muy pequeño.

2.3. Ejemplo de Escalado hasta 10^8 Variables

Supongamos que se desea evaluar un conjunto de P soluciones de 10^8 variables en la función $f_4(\boldsymbol{x})$ del CEC 2013 en 8 GPUs:

$$f_4(\boldsymbol{x}) = \sum_{i=1}^{|S|-1} w_i \cdot f_{elliptic}(\boldsymbol{z_i}) + f_{elliptic}(\boldsymbol{z_{|S|}}) \tag{1}$$

donde $S = \{50, 25, 25, 100, 50, 25, 25, 700\}$, $f_{elliptic}(\cdot)$ es una función base, w_i es un peso, y $\boldsymbol{z_i}$ es un conjunto de variables seleccionando S_i de $\boldsymbol{x_i}$ y que sufren transformaciones de rotación y generación de irregularidades locales. La rotación no se aplica para el último conjunto según la definición en CEC 2013 ($S_{|S|} = 700$). Nótese que S indica las componentes sobre las que la función es aditivamente descomponible, de forma que la función tiene exactamente $|S|$ componentes.

Para escalar la función hasta 10^8 variables, replicamos su definición 10^5 veces y cada GPU se encarga de calcular el resultado de $10^5/8 = 12{,}500$ réplicas por solución. Los datos asociados con la función se cargan en la GPU una sola vez al inicio, que son los pesos w_i, la definición de la función $f_{elliptic}$, las matrices de rotación (nótese que la mayor es sólo de 100×100 valores), y la información asociada con las transformaciones para introducir irregularidades locales. Además, se cargan los conjuntos correspondientes de los valores de las variables de decisión de las P soluciones de dimensión 10^8 en cada GPU. Todo ello requiere $P \times 10^8/8$ variables de punto flotante, que son 2,384MB en simple precisión o 4,768MB con precisión doble en cada GPU, si P es igual a 100 soluciones.

Entonces se crean las tareas para la evaluación en paralelo de la función. Particularmente, cada GPU calcula $|S| \times P \times 10^5/8$ resultados, uno por componente de la función, solución y réplica. Dado que el código asociado a cada componente es el mismo, se pueden ejecutar en paralelo hasta $P \times 10^5/8$ tareas en la misma GPU, de forma que las tareas asociadas a diferentes componentes se distribuirían en los diferentes procesadores de la GPU. Los resultados de las tareas se agregan a dos niveles, por componentes y por réplicas de la función,

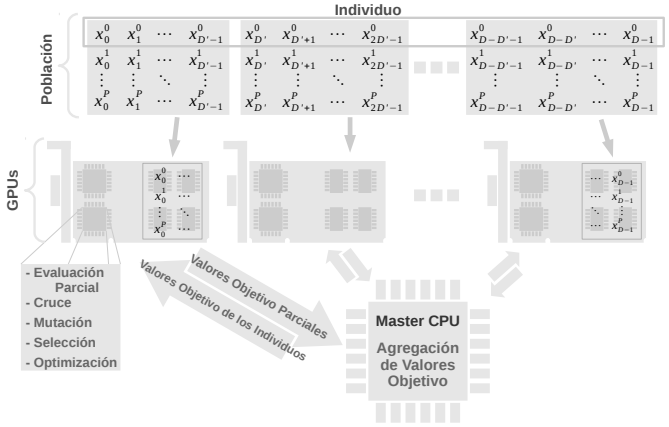

Figura 1. Esquema global de NGPUs-MA-SW-Chains

produciendo P valores parciales en cada GPU. Finalmente, el valor objetivo real de cada solución se calcula en el proceso principal, agregando los 8 valores parciales de cada solución.

La ventaja de este tipo de distribución es que no requiere la transferencia de cantidades importantes de datos entre las diferentes GPUs, o entre las GPUs y el procesador que ejecuta el proceso principal. Además, es altamente escalable y permite abordar dimensionalidades incluso mayores simplemente añadiendo más GPUs.

3. Algoritmo Memético sobre Varias GPUs

En esta sección presentamos una extensión del algoritmo MA-SW-Chains [12] para su ejecución en un sistema con múltiples GPUs. Éste método es un algoritmo memético estacionario que genera sólo un descendiente, e igualmente aplica una optimización local sobre una única solución, por iteración. En nuestro caso, el modelo se ha modificado para obtener el mayor beneficio de las múltiples GPUs. En particular, nuestro modelo produce un descendiente por cada solución de la población, y a todo miembro de la población se le aplica la optimización local. Debemos mencionar que, aunque Lastra y otros ya publicaron una extensión de este modelo para su ejecución en una GPU [7], nuestro modelo se describe más fácilmente como una extensión del memético original MA-SW-Chains [12], que desde la versión sobre una GPU.

