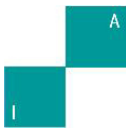
Inteligencia Artificial V 16, N 51(2013), 1-14



INTELIGENCIA ARTIFICIAL

http://journal.iberamia.org/

Algoritmos metaheurísticos en el problema del particionado hardware/software de sistemas embebidos

Humberto Díaz Pando[1], Sergio Cuenca Asensi[2], Roberto Sepúlveda Lima[3], Alejandro Rosete Suárez[4], Jenny Fajardo Calderín[5]

[1,3,4,5]Departamento de Inteligencia Artificial e Infraestructura de Sistemas Informáticos
Instituto Superior Politécnico José Antonio Echeverría. La Habana, Cuba.
{hdiazp,sepul,rosete,jfajardo}@ceis.cujae.edu.cu

[2]Departamento de Tecnología Informática y Computación
Universidad de Alicante. Alicante, España.
sergio@dtic.ua.es

Resumen El particionado hardware/software es una tarea fundamental en el co-diseño de sistemas embebidos. En ella se decide, teniendo en cuenta las métricas de diseño, qué componentes se ejecutarán en un procesador de propósito general (software) y cuáles en un hardware específico. En los últimos años se han propuesto diversas soluciones al problema del particionado dirigidas por algoritmos metaheurísticos. Sin embargo, debido a la diversidad de modelos y métricas utilizadas, la elección del algoritmo más apropiado sigue siendo un problema abierto. En este trabajo se presenta una comparación de seis algoritmos metaheurísticos: Búsqueda aleatoria (Random search), Búsqueda tabú (Tabu search), Recocido simulado (Simulated annealing), Escalador de colinas estocástico (Stochastic hill climbing), Algoritmo genético (Genetic algorithm) y Estrategia evolutiva (Evolution strategy). El modelo utilizado en la comparación está dirigido a minimizar el área ocupada y el tiempo de ejecución, las restricciones del modelo son consideradas como penalizaciones para incluir en el espacio de búsqueda otras soluciones. Los resultados muestran que los algoritmos Escalador de colinas estocástico y Estrategia evolutiva son los que mejores resultados obtienen en general, seguidos por el Algoritmo genético.

Keywords: Co-diseño hardware/software, Particionado hardware/software, Algoritmos metaheurísticos, Búsqueda aleatoria, Búsqueda tabú, Recocido simulado, Escalador de colinas estocástico, Algoritmo genético y Estrategia evolutiva.

Abstract Hardware/software partitioning is a key task for embedded system co-design. The main goal of this task is to decide which components of an application will be executed in a general purpose processor (software) and which ones on a specific hardware. In last years, several approaches have been proposed for solving the HSP problem, directed by metaheuristics algorithms. However, due to diversity of models and metrics used, the choice of the best suited algorithm is an open problem yet. This article presents a comparison of six metaheuristics algorithms: Random Search, Tabu Search, Simulated Annealing, Hill Climbing, Genetic Algorithm and Evolution Strategy. The model presented is aimed to minimize the hardware area and execution time, restrictions are considered as penalizations in order to include in the search space other solutions. Results show that Hill climbing and Evolution strategy are the algorithms that obtains best results in general, followed by the Genetic Algorithm.

Keywords:Hardware/software co-design, Hardware/software partitioning, Metaheuristic algorithms, Random search, Tabu search, Simulated annealing, Stochastic hill climbing, Genetic algorithm, Evolution strategy.

ISSN: 1988-3064(on-line)

1. Introducción

En la actualidad muchos son los escenarios donde se pueden encontrar dispositivos que incluyen sistemas embebidos (SE) para controlar su funcionamiento. Estos sistemas tienen tres características fundamentales [34]: (1) ejecutan una única funcionalidad, (2) están ligados a fuertes restricciones y (3) deben ser reactivos y/o responder en tiempo real. Al tratarse de sistemas sobre los que se ejecuta una sola aplicación, tiene sentido plantearse la modificación de la configuración hw/sw de forma que se optimice la ejecución de aquella. Es habitual, por tanto, que los términos aplicación y sistema se usen de forma intercambiable en este contexto. La segunda característica se refiere a que el diseño de un SE está guiado por el cumplimiento de las diferentes métricas de diseño que se establezcan. Existen varias métricas que pueden ser utilizadas para guiar el diseño de los SE, entre las que se encuentran: tamaño, rendimiento, coste por unidad, flexibilidad, consumo de potencia, entre otras; dándose muchas veces el caso de que para el diseño de un mismo sistema interviene más de una métrica. Esto implica que el proceso de diseño es un proceso complejo en sí mismo, ya que persigue llegar a una solución de compromiso entre las diferentes métricas.

El diseño de SEs de cierta complejidad supone, en la mayoría de los casos, el desarrollo de parte del sistema sobre un microprocesador (Sw) y otra parte en hardware a medida (Hw). Tradicionalmente el diseño del Hw y el Sw se desarrollan por separado y en etapas tempranas. Este procedimiento no permite asegurar el cumplimiento de los requisitos, además de generar iteraciones muy costosas para refinar el diseño. La tendencia actual es utilizar una descripción unificada, co-diseño [7], para el Hw y el Sw que permita, además de verificar la corrección del diseño, explorar las distintas posibilidades de particionado sin necesidad de pasar por la costosa fase de implementación.

Una de las etapas más importantes dentro del proceso de co-diseño es el Particionado Hardware/Software (PHS) [7] [36]. En esta etapa se define cual será la configuración final que adoptará el sistema; es decir, cuales de los bloques funcionales que serán implementados en Sw y cuales de ellos serán implementados en Hw. Generalmente la decisión se toma en base a la experiencia del diseñador y/o realizando una somera exploración del espacio de diseño. Este procedimiento, además de no ajustarse a ninguna metodología, no asegura un resultado óptimo, ya que para obtener la mejor configuración es necesario resolver un problema de optimización que en la mayoría de sus formulaciones es NP-fuerte [3].

Para sustituir estos métodos "ad-hoc"han sido propuestos varios modelos [16] [17] [18] [21] [4] [2] [24] que permiten llegar a una solución, que si bien en algunos casos pudiera ser un óptimo local, es aceptada como suficientemente buena. La mayoría de estos modelos utilizan algoritmos metaheurísticos de optimización (recocido simulado, búsqueda tabú, algoritmos genéticos, entre otros), los cuales permiten explorar de forma más exhaustiva el espacio de diseño en busca de la solución adecuada. En gran parte de los casos, estos modelos están guiados por la optimización de una sola métrica de diseño (área de hardware utilizada, tiempo de ejecución, consumo de potencia, etc.) restringiendo otras según el modelo definido y de acuerdo con los intereses del sistema en cuestión. La diversidad de algoritmos utilizados unido a la carencia de bancos de prueba impide la selección acertada de uno en particular que se adapte mejor al problema PHS.

