Hardware/Software Partitioning of Coarse-Grained Reconfigurable System Using Evolutionary Ant Colony Optimization

Dawei Wang, Yafei Cao, Sikun Li

College of Computer Science, University of Defence Technology,
410073 Changsha, P.R.China
daweiwang@nudt.edu.cn, yfcao@nudt.edu.cn, lisikun@263.net.cn

Abstract. We present a hardware/software partitioning approach for coarse-grained reconfigurable SoC design using an evolutionary Ant Colony Optimization with Genetic Algorithm, namely eACOGA. The architecture of reconfigurable SoC mainly consists of micro-processor, dynamic reconfigurable process element array and memories. We build an automatic configuration and optimization framework for coarse-grained reconfigurable SoC design, the kernel of which is eACOGA algorithm. eACOGA uses genetic algorithm to evolve the parameters of ant colony optimization, and makes use of such advantages as positive feedback and efficient convergence of ant colony optimization to search for optimal partitioning solutions. It utilizes the advantages of both ant colony optimization and genetic algorithm, which avoids dropping into local optimization solutions. Experiments show eACOGA improves the quality and efficiency of coarse-grained reconfigurable SoC hardware/software partitioning.

1 Introduction

Due to its potential to greatly accelerate a wide variety of applications, reconfigurable computing has become a subject of a great deal of research [1] [2]. Coarse-grained reconfigurable architectures can achieve more speedup and power saving than fine-grained reconfigurable architectures, while cost more design efforts. With more and more coarse-grained reconfigurable architectures have been proposed [3-5], the new SoC architectures patterns like "RISC + reconfigurable arrays" bring new challenges on existing SoC design methodology and tools.

Hardware/software partitioning is the key problem of coarse-grained reconfigurable SoC design. Its main task is to map tasks to hardware/software architectures for the high performance of SoC System. Recently, most research efforts use heuristic algorithms for hardware/software partitioning. These algorithms use heuristic information to guide the search process for converging to the optimal solutions step by step, so that they have the capability of handling the delay/area/power consumption trade-off.

In this paper we describe a new hardware/software partitioning approach for coarse-grained reconfigurable SoC using ant colony optimization. We have developed a framework for the automatic configuration and optimization of hardware/software

Advances in Artificial Intelligence and Applications
Research in Computer Science 32, 2007, pp. 45–55

Received 13/06/07
Accepted 31/08/07
Final version 20/09/07

partitioning. A key component of this framework is an evolutionary ant colony optimization algorithm for hardware/software partitioning. Experimental results show that our algorithm is effective in finding close to optimal solutions.

The rest of the paper is organized into 6 sections. Section 2 overviews the previous work. Section 3 shortly introduces problem definition for coarse-grained reconfigurable SoC partitioning. In section 4 we present the ant colony optimization algorithm for coarse-grained reconfigurable SoC and integrate it into an automatic reconfigurable partitioning flow. In section 5, we apply our algorithm to some typical SoC applications and give the obtained results. Finally, in section 6 we present the conclusions.

2 Related Work

There are a great number of approaches to HW/SW co-design of SoC systems, which use different techniques for partitioning. A hybrid genetic algorithm is presented for constrained hardware-software partitioning [6]. Given a high-level program and an area constraint, it considers different granularities to discover the most interesting blocks to be implemented in ad hoc functional units. G.Wang uses ant colony optimization algorithm, in which intelligent ants can cooperatively search for the global best optimal solutions using both heuristic information and pheromone information [7].

There are many approaches to HW/SW partitioning of fine-grained reconfigurable SoC. A new automated compile-time partitioning and scheduling algorithm is proposed in [8], in which three meta-heuristic algorithms, namely genetic algorithm, simulated annealing, and tabu search, are compared. Simple heuristics are presented to partition the real-time applications at task level for scheduling between CPU and FPGA so that they meet the deadline constraints for a given total system power [9]. This fast partitioning mechanism would enable dynamic reconfiguration of systems for power and performance tuning.

However, there are not many research efforts on the partitioning of coarse-grained reconfigurable SoC systems. A hyper graph covering algorithm is proposed to translate high level C program into hyper graph and use the clustering and allocation algorithm for mapping high level application on coarse-grained reconfigurable architecture [10]. While our approach makes use of ACO (Ant Colony Optimization), which is a cooperative heuristic searching algorithm by the observation on the behavior of ants, and we evolve the parameters of ACO with GA (Genetic Algorithm) to avoid the stagnancy of search.

