A Compilation System Development for Regular Reconfigurable Architecture using Abstract Graph Model

HyunJin Kim  Hyejeong Hong  Sungho Kang
Department of Electrical and Electronic Engineering
Yonsei University
Seoul, Korea
Hyunjin2.kim@gmail.com, hjhone@soc.yonsei.ac.kr, shkang@yonsei.ac.kr

Abstract - The compiler is the key to the success of the reconfigurable computing. Moreover, the high-level synthesis is emerging as one of the realistic development paradigm. The Reconfigurable Architecture Compiler (RAC) is developed to get the resource mapping for applications. This paper describes a development procedure using the abstracted graph model based on the Boost graph library. The details about the proposed compilation flow are fully presented. The mapping results show that complex communications more than 1-hop are not effective.

Keywords: Reconfigurable Computing, Compiler, high-level synthesis, Directed Acyclic Graph, Boost Graph Library.

1 Introduction

The reconfigurable computing has been emerging as the solution as the intermediate form between ASICs and general purpose processors. The fine-grained reconfigurable architectures such as old-fashioned FPGAs were developed as the counterpart of the ASICs, e.g., the reconfigurable test platform of ASIC prototypes. However, the great flexibility reduces the performance due to the large redundant flexible bits. Moreover, FPGAs don’t consider the mass production for applications in the field. Unlike the fine-grained reconfigurable hardware, the coarse-grained reconfigurable architectures pick out the multi-bit or word-level configurations, which reduce the hardware complexity and the configuration data. Moreover, Dehon’s technical reports asserted that the coarse-grained architectures sacrifice the flexibility of bit-level programmability with a multi-bit granularity [1], but the performance factors related to the inner delay, the area cost and the configuration time will be enhanced with the low complexity. Nowadays, the heterogeneous reconfigurable architectures are very popular in the-state-of-the-art FPGAs. For example, the FPGAs in the virtex-4 series of Xilinx contain the heterogeneous architecture with multi digital signal processing (DSP) cores. Moreover, the 256 XtremeDSP slices are contained in SX55 line-up [2].

The automatic high-level synthesis is one of the intensively studied research topics due to the increasing design complexity these days. To keep the advantage of short configuration time, the high-level synthesis is the key to the success of the reconfigurable computing. Unlike the ASIC design based on the standard cell, the reconfigurability and the multi-bit granularity are suitable for the high-level synthesis of the reconfigurable computing. Moreover, the development of the front-end compilation tools promotes the development of the specific compilation suites for the reconfigurable computing.

In this paper, the compiler for the reconfigurable computing named as Reconfigurable Architecture Compiler (RAC) is described. The RAC gets the applications as form of the directed acyclic graph (DAGs), which describe the loop body in applications. Based on the input parameter, the RAC executes the scheduling and mapping. The proposed tools use the Boost graph library due to the abstracted graph model of the application [8]. As a consequence, the RAC output the resource usage table described by the graph model. Each module in the RAC is modular type for the easiness of the tool rearrangement.

This paper is structured as follows. Section 2 shows the motivations toward the high-level synthesis for the reconfigurable computing. The compilation flow is fully described in details in Section 3. Section 4 shows the details about the mapping result analysis.

2 Motivations

The data computing parallelization on coarse-grained reconfigurable architectures has been researched toward the compilation of abstraction-level language codes. In this case, operations and communications can be modeled as the nodes and edges respectively. There are many kinds of graph models for the data computing, where one of the most famous graph models is the DAG. The data computing can be modeled using DAGs or the reiteration of DAGs. Therefore, the efficient mapping of DAGs on coarse-grained reconfigurable architectures has
been considered as the key issue to DSP applications. The
Hartenstein’s survey shows that many coarse-grained
reconfigurable architectures are relatively fixed based on
regular architecture templates, which are considered
physically acceptable due to rich communication
resources and nearest neighborhood links [3]. The
resource mapping of the extracted data flow was
researched in coarse-grained reconfigurable architecture
[4] [5] [6]. In [4], the investigation about the effectiveness
about the diverse network topologies for the mesh-based
reconfigurable architecture didn’t consider the details
about the scheduling heuristics. The genetic algorithm
was used in the resource mapping for the bundle of
events. Like the resource mapping in [5], the resource
allocation didn’t follow the time-exclusive usage for each
communication resource [5]. The topological placement
mapped the operations onto regular reconfigurable
architecture, where the liaison processing element (PE)
simplifies the resource routing in [6]. However, the
optimal solution could not be obtained due to the sparsity of
the resource mapping. Therefore, the efficient resource
usage was impossible.