La principal contribución de este trabajo consiste en la aplicación de algoritmos metaheurísticos de los que no se tienen evidencias sobre su utilización previa para el problema PHS y su comparación con otras metaheurísticas que sí han sido empleadas, obteniéndose interesantes resultados. Para ello, se propone utilizar la métrica Índice de rendimiento, definida en [25], la cual combina dos importantes restricciones en el diseño de sistemas embebidos (el coste de hardware (área) y el tiempo de ejecución). En la sección 2 se revisan los trabajos más relevantes relacionados con la problemática PHS. En la sección 3 se expone el modelo de particionado utilizado en este artículo. En la sección 4 se describen los experimentos realizados y los resultados obtenidos. Finalmente se exponen las conclusiones y se define el trabajo futuro.

2. Trabajos relacionados

En las últimas décadas varios han sido los trabajos donde se han propuesto soluciones al problema PHS. Generalmente todas las propuestas introducen variaciones en uno o varios de los tres elementos básicos que componen este problema: (1) Especificación y modelado de la aplicación, (2) Definición de la función objetivo y las restricciones del sistema y (3) Algoritmo de búsqueda de solución

Para especificar el sistema es necesario definir su granularidad y la implementación inicial del mismo [1] [36] [14] [30] [21]. La granularidad según [14] es el tamaño que será considerado para un bloque funcional del sistema (instrucciones, bloques básicos, bloques de control, funciones o procedimientos); deben tenerse en cuenta durante el particionado. Además, el sistema puede ser implementado inicialmente en software o hardware. Si se implementa en software (o hardware) la estrategia durante el particionado es migrar los bloques funcionales hacia el hardware (o software) hasta que se logre encontrar la solución que cumpla con las métricas de diseño impuestas. La descripción de la implementación inicial puede hacerse utilizando diversos lenguajes, preferiblemente uno de alto nivel, con los que el diseñador especifica las funcionalidades que tendrá el sistema; los más utilizados suelen ser C, SystemC y VHDL/Verilog.

El modelado del sistema [6] define de qué forma será representado antes y durante el proceso de particionado. Entre los modelos más utilizados están los grafos de control de flujo (Control flow graph), grafos de flujo de datos (Data flow graph) y los grafos de llamadas (Call graph). Muchos de estos modelos computacionales están condicionados por la granularidad escogida. Por ejemplo, si se selecciona una granularidad a nivel de funciones o procedimientos el modelo pudiera ser un grafo de llamada en el cual los nodos representan las funciones del sistema y los arcos representan la llamada de una función a otra. Es importante destacar que la conclusión a la que llegan los autores en [6] es que la selección de un modelo depende del tipo de sistema. En [21] los autores exponen varios aspectos de algunas propuestas de particionado donde se puede apreciar la diversidad de modelos que pueden ser utilizados.

Por último, para buscar la solución óptima, o una cercana a esta, es necesario definir la función de coste, las restricciones y el algoritmo que se encargará de resolver el problema de optimización. En los dos primeros aspectos están involucradas las métricas que se obtendrán a partir de los requerimientos que se especifiquen para el sistema. Existen diferentes métricas a considerar en el diseño de un sistema embebido (tiempo de ejecución, consumo de potencia, tamaño, etc.) [34], en este sentido van dirigidas algunas de las propuestas que utilizan dichas métricas en la función objetivo o en las restricciones, según el modelo que se proponga. En [24] el autor hace una abstracción de estas métricas considerando solamente tres grandes grupos: (1) coste de hardware, (2) coste de software y (3) coste de comunicación entre los bloques Hw y Sw. En [13] los autores proponen como función objetivo minimizar la utilización del bus y del procesador y el tamaño del hardware, mientras que las restricciones están basadas en métricas de tiempo relacionadas con la ejecución de las operaciones. En el sistema Cosyma [9] se optimiza el tiempo de ejecución bajo restricciones de tiempo y cantidad de ejecuciones de bloques básicos. En el sistema LYCOS propuesto en [23], así como en [16] los autores buscan maximizar la aceleración que se produce al mover un nodo de software a hardware, teniendo en cuenta el área total ocupada por el diseño. En [15] los autores plantean minimizar el área de hardware ocupada teniendo en cuenta el consumo de potencia y el tiempo de ejecución del sistema. En [17] y [18] se propone minimizar el tiempo de ejecución, en el cual se incluyen los costos de comunicación entre nodos adyacentes, utilizando como restricción el área ocupada por el sistema. En [12] los autores utilizan el tiempo de ejecución de las funciones, el costo de comunicación entre las funciones y la proximidad de las funciones para determinar qué tareas pueden ejecutarse en un procesador.

En cuanto a las propuestas relacionadas con los aspectos algorítmicos, la mayoría utiliza heurísticas genéricas, aunque hay otros trabajos que utilizan heurísticas especialmente diseñadas y otros algoritmos. Entre las primeras, varios autores presentan aproximaciones basadas en estrategias voraces [1] [33]. También se han realizado diversas propuestas basadas en los algoritmos de búsqueda tabú y aleatoria [35] [8] [33] [17], mientras que otros plantean soluciones basadas en recocido simulado [35] [9] [33] [21] [8], algoritmos genéticos [17] [27] [35] y enjambre de partículas [4] [2]. López *et al.* [21] proponen la utilización de un sistema experto para resolver el problema. López-Vallejo y López [21] y Frank Vahid [33] utilizan el algoritmo Kernighan/Lin. Varios autores [33] [21] [12] emplean el agrupamiento jerárquico y Gupta y De Micheli [13] proponen la técnica de migración de grupos. También existen trabajos que emplean heurísticas de propósito específico tal es el caso de la propuesta de Jigang y Srikanthan [15] [17]. Finalmente, otras propuestas que hacen uso de algoritmos basados en programación dinámica [15] y en programación lineal [16] [18] [23].

