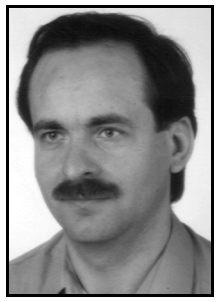
Stanisław DENIZIAK, Adam GÓRSKI
POLITECHNIKA KRAKOWSKA, KATEDRA INFORMATYKI TECHNICZNEJ

Kosynteza rozproszonych systemów wbudowanych metodą programowania genetycznego

Dr hab. inż. Stanisław DENIZIAK

Ukończył studia na Wydziale Elektroniki Politechniki Warszawskiej, obronił pracę doktorską w 1994r, a habilitacyjną w 2006r. Jest profesorem nadzwyczajnym w Katedrze Informatyki Technicznej Politechniki Krakowskiej oraz profesorem wizytującym w Katedrze Informatyki Politechniki Świętokrzyskiej. Jego zainteresowania naukowe to metody szybkiego prototypowania systemów informatycznych, projektowanie systemów wbudowanych, testowanie i diagnostyka systemów cyfrowych.

e-mail: pedenizi@cyf-kr.edu.pl



Mgr inż. Adam GÓRSKI

Ukończył studia na Wydziale Inżynierii Elektrycznej i Komputerowej Politechniki Krakowskiej. Jego zainteresowania naukowe to: algorytmy genetyczne, projektowanie systemów wbudowanych.

e-mail: adamgorsk@gmail.com

Streszczenie

W pracy zaprezentowana jest nowa metoda kosyntezy systemów wbudowanych specyfikowanych za pomocą grafów zadań, bazująca na metodzie programowania genetycznego. Przedstawione są propozycje reprezentowania procesu konstrukcji takiego systemu w formie drzewa stanowiącego tzw. genotyp. Następnie na drodze ewolucji (krzyżowania, mutacji, selekcji) generowane są kolejne „pokolenia" drzew, konstruujących systemy o coraz lepszych parametrach. W odróżnieniu od tradycyjnego podejścia genetycznego w metodzie programowania genetycznego (DGP) operuje się nie bezpośrednio na cechach rozwiązania (czyli tzw. fenotypach) ale na genotypach odpowiadających za tworzenie rozwiązań o wskazanych cechach. Przedstawione wyniki wykonanych eksperymentów świadczą o dużych możliwościach metody DGP również w zakresie kosyntezy.

Słowa kluczowe: programowanie genetyczne, kosynteza.

Hardware/software Co-Synthesis of Distributed Embedded Systems Using Genetic Programming

Abstract

This work presents a novel approach to hardware-software co-synthesis of distributed embedded systems, based on the developmental genetic programming. Unlike other genetic approaches where chromosomes represent solutions, in our method chromosomes represent system construction procedures. Thus, not the system architecture but the co-synthesis process is evolved. Finally a tree describing a construction of the final solution is obtained. The optimization process will be illustrated with examples. According to our best knowledge it is the first DGP approach that deals with the hardware-software co-synthesis.

Keywords: genetic programming, hardware/software co-design.

1. Wprowadzenie

Kosynteza [1] polega na automatycznej generacji architektury systemu wbudowanego na podstawie specyfikacji przedstawionej w formie współbieżnych procesów. Celem kosyntezy jest optymalizacja właściwości systemu takich jak: koszt, czas wykonania lub pobór mocy. Większość współczesnych metod kosyntezy zakłada architekturę rozproszoną, złożoną z elementów obliczeniowych (processing elements PE), które są komponentami sprzętowymi (HC) bądź software'owymi (PP). Proces kosyntezy składa się z następujących zadań: alokacja zasobów, przyporządkowanie zadań do zasobów, szeregowanie zadań i transmisji.

Znaczna większość istniejących algorytmów kosyntezy [2, 3] są to algorytmy rafinacyjne, startujące od suboptymalnego rozwiązania i starające udoskonalić jakość systemu poprzez jego lokalne zmiany. Jako rozwiązanie początkowe zwykle wybierana jest najszybsza architektura, gdzie każdy proces jest wykonywany przez inny PE.

