International Journal of Control and Automation
Vol. 8, No. 1 (2015), pp. 29-36
http://dx.doi.org/10.14257/ijca.2015.8.1.03

Hardware/Software Partitioning Based on Hybrid Genetic and Tabu Search in the Dynamically Reconfigurable System

Lanying Li *, Jianda Sun, Weijia Li, Zhiqiang Lv and Fei Guan

The College of Computer Science and Technology ,Harbin University of Science and Technology, Harbin 150080,China
[1]lulu08521@sina.com

Abstract

This paper proposes a hardware/software partitioning algorithm which can be applied to the dynamically reconfigurable system. Firstly, based on reconfigurable system structure, this paper brings forward a kind of system model and its task description ,then use genetic/tabu search (GATS) integration strategy, in the condition of resource constrain of the reconfigurable systems, specific applications will be mapped to software and hardware platform. Secondly, using configuration prefetching and scheduling strategy it will be found out that the shortest time assignment and the execution order of the entire task flow diagram in the partition result. The experiment results have shown that this method can effectively map the task graph to reconfigurable system, and is a kind of method with high performance.

Keywords: *reconfigurable system; hardware/software partitioning; genetic algorithm; tabu search*

1. Introduction

In the design of embedded system, hardware/software partitioning is important in the system design stage. Its primary task is dividing the system functions into the hardware/software part of target structure while can meet all kinds of restrains, and provide the best compromising scheme in hardware/software partitioning. In recent years, with the progress of reconfigurable hardware and its ability for runtime reconfigure, more and more embedded systems use the reconfigurable system structure [1-2]. Reconfigurable hardware's runtime reconfigurable ability makes the function of hardware changeable according to the execution of the program. This flexibility breaks the traditional application pattern of hardware, meanwhile making the traditional hardware/software partitioning unsuited to reconfigurable system. The hardware/software partitioning of reconfigurable system must considers the distinctions of reconfigurable hardware, such as dynamically partial reconfiguration and delays in reconfiguration [3-4].

2. System Model

The model of dynamically reconfigurable system, which is showed in Figure.1, is made of microprocessor, configuration controller, FPGA, memory and configuration memory [5]. In order to free microprocessor from FPGA's configuration tasks, and let them truly execute different tasks at the same time, there will be an additional configuration controller which configures the FPGA in the system. When microprocessor reaches an instruction of hardware configuration, it delivers relevant data to configuration controller, and the latter gets the data from configuration memory before downloading it to FPGA. Microprocessor and reconfigurable hardware communicate with each other by sharing memory. Therefore, the communication time includes the time of data delivery and also the time of data read/write. Memory in this system has no limitations, but CLB in

ISSN: 2005-4297 IJCA

reconfigurable hardware does. Tasks are executed in serial in microprocessor, multiple tasks executed at the same time in reconfigurable hardware, and microprocessor works at the same time with reconfigurable hardware. In dynamically reconfigurable system, application is described as a DAG considering dynamically partial reconfiguration and configuration delay. Firstly, a large-scale application is divided into multiple subtasks which have suitable granularity and restrain connection, and the execution of task is non-preemptive in either microprocessor or FPGA. According to the restrain relationship between these subtasks, we can draw a DAG which shows the restrain relationship between tasks [5]. The node in DAG stands for basic task-dispatch module, the edges stands for communication and precedence relationship between tasks. The weight of edge stands for the communication cost. Every basic task-dispatch module gets data from their father nodes, and deliver it to their son nodes. Every node attribute B_i is defined as a following quintuple:

$$B_i =< B_i^{sw}, B_i^{hw}, B_i^{rc}, B_i^{clb}, B_i^{t} > \quad (1)$$

During formula (1), i stands for node number of task, B_i^{sw} stands for estimated value of task node i 's executing time in microprocessor, B_i^{hw} stands for estimated value of task node i 's executing time in reconfigurable hardware , B_i^{rc} stands for configuration time of task node i's executing, B_i^{clb} stands for CLB needed for the executing of task node i , B_i^{t} stands for type number of task node i . B_i^{t} is the number of task classification group in which the tasks have the same parameters but different node number.

