Transition-aware Task Scheduling and Configuration Selection in Reconfigurable Embedded Systems

Hessam Kooti
Google Inc.
Mountain View, CA, USA
hkooti@google.com

Eli Bozorgzadeh
Computer Science Department
University of California, Irvine, CA, USA
eli@ics.uci.edu

Abstract—This paper presents a novel graph representation that captures the transition overhead due to runtime configuration of underlying hardware in reconfigurable embedded systems resulting from various configuration schemes such as FPGA-like reconfiguration or dynamic voltage/frequency scaling (DVFS). We propose an intuitive heuristic to solve combined configuration selection and task scheduling problem on this graph. In addition, when applied to DVFS, our algorithm provides simultaneous task ordering and configuration selection of the system which outperforms the state-of-the-art DVFS methods applied after task ordering.

I. INTRODUCTION

In order to cope with time-varying input characteristics and demand for upgradability, embedded systems are urged to provide tremendous flexibility through extensive configurations. Such requirements along with performance, power, and reliability are driving embedded systems to adopt hardware reconfigurability and self-adaptive system methodologies. While hardware configurations such as dynamic voltage/frequency scaling are mainly aimed for energy-aware and battery-aware runtime adaptation of the system [1,2,3,4], runtime reconfiguration of programmable devices such as FPGA devices [5,6,7] are deployed to modify the functionality or the processes running on underlying hardware. However, runtime reconfiguration, can incur stringent constraints that lead to failure to meet real time performance requirements of applications. For example, due to reconfiguration, the system may fail to process the incoming input frame or packet. In order to provide seamless quality of service, runtime adaptation needs to be planned early in system design flow. The software layer such as middleware and operating system cannot ignore this overhead during task scheduling and resource management.

In this paper, we propose a novel graph representation of input non-preemptive tasks and underlying embedded system architecture and provide an efficient and intuitive algorithm to provide a feasible schedule of the tasks on embedded system with runtime reconfiguration of underlying hardware. Our proposed graph is generic and can be used for task scheduling on embedded systems with various reconfiguration schemes such as DVFS [1,2,3], FPGA-like partial reconfiguration [7], etc. The proposed graph model and algorithm combines the scheduling with configuration selection of the system while considering the delay and energy overhead due to transition from one state to another.

*This work is funded by NSF CAREER # 0846129.

II. TARGET ARCHITECTURE

We target a heterogeneous embedded system platform consisting of several Processing Elements (PEs). Each PE can be an embedded processor with DVFS, a reconfigurable processor (soft processors), processors with reconfigurable coprocessors and/or memory, or a reconfigurable hardware (fine or coarse grained) (Figure 1). For each PE, there is a reconfigurable part (or circuitry) which is reconfigured/updated during the runtime. The reconfiguration can target functionality (e.g., FPGA partial reconfiguration) or physical characteristic of underlying hardware (e.g., dynamic voltage scaling). The PEs share a data memory through a shared bus. Based on the task scheduling timings from the *scheduler*, the *reconfiguration manager* forces the PEs to reconfigure appropriately. The inputs to this system are a set of periodic tasks with known period, execution time and deadline. We assume that we are given the information about the task input for a given time interval on which we solve the scheduling problem statically.

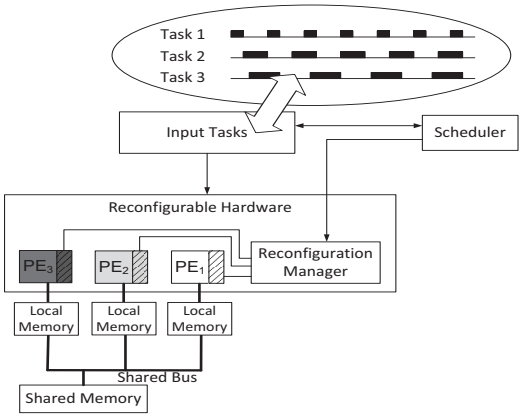

Figure 1: Target reconfigurable embedded system

III. TRANSITION-AWARE GRAPH REPRESENTATION

In a feasible schedule, the execution of the tasks must be completed by their corresponding deadlines, while only one task is allowed to be executed on each PE at a time. In the proposed graph representation, a node v_i represents the execution of task i on one PE. In some cases, we have different options to execute a task on a PE. These options can include different hardware implementations or different operation modes to run the task. The graph has to capture the execution of tasks under different implementations/modes. In this case, a set of nodes represents the execution of the task on the PE, referred to as a *super node*. For example node $v^k_{i,m}$ represents

execution of task *i* under operation mode *m* on PE *k*. In Figure 2, only the super nodes are shown except for node v_8. The subset of nodes in super node v_8 shows that task 8 can be executed on PE1 and PE2 under three different configurations.