A modo de resumen se puede plantear que la mayoría de las contribuciones están orientadas a los modelos para el particionado y a los aspectos algorítmicos. En el primer caso se aprecia una amplia diversidad de modelos que utilizan varias métricas entre las que están el área ocupada y el tiempo de respuesta del sistema. Estas dos métricas son las que con mas frecuencia se utilizan en la función objetivo

y en las restricciones que modelan el problema. La tabla 1 muestra un resumen de los métodos revisados. Como puede apreciarse, existen algunas heurísticas de propósito general que todavía no has sido evaluadas para el problema PHS. Tal es el caso del Escalador de colinas estocástico y la Estrategia evolutiva, que han sido utilizadas con buenos resultados en otros problemas (por ejemplo [28]), y que en general no pueden ser descartados según el Teorema NFL [37]. Finalmente cabe destacar la imposibilidad de comparar de forma coherente los resultados, debido a las diferencias en el modelado del problema y en los bancos de prueba utilizados. Este hecho evidencia la necesidad de realizar una formulación única para el problema PHS.

Tabla 1: Resumen de las contribuciones al PHS

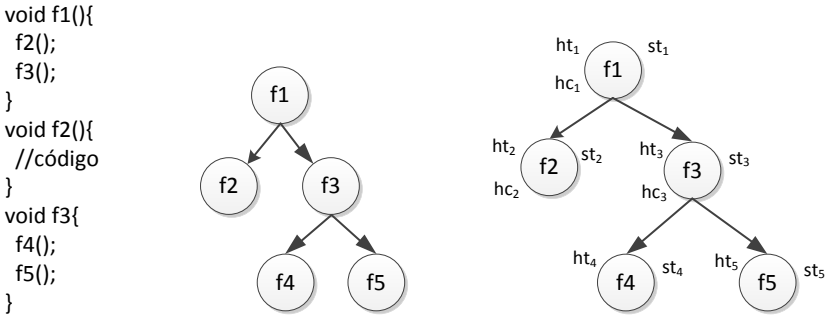
Algoritmo	Año							
	1993	1997	2003	2004	2006	2007	2008	2010
Búsqueda Aleatoria		[33]						
Búsqueda tabú		[8]					[17]	
Goloso		[33]		[1]			[17]	
Recocido simulado		[33] [8]	[21]					
Algoritmo genético	[9]						[17]	
PSO						[2]	[4]	
Heurísticos específicos					[15]		[17]	
KL		[33]	[21]					
Migración de grupos	[13]							
Agrupamiento jerárquico		[33]	[21]					[12]
Programación dinámica					[15]			
Programación lineal		[23]			[16]		[18]	
Comparaciones		[33]	[21]				[17]	

2.1. Estrategias para la validación de las propuestas PHS.

Como parte de la identificación y estudio de los trabajos relacionados, es necesario resaltar la forma en que son validados los modelos en los trabajos propuestos; de forma tal que sea posible identificar la manera más adecuada de validar el modelo presentado en este trabajo. Se identifican dos estrategias fundamentales de validación. La primera de las variantes consiste en realizar un conjunto de pruebas sobre el modelo y cada uno de los algoritmos propuestos, con el propósito de ajustar o afinar los parámetros de estos para ese modelo en particular [9] [13] [14] [1] [20] [27].

La segunda está dirigida a establecer referentes comparativos ya sea entre varias propuestas o entre varios algoritmos en una misma propuesta. En este caso se encuentra el trabajo de [16], donde se realiza una modificación al algoritmo propuesto por [23], igualmente en [18] extienden el modelo propuesto en [16]. Por último en [17] se compara el algoritmo propuesto con los algoritmos utilizados por [35], empleando los mismos valores de parámetros. Existen varios trabajos que plantean la comparativa de varios algoritmos para un mismo modelo [32] [8] [21] [31] [35] [3] [39] [2], tal es el caso del presente trabajo. Además en las propuestas de [19] [26] [4] [15] se utilizan algoritmos exactos como referentes comparativos para los heurísticos; obteniéndose medidas cuantitativas de la optimización obtenida por cada algoritmo.

Para ejecutar ambas variantes de validación es necesario decidir qué tipo de aplicación utilizar de acuerdo con los propósitos de la validación, aplicaciones simuladas o aplicaciones reales. La primera de estas variantes consiste en generar aplicaciones simuladas, o sea generar grafos de control de flujo o grafos de llamadas (según la granularidad definida), los cuales representan la estructura de una aplicación cualquiera. Para cada uno de los nodos de estos grafos se generan de forma aleatoria valores que representan cada uno de los costes definidos. En este caso se encuentran las propuestas de: [8] [32] [19] [22] [31] [35] [26] [3] [18] [17] [20] [15] [27] [39] [2]. Como puede apreciarse, este tipo de enfoque es bastante utilizado; además se ha podido apreciar que en otros contextos es igual de utilizado. Tales son los casos de [38] [11], en los que además de usar el enfoque simulado utilizan para validar su propuesta la ejecución de un conjunto de pruebas sobre el algoritmo.



(a) Código a particionar. (b) Grafo de llamadas. (c) Grafo de llamadas anotado.

Figura 1: Ejemplo de diseño y su representación.

En la otra variante son utilizadas aplicaciones típicas de distintos dominios de aplicación en los sistemas embebidos, por ejemplo: procesamiento digital de señales [14] [9] [31] [35] [21] [3] que implementa el [29] [20] [32] [9] , criptografía [26] [4] [3], comunicaciones [13] [8] [32] [19] [1] [3], entre otras. Para esta variante es imprescindible contar con herramientas que estimen cada uno de los costes, el principal problema de estas herramientas radica en que se requiere de la combinación de varias herramientas en un mismo entorno de desarrollo y depende de las arquitectura de Hw que se utilicen.

3. Modelo para el problema del particionado hardware/software

Teniendo en cuenta los trabajos previos y la especificidad de los SEs, se ha definido un modelo PHS que combina las siguientes características: granularidad, métricas asociadas a los bloques funcionales, modelo computacional utilizado, codificación de la solución, dominio de las variables y función objetivo. De esta forma, se define el diseño a particionar como:

$$D = (F, Gc, C) \quad (1)$$
$$F = \{f_1, f_2, ..., f_n\} \quad (2)$$
$$Gc = \{N, A\} \quad (3)$$
$$C = \{c_1, c_2, ..., c_n\} \quad (4)$$

Donde F representa el conjunto de los métodos o funciones de los que consta el problema, siguiendo una granularidad gruesa. Está comúnmente aceptado que el modelado de grano grueso es más apropiado para los SE ya que el utilizar grano fino generaría muchas transacciones entre Hw y Sw; lo cual pudiera dar lugar a peores resultados. Gc es el grafo de llamadas, cuyos nodos $N = F$ corresponden con los propios métodos y los arcos aij representan una llamada desde la función f_i a la función f_j. C es el coste asociado a cada nodo o función y se expresa por la terna (st_i, ha_i, ht_i). La figura 1 muestra un ejemplo sencillo de código correspondiente al sistema a particionar y su representación.