3 Hw/Sw Partitioning of Coarse-grained Reconfigurable SOC

3.1 Problem Formulation for Reconfigurable SoC Partitioning

Usually, we use TG (Task Graph) to describe the behaviors of SoC system, rAG (reconfigurable Architecture Graph) to describe the architectures of reconfigurable SoC

system, and the mapping from TG to rAG to describe hardware/software partitioning of reconfigurable SoC [11].

Definition 1 (TG) A task graph $TG = (T, E, C, P)$ consists of a set of nodes T, a set of directed edges $E \subseteq (T \times T)$, the configuration of nodes C, and a set of node ports P.

The nodes T represents the tasks of SoC system, and the edges E represents control/data flow of tasks' communication.

Definition 2 (rAG) A reconfigurable architecture graph $rAG = (C^{I}, C^{I/E}, C^{M}, RN)$ consists of microprocessor computation resources C^{I}, reconfigurable computation resources $C^{I/E}$, memories C^{M}, and route networks RN.

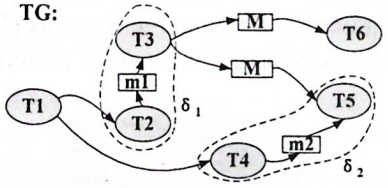
Definition 3 (*k* way partition) Given a set of modules $M = \{m_1, m_2, \ldots, m_n\}$, a k way partition problem is to find a set of clusters $P = \{p_1, p_2, \ldots, p_k\}$, which meets:

$$\begin{cases} p_i \subseteq M, & 1 \le i \le k \\ \bigcup_{i=1}^{k} p_i = M & \\ p_i \bigcap p_j = \Phi, & 1 \le i, j \le k, i \ne j \end{cases} \tag{1}$$

In the problem of reconfigurable SoC hardware/software partitioning, M presents TG, and P presents rAG. Generally, the performance objects of hardware/software partitioning are various due to the complexity of application field. In this paper, we assume the performance object is to minimize the weighted cost sum of running time and area of all the tasks. In this way, we define the objective function as follows:

Objective Cost Function $= \sum w_t * time(i) + w_a * area(i)$:

In above formulation, w_t presents weight for time, w_a presents weight for area, time(i) presents time cost of task i, and area(i) presents area cost of task i. A typical partitioning for coarse-grained reconfigurable SoC is shown in figure 1.



Mapping:

V	C	V	C
T1	Processor	T5	PEs
T2	PEs	T6	Processor
T3	PEs	m1	LM
T4	PEs	m2	LM

$\Delta = \{\delta_1, \delta_2\}$
$\Phi' = \{(T2, \delta_1), (T3, \delta_1), (T4, \delta_2), (T5, \delta_2)\}$
$\Psi' = \{(\delta_1, PEs), (\delta_2, PEs)\}$

Fig. 1. Problem formulation for reconfigurable SoC partition

We use DAG (Directed Acyclic Graph) based bi-coloring model to describe the problem of hardware/software partitioning. Figure 2 shows that tasks mapping to microprocessor are rendered by color c1, and tasks mapping to PEs are rendered by color c2. The work of ants is to find the best coloring of all tasks that minimizing the cost of partitioning. eACOGA algorithm uses dynamic combination of ant colony optimization and genetic algorithm to optimize the search for the best coloring.

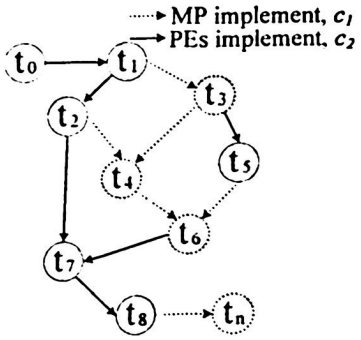


Fig. 2. Direct acyclic graph based bi-coloring model. The DAG consists of tasks nodes and their edges. All the edges are colored by two color, c_1 and c_2, which indicates mapping the tasks onto microprocessor (MP) or PEs.

3.2 Target Architecture for Reconfigurable Partitioning

We deal with reconfigurable SoC hardware/software partition by two main steps: mapping tasks to microprocessor or reconfigurable array, and partial partitioning for the tasks mapping on reconfigurable arrays.