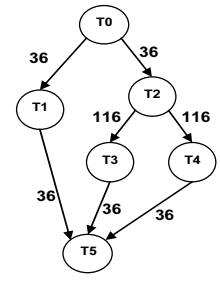
Modyfikacje polegają na przemieszczaniu zadań do innych PE, usuwaniu i dodawaniu PE, itp.. Niektóre algorytmy rafinacyjne są zdolne do opuszczania lokalnego minimum [3] optymalizowanych funkcji, ale rezultaty wciąż są subotymalne. Algorytmy konstrukcyjne budują system poprzez alokację nowych komponentów [4]. Mają niską złożoność obliczeniową, ale mają też tendencję do zatrzymywania się w lokalnych minimach optymalizowanych parametrów. Jakość uzyskanych wyników można poprawić poprzez cofanie się w konstrukcji rozwiązania [4], jednak stwarza to niebezpieczeństwo zapętlenia się algorytmu a ponadto znacznie zwiększa jego złożoność obliczeniową. Algorytmy probabilistyczne mają zdolność do wydobywania się z lokalnych minimów. W szczególności dosyć dobre wyniki uzyskano z wykorzystaniem algorytmów genetycznych [5]. Metody genetyczne są też często używane do optymalizacji w ramach wybranych problemów syntezy systemowej[6, 7].

W pracy tej prezentujemy nowatorską metodę kosyntezy sprzętowo-softwarowej opartą na programowaniu genetycznym [8]. Wszystkie istniejące metody definiują ewolucję poprzez rafinację architektury, w której chromosomy odpowiadają rozwiązaniom. W naszym algorytmie ewolucji podlega proces kosyntezy, a chromosomy reprezentują decyzje projektowe. Dzięki temu jako rezultat ewolucji otrzymujemy metodę konstruowania docelowego systemu.

Rozdział 2 przedstawia założenia stosowane w pracy. W rozdziale 3 opisany jest prezentowany algorytm. W rozdziale 4 podamy wyniki eksperymentów. Pracę podsumowuje rozdział 6.

2. Założenia

Zachowanie systemu wbudowanego opisuje graf zadań $G=\{V,E\}$, który jest acyklicznym grafem skierowanym. Każdy węzeł $v_i \in V$ reprezentuje zadanie a krawędź $e_{i,j} \in E$ opisuje zależność między zadaniami v_i oraz v_j. Każda krawędź posiada etykietę $d_{i,j}$ opisującą ilość danych, która musi być przesłana pomiędzy połączonymi zadaniami. Przykładowy graf zadań został przedstawiony na rys. 1.



Rys. 1. Przykładowy graf zadań
Fig. 1. Sample task graph.

Zakładamy, że istnieje baza danych PE i szyn komunikacyjnych CL, zawierająca czasy wykonania zadań (t), powierzchnię modułów (s) dla implementacji w formie SOC (System on Chip), oraz przepustowość kanałów komunikacyjnych (b). Tabela 1 pokazuje przykładową bazę zasobów dla systemu opisanego przez graf zadań z rys. 1. Z każdym zadaniem v_i może być związane wymaganie czasowe c_i, oznaczające chwilę czasową, do której zadanie musi zdążyć się wykonać. Wszystkie wymagania czasowe muszą być dotrzymane przez architekturę docelową. Niech ts_i będzie czasem rozpoczęcia wykonywania zadania v_i. Docelowy system jest poprawny wtedy i tylko wtedy, gdy spełnione są następujące warunki:

$$\bigvee_{e_{i,j}} ts_j \geq ts_i + t_{i,r_i} + tc_{i,j}, \tag{1}$$

$$\bigvee_{i} ts_i + t_{i,r_i} \leq c_i, \tag{2}$$

gdzie:

$$tc_{i,j} = \left\lceil \frac{d_{i,j}}{b_{CL_{i,j}}} \right\rceil. \tag{3}$$

r w (1) i (2) oznacza typ PE wykonujący zadanie v_i, $CL_{i,j}$ jest typem szyny komunikacyjnej, jeśli zadania v_i oraz v_j są przyporządkowane do tego samego PE, wówczas $tc_{i,j}$=0. Warunek (1) zakłada poprawne uporządkowanie zadań, podczas gdy (2) zapewnia spełnienie wszystkich wymagań czasowych.

