Multi-objective hardware/software partitioning technique for dynamic and partial reconfigurable system-on-chip using genetic algorithm

N.Janakiraman[#1], P.Nirmal Kumar[#2]

[#1]Associate Professor, Department of ECE, K.L.N. College of Engineering, Madurai, INDIA

[#2]Associate Professor, Department of ECE, Anna University – Guindy campus, Chennai, INDIA

[#1]janakiramanforu@yahoo.com, [#2]nirmal@annauniv.edu

Abstract – Hardware/software partitioning is a common method used to reduce the design complexity of a reconfigurable system. Also, it is a major critical issue in hardware/software co-design flow and high influence on the system performance. This paper presents a novel method to solve the hardware/software partitioning problems in dynamic partial reconfiguration of system-on-chip (SoC) and observes the common traits of the superior contributions using genetic algorithm (GA). This method is stochastic in nature and has been successfully applied to solve many non-trivial polynomial hard problems. It is based on the appropriate formulation of a general system model, being therefore independent of either the particular co-design problem or the specific partitioning procedure. These algorithms can perform decomposition and scheduling of the target application among available computational resources at runtime. The former have been entirely proposed by the authors in previous works, while the later have been properly extended to deal with system-level issues. The performance of all approaches is compared using benchmark data provided by MCNC standard cell placement benchmark netlists. This paper has shown the solution methodology in the basis of quality and convergence rate. Consequently, it is extremely important to choose the most suitable technique for the particular co-design problem that is being confronted.

Keyword-Hardware/software partitioning, Genetic algorithm, Dynamic partial reconfiguration, System-on-chip

I. Introduction

Hardware/software partitioning is a method of dividing a complex heterogeneous system into hardware co-processor functions and its compatible software programs. It is a prominent practice that can realize results greater than the software-only or hardware-only solutions in SoC design. This technique can improve the system performance [1] and reduce the total energy consumption [2]. The proposed partial dynamic reconfiguration method does not depend on any tool. It uses a set of algorithms to detect crucial code regions, compilation/synthesize of hardware/software modules, and updating of communication logic. Hence, it could tune up the system to give full efficiency without disruption of other SoC-related operations. Here, the GA is used for optimization process. This is essential in system-level design, since decision-making process affects the total performance of system. This paper presents a novel system partitioning technique with in-depth analysis. The paper is organized as follows. Section 2 briefs about the previous works in this field. Section 3 presents the proposed system model for partitioning problem. Section 4 gives the results and its analysis. Section 5 concludes the paper and discusses about the future work. Last section provides the list of references.

II. Related Works

When compared to dynamic partitioning using standard software, the run-time (or) partial dynamic reconfigurable systems had attained superior performance with manually specified predetermined hardware regions. Multiple choices of preplanned reconfigurations were rapidly executed in a run-time reconfigurable system using PipeRench architecture [3] and dynamically programmable gate arrays (DPGA) [4]. The binary-level partitioning technique [5] was provided a good solution compared to source-level partitioning methods due to the functionality of any high-level language and software compiler. Since the satisfaction of performance was not considered for the cost function of this system, it may be failed to find out local minima. A mapping technique for nodes and hardware/software components was developed in [6] called GCLP algorithm. The hardware cost was minimized by the incorporation of hill-climbing heuristic algorithm with the hardware/software partitioning algorithm [7].

III.System Model For Partitioning

The problem resolution requires the system model definition to represent the important issues in the hardware/software co-design for a specific problem [8]. The system partitioning problem model is represented by the task graph (TG) flow diagram. TG is a model of directed and acyclic graph (DAG) flow with weight vectors. Formally, it is defined as $G=(V,E)$, where '*V*' represents the nodes and '*E*' represents the edges. The flow direction is represented by each edge. Due to reducing the complexity of TG, it can be modified as one starting node and one ending node. Figure1 represents the overview of the partitioning procedure. Design constraints and design specifications are given as the input to the partitioning process as a high-level specification language. The nodes can act as giant pieces of information like tasks and processes of coarse granularity or tiny types like instructions and operations of fine granularity approach.