To meet the needs of various applications, many reconfigurable architectures have been proposed, which have different details of design implementation, such as the topology of reconfigurable array, communication protocol, and the strategy of reconfiguration. To facilitate the research of reconfigurable partition problem, we abstract reconfigurable architecture and take it as the target architecture for partitioning. We design a RAAM (Reconfigurable Architecture Abstract Model) for the hardware/software partitioning of coarse-grained reconfigurable Systems [12]. RAAM has many configurable parameters, so it can support most kinds of reconfigurable architectures.

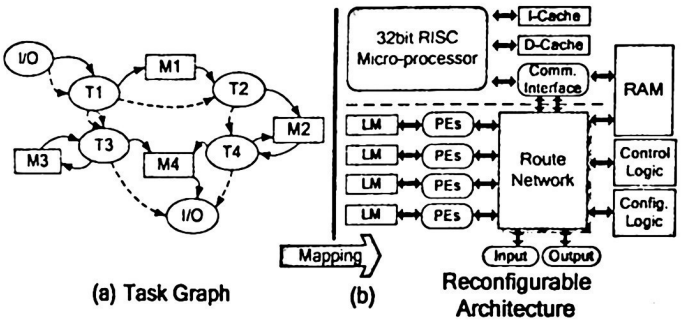


Fig. 3. Reconfigurable architecture abstract model for hardware/software partitioning. (a) task graph describes the function and behavior of reconfigurable system; (b) reconfigurable architecture consists mainly of microprocessor and coarse-grained reconfigurable PEs array.

Figure 3(a) shows the task graph that describes the function and behavior of reconfigurable systems. There are three types of task nodes: computing task, storage seg-

ment and communication task. The task nodes are connected by dependencies or channels. Figure 3(b) shows the reconfigurable architecture that consists of microprocessor and reconfigurable array. The microprocessor uses 32-bit RISC instruction set and some enhanced instructions for special propose processing. Reconfigurable array consists of PEs (Process Elements), route network, LM (Local Memory), configurable logic and control logic, etc. PEs array based reconfigurable architectures are popular because of their advantage of high performance, low power and good flexibility for embedded system applications.

4 Ant Colony Optimization For Coarse-Grained Reconfigurable SoC Partitioning

4.1 eACOGA Algorithm

ACO algorithm can find better solutions of partitioning more effectively [13]. But the strategy of random selection in constructing solutions lead to slow convergence speed. Furthermore, the principle of positive feedback can not only strengthen the solutions with better performance, but also bring on the stagnancy of search. The causation is that the main configurable parameters of the algorithm, such as α, β, ρ, Q, are set to fixed value when initializing, and it has no adaptability to various applications.

Based on existing basic ACO algorithm ([7]), we present an eACOGA approach of hardware/software partitioning for reconfigurable SoC. GA can evolve the configuration parameters of ACO algorithm by cross operation and variation operation ([9]). So eACOGA can evolve and optimize itself to search global optimal solutions.

We define the rules of eACOGA as follows:

Objective Function and Fitness Function: We define objective function as S_{best} = arg min C_p, fitness function as Fitness (p) = $1/C_p$. Where, C_p figures the cost of partition p.

Configure RAAM: According to the need of specific applications, design experts configure the task graph and reconfigurable architecture. In this step, the parameters of the tasks and architectures should be decided.

Strategy of Render to DAG: For any nodes except t_n, ants try to render the color of t_j, the subsequence of t_i. Ants achieve the work according to the global heuristic information ($\tau_{ij}(k)$) of edge e_{ij} and the local heuristic information ($\eta_j(k)$) of node t_j. The ants on node t_i will render the color of node t_j as c_k at the probability of:

$$p_{ij}(k) = \frac{\tau_{ij}(k)^{\alpha} \eta_j(k)^{\beta}}{\sum_{l=1,2} \tau_{ij}(l)^{\alpha} \eta_j(l)^{\beta}} \tag{2}$$

Where, $\tau_{ij}(k)$ is the pheromone on edge e_{ij}, α and β is the factor of them and $\eta_j(k)$ is defined as follows:

$$\eta_j(k) = 1/((w_t * time_t(k)) + (w_a * area_j(k))) \tag{3}$$

Use Genetic Algorithm to Evolve Parameters: We use genetic algorithm to evolve the parameters of ant colony optimization, such as α, β, ρ, Q. First, Configure the probability factor of cross and variation operation according to the size of population and the generation of evolution. Then by taking α, β, ρ, Q as the variable of fitness function, the best optimal partition cost as fitness function and the course of ACO as the individual, we optimize α, β, ρ, Q repeatedly until finding the best optimal solutions.