Según este modelo, una solución al problema del particionado se expresa como un conjunto de elementos binarios $X = \{x_1, x_2, ..., x_n\}$ donde el elemento i representa el tipo de implementación asignada a la función p_i; si $x_i = 1$ indica implementación Hw e implementación Sw en caso contrario. Teniendo en cuenta lo anteriormente planteado es posible calcular el área de Hw utilizada (A) por el diseño como:

$$A = \left(\sum_{i=1}^{n} x_i ha_i \right) + CostoArq \quad (5)$$

CostoArq representa el coste básico de la arquitectura. Este coste está asociado al hecho de que cualquier solución, incluso una solución completamente Sw, necesita de unas infraestructuras Hw mínimas (periféricos, memoria, procesador, etc.)

Finalmente se define el tiempo de ejecución del sistema (T) como:

$$T = \sum_{i=1}^{n}[(1 - x_i)st_i + x_i ht_i] \tag{6}$$

En este trabajo se utiliza como métrica el Índice de rendimiento (Ir) propuesto por [25]. Esta métrica permite evaluar conjuntamente el impacto de una partición determinada en el coste hardware (A- área ocupada) y en el tiempo de ejecución de la aplicación (T). De esta forma el proceso de exploración es similar al que en la práctica lleva a cabo un diseñador humano a la hora de abordar la partición de un sistema embebido. Además las restricciones del sistema se consideran como penalizaciones, con el fin de incluir en el espacio de búsqueda soluciones que, si bien no forman parte de éste, puedan ayudar a llegar a la solución deseada. De esta forma se define la función objetivo como:

$$\text{mín}\, Ir = A * T + P_A + P_T. \tag{7}$$

Los términos P_A y P_T que aparecen en la ecuación anterior representan las penalizaciones de área y tiempo respectivamente que se aplicarían en caso de que la evaluación de las dos métricas para una solución estén por encima de un umbral definido por el diseñador. Ambas penalizaciones se definen por partes como:

$$P_A = \begin{cases} A/A_{max} & A > A_{max} \\ 0 & A \leqslant A_{max} \end{cases} \qquad P_T = \begin{cases} T/T_{max} & T > T_{max} \\ 0 & T \leqslant T_{max} \end{cases}$$

4. Experimentos y resultados

Como ya se ha comentado anteriormente, no existen bancos de prueba consensuados y bien definidos que permitan realizar una comparación efectiva entre algoritmos. Para solventar esta dificultad adoptaremos la estrategia comúnmente aceptada de utilizar benchmarks sintéticos o simulados. De esta forma los resultados obtenidos serán extrapolables e independientes del dominio de aplicación y de las plataformas finales de implementación.

4.1. Exploración del espacio de soluciones

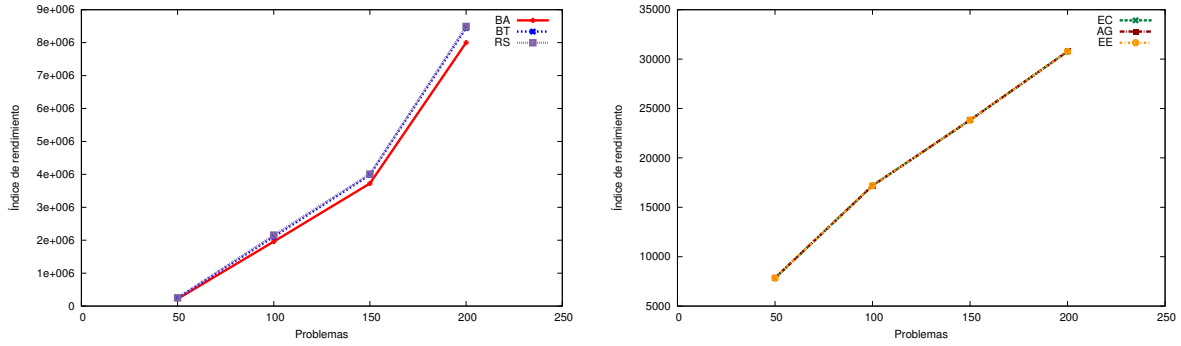
Los experimentos se ejecutaron sobre cuatro sistemas o problemas representados por grafos de 50, 100, 150 y 200 nodos. Los valores de tiempo de ejecución en software (st_i), área de hardware ocupada (ha_i) y el tiempo de ejecución en hardware (ht_i) fueron generados aleatoriamente, al igual que en [9] [23] [35] [3]. El tiempo de ejecución en software (st_i) se generó aleatoriamente en el rango [1,30], mientras que para el área de hardware (ha_i) los valores generados fueron en el rango [1,100]. Como el tiempo de ejecución de un nodo software generalmente es mayor que su contraparte hardware, el tiempo de ejecución del nodo en hardware (ht_i) se generó aleatoriamente en el rango $[1, \frac{1}{2} * st_i]$.

Con estas aplicaciones sintéticas (problemas) se realizaron 30 exploraciones del espacio de particionado utilizando cada uno de los 6 algoritmos de búsqueda. En cada exploración se generaron 50.000 particiones y las consiguientes evaluaciones de la función objetivo. La diferencia entre las distintas exploraciones llevadas a cabo para un algoritmo determinado, estriba en el punto de partida de la exploración, siendo este establecido aleatoriamente. La tabla 2 resume las características principales de los problemas utilizados en los experimentos. Como puede apreciarse, se estableció el mismo $CosteArq$ (área mínima) para todos los problemas.

Para realizar los experimentos se utilizaron algoritmos metaheurísticos basados en un punto (Búsqueda aleatoria (BA), Búsqueda tabú (BT), Escalador de colinas estocástico (EC), Recocido simulado (RS)) y poblacionales (Algoritmo genético (AG), Estrategia evolutiva (EE)). La codificación de los algoritmos se realizó mediante la biblioteca Biciam [10], la cual implementa un modelo unificado de algoritmos metaheurísticos. En el caso de Recocido simulado los parámetros utilizados fueron: temperatura inicial = 15, temperatura final = 0 y un $\alpha = 0{,}93$. Para el caso del Algoritmo genético la población inicial es de 50 individuos, el factor de truncamiento es del 30 % de la población inicial, mientras que la probabilidad

Tabla 2: Características de los problemas generados

Problema	Cantidad de nodos	Área Máxima	Área Mínima	Tiempo Máximo	Tiempo Mínimo
1	50	2232	10	784	189
2	100	5564	10	1719	440
3	150	7048	10	2383	648
4	200	10757	10	3077	865



(a) Algoritmos: BA, BT y RS

(b) Algoritmos: EC, AG y EE

Figura 2: Calidad de la solución obtenida por cada algoritmo.

de mutación y cruzamiento es de 0.9 para ambos casos. Por último, para el algoritmo de Estrategia evolutiva la población inicial y el factor de truncamiento son similares al Algoritmo genético, utilizando una probabilidad de mutación de 1. Es importante aclarar que los parámetros empleados en todos los algoritmos fueron los que mejor resultados ofrecieron después de analizar diversos valores.