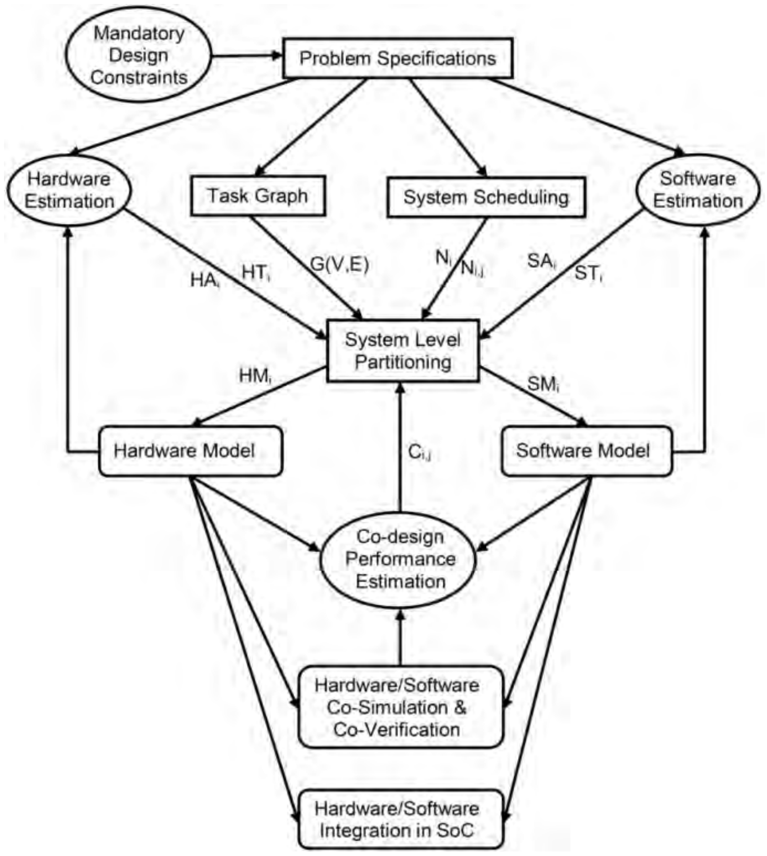

Fig.1. System Model for Partitioning

After the system space estimation, every node is tagged with some attributes. Giant pieces of data for a node $(V_{i,j})$ are represented by 5 attributes as follows:

(1) Hardware area $(HA_{i,j})$

(2) Hardware implementation time $(HT_{i,j})$

(3) Software memory size $(SS_{i,j})$

(4) Software execution time $(ST_{i,j})$

(5) The average execution time in numbers $(N_{i,j})$

Shortly,

Hardware module $(\mathrm{HM_{i,j}})=(\mathrm{HA_{i,j}})+(\mathrm{HT_{i,j}})+(\mathrm{N_{i,j}})$

Software module $(\mathrm{SM_{i,j}})=(\mathrm{SS_{i,j}})+(\mathrm{ST_{i,j}})+(\mathrm{N_{i,j}})$

Communication values $(C_{i,j})$ of every node are represented by three components as follows.

(1) Transfer time $\left(TT_{i,j}\right)$

(2) Synchronization time $\left(SynT_{i,j}\right)$

(3) The average communication time in numbers $\left(M_{i,j}\right)$

Shortly,

Communication value of node $\left(\mathrm{C_{i,j}}\right)=\left(\mathrm{TT_{i,j}}\right)+\left(\mathrm{SynT_{i,j}}\right)+\left(\mathrm{M_{i,j}}\right)$

$$\mathrm{C_{i,j}}=\frac{\left(\mathrm{N_i}*\Delta\mathrm{TT_i}\right)+\left(\mathrm{N_j}*\Delta\mathrm{TT_j}\right)+\left(\mathrm{SynT_{i,j}}\right)}{\left(\mathrm{HT_i}\right)+\left(\mathrm{HT_j}\right)};$$

where $(\Delta\mathrm{TT_i})=\left(\mathrm{ST_i}\right)-\left(\mathrm{HT_i}\right)$ and $(\Delta\mathrm{TT_j})=\left(\mathrm{ST_j}\right)-\left(\mathrm{HT_j}\right)$

