Mapping Applications to an FPFA Tile

Michèl A.J. Rosien, Yuanqing Guo, Gerard J.M. Smit, Thijs Krol
Department of Computer Science,
University of Twente, Enschede, the Netherlands
email: {rosien, yguo, smit, krol}@cs.utwente.nl

Abstract— **This paper introduces a transformational design method which can be used to map code written in a high level source language, like C, to a coarse grain reconfigurable architecture. The source code is first translated into a Control Dataflow graph (CDFG), which is minimized using a set of behaviour preserving transformations such as dependency analysis, common subexpression elimination, etc. After applying graph clustering, scheduling and allocation transformations on this minimized graph, it can be mapped onto the target architecture.**

I. Introduction

In the CHAMELEON/GECKO project we are designing a heterogeneous reconfigurable System-On-a-Chip (SOC). This SOC contains a general-purpose processor (ARM core), a bit-level reconfigurable part (FPGA) and several word-level reconfigurable parts Field Programmable Function Array (FPFA) tiles; (see Section II). The objective of this paper is to show that a transformational design method can be used to map processes, written in a high level language (C/C++), to an FPFA tile.

II. The target architecture: FPFA

Each word-level reconfigurable part of our architecture, the FPFA processor tile (see [3]), consists of five identical Processing Parts (PPs), which share a control unit (see Fig. 1). An individual PP contains an arithmetic and logic unit

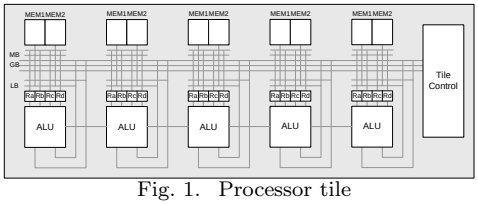


Fig. 1. Processor tile

(ALU), four input register banks named Ra, Rb, Rc, Rd and two memories called MEM1 and MEM2. Each register bank consists of four registers. Each memory has 512 entries. A crossbar-switch makes flexible routing between the ALUs, registers and memories possible. The crossbar enables an ALU to write back their result to any register or memory within a tile.

III. Definition of a CDFG

In our toolset the input language is first translated into a Control Data Flow Graph(CDFG). A CDFG is a graph that represents the operations (for example C operators and function calls) and the dataflow between those operations. This data includes operands of mathematical operations, the statespace (see section IV) and control information which is used, for example, to control MUXes which in turn control the iteration and selection statements. Examples of such statements are: the **if** statement and the **while** statement.

IV. Representation of C memory model

To be able to map a C program to a CDFG it is necessary to map the linear, random access memory model of C to the hypergraph model. A mathematical abstraction of this memory model, called *the statespace*, is introduced in [1]. The statespace is a set of tuples: $\{(ad, da), (ad, da), ...\}$. A tuple consists of an *ad* field, which represents the address, and a *da* field which represents the data at that address. This data can be anything, including a tuple of this type again. Interaction with the statespace is done using three primitive hypergraphs. These primitive hypergraphs allow writing data to the statespace and reading or deleting data from the statespace.

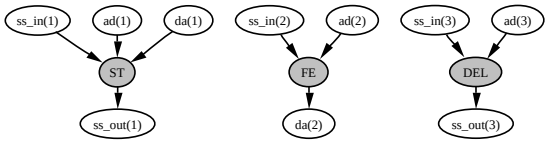


Fig. 2. Primitive operations on the statespace.

Fig. 2 shows the three primitive operations on the statespace.

- **ST**: "Store", Stores a tuple on the statespace.
- **FE**: "fetch", Reads a tuple from the statespace.
- **DEL**: "Delete", Deletes a tuple from the statespace.

V. Example of a generated CDFG

This section will show a CDFG generated from a piece of FIR filter code. The source code is shown below. The code consists mainly of a simple while loop which adds and multiplies values from two arrays.

```
void main() {
    sum = 0; i = 0;
    while (i < 5) {
        sum = sum + a[i] * c[i]; i = i + 1;
    }
}
```

VI. Mapping and Scheduling of Directed Acyclic Graphs on An FPFA Tile

We use a three phase decomposition algorithm based on the two-phased decomposition of multiprocessor scheduling introduced by Sarkar [4]: (1) Task clustering and ALU data-path mapping; (2) Scheduling the clusters on the 5 physical ALUs of an FPFA tile; (3) Resource allocation.