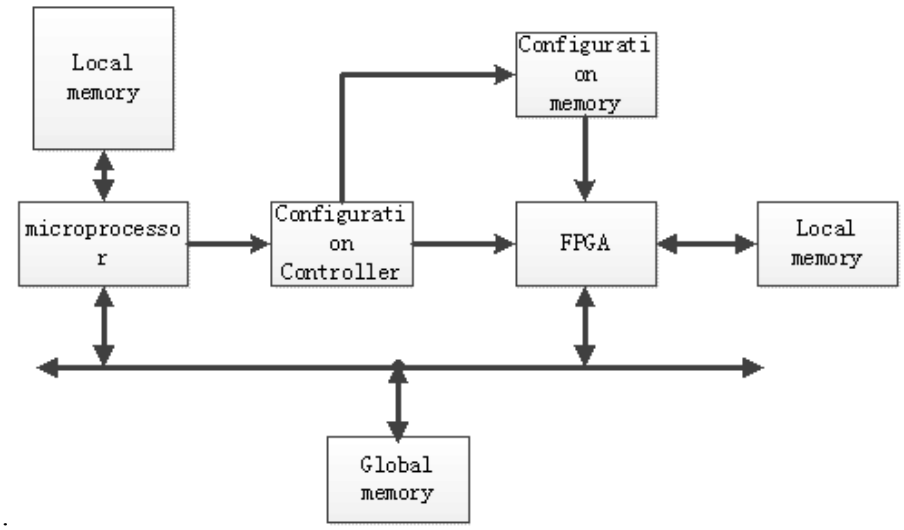


Figure 1. Model of System Structure

3. Hardware/Software Partitioning

Hardware/software partitioning and realization change with different application. The solution is to find out time/space mapping from the application described by task DAG to microprocessor and reconfigurable hardware, and it's indeed a kind of global optimization problem [6-7].In order to minimize system cost under time restrain, we will use a mixed strategy(GATA) of GA and TS to handle the task partitioning.

Glover, the founder of TS, gave a theoretical analysis and discuss on the necessity and possibility of mixture of GA and TS, which is generally known as theory basis of mixture of GA and TS [8]. According to Glover's theory and the analysis of GA and TS, we propose a mixed strategy GATS, and apply it to hardware/software partitioning of

reconfigurable embedded system. Because of GA's weak hill climbing ability and TS's strong hill climbing ability, we use TS algorithm to improve GA's hill climbing ability, which means taking TS as the mutation operator TSM of GA. GATS has the advantages of multiple starting point from GA and memory function and strong hill climbing ability from TS [9].

The initial parameters are provided firstly (including maximum iterations times $\max generation$, group size $popsize$, the exchange probability pc and mutant probability pm).We generate initial group randomly with GA algorithm and then code it with binary system. After that, we do operations such as selection and overlap to the individual, and call TS algorithm in mutating stage of GA. The whole process of this algorithm is showed in Figure.2 (a). Among that, as the mutation operator, TS is the core in the algorithm. The process of TS is showed in Figure.2 (b). If the random-generating probability $p < p_m$, call TS. Tabu searching involves concepts such as neighbor, tabu list, tabu length, candidate and aspiration criterion, etc. The neighborhood solution is got in GATS's mutation operator with 0-1 opposite factors, and X_0 is the optimal state which meets the special pardon principle. X_m is the optimal neighborhood solution which is absent from the tabu list. The whole process of GATS algorithm is showed in Figure 2.