Efficiency of the hardware/software system partitioning process is based on the target architecture and its mapping technique. Hence, this work considers the 'Dynamically Reconfigurable Architecture for Mobile Systems' (DReAM) as target architecture. Execution of hardware and software processes should be concurrently in the standard processor and the application-specific co-processor. This partitioning process concludes the assignment of modules to implement the hardware and software stages, implementation schedule (timing), and the communication interface between software and hardware modules. In general, this partitioning solution can be validated by the measurement of eminent attributes like performance and cost parameters. Hence, this paper used as three quality attributes related to design elements as follows:

(1) The estimated hardware area is A_E, and the maximum available area is A.

(2) The estimated design latency is T_E, and the maximum allowed latency is T.

(3) The estimated software (or) memory space is M_E, and the maximum available space is M.

Static-list scheduling method is used for the scheduling process [9]. It is a subtype of resource-constrained scheduling algorithm. This scheduler considers the timing estimation of every vertex and its interconnections. This scheduler unit provides the design latency (T_E) and the cost of communication for hardware–software co-design. Based on the hardware and software implementations, another four parameters are considered for co-design realization.

When the entire system is implemented in hardware,

(1) The minimum design latency is *MinT*.

(2) The maximum hardware area is *MaxA*.

When the entire system is implemented in software,

(1) The maximum design latency is *MaxT*.

(2) The maximum memory space is *MaxM*.

These parameters are used to create the bounding constraints for the design space.

$0 \le A \le MaxA$; $0 \le M \le MaxM$; $MinT \le T \le MaxT$.

A. *System Operations*

The design specifications are given in the format of ISPD98 benchmark suite [10] circuit netlist. This partitioning process has three stages.

In first stage, the processing of design specifications is divided into three subtasks. The first subtask is the separation of hardware $(HA_i \,\&\, HT_i)$ and software $(SS_i \,\&\, ST_i)$ estimations from the design specifications. The second subtask is to translate the design specifications into a hypergraph-based control data flow graph (CDFG) representation $G=\left(V,E\right)$. The third subtask is scheduling $(N_i \,\&\, N_{i,j})$ of each operations in the CDFG with satisfaction of the design constraints and the priority of operations.

In second stage, the outputs of these three tasks are given into the system-level partitioning module through the registers. It has three functionalities. The operational-level analysis is the first process, used to classify the tasks whether it is suitable for hardware realization or software execution. Next, the allocation process is used to allocate the required supporting entities like functional units, interconnections, and storage elements for the scheduled hardware and software systems. This allocation is based on the speed constraint (i.e., parallel processing) and the area constraint (i.e., dynamic partial reconfiguration). Finally, an absolute data path is generated by integrating components in the basis of hardware and software partitions. Then, the partitioning data are given to the specific hardware (HM_i) and software (SM_i) models.

In third stage, the hardware and software models are executed separately and the outcomes are compared with their estimated values (i.e., first stage). If any controversy arises, the feedbacks are given to the second-stage process. This looping process is continued till the satisfaction of all criterions.

Next, the performance $(C_{i,j})$ of hardware–software co-design is estimated and compared with target performance metrics. If any misalignment arises, the feedback is indicated to the system-level partitioning stage. Then, the entire second and third stages are recompiled, till the achievement of target performance measures. Finally, the hardware/software co-simulation and co-verification is performed, and then, the SoC is realized.

B. Hardware/Software Estimation

The CDFG file is given to the input of both hardware and software estimations with the settings of target technology files and processor specifications. The hardware execution is a parallel process since the specifications are modeled in VHDL library. The software execution is a sequential process since the specifications are modeled in C code. The GA technique is used to optimize these parallel and sequential processes. Hardware estimation is based on the high-level synthesizable components, to share the control and data path between hardware and software processes. GA is used to optimize this resource sharing process [11]. The quality measures are closely associated with performance metrics like execution, implementation, transfer, and synchronization times commonly called reaction time. This reaction time is associated with each node in each execution of local DFG. For convenient, the CDFG is split into several small DFGs called local DFGs.