Each *transition* edge represents the overhead between two consecutive tasks on a PE. This overhead represents a reconfiguration time overhead in a reconfigurable platform or the delay overhead during voltage scaling in a PE. Such edges exist only between the nodes with the same *k* index (i.e. being mapped to the same PE) (Figure 2). The transition overhead as modeled in this graph does not need to be same for all PE configurations and can vary. For example, if reconfiguration overhead for the next task execution depends on the current configuration of the system and the current task being executed, the corresponding edge will carry this overhead as the weight. A directed path along the transition edges in this graph imposes an ordering on execution of the corresponding tasks on a PE. In case there is a data dependency between the tasks, the transition edges can be refined further to impose the partial ordering imposed by data dependency.

There is another set of edges, called *data* edges, to represent the communication overhead for data transfer between the PEs (through a shared memory in this example). The data edges representing the communication overhead are only between the nodes with different *k* indexes (i.e. tasks belonging to different PEs). The weight associated to these edges is equal to the communication time overhead between the two end nodes. These edges (communication overhead edges) are shown as dotted edges in Figure 22.

We insert a dummy source node in the graph to indicate the initial state of the system in each period and a dummy sink node to indicate the final state of the system at the end of each period. There is a directed edge from the dummy source node to each node in the graph (i.e. any task can be the first task on a PE) and a directed edge from each node to the dummy sink node (i.e. any task can be the last task on a PE).

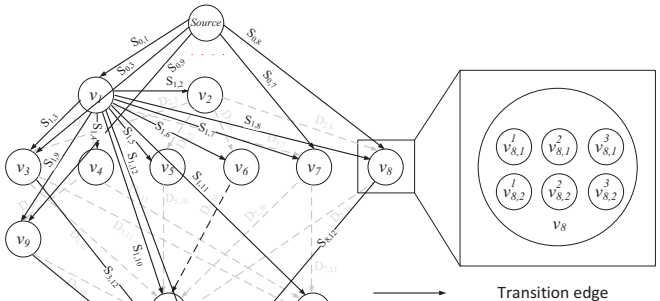

Figure 2. Graph Representation and Detailed view of a super node

Given the period, the execution time and the relative deadline of each task, we compute the time window (i.e. $[a_i, b_i]$) during which each task should start execution in order to finish execution before its deadline. For node $v_{i,m}^k$, the terms $[a_i, b_{i,m}^k]$ $t_{i,m}^k$, $e_{i,m}^k$ and $c_{i,m}$ represent execution start time window, execution start time, execution time, and cost of task execution (e.g. energy consumption), respectively. The weight $s_{i,m;j,n}^k$ and cost $ce_{i,m;j,n}^k$ represents the transition overhead between $v_{i,m}^k$ and $v_{j,n}^k$, and the cost associated with this transition, respectively. In order to have a feasible schedule, two conditions should be met: 1) start time of each task should be inside the corresponding time window 2) time overheads before each task should be taken into account.

If we find a path in this graph starting from the source node and ending at the sink node, along which each super node is visited once (or labeled with a start time) such that the aforementioned conditions are satisfied, a feasible schedule is provided for the tasks along the path on a PE. Any directed path in this graph corresponds to execution of a set of task on a PE and it must include at most one node from each super node corresponding to the selected tasks. All the nodes along a path have the same *k* index. On multiple PEs, we need to find multiple *disjoint* paths and cover all the super nodes in order to provide a feasible schedule for all the tasks. Along a path, the cost (e.g., energy consumption) is calculated by adding the weights on the edges and the nodes (e.g., energy consumption of the nodes and edges) along the path. For energy minimization purposes, the goal is to find a set of disjoint paths that covers all the *super nodes* while the total path costs (energy consumption of the system) are minimized (min cost). In addition, the disjoint paths in this graph not only represent a task ordering on PEs, but also determine the configuration of the PE for each task execution. Hence, the configuration selection and task ordering (or scheduling) is simultaneously solved. For example, if configurations refer to operating voltage/frequency of the processor for each task, task ordering and voltage/frequency selection are tackled at the same time in this graph as opposed to two sequential steps [2,3].