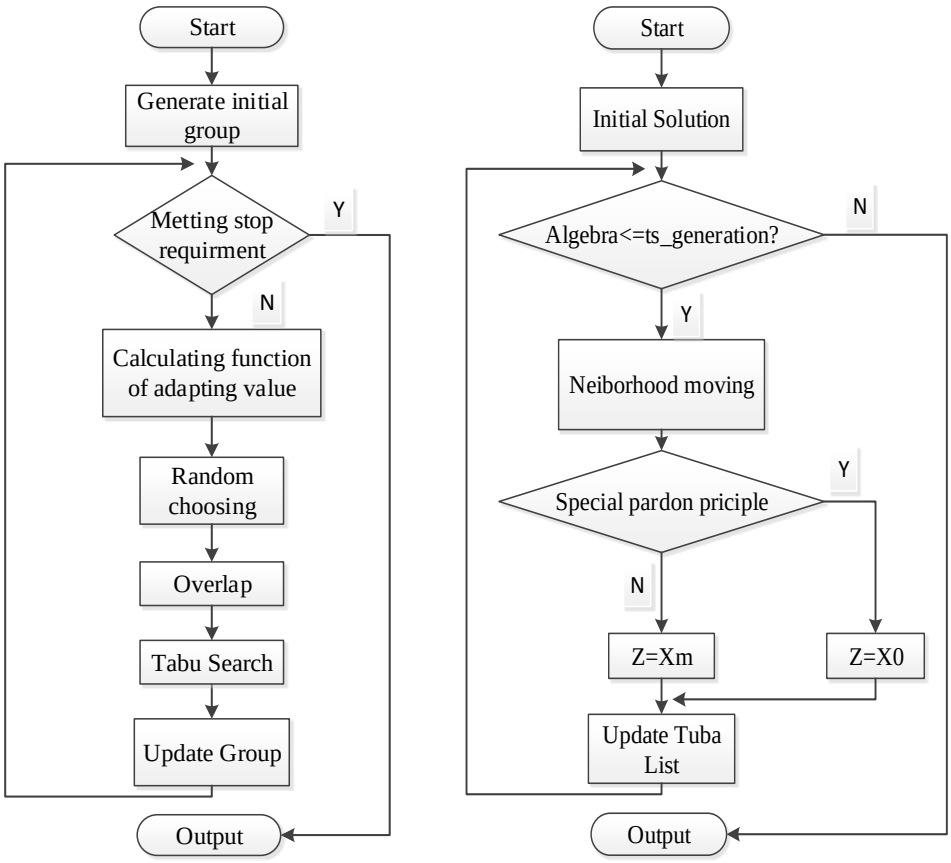
4. Scheduling Algorithm

The purpose of scheduling is finding out the least-time-cost dispatch and execution sequence of tasks according to the partitioning result got from the mixed algorithm of genetic algorithm and tabu search [9]. Now most DAG- scheduling algorithm is based on a so called list scheduling technology [10]. There are two stages in list scheduling algorithm. Firstly, the priority of each task is decided, then comes the stage of processor choosing, which chooses the appropriate processor to run the ready task which has the highest priority. Its basic idea is using a scheduling sequence to save all ready tasks, and when there is vacant operation resource, choose one task which has the highest priority and run it at the vacant operation resource.

4.1. Priority of Scheduling Algorithm

The quality of scheduling algorithm mostly depends on how it defines each task's scheduling priority. The common standard includes ASAP, ALAP, etc. But these priorities are often limited by their static properties, and it's calculated before scheduling. During the scheduling, the static property can't describe task's dynamic property accurately.

Therefore, the formula $priority(V_i) = -(dyna\_ASAP(V_i) + ALAP(V_i))$ [11] is used in this paper to describe the dynamic priority. During that, $Dyna\_ASAP$ stands for starting time of dynamic $ASAP$, and the value of $ALAP$, which is got by static calculating, remains the same. As to task flow graph which is expressed by reverse adjacency, we calculate the critical path, then decide the value of ASAP and ALAP of every task node, and each node's ASAP takes the maximum value among these from multiple paths, every node's ALAP takes the minimum value among these from multiple paths. The communication time among tasks is allocated to source tasks and purpose tasks which is communicating. Greater ASAP value means the later scheduling of tasks, therefore has lower priority, and greater ALAP means later execution and low priority.