The response times for

Routine statements, $\mathrm{T_{RS} = T_{DFG}}$

Conditional statements, $\mathrm{T_{CS}} = \sum_n \mathrm{P_n T_{DFGn}}$;

n – Number of iterations

$\mathrm{P_n}$ – Probabilities of iterations of outcomes.

Looping statements, $\mathrm{T_{LS} = nT_{DFG}}$;

$$\mathrm{T_{CDFG} = F(T_{DFG1}, F_{DFG1}, \ldots, T_{DFGi}, F_{DFGi}) + F(T_{DFG1}, F_{DFG1}, \ldots, T_{DFGj}, F_{DFGj})}$$

$$\mathrm{MinT} = \alpha\left[\mathrm{Max}\left(\mathrm{A*C}_{\mathrm{i,j}}\right) + \sum_{\mathrm{i}} \mathrm{T_i N_{i,j}}\right]$$

$\mathrm{T_i}$ – Time delay for each node

α – Co-estimation factor

$$\mathrm{MaxT} = \beta\,\mathrm{MinT} + \sum_{\mathrm{i}} \mathrm{T_i}\left[\mathrm{N_i} \sum_{\mathrm{j=1}}^{\mathrm{R_i}} {}_{\mathrm{i,j}}\right]$$

$\mathrm{R_i}$ – Required components of each node ‘i’

β – Constant, since MaxT is a higher-order term

$\mathrm{F_i}$ – Number of fixed components for each node ‘i’

$$\mathrm{T_{CDFG}} = \beta\,\mathrm{MinT} + \sum_{\mathrm{i}} \mathrm{N_i}\left[\frac{\mathrm{T_i}}{\mathrm{F_i}} \sum_{\mathrm{j=F_i+1}}^{\mathrm{R_i}} {}_{\mathrm{i,j}}\right]$$

C. Register Estimation:- [12]

Many input multiplexers = $(\mathrm{i^*MUXs})$

State machines based control logic is used to control lines, $\log_2 \mathrm{i}$

ROM size, $\left(\mathrm{STA}*\left[(1+\log_2 \mathrm{i})\left(\mathrm{REG} + \sum_{\mathrm{i}} \mathrm{F_i}\right) + \log_2 \mathrm{S}\right]\right)$ bits

STA – Number of states & REG – Number of registers.

Software estimation is based on the calculation of memory space occupied by instruction set and user-defined data types and data structures. The average queuing time for each memory access can be modeled as

T_q, and the number of access is represented by N_{mem}. This calculation is necessary to estimate $\left(TT_{i,j}\right)$ and $\left(SynT_{i,j}\right)$.

Hardware estimation $\left(T_{HM}\right) = \alpha\left(T_{(CDFG,HM)}\right) + T_q\left(N_{mem,HM}\right)$

Software estimation $\left(T_{SM}\right) = \left(T_{(CDFG,SM)}\right) + T_q\left(N_{(mem,SM)}\right)$

Co-estimation $\left(T_{HM/SM}\right) = \sigma\,\varphi\left(T_q\right) + \left(\frac{N_{mem}}{T_q}\right)$; where σ and φ are complex structures.

IV. Genetic Algorithm

Genetic algorithms (GAs) are evolutionary techniques based on the Charles Darwin's survival of the fitness. These are transformative computational techniques working in the concept of evolution of natural organs. In all organic techniques the fundamental data are taken in the form of sequences or chromosomes. Like that, genetic methods are also has chromosomes or strings, which is a collection of several data or change alternatives for each individual or gene. The quantity and quality of data are decided by population size of the GA. This is a key factor to find the several alternate solutions to a particular problem and that evolution takes more number of generations. In general, GA accepts problem in the format of chromosomes. Hence the circuit netlist or hypergraph format problems are converted in the form of chromosome.