In our previous work, we have proposed mathematical programming to solve the task scheduling and configuration selection problem on our proposed graph to incorporate partial reconfiguration overhead in FPGAs [5] and energy minimization using DVFS [1] in embedded reconfigurable multiprocessors. In this paper, we provide an extended and generalized graph representation. While mathematical programming solution is optimum but it is very computationally expensive as a solution and its runtime is not affordable to be used by embedded system design exploration tools. Hence, we propose a heuristic algorithm to solve the problem more efficiently. In this work, we propose to formulate the problem similar to a special class of network flow problem referred to as Vehicle Routing Problem with Time Window (VRPTW) [8]. However, VRPTW problem in [8] does not consider multiple operation modes, other constraints regarding reconfigurable architectures as modeled on our graph representation. We propose how to adopt the vehicle routing algorithm to find disjoint paths in order to solve the min cost task scheduling problem on our graph representation of reconfigurable system. We present this algorithm with the objective of energy minimization (cost minimization) and apply it to DVFS problem in embedded systems [1]. However, the proposed heuristic is not restricted to DVFS and can be simply applied to task scheduling on FPGA-like systems with runtime reconfiguration.

IV. OUR HEURISTIC TRANSITION-AWARE SCHEDULING

Since schedulability is not guaranteed, we first aim for maximum schedulability, and then focus on energy minimization. Our proposed heuristic is based on a heuristic

for solving VRPTW ([9,10]). However, in our problem each super node (representing the execution of a job) is a set of nodes in the graph, and our heuristic selects the best operation mode of the PE for each task instance referred to as a *job*.

Figure 4 presents pseudo-code of our heuristic algorithm. In our heuristic, the first part (lines 1-13) focuses on feasibility of the real-time scheduling. We first relax the constraint on the number of the paths (i.e. number of PEs) and find a feasible solution such that all the super nodes are visited (i.e. all the jobs are executed). For this step, we use Push Forward Insertion Heuristic (PFIH) algorithm [10] (line 1 in Figure 4). Next step is to restrict the number of the paths. If the number of paths provided by PFIH is greater than the number of PEs, we will keep deleting the paths with minimum number of nodes until the number of paths is equal to the number of PEs.

Next, we need to allocate the deleted super nodes on the remaining paths. Note that when we want to add a super node to the existing paths, there is no need to add the same node which was deleted and we can add any node from that super node. First, based on an insertion cost function, we find the best location on existing paths to insert each deleted super node (i.e. for each node inside the deleted super nodes we find the best location). If no feasible position is found, we forcefully insert the node with minimum insertion cost in the best position by removing some existing nodes from the path to make it feasible (line 8). Then we apply a series of transformational operations on scheduled nodes along the paths to increase the chance of inserting deleted super nodes to existing paths (line 9).

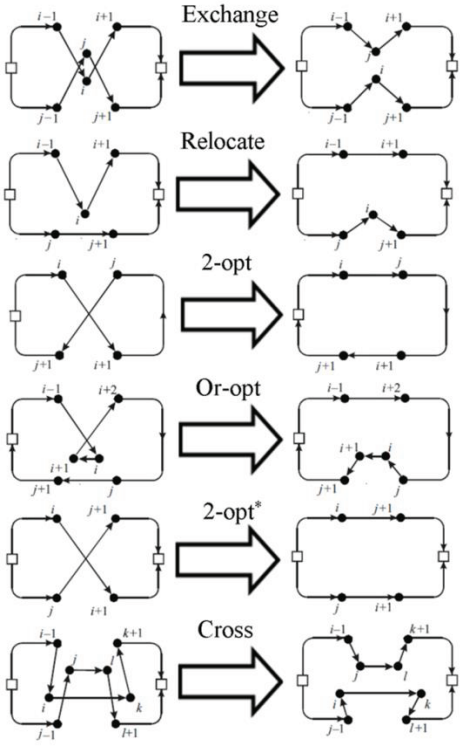

Figure 3-Transformational Operations

Error! Reference source not found. demonstrate the set of transformations used in our algorithm. The transformational operations are used in order to move to another neighbor in solution space [11]. Some transformations are performed on a single path and some of them are performed on a pair of paths among all the paths in our solution. In **Error! Reference source not found.**, in each picture, the left column shows the considered path(s) for transformation before performing the action and the right column shows the same path(s) after the transformation. For example **Error! Reference source not found.**-a shows the paths r_1 and r_2 as *{source, ... , v_{i-1}, v_i, v_{i+1}, ... , sink}* and *{source, ... , v_{j-1}, v_j, v_{j+1}, ... , sink}* before transformation and paths r_1' and r_2' as *{source, ... , v_{i-1}, v_j, v_{i+1}, ... , sink}* and *{source, ... , v_{j-1}, v_i, v_{j+1}, ... , sink}* after transformation.