La Figura 1 presenta el esquema global de nuestro modelo, NGPUs-MA-SW-Chains, y la 2, su pseudocódigo. La idea principal es dividir el genotipo de los individuos de la población verticalmente, de forma que cada GPU recibe la

```
   //Inicialización
 1 Generar la porción i-ésima de la solución j en cada GPU;
 2 f_i^j ←Evaluar la porción i-ésima de cada solución j en cada GPU;
 3 Agregación de valores objetivo en proceso principal y transmisión a GPUs;
   //Bucle principal
 4 repeat
 5 |   for N_it iteraciones do
 6 |   |   Aplicar cruce en la porción i-ésima de toda solución j en cada GPU;
 7 |   |   Aplicar mutación en la porción i-ésima de toda solución j en cada GPU;
 8 |   |   f_i^j ←Evaluar la porción i-ésima de toda solución j en cada GPU;
 9 |   |   Agregar los valores objetivo en proceso principal y transmitir a GPUs;
10 |   |   Aplicar selección en paralelo en cada GPU;
11 |   end
12 |   for N_it iteraciones do
13 |   |   Generar porción i-ésima de un vecino de toda solución j en cada GPU;
14 |   |   f_i^j ←Evaluar la porción i-ésima de toda solución j en cada GPU;
15 |   |   Agregar los valores objetivo en proceso principal y transmitir a GPUs;
16 |   |   Aplicar selección en paralelo en cada GPU;
17 |   end
18 until condición de parada;
19 return Mejor-solución-generada();
```

Figura 2. Pseudocódigo de NGPUs-MA-SW-Chains

misma porción de todas las soluciones de la población y aplica los operadores evolutivos sobre ellos. Sus operaciones son:

1. *Generación de la población inicial*: Cada GPU genera aleatoriamente una porción del genotipo de los P individuos de la población, según las restricciones de dominio de las respectivas variables de decisión (línea 1). Concretamente, cada GPU produce los valores iniciales para sus bloques de D variables correspondientes.
2. *Evaluación*: Cada GPU calcula los valores de sus réplicas de la función según se ha comentado en la Sección 2.2. Los valores parciales que produce cada GPU se agregan dentro de la misma tarjeta, de forma que cada una devuelve P valores parciales al proceso principal, una por solución (líneas 2, 8 y 14). Los valores parciales se agregan posteriormente en el proceso principal, obteniendo el valor objetivo real de cada solución de la población, y éste se manda a las tarjetas GPU (líneas 3, 9 y 15).
3. *Cruce y mutación*: Los operadores genéticos se aplican en paralelo sobre las porciones de genotipo que maneja cada GPU (líneas 6 y 7). Esto es posible dado que las operaciones de MA-SW-Chains, BLX-α y BGA [7], no dependen de interacciones entre las variables. Para explotar al máximo el paralelismo, cada porción del genotipo se cruza con otra de la población seleccionada según emparejamiento variado negativo [7], lo cual promueve el cruce entre genotipos diferentes (el más distante entre tres elegidos aleatoriamente de la población). Sin embargo, el cálculo de las distancias se restringe a la porción del genotipo que maneja la GPU, para evitar costes de comunicación. Es importante destacar que de esta forma, las soluciones normalmente se cruzan

con diferentes soluciones en el mismo instante, cada porción con otra de un individuo diferente. Seguidamente, el genotipo resultante se muta.

4. *Selección*: Una vez que se han generado los descendientes, mutado y evaluado, se seleccionan los P mejores individuos para formar la nueva población (líneas 10 y 16)
5. *Optimización*: Toda porción del genotipo de todo individuo se optimiza en paralelo por N_{it} iteraciones (líneas 12-17). Se utiliza el método de Solis y Wets, como en el MA-SW-Chains original [12]. En cada iteración, se genera una nueva porción del genotipo, de entre las variables que gestiona la GPU para cada individuo, y se evalúan. El proceso principal agrega los valores objetivo parciales y los devuelve a las tarjetas. Finalmente, cada porción candidata se acepta si el nuevo valor objetivo de la solución completa es mejor.

4. Experimentos

Esta sección presenta los experimentos llevados a cabo para comprobar si la disponibilidad de múltiples GPUs, particularmente el incremento de memoria correspondiente, posibilita abordar funciones aditivamente descomponibles de muy elevada dimensionalidad.