Population Set: This paper deals the circuit in the format of hypergraph. Each component (node or vertex) and inter-connection (net or hyper-edge) is encoded in the format of 32-bit binary digits called chromosomes or individuals. These chromosomes are arranged in the adjacency matrix format which represents the corresponding node and net positions using spanning tree algorithm [13]. This is a randomly generated initial population set or matting pool with the user specified population size. The cost (number of cuts) of each individual (partition) is calculated and stored in the registers.

Fitness Evaluation: Based on the cost value, the fitness function is evaluated for each individual (i.e. Fitness = Total Number of Nets – Cut Size). This fitness evaluation influences the individual's selection for next generation. Here the rank based selection process is used to select individuals for creating a mating pool.

Crossover: The crossover operators are try to combine the attributes of highly fit individuals in the mating pool, to create offspring which is expected to be better than their parents fitness value. Here the uniform crossover operator is used for mating process with varying range of crossover probability value (0.91-0.98). If the newly generated individuals or children have high fitness value than original population, then the lowest fit individuals are replaced by new individuals, otherwise, the original population doesn't altered (elitism).

Pseudo-code for Crossover

```
begin
    K ← Individual;
    K ← 0;
    while (K < Population Size) do
    $C_K$ ← Random number between 0 and 1;
        If ( $C_K$ < 0.91) then
            Select individual K as one parent for crossover
        endif
    K ← K + 1;
    endwhile
end
```

Mutation: The mutation operator is randomly performed on the bits with very small probability value (0.05), due to the prevention of sudden changes in population set. Then this modified population set is evaluated for its fitness function.

The stopping criterion for this entire process is set to 100 runs.

V. ANALYSES OF RESULTS

All the hardware/software partitioning algorithms have been experimented in a set of benchmark suites provided by ISPD'98, whose characterization is shown in Table1. Size and values of the system graph should bound within the design space. All these examples are illustrated in the form of directed and acyclic graphs to specify the certain coarse–grain tasks. Every example has been tested in different constraints, but it always within the specified boundary conditions. The results are summarized in Table2. These results will be analyzed from both qualitative and quantitative perspectives. The qualitative aspects will be mainly represented by the resulting cost of the solutions obtained from each method, under different constraints. The quantitative issues will be shown by means of the computation time resulting from each technique.

TABLE I
DESIGN CHARACTERISTICS FOR ISPD'98 BENCHMARK SUITE

Circuit	# Cells	# Pads	#Modules	# Nets	# Pins
ibm01	12506	246	12752	14111	50566
ibm02	19342	259	19601	19584	81199
ibm03	22853	283	23136	27401	93573
ibm04	27220	287	27507	31970	105859
Ibm05	28146	1201	29347	28446	126308

TABLE III
RESULTS ACQUIRED WITH THE ISPD'98 EXAMPLES

Example	Constraints			Genetic Algorithm			
	Area (CLBs)	Time (ns)	Memory (Bytes)	AE	TE	ME	Fitness
ibm01	121800	10200	52670	118146	9384	46350	0.9233
	103080	8670	44770	101637	8020	41189	0.9437
ibm02	154700	12600	55980	140170	11230	49823	1.0000
	193375	15750	48980	172104	15435	51429	0.9733
ibm03	171200	14200	48090	154896	12040	38953	1.0000
	111280	9230	57708	103521	8769	54823	1.0000
ibm04	182200	15900	56460	173090	14469	62106	0.9866
	258724	19239	50814	234597	16546	46749	0.9600
ibm05	198300	16800	62210	180453	13776	58478	0.8900
	97167	12432	81495	92309	10940	84755	0.9566

VI. CONCLUSION AND FUTURE WORK

In this paper, the commonly used biologically inspired optimization algorithm, which addresses the hardware/software partitioning problem for SOC designs, is implemented using clustering approach as well as their performance is evaluated. This evaluation process does not have any constraints on the cluster size and the number of clusters. Hence, this evaluation approach is quiet suitable to be used in reducing the design complexity of systems. This paper had shown how this problem can be solved by means of very different partitioning techniques at runtime of the system (dynamic partial reconfiguration). The problem resolution has been based on the definition of a common system model that allows the comparison of different procedures. These extensions have improved previous implementations, because they include some issues previously not considered. The constraints of these algorithms have been integrated into the cost function in a general and efficient way. This genetic algorithm-based dynamic partitioning technique has produced an average of 16.19 % accuracy in hardware/software partitioning compared to [14] and [15].