(a) Main Process of GATS Algorithm (b) Process of Individual Tabu Search

Figure 2. Process of GATS Algorithm

The algorithm of calculating dynamic priority is as follows: as for each ready task, find the recent-done time from all of its former tasks, then search reconfigurable hardware by recent-done time until finding a room for this task. Then give the time to $Dyna\_ASAP$, and calculate dynamic priority according to it.

4.2 Configuration Prefetching

The feature of dynamically reconfigurable system is that the hardware structure or device can change its function and connection fast. The system's work time is divided into operation time and preparation time when using dynamic reconfiguration. The real operation time means time for module's work and communication. The preparation time is the delay in the function shift caused by configuration. Now dynamically reconfigurable system faces the choke point of long preparation time. The ways to reduce preparation time includes increasing the device's speed for reconfiguration, and reducing the configuration time of software hides. Therefore, we adopt Configuration Prefetching Scheme to hide the system's configuration time. Its basic idea is: when some node in DAG is under dispatch and execution, to configure its latter node in advance, and use a configuration waiting queue to save these configuration nodes which need to be configured but can't start immediately because of the occupation of FPGA's configuration ports.

Algorithm to calculate each node's configuration prefetching list is as follows:

(1) Preset a task's flow graph G and all nodes' execution plan P of a application program.

(2) Make Reverse traversal to G, as for each node V, if V is executed in FPGA, then insert V into its own configuration prefetching list.

(3) Insert each node in V's configuration prefetching list into every configuration prefetching list of V's direct predecessor node.

(4) As for each node in G, arrange every node in its configuration prefetching list from high to low according to the priority got from the 4.1 section.

(5) Make forward traversal to G, as for each node V and every node K from V's configuration prefetching list, if some node V' can be found for every path from G's starting node to V, which makes K a member of configuration prefetching list of V', then delete K from V's configuration prefetching list.

(6) Traverse each node in V's configuration prefetching list, delete the latter node which can't hide its configuration cost.

(7) Return configuration prefetching list of each node in G..

4.3. Scheduling Algorithm

Scheduling algorithm is as follows:

(1).Insert each entrance node in G into scheduling queue.

(2).Check node which has finished configuration in FPGA in advance. If all former nodes have finished execution, then start executing this node.

(3).If the microprocessor is free, select node V which has the highest priority and appointed to execute with microprocessor from scheduling queue, then insert nodes in V's configuration prefetching list into configuration waiting queue.

(4).If some node K has completed its configuration, and all direct precursors have been done, execute K immediately.

(5).If the configuration port of FPGA is free, select the node V which with highest priority and can be satisfied with the rest resource of FPGA in this moment from configuration waiting queue, then start configuration and update the resource use record of FPGA, after that insert the node in configuration prefetching list into configuration waiting queue.

(6).Insert new node which is ready for scheduling into scheduling queue.

(7).Repeat (2)~(6) until the scheduling queue is empty.

(8).Return execution time of the last node.

5. Experiment and Analysis

To verify the effectiveness of hardware/software partitioning algorithm and configurable scheduling scheme which are proposed in this paper, we use TGFF toolkit to create DAG randomly as task graph, and divide nodes into different types of task, then set different runtime, area and so on as cost for different types of task.

Suppose the target system is made of single processor and Virtex □ series xc2v1000 FPGA, which contains 1280 CLBs. The experimental environment is Intel 1GHz Processor, 512MB Memory, Linux Operating System, and GUN compiler. To simplify the experiment, we make reasonable assumption as follows:

(1) Microprocessor and FPGA are connected to the shared memory at same rate, therefore we can allocate the communication time to two tasks which communicates with each other.

(2) The cost of runtime and area which tasks need to run in microprocessor and FPGA is known and invariant.

(3) The resource utilization rate is 70%.

Owing to the effect of controls parameters to the result, we use the same controls parameters in the comparison process to guarantee comparability. Among that, GA's maximum iterations: $max\,generation = 3n$ (n is the node's quantity in DAG), group

size $popsize = n$, chrome length $chromelength = n$, crossover probability $p_c = 0.8$, mutation probability $p_m = 0.01$. And TS's maximum iterations $ts\_generation = 3n$, neighbourhood number $neigh\_chrom = n$, tabu length $tabu\_length = \sqrt{n}$ 。 These data will be referred in GATS to reduce the calculated amount. In the TS part of GATS, the maximum iterations is 100, but the whole GATS's maximum iterations is still max generation. The end condition of all three algorithms is judging if it reaches the maximum iterations.