Hemos considerado las funciones del CEC 2013 [9], que consisten en 15 funciones de minimización, donde el óptimo global se encuentra desplazado del centro del espacio de búsqueda, los valores objetivo también se han desplazado, y muchas de ellas usan matrices de rotación para crear interdependencias entre variables dentro de sus componentes. Estas funciones se han escalado hasta 10^8 variables mediante la replica de la función como se comenta en la Sección 2.2. Además se han realizado 10 ejecuciones independientes de los algoritmos en cada función.

Los parámetros de NGPUs-MA-SW-Chains (Sección 3) se han ajustado a $P = 25$, operador de cruce BLX-α con $\alpha = 0{,}5$ y $N_{it} = 500$, es decir, alternativamente aplica 500 iteraciones de la fase evolutiva y 500 iteraciones de la optimización local. Sus resultados se han comparado con los de una búsqueda aleatoria que también explota las existencia de múltiples GPUs, NGPUs-RandomSearch. Ésta explota la disponibilidad de múltiples GPUs para generar en paralelo porciones de soluciones aleatorias, de la misma forma que NGPUs-MA-SW-Chains genera la población inicial. Se ha considerado la búsqueda aleatoria como referencia por la inexistencia, a nuestro conocimiento, de otros métodos que puedan abordar problemas con tantas variables de decisión de punto flotante. En particular, ni el original MA-SW-Chains, ni su versión con una GPU pueden aplicarse en este caso. Ambos algoritmos se ejecutan para un máximo de 500,000 evaluaciones por ejecución.

Nótese que cuando el espacio de búsqueda es extremadamente grande, y los recursos computacionales limitados (pocas evaluaciones), la necesidad de encontrar regiones de búsqueda relativamente buenas puede volverse especialmente más significante que la necesidad de obtener lo mejor de las regiones encontradas.

Cuadro 1. Resultados con 10^8 dimensiones

	F1	F2	F3	F4	F5
NGPUs-MA-SW-Chains	**8.42e+15**	**1.95e+9**	2.17e+1	**2.44e+18**	**2.55e+12**
Std	8.65e+14	7.59e+6	8.36e-3	1.73e+17	1.21e+10
NGPU Random Search	2.43e+16	6.00e+9	**2.17e+1**	1.44e+19	8.09e+12
Std	2.64e+11	4.38e+4	0	8.24e+15	2.91e+9
	F6	F7	F8	F9	F10
NGPUs-MA-SW-Chains	1.08e+11	**1.81e+19**	**1.95e+23**	**2.07e+14**	**9.74e+12**
Std	3.42e+6	4.31e+21	2.15e+22	1.52e+13	1.22e+10
NGPU Random Search	**1.08e+11**	1.39e+24	8.86e+23	5.59e+14	9.82e+12
Std	3.00e+5	2.51e+22	5.08e+20	7.48e+10	5.46e+7
	F11	F12	F13	F14	F15
NGPUs-MA-SW-Chains	**7.46e+21**	**4.59e+16**	**2.39e+21**	**9.67e+22**	**2.68e+19**
Std	1.96e+21	3.29e+15	1.33e+21	6.16e+22	1.80e+18
NGPU Random Search	1.70e+26	2.00e+17	1.32e+25	1.56e+26	7.65e+19
Std	2.43e+24	4.90e+11	2.13e+23	1.53e+24	5.56e+16

Esto significa que la exploración del espacio de búsqueda se vuelve de especial interés, y cualquier esfuerzo invertido en explotación puede reducir el rendimiento del algoritmo. Por ello, este experimento prueba si la combinación de información y explotación que llevan a cabo las operaciones de NGPUs-MA-SW-Chains son aún suficientemente beneficiosas en estos problemas de extrema dimensionalidad, explotando algún conocimiento de los problemas de optimización de variables reales, con respecto a un método de exploración puro e inconsciente.

El Cuadro 1 muestra los resultados. El test de Wilcoxon al 0.05 encontró diferencias significativas en los resultados de cada una de las funciones. Observamos que NGPUs-MA-SW-Chains obtiene mejores resultados medios en todas las funciones (iguales en F3 y F6), pero generalmente peores desviaciones típicas en sus resultados. Por tanto, podemos concluir que en general, sus operaciones, que combinan la información de soluciones previas mediante los operadores genéticos y la optimización local, son aún beneficiosos en estos contextos de extrema dimensionalidad. Además, podemos destacar que:

- Las diferencias en los resultados entre estos dos algoritmos son de varios órdenes de magnitud en algunas funciones (F7, F11, F13 y F14) y modestas o mínimas en otras (F1, F2, F4, F5, F8, F9, F10, F12 y F15). Aunque NGPUs-MA-SW-Chains se muestra mejor que la búsqueda aleatoria, la existencia de estas mejoras mínimas confirma la hipótesis de que invertir excesivos recursos en explotación de soluciones previas, cuando se tratan espacios de búsqueda tan grandes, puede no ofrecer mejoras significativas frente a la simple exploración limitada del espacio.
- Que la búsqueda aleatoria consiga desviaciones estándar mejores indica que ésta es más estable produciendo resultados similares, aunque de peor calidad, que NGPUs-MA-SW-Chains. Nuestra hipótesis es que el rendimiento de NGPUs-MA-SW-Chains puede depender fuertemente del reducido número de regiones de búsqueda explorados, de forma que sus resultados pueden variar mucho más que los de la búsqueda aleatoria entre una ejecución y otra.