A future study could extend the system model to encompass other quality attributes, like power consumption, influence of communications, and the degree of parallelism. Also, the hybrid algorithms of these biologically inspired algorithms and their compilation are currently under study.

ACKNOWLEDGMENT

This work was supported in part by All India Council for Technical Education-Quality Improvement Programme scheme 2010. Access to research and computing facilities was provided by the Anna University and K.L.N. College of Engineering.

REFERENCES

[1] D.D. Gajski, F. Vahid, S. Narayan and J. Gong, "SpecSyn—an environment supporting the specify-explore-refine paradigm for Hardware/Software system design," IEEE Trans. VLSI Syst., vol. 6(1), pp. 84–100, 1998.

[2] J. Henkel, "A low power Hardware/Software partitioning approach for core-based embedded systems," in Proc. of the 36th ACM/IEEE Conf. on Design Automation, 1999, pp. 122–127.

[3] S.C. Goldstein, H. Schmit, M. Budiu, M. Moe and R.R. Taylor, "PipeRench—a reconfigurable architecture and compiler," IEEE Computer, vol. 33, pp. 70–77, 2000.

[4] A. DeHon, "DPGA-coupled microprocessors-commodity ICs for the early 21st century," in Proc. of FCCM, 1994.

[5] G. Stitt and F. Vahid, "Hardware/Software partitioning of software binaries," in IEEE/ACM Inter. Conf. on Computer Aided Design, 2002, pp. 164–170.

[6] G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar, "Multilevel hypergraph partitioning-application in VLSI domain," IEEE Trans. VLSI Syst., vol. 20(1), 1999.

[7] C.J. Alpert, "The ISPD98 circuit benchmark suite," in Proc. of the Inter. Symp. on Physical Design, 1998, pp. 80–85.

[8] Y. Jiang, H. Zhang, X. Jiao, X. Song, W.N.N. Hung, M. Gu and J. Sun, "Uncertain model and algorithm for Hardware/Software partitioning," IEEE Comp. Soc. Annu. Symp. VLSI, 2012, pp. 243–248.

[9] A. Al-Wattar, S. Areibi and F. Saffih, "Efficient on-line Hardware/Software task scheduling for dynamic run-time reconfigurable systems," in 26th Inter. Parallel and Distributed Processing Symp. Workshops & PhD Forum, 2012, pp. 401–406.

[10] D.E. Goldberg, Genetic Algorithms in Search Optimization and Machine Learning, Pearson Education, 2004.

[11] W. Sheng, W. He, J. Jiang and Z. Mao, "Pareto optimal temporal partition methodology for reconfigurable architectures based on multi-objective genetic algorithm," in 26th Inter. Parallel and Distributed Processing Symp. Workshops and PhD Forum, 2012, pp. 425–430.

[12] P. Mazumder and E.M. Rudnik, Genetic Algorithms for VLSI Design Layout and Test Automation, Pearson Education, 2003.

[13] J. Yu, Z. Hehua, J. Xun, S. Xiaoyu, N.N.H. William, G. Ming and S. Jiaguang, "Uncertain Model and Algorithm for Hardware/Software Partitioning," IEEE Computer Society Annual Symp. on VLSI, 2012, pp. 243-248.

[14] L. Luo, H. He, Q. Dou and W. Xu, "Hardware/Software partitioning for heterogeneous multicore SoC using genetic algorithm," in 2nd Inter. Conf. on Intelligent System Design and Engg. App., 2011, pp. 1267–1270.

[15] L. Su and X. Zhang, "Research on an SOC Software/Hardware partition algorithm based on undirected graphs theory" in IEEE Inter. Conf. on Computer Science and Automation Engg., 2012, pp. 274–278.