Tab. 1. Baza danych zasobów
Tab. 1. Resource database

	PP1 S=300		**PP2** S=400		**HC1$_1$**		**HC2$_1$**	
	t	**s**	**t**	**s**	**t**	**s**	**t**	**s**
T0	200	6	150	6	60	180	40	250
T1	50	4	40	3	24	90	15	150
T2	-	-	360	18	250	200	150	500
T3	250	13	220	14	150	140	100	300
T4	150	12	160	15	70	20	30	50
T5	60	5	55	5	35	110	-	-
CL1 b=8	s=2		s=2		s=10			
CL2 b=16	-		s=2		s=15			

Niech architektura docelowa dla systemu opisanego grafem zadań zawierającym n procesów, składa się z m programowalnych procesorów, p szyn komunikacyjnych, wtedy całkowita powierzchnia zajmowana przez system SoC wyraża się wzorem:

$$A = \sum_{i=1}^{m} S_{PE_i} + \sum_{j=1}^{n} s_j + \sum_{k=1}^{p} \sum_{l=1}^{P_k} s_{CL_k,PC_l}, \tag{4}$$

gdzie: CL_k jest typem k-tej magistrali podłączonej do P_k i PC_l jest typem l-tego PE podłączonego do CL_k. Celem kosyntezy jest znalezienie architektury z najmniejszym A zapewniającym zachowanie warunków (1) i (2).

3. Opis algorytmu

Zgodnie z zasadą rozwojowego programowania genetycznego [8] w opracowywanym algorytmie ewolucji podlega drzewo (genotyp) opisujące konstruowanie projektowanego systemu. Korzeń określa konstrukcję systemu embrionicznego, węzły odpowiadają funkcjom budującym system. Aby zapewnić implementację wszystkich zadań, struktura genotypu jest drzewem odwzorowującym graf zadań, gdzie każdy węzeł implementuje zadanie odpowiadające jego odpowiednikowi w grafie zadań. Struktura tego drzewa jest zachowana po mutacji i krzyżowaniu.

Embrion implementuje pierwsze zadanie z podanego grafu zadań, przez co liczba możliwych embrionów jest liczbą elementów PE w bibliotece zasobów. Dla każdego rozwiązania embrion jest wybierany losowo na początku projektowania systemu. Dla przykładu z rys. 1, każdy z czterech PE z tabeli 1 może być embrionem.

Każdy węzeł w drzewie genotypu reprezentuje funkcję implementującą kolejne zadanie z grafu zadań. Każda funkcja składa się z następujących kroków:

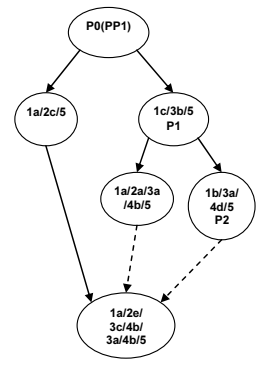
1. Alokacja nowego PE: ten krok jest opcjonalny.
2. Przyporządkowanie zadania do PE: ten krok zawsze musi być wykonany. Jeśli poprzedni krok został wykonany wtedy zadanie jest przyporządkowane do nowego PE, w przeciwnym razie do dowolnego PE z aktualnej architektury.
3. Alokacja nowego CL: ten krok jest opcjonalny.
4. Przyporządkowanie transmisji do CL: kroki 3 i 4 są powtarzane dla każdej krawędzi wchodzącej do węzła odpowiadającego implementowanemu zadaniu.
5. Szeregowanie zadań: ten krok jest wykonywany tylko wtedy, gdy istnieje przynajmniej jeden PP z więcej niż jednym przyporządkowanym zadaniem.

Najpierw generowane jest pokolenie początkowe złożone z losowo wygenerowanych genotypów. Wielkość pokolenia początkowego definiuje parametr α i wynosi: Π=α**n***e* (*n* jest liczbą zadań, natomiast *e* liczbą możliwych embrionów). Dla każdej funkcji wszystkie kroki są generowane losowo zgodnie z tab. 2. Ostatnia kolumna przedstawia prawdopodobieństwo wyboru danej opcji.

Tab. 2. Opcje wyboru dla konstruowanego systemu
Tab. 2. System-construction options

Krok	Opcja	P
1.	a.Żaden	0.6
	b.Najmniejsza powierzchnia	0.1
	c.Najszybszy	0.1
	d.Najmniejsze *t*s*	0.1
	e.Najrzadziej używany	0.1
2.	a.Najmniejsza powierzchnia	0.2
	b.Najszybszy	0.2
	c.Najmniejsze wykorzystanie	0.2
	d.Najdłużej bezczynny	0.2
	e.Tak jak dla poprzednika	0.2
3.	a.Żadna	0.5
	b.Najmniejsza powierzchnia	0.2
	c.Największe *b*	0.2
	d.Najrzadziej używana	0.1
4.	a.Najmniejsza powierzchnia	0.3
	b.Najszybsza	0.3
	c.Najmniejsze wykorzystanie	0.2
	d.Najdłużej bezczynna	0.2
5.	Szeregowanie listowe	1.0

System jest konstruowany przez wykonywanie funkcji według poziomu węzła w drzewie genotypu. Węzły znajdujące się na tym samym poziomie są wykonywane od lewej do prawej. Zawsze wybierane są tylko opcje zapewniające spełnienie ograniczeń czasowych. rys. 2. przedstawia przykładowy genotyp dla grafu zadań z rys. 1.