La figura 2 muestra el comportamiento de los algoritmos respecto al promedio de los resultados obtenidos en la evaluación de la función objetivo para cada uno de los problemas analizados. Como puede apreciarse, para grafos de 50 nodos los algoritmos alcanzan resultados similares; mientras que para el resto de los grafos si existen diferencias significativas entre dos grupos de algoritmos (BA, BT y RS, no alcanzan nunca los resultados del resto de algoritmos). Además, para el caso de los algoritmos EC, AG y EE, estos alcanzan los mismos valores en la función objetivo en cada uno de los problemas tratados.

Debido a la similitud de resultados de los algoritmos EC, AG y EE, se realizó un segundo conjunto de experimentos con el fin de esclarecer las diferencias entre ellos. En este experimento se obtuvo detalladamente la cantidad de iteraciones promedio necesarias para cada algoritmo converger a su valor mínimo. Como puede apreciarse en la tabla 3, el algoritmo EC es el que más rápido converge en todos los problemas, lo que implica que este algoritmo obtiene las mejores soluciones en menor tiempo. Es importante hacer notar que los resultados obtenidos en este trabajo son similares a los obtenidos en [28] para el trazado de grafos.

Tabla 3: Número de iteraciones mínimas como promedio para converger al valor mínimo

Cantidad de nodos	Escalador de colinas	Algoritmo genético	Estrategia evolutiva
50	**828**	1089	1264
100	**1627**	1856	2143
150	**2438**	4297	3642
200	**2808**	4810	6032

Finalmente se diseñó un tercer experimento con el objetivo de esclarecer la causa del buen comportamiento del EC. Este experimento consistió en ejecutar 30 veces el algoritmo búsqueda aleatoria con 10.000 evaluaciones por cada una de las ejecuciones. El objetivo de este experimento es estimar si en el espacio de soluciones de este problema no existen muchos óptimos locales que provoquen que el EC se estanque en un óptimo local y le impida llegar a una mejor solución. El procedimiento para calcular la cantidad de óptimos locales fue explorar la vecindad de cada una de las 30.000 soluciones generadas y ver cuantas de estas fueron óptimos locales. Los resultados obtenidos arrojaron que como promedio solo el 2 % de las soluciones generadas fueron óptimos locales, con lo que se puede concluir que en el espacio de soluciones no existe una cantidad suficiente de óptimos locales que provoquen un estancamiento en el EC.

4.2. Análisis multi-objetivo de los resultados.

Teniendo en cuenta que en la función objetivo coexisten dos métricas contrapuestas, se realizó un análisis de los resultados desde el punto de vista multiobjetivo. En la primera parte del análisis se determinó para cada algoritmo el frente optimal de Pareto [5]. En una segunda etapa, se elaboró un frente optimal de Pareto unificado a partir de las soluciones no dominadas de cada algoritmo. Es importante señalar que estos análisis se realizaron sobre los cuatro problemas utilizados con anterioridad (50, 100, 150 y 200 nodos).

Análisis de los Frentes de Pareto. Para cada algoritmo se determinó su Frente de Pareto, seleccionando de entre todas las soluciones generadas sólo las del tipo "no dominada". El análisis de este frente permite establecer comparativas entre los diferentes métodos de exploración. Para ello se tuvieron en cuenta tres factores: el porcentaje de soluciones no dominadas, el tamaño del frente y la distribución de las soluciones.

En la figura 3 es posible apreciar el comportamiento de los algoritmos para cada uno de los problemas tratados con respecto al porcentaje de soluciones no dominadas. Cuanto mayor es este porcentaje, mayor es la calidad del algoritmo de exploración. Esta medida se define como:

$$PSnD_{XX} = \frac{SnD_{XX}}{U_{XX}} * 100$$

Donde U_{XX} representa la cantidad de soluciones únicas generadas por el algoritmo XX, SnD_{XX} la cantidad de soluciones no dominadas del algoritmo XX y $PSnD_{XX}$ el porcentaje de soluciones no dominadas del algoritmo XX.

Es importante señalar que los algoritmos BA, BT y RS, si bien fueron los que mayor cantidad de soluciones únicas generaron, fueron los que menor porcentaje de no dominadas tuvieron, es decir, fueron capaces de encontrar muchas soluciones pero el espacio donde se movieron estaba alejado de su correspondiente frente de Pareto. En cambio los algoritmos EC, EE y AG generaron menos soluciones únicas pero con un mayor porcentaje de soluciones no dominadas. Se debe resaltar que el algoritmo EC fue el que mayor porcentaje obtuvo en la mayoría de los casos, tal y como puede apreciarse, seguido por el algoritmo EE y AG.

A continuación se presentan dos elementos que aportarán una caracterización cualitativa del frente de Pareto en cada algoritmo, siendo estas el tamaño del frente y la distribución de las soluciones sobre él. La importancia de emplear ambas es que dan una idea del espacio que abarca el frente y dentro de este espacio como se compartan las soluciones en cada parte. Las características esperadas para cualquier frente es que sea amplio y que las soluciones se encuentren bien distribuidas sobre él.

Al realizar un análisis visual de los frentes obtenidos, se aprecia que para los cuatro problemas estudiados los algoritmos se comportan de forma similar. En cuanto al tamaño del frente los algoritmos EC, EE y AG fueron los que obtuvieron un frente más amplio, mientras que para BA, BT y RS fue más compacto. La distribución de las soluciones en los tres primeros fue aceptable con una mayor dispersión hacia uno de los extremos, mientras que para el resto sí estuvieron bien distribuidas a lo ancho de todo el frente. Para ejemplificar este comportamiento, la figura 4 muestra el frente de Pareto para cada algoritmo en el problema de 150 nodos.

Frente de Pareto unificado.