The deterministic solutions for the scheduling and
the mapping are NP-Complete. The mathematical
programming methods cannot finish the scheduling and
the mapping due to the exponentially increasing
computation complexity, so that some heuristics were
proposed to get solutions in a reasonable time [7]. An
integrated solution, where the scheduling and the mapping
are not separated, causes the hard computation problem.
An optimal solution as the integrated solution needs very
large design space. Moreover, unlike the resource model
in [7], the resource-constraints in coarse-grained
reconfigurable architectures are more complex due to its
abundant routing resource. To limit the design space
related to reconfigurable architectures, the separation
between the scheduling and the mapping is considered.
Based on the previous works of the high-level synthesis
for logic, the separation can be helpful for reducing the
complexity in the scheduling and the mapping intuitively.
As a general purpose heuristic, the list scheduling is
suitable for the separation. The basic idea of list
scheduling is to make a precedence list of operations with
some priority rules, and then repeatedly execute resource
selection for each operation until a valid schedule is
obtained. The design space can be limited because
operations can be ordered with some priorities in the list
scheduling and the operations are iteratively mapped, so
that the list scheduling has been exploited in the resource
mapping of coarse-grained reconfigurable architectures
[4] [5]. In the list scheduling, the mapping is executed
depending on the priority list. The scheduling is the time
ordering which inserts the operations in a time slot on
their assigned resources. Unlike the high-level synthesis
in logic, the separation for the reconfigurable hardware is
unnatural due to the zero-sum property of the regular
architecture and the co-relation between the task and the
communication. The data flow should be mapped onto the
regular reconfigurable architecture simultaneously with
operations in a time slot, so that the integrated solution
should be implemented in the RAC.

3 Proposed Compilation System

The RAC gets an application as an abstract graph
and the information described with the extended markup
language (XML). The objective of our compilation
systems is the mapping of operations onto processing
elements and the scheduling of communications with
cycle-based accuracy in a reconfigurable architecture.
This section shows the details about the libraries used in
the RAC and the compilation flow.

3.1 Standard Template and Boost Graph Library

The RAC is programmed with C++ language. The
standard C++ suite contains various high-level tools such
as containers and algorithms. Unlike the C language, the
containers provide their own member functions. For
example, find member function in multimap container is
faster than the independent sort algorithm. Moreover,
various algorithms are useful to make complex routines.
For example, the sort algorithm is used to make the
priority list.

The Boost graph library is one of the most famous
third-party C++ libraries. Unlike other graph libraries, the
Boost graph library is based on the Standard Template
Library (STL) as much as possible. This library not only
provides the graph data structure, but also makes the
usage of the user-property maps very easy. Therefore, a
programmer can just develop his own graph model using
the STL and the user-properties. Moreover, various
algorithms related to the graph search and the property
manipulations are provided. For example, the RAC
calculates distances and slacks using the Bellman-Ford
shortest search algorithm using minus distance weights
[8]. That means the longest search algorithm can be
implemented by just using the minus distance weights.

3.2 Compilation Flow and Hardware Model

As the program files, C source codes can be
manually transformed into DAGs using SUIF font-end [9].
The DAGs can be described as dot graphical format files.
The Algorithm and the Platform XML files contain details
about algorithm parameters and general or user-defined
configuration information. After parsing both XML files,
the heuristic list scheduler analyzes DAGs and generates a
prioritized operation list. Moreover, the regular
architecture can be automatically generated using graph
model template. The XML file is very helpful for adding
the user-defined generation parameters about ad-hoc
communication links and specific processing elements.
The automatic generation of the abstract graph model is possible due to the systematic regular architecture. However, user-defined PEs or communications can be inserted depending on the user-defined parameters in Platform XML file. The hop means the normalized distance among switches, which control the communications in time slots with the configuration data. Figure 2 describes the concepts of the regular hop model. The Switch X1 is a neighborhood of the Switch X2, so that the hop is zero as it stands. The Switch X3 is not the neighborhood of the Switch X1 physically and logically. Intuitively, the latencies can be estimated using the hop model. Based on the pipelined communication model, the communication latency between neighboring PEs can be \( n \) cycles if the \( n-1 \) repeaters are inserted in a communication channel. The hop models can represent the physical separation. Therefore, it is a valid assumption that the latency of the 1-hop communication channels is one cycle greater than that of the 0-hop communication channels. The regular communication model can be described as the specific linear equations.

<table>
<thead>
<tr>
<th>1-hop, ( n+1 ) cycles</th>
<th>0-hop, ( n ) cycles</th>
<th>1-hop, ( n ) cycles</th>
<th>0-hop, ( n ) cycles</th>
</tr>
</thead>
</table>

Figure 2. Proposed Communication Model

Before mentioning the algorithm in the proposed RAC, the assumed properties in the abstract graph model are briefly described as follows:

- The latencies in PEs are normally pre-fixed in natural numbers.
- The proposed mapping is just static and non-pre-emptive, so that the configuration data should configure each PE and switch in a time slot.
- The specific operations should be just described with the operation types and the latencies.
- For the inputs and the outputs, each PE can implicitly access the global memory.

