A Genetic Algorithm Approach to Static Task Scheduling in a Reconfigurable Hardware Environment

S. M. Loo, B. Earl Wells, J. D. Winningham
{smloo, wells, jdw}@ece.uah.edu
Department of Electrical and Computer Engineering
University of Alabama in Huntsville
Huntsville, AL 35899

Abstract
This paper presents a basic framework for applying static task scheduling techniques to arbitrarily-structured task systems whose targeted execution environment is comprised of finite amounts of reconfigurable hardware. Such reconfigurable hardware is characterized by the fact that its structure and logical functionality can be altered any time after the hardware devices are constructed. Such an environment is assumed to allow for the use of multiple sequential processing elements, task-specific logic and a communication network within the reconfigurable hardware. This research focuses upon the application of scheduling theory to a static/deterministic environment where all the task execution times and communication times can be estimated with a great degree of certainty, and the system configuration is determined prior to the execution of the application. The goal of this strategy is to create schedules that have minimal overall execution time under the constraint that the implementation of such schedules does not require more resources of any type than is present within the reconfigurable hardware. This scheduling technique conforms in many ways to those applied to the heterogeneous parallel and distributed processing domains but with the addition that the scheduling routines also determine the configuration of the reconfigurable hardware as well as creating an ordered list of tasks and communications for each sequential processing element that is employed. This hardware configuration information includes the type of processors employed, the type of communication that is supported by each data link (buffered or non-buffered), and the interconnection topology needed to support these data links. This paper highlights how a genetic algorithm can be utilized to create schedules for such an environment.

1. INTRODUCTION

The use of pre-fabricated reconfigurable hardware allows for rapid implementation of sizable system designs without the need to create custom hardware [1]. Such reconfigurable hardware is general enough to benefit from the economies-of-scale production advantages afforded to popular commercial off-the-shelf products while providing a degree of flexibility formally found only in software based systems. This flexibility allows the hardware structures for given portions of an application to be specialized and optimized to achieve a performance that can be orders of magnitude greater than that which can be achieved within most traditional processing systems that employ a Von Neumann style architecture.

It is also important to note that the logic resources available within commercial off-the-shelf reconfigurable hardware are limited by the size (feature size, density, input/output connections, etc.) of the underlying reconfigurable device(s) that are employed. For most applications this means that it is impossible to configure the reconfigurable logic in a manner where all portions of the design are implemented in an optimal fashion from a performance point of view. Such performance optimal implementations would consume enormous amounts of reconfigurable hardware resources – probably orders of magnitude larger than can be made available. Unfortunately this problem is unlikely to be solved by improvements in device technology since the application problems themselves continue to increase in size along with their required performance.

The desired compromise is to carefully use the reconfigurable medium in a manner that is cognizant of the time space trade-off that is associated with computations performed in digital hardware. In general, high performance can be gained through increased concurrency. Increased concurrency is the result of employing more of the hardware to solve the problem. Since for any targeted reconfigurable hardware environment such resources are limited at the time of fabrication, the compromise translates into determining how much concurrency should be employed in order to meet the performance requirements of the application without exceeding the resource limits of the reconfigurable hardware medium. The key to finding this effective balance is to develop techniques that can determine, within the confines of the resource limits, which portions of the problem must have increased levels of concurrency to meet the overall performance constraints and which portion of the problem can be implemented more sequentially to save room for the higher performing portions of the design.

One of the most efficient organizations from a hardware resource usage point of view is a traditional Von Neumann style processing configuration. The sequential execution of individual instructions allow the functional units that are implemented in hardware to be reused many times in a time shared manner by many portions of the application. This savings in resources overcomes the added hardware that must be included to perform such operations as instruction fetch, decode, and sequencing. They are very flexible, support very irregularly structured computation but suffer from the performance constraints associated with highly sequential operation. Fortunately the current state of commercial off-the-shelf reconfigurable devices such as Field Programmable Gate Arrays (FPGAs) has progressed to the point where multiple sizable Von Neumann style processing core units can fit within commercially available reconfigurable hardware. This means that incorporating one or more such main stream units within the reconfigurable hardware, along with other hardware elements, can be a viable option to solve the time/pace trade-off.