Finalmente, un análisis estadístico mediante el test de Wilcoxon confirmó posteriormente la superioridad de NGPUs-MA-SW-Chains con valores: R+: 118.5, R-=1.5, y valor crítico al 0.01 de 15.

5. Conclusiones

En este trabajo hemos abordado el uso de una arquitectura computacional escalable y distribuida con múltiples GPUs para la optimización de problemas con variables de punto flotante de dimensionalidad extrema. Hemos presentado un modelo de escalado para la evaluación de funciones aditivamente descomponibles que nos ha permitido presentar resultados para hasta 10^8 variables de decisión, lo que supone dos órdenes de magnitud superiores al máximo que se ha abordado en la literatura, a nuestro conocimiento. Adicionalmente, hemos presentado un algoritmo memético que hace uso de las múltiples GPUs en la búsqueda de soluciones para éstos problemas.

Agradecimientos

Este trabajo se ha financiado mediante el Proyecto TIN-2014-55252-P del Ministerio de Economía y Competitividad de España y fondos FEDER.

Referencias

1. "NVIDIA CUDA programming and best practices guide." [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html
2. A. Cano, A. Zafra, and S. Ventura, "A parallel genetic programming algorithm for classification," in *Proceedings of the 6th International Conference on Hybrid Artificial Intelligent Systems (HAIS). Lecture Notes in Computer Science*, vol. 6678 LNAI, no. PART 1, 2011, pp. 172–181.
3. A. Cano, S. Ventura, and K. Cios, "Scalable CAIM discretization on multiple GPUs using concurrent kernels," *Journal of Supercomputing*, vol. 69, no. 1, pp. 273–292, 2014.
4. A. Cano, A. Zafra, and S. Ventura, "Speeding up multiple instance learning classification rules on GPUs," *Knowledge and Information Systems*, vol. 44, no. 1, pp. 127–145, 2015.
5. A. Cano, D. T. Nguyen, S. Ventura, and K. J. Cios, "ur-CAIM: improved CAIM discretization for unbalanced and balanced data," *Soft Computing*, vol. 20, no. 1, pp. 173–188, 2016.
6. A. Cano, A. Zafra, and S. Ventura, "An interpretable classification rule mining algorithm," *Information Sciences*, vol. 240, pp. 1–20, 2013.
7. M. Lastra, D. Molina, and J. M. Benítez, "A high performance memetic algorithm for extremely high-dimensional problems," *Information Sciences*, vol. 293, pp. 35–58, 2015.
8. K. Tang, X. Yao, P. Suganthan, C. MacNish, Y. Chen, C. Chen, and Z. Yang, "Benchmark Functions for the CEC'2008 Special Session and Competition on Large Scale Global Optimization," Technical Report, Nature Inspired Computation and Applications Laboratory, Tech. Rep., 2007.

9. X. Li, K. Tang, M. N. Omidvar, Z. Yang, and K. Qin, "Benchmark Functions for the CEC'2013 Special Session and Competition on Large-Scale Global Optimization," Evolutionary Computation and Machine Learning Group, RMIT University, Australia, Tech. Rep., 2013.
10. R. K. Thompson and A. H. Wright, "Additively Decomposable Fitness Functions," Dept. of Computer Science, University of Montana, Tech. Rep., 1996.
11. A. Cano, C. García-Martínez, and S. Ventura, "Extremely High-dimensional Optimization with MapReduce: Scaling Functions and Algorithm," *Information Sciences*, vol. Submitted, pp. 1–31, 2015.
12. D. Molina, M. Lozano, C. García-Martínez, and F. Herrera, "Memetic algorithms for continuous optimisation based on local search chains," *Evolutionary Computation*, vol. 18, no. 1, pp. 27–63, 2010.
13. D. Molina, M. Lozano, A. Sánchez, and F. Herrera, "Memetic algorithms based on local search chains for large scale continuous optimisation problems: MA-SSW-Chains," *Soft Computing*, vol. 15, pp. 2201–2220, 2011.