Pheromone Setting and Refreshing: We adopt MMAS (Max-Min Ant System) introduced by Thomas Stuzle ([14]), the refreshing equation of pheromone is:

$$\tau_{ij}(k)=\begin{cases}(1-\rho)*\tau_{ij}(k)+\Delta\tau_{ij}(k)_{best}\\ \tau_{ij}(k)_{max}, \text{if } \tau_{ij}(k)>\tau_{ij}(k)_{max}\\ \tau_{ij}(k)_{min}, \text{if } \tau_{ij}(k)<\tau_{ij}(k)_{min}\end{cases} \quad (4)$$

Where, ρ is the evaporation ratio of pheromone, $\tau_{ij}(k)_{max}$ ($\tau_{ij}(k)_{min}$) is maximum (minimum) strength of c_k pheromone on edge e_{ij}, and $\Delta\tau_{ij}(k)_{best}$ is increment of c_k pheromone on edge e_{ij} done by the "best ant" in current ant system algorithm iteration. According to Ant-Cycle Model, $\Delta\tau_{ij}(k)_{best}$ is defined as:

$$\Delta\tau_{ij}(k)_{best}=\begin{cases}Q/C_{p_{best}}, \text{in } p_{best}\ e_{ij} \textit{ is rendered by } c_k\\ 0, \qquad \text{otherwise}\end{cases} \quad (5)$$

```
//Input: the configuration of RAAM and DAGs .
//Outputs: the best optimal solutions for partition
//Q, α, β, ρ is the main parameters of ACO
1  Main(){
2    Generation = 0;
3    Initialize();
4    Evaluate();
5    Keep_the_Best();
6    foreach generation
7    {
8      select();
9      crossover();
10     mutate();
11     report();
12     Evaluate();
13     elitist();
14   }
15   Output the best fitness values.
16 }

17 Evaluate(){
18 for (mem = 0; mem < POPSIZE; mem++)
19 {
20   for (i = 0; i < NVARS; i++)
21     x[i+1] = population[mem].gene[i];
22   Q = x[1]; α = x[2]; β = x[4]; ρ = x[4];
23   ACO partition = new ACO;
24   partition GetAnt();
25   partition StartSearch();
26   population[mem].fitness = CostFunction_to_Fitness;
27 }

28 ACO::GetAnt(){
30   Randomly put ant into DAGs;
31   for (i = 0; i < nAntCount; i++)
32   {
33     task = rnd( nTaskCount);
34     ants[i].AddTaskIntoTabu(task);
35   }
36 }

37 ACO::StartSearch(){
38   foreach ant
39   {
40     Select_NextNode_Accordingto_heuristic_Information();
41     Moveto_NextNode();
42     Update_Tabu_Table();
43     find out the best solution of the step and put it into temp;
44   }
45   Update_Trail();
46   Find_theBest_Solutions_Of_Partition();
47 }
```

Fig. 4. Pseudo-code for eACOGA algorithm

The pseudo-code for eACOGA algorithm is shown in Figure 4. First, the configurations of RAAM and DAGs are input, and after the execution of eACOGA algorithm the best optimal solutions for partition are output. In eACOGA, ACO is built as a class, which has two main functions: GetAnt() and StartSearch(), as shown in Line 23-25. GA randomly encodes the variables of α, β, ρ, Q, as shown in Line 18-22. GA evolves the variables continuously by the operation of select, crossover and mutate, as shown in Line 6-14. In ACO, we put the ants randomly into DAGs and begin the search for best optimal solutions, as shown in the function on Line 28-36 and Line 37-47. After evaluating the fitness function values we output the best optimal solutions, as shown on Line 12 and Line 15.

4.2 Automatic Partitioning Flow

We have designed an automatic partitioning flow for mapping applications on reconfigurable SoC, as shown in Figure 5.