Algorithm 1. Heuristic Transition-aware Task Scheduling

Input: Graph Representation of the input task and reconfigurable system

```
[1]  perform PFIH //find paths to cover maximum number of nodes
[2]  if (number of paths ≤ number of processors) then
[3]     go to energy minimization part (line 14)
[4]  else
[5]     remove the paths with minimum number of nodes until number
        of paths is equal to number of processors
[6]  end if
[7]  while (not all super nodes are visited) and (not to terminate) do
[8]     select an unvisited super node and insert in a path
[9]     explore the neighbor solutions to improve the timing
[10]    if (paths are not changed)
[11]       terminate
[12]    end if
[13] end while
[14] repeat //energy minimization
[15]    for (each path, p) do
[16]       while (total energy consumption p is reduced) do
[17]          apply hill-climbing with 2-opt, Or-op, Reduce and E-opt
[18]       end while
[19]    end for
[20]    for (each pair of paths, (p1,p2) ) do
[21]       while (total energy of p1 and p2 is reduced) do
[22]          apply hill-climbing with Relocate, Exchange, 2-opt* and
              cross to reduce the energy
[23]       end while
[24]    end for
[25]    if (the total energy of paths is not reduced) then
[26]       apply Generalized Ejection Chains to reduce the energy
[27]    end if
[28] until (total energy consumption of the paths is not reduced)
```

Figure 4. Heuristic Algorithm

After each operation, a cost function is used to check if new solution is better than existing solution in terms of available free time. During these operations, we will look for the best node to use from each super node. These steps (line 8 and 9) are repeated until all the nodes are visited or no more improvement is possible (lines 7 and 10-13). After maximizing the number of nodes on the paths, we will perform the second part of the heuristic to minimize the cost, i.e., energy (lines 14-31). In this step, we apply the aforementioned transformational operations sequentially to explore the neighbors in solution space. However in this part, the criterion to accept the transformation and move to the next neighboring solution is energy (cost) saving without reducing the schedulability. In this part, in addition to previous transformation, we have a new operation named *Reduce*(v_i) which tries to find the lowest cost (e.g., smallest operational voltage in DVFS) for v_i without reducing the schedulability (i.e. finding the best node in the supernode). We repeat these operations until no cost (energy) saving is possible. In the second part of our procedure we use two local searches with classical operators, an enumeration-based operator (E-opt) to minimize the energy of a single path

and a Generalized Ejection Chain (GEC)-based operator to minimize the total energy of all the paths. These operations are explained in details in [9].

V. EXPERIMENTAL RESULTS

We applied our proposed heuristic algorithm to tackle DVFS-based energy minimization problem in embedded systems. In our previous work [1], using our graph representation, we presented an optimal ILP-based network flow based solution for simultaneous static real-time scheduling and energy minimization (DVFS). In this work, we apply the proposed heuristic algorithm to allow task re-ordering during energy minimization. We have conducted a set of experiments using generated benchmarks as well as 3 real-world application task sets. We have 7 synthetic benchmarks, each of them with 3 tasks. The three real-world applications task sets are the task sets from Computerized Numerical Control (CNC) machine controller application (8 tasks), Video phone application (4 tasks) and Avionics application (18 tasks). Tasks do not have data dependencies. Our task properties are presented in details in [1].

We considered three operation modes (processors used in [3]) for each processor in the system. We indicate each operation mode with a pair of supply and body bias voltage $mode_m = (V_{dd_m}, V_{bs_m})$. The processor has 3 modes (1.8,0), (1.5,-0.4), and (1.2,-0.6). In the real-world applications only 1 processor was enough for scheduling all the tasks, but in synthetic benchmarks we need 2 processors to be able to run all the tasks. We used the data from [1] for delay and energy overhead due to switching from one operation mode.

We implemented our heuristic in C#. The execution time of this heuristic in our simulation was less than 3 minutes on a computer with CPU Intel P4 3.4GHz and 3GB RAM. We compared our solutions with 3 existing voltage/frequency scheduling algorithms: MILP [1], VCS [2] and DVO [3]. The results from DVO determines the optimal solution while not considering ordering during voltage assignment, hence the difference of this algorithm from our optimal solution shows the importance of considering the ordering and voltage/frequency assignment at the same time.