Rys. 2. Przykładowy genotyp
Fig. 2. Sample genotype

Pierwszy węzeł oznacza embrion, w tym przypadku jest to procesor P0 typu PP1. Pozostałe węzły odpowiadają funkcjom kolejno konstruującym rozwiązanie. Każda funkcja opisana jest jako sekwencja par $n_i m_i$, gdzie n_i oznacza numer kroku, a m_i opcję tego kroku (wg tab. 2).

Zakładamy, że maksymalny czas zakończenia zadań wynosi c_5=650, wtedy system jest konstruowany następująco:

1. P0(PP1): procesor P0 (typu PP1) jest embrionem i wykonuje zadanie T0,
2. 1a/2c/5: zadanie T1 jest również przydzielone do P0, szeregowanie zadań T0→T1,
3. 1c/3b/5 P1: T2 jest implementowane na najszybszym PE ($HC2_2$), alokowana jest magistrala komunikacyjna CL1 między P0 i P1,
4. 1a/2a/3a/4b/5: T3 powinno być przyporządkowane do P0 (najmniejsza powierzchnia), ale to spowoduje naruszenie c_5, zatem następuje wybór $HC1_3$ oraz C0 do transmisji,
5. 1b/3a/4d/5 P2: T4 jest zaimplementowane na $HC1_4$ (najmniejsza powierzchnia), transmisja T2→T4 występuje za T2→T3 (T4 zakończy się wcześniej niż T3).
6. 1a/2e/3c/4b/3a/4b/5: T5 jest przyporządkowane do P0, alokowany jest CL2. Po uszeregowaniu T5 zakończy się w czasie 583.

Na drodze ewolucji poprzez użycie operacji genetycznych: reprodukcji, krzyżowania i mutacji, generowane są nowe pokolenia rozwiązań. Liczba rozwiązań w każdej populacji zawsze wynosi Π. Ewolucja jest kontrolowana przez parametry β, γ i δ w następujący sposób:

- Φ=β*Π: liczba rozwiązań uzyskana poprzez użycie reprodukcji,
- Ψ=γ*Π: liczba rozwiązań uzyskana poprzez użycie krzyżowania,
- Ω=δ*Π: liczba rozwiązań uzyskana poprzez użycie mutacji,
- β+γ+δ=1: w każdej populacji jest taka sama liczba osobników.

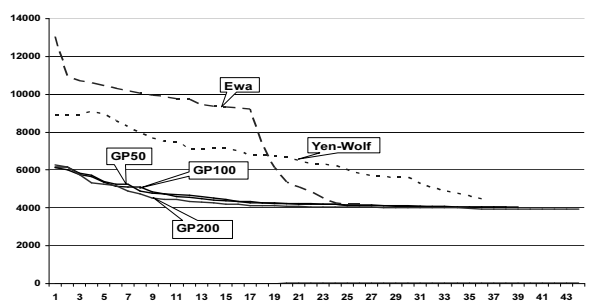
Rozwiązania są zestawione od najtańszego do najdroższego. Reprodukcja kopiuje Φ rozwiązań z aktualnej populacji. Rozwiązania są wybierane losowo, ale z różnym prawdopodobieństwem, które jest uzależnione od pozycji r w rankingu:

$$P = \frac{\Pi - r}{\Pi}, \tag{5}$$

Krzyżowanie losowo wybiera dwa rozwiązania, następnie pojedynczy punkt przecięcia jest wybierany losowo dla obu rodziców. Mutacja wybiera losowo jedno rozwiązanie, następnie losowo wybiera węzeł i podmienia go generując nowe opcje (z Tab. 2). Algorytm zatrzymuje się, jeśli najlepsze rozwiązanie nie zostało znalezione w ε kolejnych krokach.