Such an approach is taken in this research where the reconfigurable hardware is to implement both the high-speed logic that has been designed to execute the most time intensive portions of the application problem and traditional Von Neumann style processing core units to save valuable hardware resources. In this arrangement, the Von Neumann units, the high-speed logic modules, the input/output logic, and the routing between each of these hardware entities are contained within the finite resources of the reconfigurable logic. The trade-off is to determine the number and types of each of these entities that will best meet the needs of the application and fit within the available reconfigurable hard-
ware resources. The goal of the research described in this paper is to develop and evaluate solutions to this space/time trade-off in the cases where the application can be decomposed into a well behaved system of tasks that can be implemented directly in hardware or executed sequentially on one or more processing core units. The solution employs developing parallel processing style static scheduling techniques that can be applied to this finite resource reconfigurable hardware environment. In a traditional parallel processing setting, where the structure of the interprocessor connection network topology is fixed and the number of tasks in the application is greater than the number of processors that are employed, this inherently difficult problem is often divided into three interrelated phases. These phases are assigning the tasks into processing groups where the number of processing groups is equal to the number of processors, mapping each of these processing groups in a one-to-one manner to the available set of processors, and sequencing the order of execution of the tasks on each processor in a manner that insures problem correctness by strictly adhering to the partial ordering of tasks as dictated by the task system’s precedence constraints.

The scheduling model defined in this research makes use of a hardware library that consists of an arbitrary number of Von-Neumann style core units (called processing cores), task-specific logic cores, and inter-core communication elements. The application problem is represented as a system of static tasks that have a governing set of precedence relationships. This task system along with the hardware library of candidate core processors and task specific logic cores are used as inputs to the scheduling process. The scheduler then utilizes the task hardware/software execution time and resource utilization parameters along with the global resource and performance constraints to determine an assignment and schedule of the tasks to the system elements that will satisfy the various design constraints. The scheduler produces both a task execution schedule and a high-level hardware system description that contains the number and type of processing core units, the interconnect routing topology, and number/placement of communication elements. The task execution schedule dictates the order that the tasks execute (task execution sequence) within each sequential execution unit, and among tasks allocated to sequential execution units and the task specific logic cores. The high-level hardware system description is created in a manner that reflects the allocation of the tasks that was dictated by the scheduler. The use of a hardware library adds significantly to the search space that must be explored by the overall scheduling methodology as compared to traditional fixed hardware parallel processing scheduling techniques. This is countered at least in part by the virtual elimination of the mapping phase of the scheduling operation since the reconfigurable medium can be made to conform directly to the topological structure of the application. It is believed that insight gained through this well defined problem domain can be extrapolated and extended to allow formalisms to be developed for less deterministic problems such as those associated with dynamically reconfigurable schedulable systems.

2. RECONFIGURABLE SYSTEM DESIGN MODEL

A high level model is introduced that allows scheduling theory to be applied to a pre-fabricated reconfigurable hardware environment. This model is called the Reconfigurable System Design Model (RSDM) [2]. The components of the RSDM are shown in Figure 1. The heart of this model is the scheduler. Inputs to the scheduler include the hardware library and task system. The hardware library supports three types of functional units that can be placed in reconfigurable hardware. There are also inputs that reflect the hardware resource limits associated with the reconfigurable hardware medium and various design constraints that reflect the required performance of the application. The purpose of the scheduler is to generate a complete static task schedule and high-level hardware system description that satisfies all of the given constraints. The task execution schedule describes the task execution sequence of the system from a global point of view for a single major frame of execution. It does so in a manner that satisfies the precedence and resource constraints. The high-level hardware system description is created at the same time the task schedule is generated. It consists of the number and type of core processors to be implemented and the inter-reconfigurable logic communication topology of the system. In the following sections, the characteristics of these various elements are discussed.