Based on the hardware model, the algorithms in the proposed tool are described in Section 4.

4 Scheduling and Mapping

The parameters for the algorithm are provided with the Algorithm XML file. After parsing and analyzing the Algorithm XML file, the options for the scheduling and the mapping are set up. The algorithm is divided into two parts: Heuristic List Scheduler and Placement & Routing.

4.1 Heuristic List Scheduler

As the first part of the list scheduling, the Heuristic List Scheduler makes the priority list based on the input parameters. The applications are provided as the DAGs. The information extraction from the DAGs is the key to make the priority list. The first thing is grouping the operations according to the data dependency. The strongly connected components can be indexed with the Tarjan algorithm using the undirected graph model [8]. Therefore, the DAGs are modified into undirected graphs, which can be the input for the Tarjan algorithm [8]. The independent operations are prioritized with the distances from the virtual sink source [7]. This means that our proposed algorithm follows the ALAP rule as a priority rule. The bundles of algorithms in the Boost graph library are used to get the distances. The sorting and the search algorithm in STL are also used to order the operations. However, the libraries in the RAC are somewhat modular, so that the combinations of the containers and routines are adaptable for various algorithms with many kinds of parameters. The RAC starts mapping the operations in an operation group after finishing mapping operations in other prioritized groups.

4.2 Placement & Routing

Based on the priority list, the mapper executes the placement and routing for the operations and the communications. The mapper adopts the available operations in a time slot considering the communications among operations. The available mapping results can be calculated using the communication path finder routine. First, the path finder routine searches the cases of the suitable resource usage for an input edge, where a case of the resource usage contains the information about a PE and communications for an input edge. Based on the
information, the path finder routine searches the solution for another input edge iteratively. Finally, the suitable results for the operation mapping can be calculated. The mapping results are implicitly sorted in the associative container. For the simplicity of computations, only the first result is selected with the predicate used in sorting for the container. The simple edge index is used in the container for the temporary mapping result. The mapped operations and the communications are removed in the priority list. Iteratively, the mapper executes the placement and routing until the priority list is empty. If the priority list cannot be mapped in the target hardware, the priority list is divided into sub-lists keeping the data dependency. The buffers of the PEs in a previous sub-list maintain the data, so that the data dependencies are not broken.

5 Mapping Result Analysis

The proposed system provides the resource usage table of PEs and the communications in the regular reconfigurable architecture. The resource usage tables contain the index and the related time table. To prove the efficiency of the RAC, some benchmarks from the ExpressDFG system are tested [10]. The results in Table 1 show that the complex communications with 2-hop and 2-delay communication resource are not effective on 3x3 or 4x4 regular reconfigurable architectures. The execution time is under 10 seconds including the logging data extraction. The proposed RAC needs great execution time for large array due to the factorial increase of the design space.

Table 1. Benchmark Mapping Results

<table>
<thead>
<tr>
<th>Name</th>
<th>array (6x9)</th>
<th>hop/delay</th>
<th>#Cycles</th>
<th>#Comm.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL (3x3)</td>
<td>0/1</td>
<td>10</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0/1,1/1</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0/1,1/1,2/2</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>horner bezier</td>
<td>0/1</td>
<td>19</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>(3x3)</td>
<td>0/1,1/1</td>
<td>17</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0/1,1/1,2/2</td>
<td>17</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>ARF* (3x3)</td>
<td>0/1</td>
<td>36</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0/1,1/1</td>
<td>33</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0/1,1/1,2/2</td>
<td>33</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>motion vector</td>
<td>0/1</td>
<td>29</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>(4x4)</td>
<td>0/1,1/1</td>
<td>26</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0/1,1/1,2/2</td>
<td>26</td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>

The mapping of fir1 DAG on the 6x6 array needs about 10 minutes. The total mapping time on the 8x8 cannot be estimated. However, most regular reconfigurable architectures adopt hierarchical communications shown in [3]. Therefore, the size of the regular reconfigurable hardware limited to the 6x6 can be sufficient for the RAC.

6 Conclusion

In this paper, a new automatic compilation flow and the development procedure are describes in detail. The RAC adopts the list scheduling for integrating the scheduling with the mapping. The mapping results shows the analysis examples, where the complex communications are not necessary. The compilation system reflects the properties of commercial reconfigurable hardware. The analysis of the mapping results shows that the complex communications are not helpful for the latency or the communication resource usage. Therefore, the RAC provides the automatic high-level design flow for the reconfigurable hardware.

References