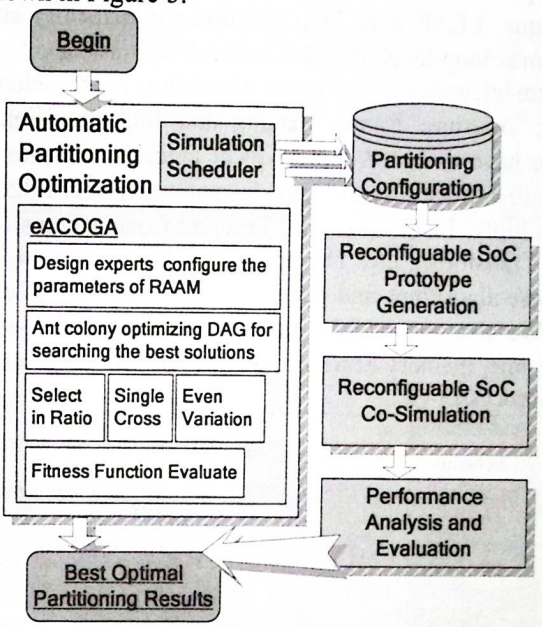


Fig. 5. Automatic partitioning flow of reconfigurable SoC. It integrates eACOGA for partitioning configuration.

First, design experts configure the tasks and reconfigurable architecture of RAAM. Then, application specific reconfigurable SoC prototype is generated according to existing reconfigurable architecture templates. Finally, we run reconfigurable SoC transaction level co-simulation and output the best optimal partitioning solutions.

The automatic partitioning flow has two main advantages:

(1).For each individual of genetic population in eACOGA, the flow of partition and reconfigurable SoC co-simulation can run automatically. When some constraints cannot be met, experts can request to stop the simulation.

(2).Transaction level simulation in SystemC can describe various behaviors of reconfigurable SoC with faster speed and nicer accuracy. Architecture template enhances reuse of existing SoC design and achieves exploration speedup well.

5 Experimental Results

5.1 Target Architecture and Benchmarks

We use eACOGA algorithm for the partitioning of a reconfigurable SoC system, which consists mainly of 32-bit RISC microprocessor called Estar and reconfigurable arrays called LEAP, as shown in Figure 7 [15]. Both Estar.and LEAP are developed by our research group. We use Estar for common computing. It has 8KB instruction cache and 8KB data cache, 266M Hz and 220mW of CPU core. Besides, we use LEAP for reconfigurable computing. It can accelerate applications through loop self-pipelining technique. LEAP steps loop iteration automatically and has the ability to exploit parallelism at loop-level, instruction-level, and task-level.

The SoC system integrates some typical algorithms in the field of Software Defined Radio, Synthetic Aperture Radar imaging and high-precision digital image encode/decode. We have designed some typical algorithms running on reconfigurable systems, such as FFT (Fast Fourier Transformation), Sobel Edge Detection, Median Filter, Matrix Multiply, FDCT (Forward Discrete Cosine Transform), IDCT (Inverse Discrete Cosine Transform), etc. The application program analysis and pick up tool can recognize these algorithms and generate them as the nodes of DAG, such as T1, T2 in Figure 6. We also generate the attributes of these nodes, such as configuration time, computing time, memory accessed, number of PEs used, and execution time on embedded microprocessor etc.

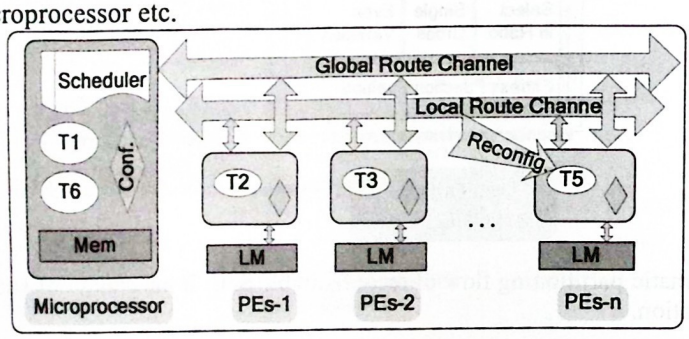


Fig. 6. A typical partitioning result of reconfigurable SoC. The task of T1 and T6 are mapped onto microprocessor, and T2, T3, and T5 are mapped onto PEs.

We have tested the performance of these typical algorithms on Estar and LEAP and translate execution time into time cost and resources used into area cost. In eACOGA algorithm, we set $w_t = 1$, $w_a = 10$, ACO iteration counts = 100, GA population size = 5, GA max generation = 50, GA cross probability factor = 0.8, and GA variation probability factor = 0.15.