Si bien las dos medidas anteriores dan una idea de la calidad de las soluciones no dominadas, no existe un algoritmo que supere claramente al resto. En la segunda etapa del análisis multiobjetivo se

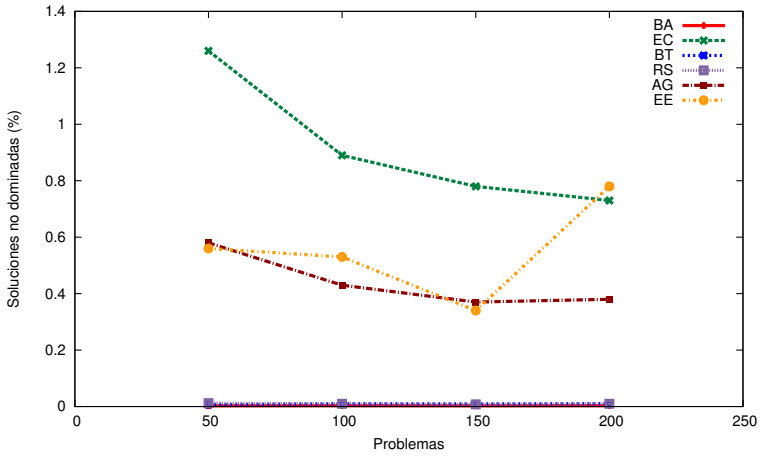
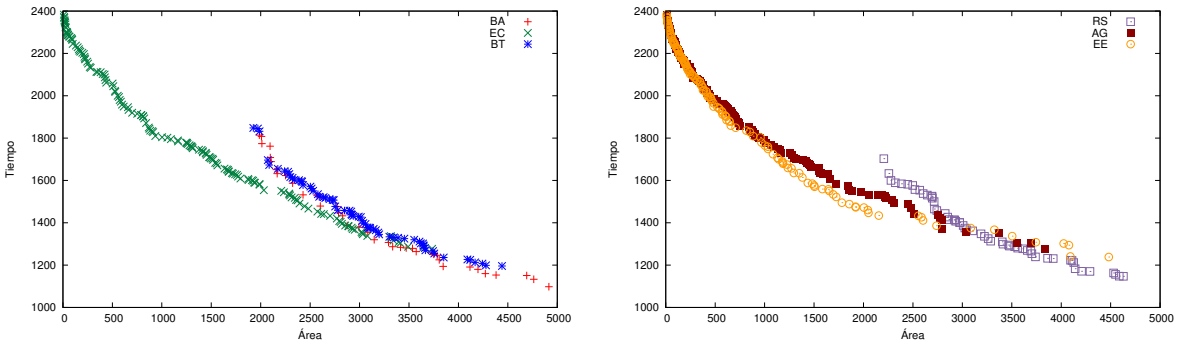


Figura 3: Comportamiento de los algoritmos con respecto al porcentaje de soluciones no dominadas generadas.



(a) Algoritmos: BA, EC y BT

(b) Algoritmos: RS, AG y EE

Figura 4: Frente optimal de Pareto de cada algoritmo para 150 nodos.

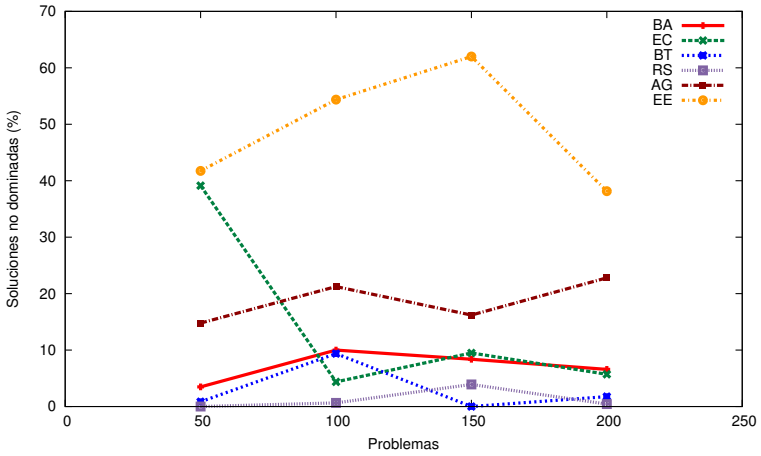


Figura 5: Comportamiento de los algoritmos con respecto al porcentaje de soluciones no dominadas en el frente unificado.

construyó un frente de Pareto unificado, a partir de las soluciones no dominadas de cada algoritmo, el cual permitirá analizar el aporte de cada algoritmo a este frente y las características del mismo.

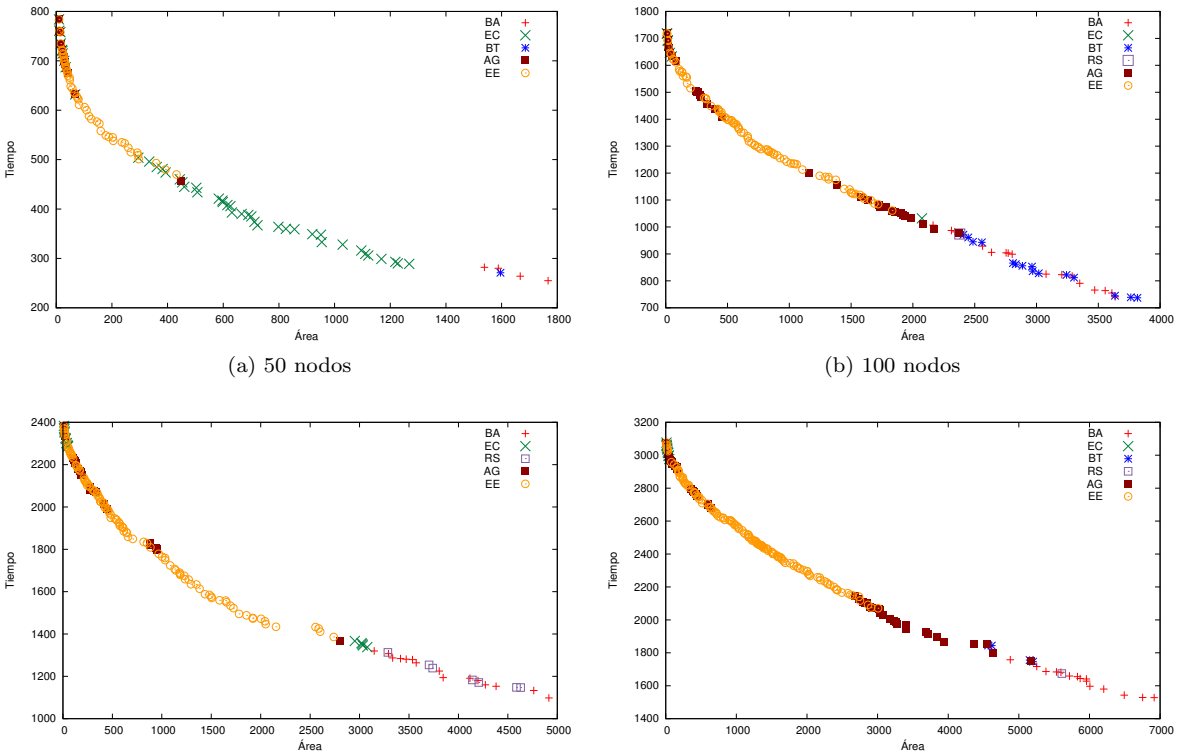
El porcentaje de soluciones no dominadas al frente unificado se determina como:

$$PSU_{XX} = \frac{SU_{XX}}{SFU} * 100$$

Donde SFU representa la cantidad de soluciones presentes en el frente unificado, SU_{XX} la cantidad de soluciones no dominadas en el frente perteneciente al algoritmo XX y PSU_{XX} el porcentaje de soluciones no dominadas del algoritmo XX en el frente unificado.

La figura 5 muestra el aporte de cada algoritmo al frente unificado. Es necesario resaltar en esta gráfica que el algoritmo EE es el que mejor porcentaje logra para todos los problemas. De manera general todos los algoritmos aportan al menos una solución en casi todos los problemas, salvo RS que para 50 nodos no aporta ninguna y BT que no lo hace para 150 nodos. Mientras que para los tres últimos problemas, los de mayor cantidad de nodos, los algoritmos EE y AG, en ese orden, son los que mayor cantidad de soluciones aportan al frente.

En la figura 6 se puede observar el frente de Pareto unificado para cada uno de los problemas tratados, a partir de considerar las contribuciones de cada algoritmo. Al analizar los gráficos es posible hacer notar que en todos los problemas los resultados muestran la tendencia de los algoritmos a dominar determinadas regiones del frente. Es decir, existen algoritmos que sus soluciones están dominadas por el área, otros que sus soluciones están dominadas por el tiempo y otros algoritmos que la mayor parte de las soluciones se encuentran en el centro del frente. La mayoría de las soluciones aportadas por los algoritmos BA, BT y RS tienden a estar dominadas por la métrica tiempo, mientras que los algoritmos EE, AG y EC tienden a dominar el extremo contrario del frente, o sea soluciones dominadas por el área. Además los algoritmos EC y AG tienden a aportar soluciones en el centro del frente, excepto para el problema de 200 y 50 nodos respectivamente. Por último destacar qeu el algoritmo EE es el que mantiene en todos los problemas una distribución y ubicación similar.



(a) 50 nodos

(b) 100 nodos

(c) 150 nodos

(d) 200 nodos

Figura 6: Frente optimal de Pareto unificado para los distintos problemas.

5. Conclusiones

En este trabajo se compararon seis algoritmos metaheurísticos en el problema del PHS de sistemas embebidos, utilizando para esto un mismo modelo con diferentes instancias de problemas para grafos de 50, 100, 150 y 200 nodos. Todas las instancias se generaron de forma sintética definiendo de forma aleatoria los valores de tiempo de ejecución en software (st_i), área de hardware ocupada (ha_i) y el tiempo de ejecución en hardware (ht_i) para cada nodo. En el modelo se utilizó, por primera vez, el Índice de rendimiento como función objetivo.

Los resultados que se obtuvieron, en un primer análisis, mostraron que el Escalador de colinas fue el algoritmo que mejor solución obtuvo en la menor cantidad de iteraciones. Lo relevante de este resultado está en que en ninguna de las publicaciones consultadas se utiliza este algoritmo para el problema PHS. Además se comprobó que los resultados obtenidos por el escalador de colina se deben a las características del espacio de soluciones donde no existen un número considerable de óptimos locales.

A partir de los resultados obtenidos en el análisis desde el punto de vista multiobjetivo se pudo comprobar, que si bien el Escalador de colinas estocástico fue el algoritmo que mayor cantidad de soluciones no dominadas obtuvo en la mayoría de los casos, el algoritmo Estrategia evolutiva aportó una mayor cantidad de este tipo de soluciones al frente de Pareto unificado.

A la vista de los resultados, se puede concluir que el tipo de solución obtenida (dominadas por área o por tiempo) está determinada por el algoritmo utilizado. Este resultado sugiere la propuesta de un algoritmo combinado que permita darle solución al problema PHS, según el tipo de solución que desee obtener el diseñador en un momento determinado. Finalmente el estudio comparativo realizado en este trabajo podría facilitar la toma de decisiones a la hora de seleccionar el algoritmo más apropiado en función de las características de la aplicación.

Referencias

[1] Pradeep Adhipathi. Model based approach to hardware/software partitioning of soc designs. Master's thesis, Faculty of the Virginia Polytechnic Institute and State University, 2004.

[2] F.-F. Amin, K. Mehdi, F. Sied Mehdi, and S. Saeed. Hw/sw partitioning using discrete particle swarm. In *17th ACM Great Lakes symposium on VLSI*, Stresa-Lago Maggiore, Italy., 2007. ACM.

[3] Péter Arató, Zoltán Ádám Mann, and András Orbán. Algorithmic aspects of hardware/software partitioning. *ACM Trans. Des. Autom. Electron. Syst.*, 10(1):136–156, January 2005.

[4] A. Bhattacharya, A. Konar, S. Das, C. Grosan, and A. Abraham. Hardware software partitioning problem in embedded system design using particle swarm optimization algorithm. In *International Conference on Complex, Intelligent and Software Intensive Systems*, pages 171–176, 2008.

[5] Yu Chen, Xiufen Zou, and Weicheng Xie. Convergence of multi-objective evolutionary algorithms to a uniformly distributed representation of the pareto front. *Inf. Sci.*, 181(16):3336–3355, August 2011.

[6] L. A. Cortés, P. Eles, and Z. Peng. A survey on hardware/software codesign representation models. Save project report, Dept. of Computer and Information Science, Linköping University, Linköping, Sweden, 1999. `http://ftp.ida.liu.se/labs/eslab/publications/pap/db/SAVE99.pdf`.

[7] Giovanni De Micheli and Rajesh K. Gupta. Hardware/software co-design. *Proceedings of the IEEE*, 85(3):349–365, 1997.

[8] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli. System level hardware/software partitioning based on simulated annealing and tabu search. *Des Autom Embed Syst*, 2(1):5–32, 1997.

[9] Rolf Ernst, Jorg Henkel, and Thomas Benner. Hardware-software cosynthesis for microcontrollers. *IEEE Des. Test*, 10(4):64–75, October 1993.