2.1 Hardware Library

The hardware library represents a high-level description of the candidate logic modules that can be used to implement an application. It supports three types of functional units that can be placed in reconfigurable hardware. These functional units include the Processor Cores (PCs), Task Specific Cores (TSCs), and Communication Core Elements (CCEs). In this model, PCs represent distinct elements in the hardware library because they have the general capability to support the Von Neumann style sequential execution of more than one task. In general, the number of tasks that they can execute is limited by the internal program and data memory that is present within the PC. This is because the model assumes that all memory elements are explicitly specified as part of the PC type definition in the hardware library. This means that there is an added dimension to the resource utilization problem. Each PC uses a fixed amount of hard resources every time an instance of it is implemented in the reconfigurable hardware. Some of these hardware resources are used for internal program and data storage. The amount of program/data storage thus in effect becomes a so called “soft” resource limit that will directly affect the number and type of tasks that the PC can execute. This is because each task in the system has assigned to it a projected “soft” resource usage requirement for each type of PC that is present in the hardware library.

TSCs are another type of functional unit that may be present in the hardware library. Unlike PCs they are not general purpose in nature but perform the specific function that is associated with the task. CCEs are the third and final type of functional units that are present in the hardware library. The model supports both synchronous (buffered) and asynchronous (non-buffered) CCEs. It is assumed that the functional units themselves utilize a common asynchronous protocol and dedicated communication ports to communicate with each other. Synchronous links between functional
units are composed of CCEs that are primarily made up of routing resources. Asynchronous communication is made possible by incorporating buffered communication elements. In this way, the interface between PCs and TSCs is uniform regardless of whether synchronous or asynchronous communication elements are used.

2.2 Task System

The second input to the RSMD model is the task system, where the application task structure and performance information is maintained as well as the soft resource requirements for each task. For this portion of the model it is assumed that the application problem has been decomposed into a set of tasks that can execute in a deterministic manner as software processes on the sequential processing units or as hardware functions within the reconﬁgurable logic. These tasks are considered to be well deﬁned in that the execution time can be determined at the time of task creation for all software and hardware manifestations. Also, all tasks are considered to be non-preemptive in nature. In this scenario, the edges in the task system contain both data and control ﬂow information which guarantee the correct system operation. In this work, it is assumed that a well deﬁned system can be unrolled into a directed acyclic graph (DAG) part and a communication part. A sample task system (major frame) with its data structure is shown in Figure 2.

2.3 High-Level Hardware System Description

Another output of the scheduler is the high-level hardware system description (HLHSD). This description indicates the number and type of processing cores, task cores, and communication core elements that are to be employed for the system and the interconnection structure that is used to interface the various functional units into a complete system. This representation can easily be translated for hardware synthesis into a structural representation of these components within a hardware description language.

2.4 Resource and Design Constraints

Another input to the RSDM are the design constraints. There are at least two types of constraints that are speciﬁed by the user. The ﬁrst is the amount of hardware resources that are available for use in the reconﬁgurable medium. This is in effect the global size constraint. The second constraint speciﬁes the level of performance that the system must possess. This is the global timing constraint -- it is in effect the maximum acceptable length of the schedule. This constraint is mandatory in real-time systems where it can be viewed as representing the global deadline associated with the major frame of the application’s task system. In general the more stringent the performance requirement (i.e. the shorter the required schedule length) or the smaller the global resource constraint, the harder it will be to create an acceptable static schedule and high-level hardware description that is capable of ﬁtting within the ﬁnite resources of the reconﬁgurable medium.

2.5 Task Schedule

One of the outputs of the scheduler is the task schedule. The task execution schedule contains the order of execution of the given tasks and the order of execution of the tasks within each PCs. The schedule length is used to determine if the implementation will meet the mandated performance requirements speciﬁed in the design constraints. During the software creation portion of the design cycle this task scheduling information is used to determine the execution order of all task modules that are implemented within each PC.