Table 1. The performance of typical algorithms. Execution time of them on Estar and LEAP are shown. cPE and mPE discribes the PE for computation and the PE for memory.

Typical Algorithms	Execution Time		Resources (cPE, mPE)
	Estar (Mcycle)	*LEAP (Kcycle)*	
512 point FFT	32.320	6.721	10c4m
1024 point FFT	72.353	12.802	10c4m
Edge Detection (320x240)	39.720	216.958	16c7m
Edge Detection (480x360)	87.898	474.205	16c7m
Median Filter (320x240)	1580.368	220.010	30c7m
Median Filter (480x360)	3590.500	478.792	30c7m
Matrix Multiply (64x64)	54.315	79.141	30c10m
Matrix Multiply (128x128)	2522.258	318.901	30c10m
FDCT	2433.389	2838.905	30c10m
IDCT	2437.417	2839.044	30c10m

5.2 Result Analysis

We generate some test DAGs, the nodes of which consist of typical algorithms in Table 1. We set the parameters region of α, β, ρ, Q respectively as [5, 1], [5, 1], [0.8, 0.2], [100, 40].

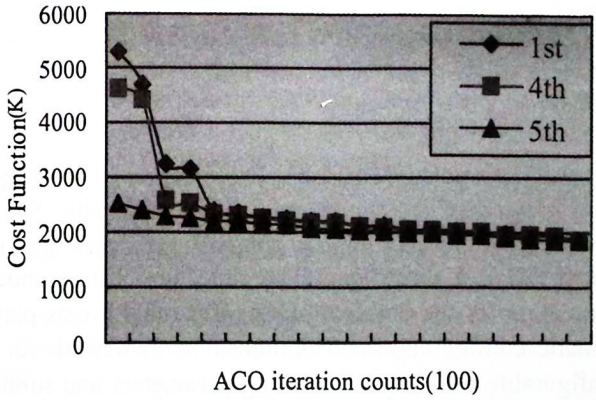


Fig. 7. The evolution curve of eACOGA for partition. The convergence speed of 5th ACO is much faster than that of 1st and 4th.

We achieve the object of hardware/software partitioning using eACOGA, as shown in Figure 7. The total number of tasks is 10, and ant counts = 6. The global best optimal solutions are 1862843, in which Q = 70.540, α = 4.836, β = 3.580, ρ = 0.738. The average value of 1st is 1918421, by evolution the average value of 4th is 1891026, and that of 5th is 1890922.

To compare the quality of eACOGA with that of other researches, we select ACO algorithm in literature [8]. We set ACO parameters as: Q = 1000, α = 1, β = 1 and ρ = 0.8. Figure 8 show that eACOGA has better ability than ACO in searching for the global best optimal solutions. The algorithm of ACO gets into local best optimal solutions (1905396). However, eACOGA can find the global best optimal solutions (1862843) effectively for the advantage of self-adaptive optimization. Besides, another algorithm we have researched, called initACO (ACO with init pheromone), has the performance between them [16].

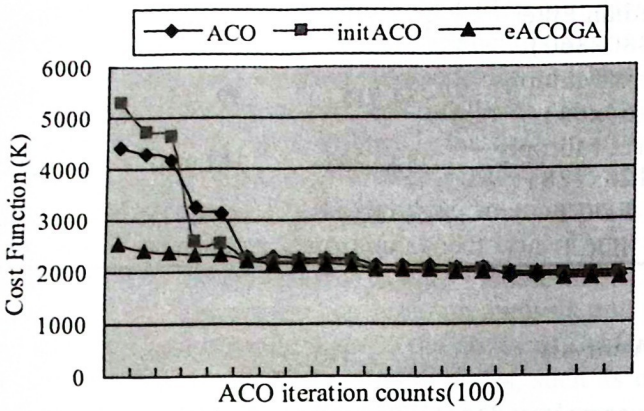


Fig. 8. Comparing eACOGA with ACO and initACO. It shows eACOGA can find optimal solutions with fewer iteration counts.