[10] Jenny Fajardo and Alejandro Rosete. Algoritmo multigenerador de soluciones para la competencia y colaboración de generadores metaheurísticos. *Revista Internacional de Investigación de Operaciones (RIIO)*, 1, 2011.

[11] C.M. Fernandes, J.J. Merelo, and A.C. Rosa. A comparative study on the performance of dissortative mating and immigrants-based strategies for evolutionary dynamic optimization. *Information Sciences*, 181:4428–4459, 2011. doi: 10.1016/j.ins.2011.05.022.

[12] Diana Göhringer, Michael Hübner, Michael Benz, and Jürgen Becker. A design methodology for application partitioning and architecture development of reconfigurable multiprocessor systems-on-chip. In *18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, 2010.

[13] Rajesh K. Gupta and Giovanni De Micheli. Hardware-software cosynthesis for digital systems. *IEEE Design & Test of Computers*, 10(3):29–41, July 1993.

[14] Jörg Henkel and Rolf Ernst. An approach to automated hardware/software partitioning using a flexible granularity that is driven by high-level estimation techniques. *IEEE Trans. Very Large Scale Integr. Syst.*, 9(2):273–290, April 2001.

[15] Wu Jigang and Thambipillai Srikanthan. Algorithmic aspects of area-efficient hardware/software partitioning. *Journal of Supercomputing*, 38(3):223–235, December 2006.

[16] Wu Jigang and Thambipillai Srikanthan. Low-complex dynamic programming algorithm for hardware/software partitioning. *Information Processing Letters*, 98(2):41–46, April 2006.

[17] Wu Jigang, Thambipillai Srikanthan, and Tao Jiao. Algorithmic aspects for functional partitioning and scheduling in hardware/software co-design. *Des Autom Embed Syst*, 12(4):345–375, 2008.

[18] Wu Jigang, Thambipillai Srikanthan, and Guang-Wei Zou. New model and algorithm for hardware/software partitioning. *Journal of Computer Science and Technology*, 23(4):644–651, July 2008.

[19] A. Kalavade and E. A. Lee. The extended partitioning problem: Hardware/software mapping, scheduling and implementation-bin selection. *Des Autom Embed Syst*, 2(2):125–164, 1997.

[20] Young-Jun Kim and Taewhan Kim. A hw/sw partitioner for multi-mode multi-task embedded applications. *Journal of VLSI Signal Processing*, 44:269–283, 2006.

[21] Marisa López-Vallejo and Juan Carlos López. On the hardware-software partitioning problem: System modeling and partitioning techniques. *ACM Trans. Des. Autom. Electron. Syst. (TODAES)*, 8(3):269–297, July 2003.

[22] M. López Vallejo, J.C. López, and C Iglesias. Hardware-software partitioning at the knowledge level. *J. Applied Intell.*, 10:173–184, 1999.

[23] J. Madsen, J. Grode, P.V. Knudsen, M.E. Petersen, and A. Haxthausen. Lycos: the lyngby cosynthesis system. *Des Autom Embed Syst*, 2(2):195–235, 1997.

[24] Zoltán Ádám Mann. *Partitioning algorithms for hardware/software co-design.* PhD thesis, Budapest Univerity of Technology and Economics, Department of Control Engineering and Information Technology, 2004.

[25] Luiza de M. Mourelle and Nadia Nedjah. Efficient cryptographic hardware using the co-design methodology. In *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04) Volume 2 - Volume 2*, ITCC '04, pages 508–, Washington, DC, USA, 2004. IEEE Computer Society.

[26] P. P. Arató, S. Zuhász, Z. A. Mann, A. Orbán, and D. Papp. Hardware/software partitioning in embedded system design. In *IEEE International Symposium on Intelligent Signal Processing*, 2003.

[27] M. Purnaprajna, M. Reformata, and W. Pedrycza. Genetic algorithms for hardware-software partitioning and optimal resource allocation. *Journal of Systems Architecture*, 53(7):339–354, 2007.

[28] Alejandro Rosete-Suárez, Alberto Nogueira-Keeling, Alberto Ochoa-Rodríguez, and Michèle Sebag. Hacia un enfoque general del trazado de grafos. *Revista Iberoamericana de Inteligencia Artificial.*, 3(8):18–26, 1999.

[29] M. Schwiegershausen, H. Kropp, and P. Pirsch. A system level hw/sw partitioning and optimization tool. In *Proceedings of the conference on European design automation*, EURO-DAC '96/EURO-VHDL '96, pages 120–125, Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

[30] Adnan Shaout, Ali H. El-Mousa, and Khalid Mattar. Models of computation for heterogeneous embedded systems. In Sio-Iong Ao and Len Gelman, editors, *Electronic Engineering and Computing Technology*, volume 60 of *Lecture Notes in Electrical Engineering*, pages 201–213. Springer Netherlands, 2010. doi: 10.1007/978-90-481-8776-8_18.

[31] V. Srinivasan, S. Radhakrishnan, and R. Vemuri. Hardware software partitioning with integrated hardware design space exploration. In *DATE*, pages 28–35, Paris, France, 1998.

[32] F. Vahid and T. D. Le. Extending the kenighan/lin heuristic for hardware and software functional partitioning. *Des Autom Embed Syst*, 2:237–261, 1997.

[33] Frank Vahid. Modifying min-cut for hardware and software functional partitioning. In *Proceedings of the 5th International Workshop on Hardware/Software Co-Design*, CODES '97, pages 43–48, Washington, DC, USA, 1997. IEEE Computer Society.

[34] Frank Vahid and Tony Givargis. *Embedded System Design: A Unified Hardware/Software Introduction.* J. Wiley and Sons, 2002.

[35] Theerayod Wiangtong, Peter Y. K. Cheung, and Wayne Luk. Comparing three heuristic search methods for functional partitioning in hardware/software codesign. *Des Autom Embed Syst*, 6(4):425–449, 2002.

[36] Wayne Wolf. A decade of hardware/software codesign. *Computer*, 36(4):38–43, April 2003.

[37] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1996.

[38] Shengxiang Yang and Xin Yao. Experimental study on population-based incremental learning algorithms for dynamic optimization problems. *Soft Computing*, 9:815–834, 2005. doi: 10.1007/s00500-004-0422-3.

[39] Yiguo Zhang, Wenjian Luo, Zeming Zhang, Bin Li, and Xufa Wang. A hardware/software partitioning algorithm based on artificial immune principles. *Applied Soft Computing*, 23(32):383–391, 2007.