Run GA, TS, GATS program separately under the same controls parameter, and set chromosome length, group size $popsize$ and the number of neighborhood solutions to 30,40,50,60,70, 80.

Table 1, Figure 3 and Figure 4 gives the result when the algorithm runs in different scale. All three algorithms in Figure 3 use configuration prefetching but don't use scheduling scheme. All three algorithms in Figure 4 use configuration prefetching and scheduling scheme, and all of them get a good partitioning result compared with Figure 3. Because GATS combine the global search ability of genetic algorithm and partial search ability of tabu search algorithm, it has a better convergence rate for all different application scales compared with GA. Configuration prefetching and schedule scheme can decrease efficiently the configuration time of the system in different application fields, therefore decrease the finishing time of task graph.

Table 1. Execution Time Comparison of Three Algorithms

Task	GA		TS		GATS	
Node	NS	S	NS	S	NS	S
30	2071	1644	2058	1950	2056	1538
40	2845	2241	2835	2743	2809	2076
50	3680	3087	3573	3402	3362	2458
60	4316	3085	4311	4001	4310	2770
70	4944	3324	4806	4522	4776	3014
80	5431	3611	5248	4986	5119	3252

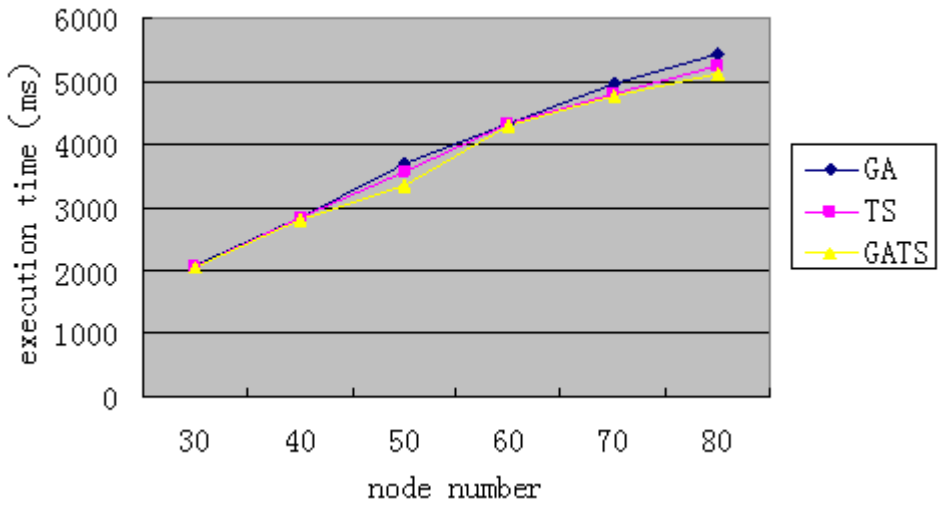


Figure 3. Comparison of Three Algorithms Execution Time (NS)