6 Conclusions

ACO has been well proved to be suitable for fine-grained reconfigurable SoC partitioning, but not mentioned for coarse-grained reconfigurable SoC. This paper proposes a hardware/software partitioning approach of coarse-grained reconfigurable system using eACOGA. A reconfigurable architecture abstract model is proposed as the target architecture for the problem of reconfigurable hw/sw partitioning. We also build an automatic configuration and optimization framework for mapping applications on reconfigurable SoC. By configuring parameters and running simulation we can get global best optimal partitioning solutions. It can save the time of waiting for manual simulation.

The algorithm of eACOGA can evolve the main control parameters (α, β, ρ, Q) of ACO, so that it can find global best optimal solutions efficiently and rapidly. It overcomes the disadvantage of ACO that be inclined to get into local best optimized solu-

tions. Furthermore, the method combining ACO and GA that yields even better results than using each of the algorithms individually.

References

1. Compton Katherine,Hauck Scott. Reconfigurable Computing: A Survey of Systems and Software. *ACM Computing Surveys*, 2002, 34(2):171-210.
2. R. Hartenstein. "A Decade of Reconfigurable Computing: a. Visionary Retrospective," In *Int'l Conf. on Design, Automa-. tion and Test in Europe (DATE'01)*, Munich, Germany,. March 12-15, 2001, 642-649.
3. Volker Baumgarten, G. Ehlers, F. May, Armin Nückel, Martin Vorbach, and Markus Weinhardt: PACT XPP - A Self-Reconfigurable Data Processing Architecture. *The Journal of Supercomputing*, 2003, 26(2): 167-184.
4. H Singh, M Lee, G Lu, F J Kurdahi, N Bagherzadeh, E Filho, R Maestre. Morhposys: case study of a reconfigurable computing system targeting multimedia applications. *Proc. Design Automation Conference (DAC'00)*, Los Angeles, California, 2000, 573-578.
5. J.R. Hauser, J. Wawrzynek. Garp: a MIPS processor with a reconfigurable coprocessor. *5th IEEE Symposium on FPGA-Based Custom Computing Machines (FCCM'97)*, 1997, 12-21.
6. Pierre-Andre Mudry, Guillaume Zufferey, Gianluca Tempesti: A Hybrid Genetic Algorithm for Constrained Hardware-Software Partitioning. *Proceedings of the IEEE Workshop on Design and Diagnostics of Electronic. Circuits and Systems*, 2006, 3-8.
7. Gang Wang, Wenrui Gong and Ryan Kastner, "System Level Partitioning for Programmable Platforms Using the Ant Colony Optimization", In *13th International Workshop on Logic and Synthesis (IWLS'04)*, 2004.
8. Theerayod Wiangtong. *Hardware/Software Partitioning and Scheduling for Reconfigurable Systems*. Ph.D. thesis, University of London, UK, 2004.
9. Pramod K., S. Acharya, R. N. Mahapatra, "A Partitioning Algorithm for Power constrained Reconfigurable Real-Time Systems", *Journal of Microprocessor and Microsystems*, 2005.
10. Yuanqing Guo, Gerard J. M. Smit, Hajo Broersma, Paul M. Heysters: A graph covering algorithm for a coarse grain reconfigurable system. *LCTES 2003*, 199-208.
11. Michael H. Eisenring. *Communication channel synthesis for heterogeneous embedded systems*. Ph.D. thesis, SWISS Federal institute of Technology ZURICH, No.14640, 2002.
12. M. Kaul, V. Srinivasan, etal, "Partitioning and Synthesis for Run-Time Reconfigurable Computers Using the SPARCS System", In *Proceedings of the 1998 Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD'98)*, NASA Goddard Space Flight Center, Sept 15-16, 1998.
13. M Dorigo, V Maniezzo, and A Colorni. Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Trans on Systems, Man and Cybernetics, Part-B*. 1996, 26(1), 29~41.
14. T Stutzle, H H Hoos, et al. MAX-MIN Ant System. *Future Generation Computer System*. 2000, 16(8): 889~91.
15. Yong Dou, Xicheng Lu, "LEAP: A Data Driven Loop Engine on Array Processor", *The 4th Int'l Conf on Parallel and Distributed Computing, Applications and Technologies (PDCAT'03)*, 2003.
16. Xiong Zhihui , Li Sikun , Chen Jihua, "Hardware/Software Partitioning Based on Ant Optimization with Initial Pheromone", Journal of Computer Research and Development, 2005, 42(12):2176-2183.