3. SCHEDULING TECHNIQUE

Due to intractable nature of this problem [2], a genetic algorithm based scheduling technique has been designed to ﬁnd a high-level hardware system description within the available resource with good schedule length.

3.1 Task and Functional Unit Representation

As part of the genetic algorithm implementation, a data structure has been designed to represent the assignment of tasks to PCs and TSCs in a hardware library. The data structure shown in Figure 3 assumes a hardware library with 3 PCs and 10 TSCs. The sample task system’s precedence matrix which deﬁnes the global precedence relationship with its data structure is shown in Figure 3.

3.2 A Valid Execution Sequence

The method to generate a legal sequence depends on task allocation to each functional unit (FU) and Precedence Matrix. The weighting scheme controls the priority ordering of task selection for scheduling. The ﬁrst task selected for processing is T4 which is allocated to PC 1 for execution. By referring to the fourth column of Precedence Matrix (Figure 3a), T4 depends on T1, T2, T3, and T5. With these known dependencies, the fourth column of precedence matrix is cross referenced with the task allocated to PC 1. Two of the preceding tasks are allocated to PC 1 which violated the precedent constraints. Thus, T4 is not eligible for scheduling. The next task selected for processing is T3. T3 depends on T1, T2, and T5 which are not allocated to PC 2. This means that T3 can be scheduled. The similar process is applied to T1, T2, T8, and T10. The next task selected for processing is T7. T7 is implemented with TSC, thus there is no local dependency and T7 can be scheduled. After that, task T9, T6, and T5 are scheduled. Since not all the tasks have been scheduled, the scheduling process continue. Task T4 is the
only task left to be scheduled. Since all the dependent tasks have been processed for execution, $T_d$ is now eligible for processing. This task execution sequence, Figure 4, is a legal sequence, but whether the high-level hardware system description is realizable is yet to be determined. For a task execution sequence to be realizable, its hard and soft resources utilization is not to be more than the amount available. To obtain the task execution schedule, the task execution sequence is inserted with task execution time and idle time to conform the precedence constraints.

Two candidates are required to produce a new candidate. In order to determine which candidates are more likely to reproduce, the likelihood value of each candidate is calculated. The likelihood value is calculated with the inverse of its fitness value divide by the sum of the multiplicative inverses of the fitness values. With the likelihood values calculated, the chromosome update is applied to generate candidates for next generation. Two techniques have been implemented: cross over and discrete gene selection. Within each update operation, two parent candidates are selected to produce a new candidate.

5. CONCLUSIONS

This paper only presents a basic overview of the genetic algorithm approach to finite resource reconfigurable hardware scheduling. A detailed presentation is beyond the scope and space limitations of this paper. Details presentation of this subject matter can be found in [2]. Extensive empirical investigations are now underway to evaluate how well this genetic algorithm approach compares to other scheduling techniques when applied to the RSDM model [2]. Other techniques that have been investigated include scheduling methods that are based upon purely random initialization of the chromosome data structure and simulated annealing. Table 1 illustrates a single case that is in many ways typical of the systems that have been investigated up to this point in the research. The system represents a single 100 task system where the ideal execution time for an infinite resource system has been calculated to be 1720 units. The resource constraints for this example task system have been set so that there will not be enough resources to implement all the tasks within TSC and none of the PCs have enough resources to execute all the system tasks. Since there are no known examples in the literature to compare the effectiveness of these techniques for the RSDM environment the best approach seems to be to compare the resulting schedule length produced by each method for a particular task/hardware Model system with the ideal and with one another. Hopefully this will form a set of reference methodologies that can be used for comparison purposes as investigations into this type of scheduling problem matures. In this case, (as in most of the cases we have tested), the genetic algorithm appears to be the most promising approach developed so far. A large number of synthetic task systems have been generated to demonstrate the robustness of the model and scheduling methods. The details can be found at [2].

6. REFERENCES