4. Wyniki wykonanych eksperymentów

W celu oceny skuteczności opracowanej metody, wykonano syntezę systemu opisanego losowym grafem zadań, złożonym z 50 węzłów. Analogiczne eksperymenty wykonano dla algorytmów rafinacyjnych: Yen-Wolf [2] oraz Ewa [3].



Rys. 3. Przebieg optymalizacji systemu
Fig. 3. Optimization flow

Rysunek 3 przedstawia przebieg optymalizacji rozwiązań, w metodzie opartej o DGP eksperymenty wykonano dla 3 wartości parametru α: 50, 100 i 200 co odpowiada populacjom o liczebności: 10000, 20000 oraz 40000. Oś X na wykresie reprezentuje kolejne pokolenia lub kolejne kroki rafinacji.

Ostateczne wyniki syntezy przedstawione są w tabeli 3, przedstawiony jest koszt architektury oraz czas wykonania wszystkich zadań. Dla wszystkich przypadków w algorytmie GPP uzyskano tańsze rozwiązania.

Tab. 3. Wyniki syntezy
Tab. 3. Synthesis results

	Yen-Wolf	Ewa	**GP50**	**GP100**	**GP200**
Koszt	4488	4191	**4062**	**4044**	**3939**
Czas	2189	1959	**1933**	**1933**	**1994**

5. Wnioski

W pracy przedstawiono metodę kosyntezy systemów specyfikowanych za pomocą grafu zadań i implementowanych w technologii SOC. Metoda oparta jest o rozwojowe programowanie genetyczne. Według naszej wiedzy jest to pierwsze użycie algorytmu DGP do problemu kosyntezy. Podstawową różnicą w odniesieniu do innych podejść genetycznych jest to, że ewolucji nie podlegają rozwiązania, ale metoda konstruująca te rozwiązania. Okazało się, że metoda DGP jest również skuteczna dla problemu kosyntezy. Już we wstępnych eksperymentach uzyskano rozwiązania lepsze niż w dotychczas stosowanych metodach.

Dalsze prace będą miały na celu badanie innych metod reprezentacji problemu kosyntezy dla potrzeb DGP. Przede wszystkim planuje się opracowanie i ocenę bardziej zaawansowanych metod konstrukcji systemu, odpowiadających węzłom w drzewie genotypu. Zostaną przebadane również alternatywne wersje operatorów genetycznych i wpływ parametrów algorytmu na jakość uzyskanego rozwiązania.

6. Literatura

[1] T.-Y.Yen, W.Wolf, "Hardware-Software Co-synthesis of Distributed Embedded Systems", Springer, 1997.
[2] Yen T.-Y., Wolf W.H., Sensivity-Driven Co-Synthesis of Distributed Embedded Systems, Proc. of the Int. Symposium on System Synthesis, 1995, pp.4-9.
[3] S.Deniziak, "Cost-efficient synthesis of multiprocessor heterogeneous systems", Control and Cybernetics, Vol.33, No. 2, 2004, pp.341-355.
[4] Dave B.P., Lakshminarayana G., Jha N.K., "COSYN: Hardware-Software Co-Synthesis of Embedded Systems", Proc. of the Design Automation Conference, 1997, pp.703-708.
[5] Dick R.P., Jha N.K., "MOGAC: A Multiobjective Genetic Algorithm for the Co-Synthesis of Hardware-Software Embedded Systems", IEEE Trans. on CAD, Vol.17, No.10, 1998
[6] M.Purnaprajna, M.Reformat, W.Pedrycz,"Genetic Algorithms for hardware-software partitioning and optimal resource allocation", Journal of Systems Architecture, No.53, 2007, pp.339-354.
[7] G.W. Grewal, T.C. Wilson, "An enhanced genetic algorithm for solving the high-level synthesis problems of scheduling, allocation and binding", International Journal of Computational Intelligence and Applications 1 (2001), pp.91–110.
[8] J.R. Koza, "Genetic Programming: On the Programming of Computers by Means of Natural Selection", MIT Press, Cambridge, MA, 1992.
[9] J.R.Koza, R.Poli, "Genetic Programming", In Edmund Burke and Graham Kendal, editors. "Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques", Chapter 5, Springer, 2005.
[10] J. R.Koza, M. A.Keane, M. J.Streeter, W.Mydlowec, J.Yu, G.Lanza, "Genetic Programming IV: Routine Human-Competitive Machine Intelligence", Kluwer, 2003.

Artykuł recenzowany