TABLE I. NORMALIZED ENERGY DISSIPATION

	Processors with 3 operation modes				
	no-DVFS	***VCS [2]***	***DVO [3]***	***Our heuristic***	***MILP[1]***
Task set 1	100	78.95	78.89	**70.93**	47.52
Task set 2	100	82.21	62.76	**52.76**	47.79
Task set 3	100	69.73	60.17	**57.56**	48.32
Task set 4	100	45.58	45.58	**45.58**	45.58
Task set 5	100	45.47	45.47	**45.47**	45.47
Task set 6	100	70.89	62.06	**60.80**	45.83
Task set 7	100	83.02	75.33	**65.26**	62.61
CNC	100	64.03	58.16	**49.46**	49.46
Video phone	100	80.97	71.73	**68.49**	45.50
Avionics	100	52.42	45.89	**45.89**	45.89
Average	100	67.33	60.61	**56.22**	48.40

Error! Reference source not found. show the results for our simulations on synthetic benchmarks and real-world application for the processor with 3 modes of operation, respectively. The first column is the energy consumption of the system in maximum voltage (no-DVFS). Second and third columns are the energy consumption while using VCS and DVO algorithms, respectively. Third and fourth columns are our solutions for DVFS/ABB in real-time scheduling. All the values are normalized to energy consumption of no-DVFS case. Our heuristic reduces the energy by 43.78%, 11.11% and 4.39% (on average) in comparison with no-DVFS, VCS and DVO algorithms, respectively. In this case the difference between the optimal result and our heuristic is 7.82%. For avionics application, the results from our solutions and the DVO algorithm are same. This is the result of the fact that the time constraints in this application are very relaxed. Also there are two cases (Task set 3 and 6) in Table I which our heuristic works slightly worse than DVO and that is because DVO is an ILP-based optimal scheduler and our heuristic is not optimal. Our MILP solution which is optimal is working better than DVO in these cases.

VI. CONCLUSIONS

In this paper, we presented a novel graph representation for task scheduling on reconfigurable embedded systems. The graph can capture reconfiguration delay and cost overhead resulting from various configuration schemes such as DVFS or FPGA partial reconfiguration. Configuration selection and task ordering are simultaneously tackled in the proposed heuristic scheduling algorithm. We can benefit from simultaneous task ordering and DVFS as opposed to applying DVFS on a set of ordered tasks as commonly presented in related work and our results show significant energy reduction when applying combined task ordering and voltage/frequency selection using our proposed heuristic algorithm.

REFERENCES

[1] H. Kooti, E. Bozorgzadeh, "*Unified Theory of Real-Time Task Scheduling and Dynamic Voltage/Frequency Scaling on MPSoCs*", in IEEE International Conference on Computer-Aided Design, 2010

[2] F. Dabiri, A. Vahdatpour, M. Potkonjak and M. Sarrafzadeh, "Energy Minimization for Real-time Systems with Non-Convex and Discrete Operation Modes," in *IEEE Design, Automation and Test in Europe*, 2009.

[3] A. Andrei, M. Schmitz, P. Eles, Z. Peng and B. M. Al-Hashimi, "Overhead-Conscious Voltage Selection for Dynamic and Leakage Energy Reduction of Time-Constrained Systems," in *IEEE Design, Automation and Test in Europe*, 2004.

[4] P. Rong and M. Pedram, "Energy-Aware Task Scheduling and Dynamic Voltage Scaling in a Real-Time System," *Journal of Low Power Eelectronics,* vol. 4, no. 1, pp. 1-10, 2008.

[5] H. Kooti, E. Bozorgzadeh, S. Liao and L. Bao, "Transition-aware Real-Time Task Scheduling for Reconfigurable Embedded Systems," in *Design, Automation and Test in Europe*, 2010.

[6] D. B. Stewart and P. K. Khosla, "Real-Time Scheduling of Dynamically Reconfigurable Systems," in *IEEE International Conference on Systems Engineering*, 1991.

[7] O. Diessel, H. ElGindy, M. Middendorf, H. Schmeck and B. Schmidt, "Dynamic scheduling of tasks on partially reconfigurable FPGAs," *IEE Proceedings Computers and Digital Techniques,* vol. 147, no. 3, pp. 181-188, 2000.

[8] N. Azi, M. Gendreau and J.Y. Potvin, "An exact algorithm for a single-vehicle routing problem with time windows and multiple routes," in *European journal of operational research*, 2006.

[9] A. Lim and X. Zhang, "A Two-Stage Heuristic for the Vehicle Routing Problem with Time Windows and a Limited Number of Vehicles," in *HICSS*, 2005.

[10] M. M. Solomon, "Algorithms for the vehicle routing and scheduling problems with time window constraints," *Operations Research,* vol. 35, no. 2, pp. 254-265, 1987.

[11] O. Bräysy, "Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms," *Transportation Science,* vol. 39, no. 1, pp. 104-118, 2005.