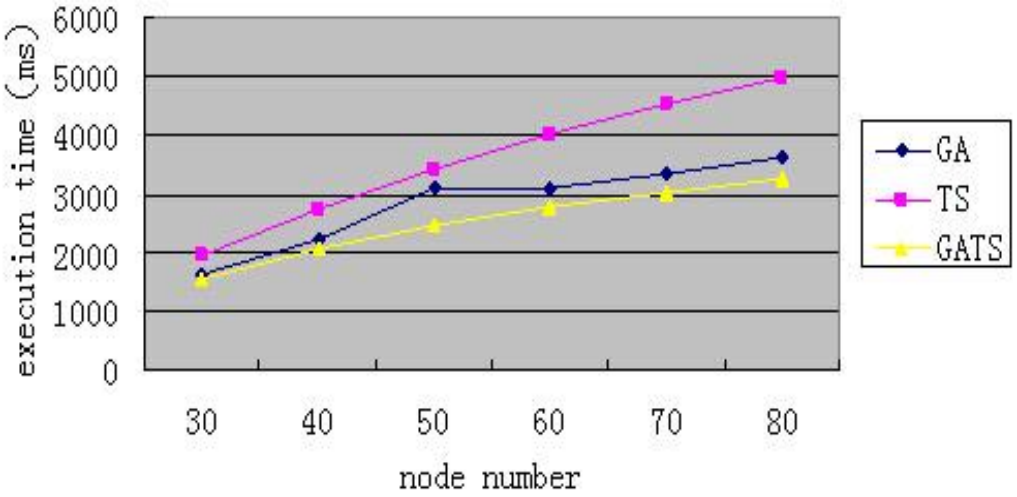


Figure 4. Comparison of Three Algorithms Execution Time(S)

6. Conclusion

Reconfigurable system can use the speed and flexibility of reconfigurable hardware efficiently, and provide a good computing platform for applications. Hardware/software partitioning is the key technique to use reconfigurable system efficiently. We realize the time-space reflection from application to reconfigurable system efficiently based on the features of reconfigurable hardware, the combination of genetic algorithm and tabu search algorithm, configuration prefetching and schedule scheme.

The method proposed in this paper aims at single microprocessor and one-chip reconfigurable system for now, and it divides and reflects application programs under restrains of time and certain area. Hardware/software partitioning and reflection for dynamically reconfigurable system with multiple microprocessor and multiple-chip FPGA will be the emphasis of next stage's research.

Acknowledgement

Supported by National Collegiate Innovation and Entrepreneurship Training Program (No: 201310214013).

References

[1] R. Ernst, J. Henkel and T. Benner, "Hardware-software Co-synthesis for Microcontrollers", IEEE Design &Test of Computers, vol. 10, no. 4, **(1993)**, pp.64-75.
[2] P. Yipin, L. Ming and Y. Jun, "Hardware-software Partitioning Research Based on Resource Constraints", Circuits and Systems, vol. 10, no. 3, **(2005)**, pp. 80-84.
[3] Y. Zou, Z. Zhuang and H. Chen, "HW-SW Partitioning Based on Genetic Algorithm", Proceedings of the Congress on Evolutionary Computation, **(2010)**, pp. 628-633.
[4] H. Comptonk, "Reconfigurable Computing: a Survey of Systems and Software", ACM Computing Surveys, vol. 34, no. 2, **(2002)**, pp. 171-210.
[5] J. Todmant, A. Constantindesg and E. Wiltonsj, "Reconfigurable Computing: Architectures and Design Methods, IEE Proc of Computers and Digital Techniques, vol. 152, no. 2. **(2005)**, pp.193-207.
[6] J. Jianchun, "Research on the Critical Problem of Embedded Software for Heterogeneous Multi-core", ChongQing University, **(2011)**.
[7] J. Yiming, "Hardware-software Partitioning Algorithms Research on Reconfigurable SOC for the Different Constraints", HuNang University, **(2013)**.
[8] F. Glover, J. Kelly, M. Laguna, "Genetic Algorithms and Tabu Search: Hybrids for Optimization", Comput & Ops Res, vol. 22, no.1, **(1995)**, pp.111-134.
[9] B. Mei, P. Schaumont and S. A. Vemalde, "Hardware/software Partitioning and Scheduling Algorithm for Dynamically Reconfigurable Embedded Systems", Proceedings of ProR ISC, **(2000)**.

[10] L. Gingmei and J. Shengnan, "Research on the Static Task Scheduling for Heterogeneous Multi-core Processor", Journal of Computer Engineering and Design, vol. 34, no.1, **(2013)**, pp.178-184.
[11] L. Renfa, L. Yan and X. Cheng, "Reasearch Progresses on Task Scheduling for Multiprocessor SOC", Journal of Computer Research and Development, vol. 45, no. 9, **(2008)**, pp. 1620-1629.