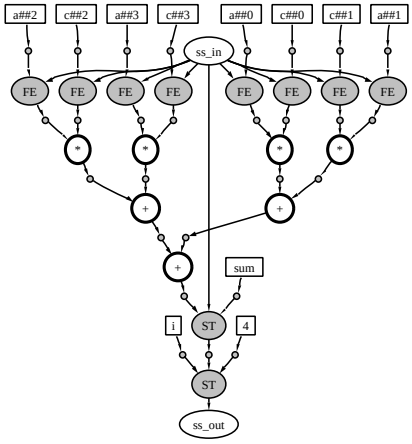


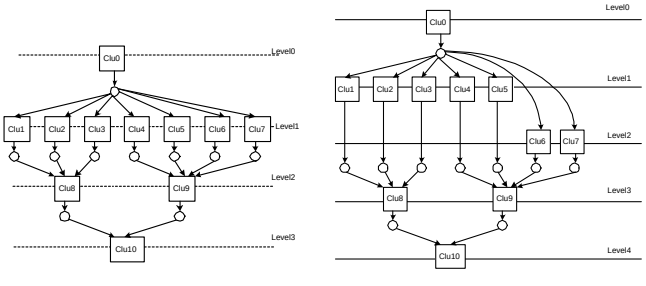
Fig. 3. Translation of the FIR filter code. After complete loop unrolling and full simplification.

A. Clustering and data-path mapping

In the clustering phase the task graph is partitioned and mapped to an unbounded number of fully connected ALUs, which can perform inter-ALU communication simultaneously. This clustering and mapping scheme is based on the ALU data-path of our FPFA.

B. Scheduling of clusters

In the scheduling phase, the graph obtained from the clustering phase is scheduled according to the maximum number of ALUs (in our case 5). This means that at most 5 clusters can be on the same level. In a clustered graph, the longest path is referred to as *critical path*. All nodes on the critical path have an incremental level number. The clusters that do not belong to any critical path can be moved up and down within the range where the dependence relations among the tasks are satisfied (see Fig. 4). Here we adopt a heuristic procedure in which the clusters are scheduled level by level. The complexity is thus linear to the number of clusters.



(a) Before scheduling (b) After scheduling

Fig. 4. Insert a new level when necessary

C. Resource allocation

In the allocation phase, the scheduled graph is mapped to the resources where locality of reference is exploited, which is important for performance and energy reasons. The main challenge in this phase is the limitation of the size of register banks and memories, the number of buses of the crossbar and the number of reading and writing ports of memories and register banks. We adopt a heuristic resource allocation method, whose pseudocode is listed in Fig. 5. Please check [5] for details.

```
//Input: Scheduled Clustered Graph G
//Output: The job of an FPFA tile for each clock cycle
function ResourseAllocation(G) {
      for each level in G do  Allocate(level);
}
function Allocate(currentLevel) {
    Allocate ALUs of the current clock cycle
    for each output  do store it to a memory;
    for each input of current level
    do  try to move it to proper register at the clock cycle which is four steps
          before; If failed,  do it three steps before; then two steps before;  one
          step before.
    if  some inputs are not moved successfully
    then insert one or more clock cycles before the current one to load inputs
}
```

Fig. 5. Pseudocode of the heuristic allocation algorithm

The clusters in the scheduled graph are allocated level by level. The computational complexity of this allocation method is also linear to the number of clusters.

VII. Conclusion and Future Work

In this paper we presented a transformational method to map a process written in a high level language, such as C, to one FPFA tile. The mapping procedure is divided into four steps: translating the source code to a CDFG, clustering, scheduling and resource allocation. High performance and low power consumption are achieved by exploiting maximum parallelism and locality of reference respectively. In conclusion, using this mapping scheme, the potential advantages of FPFA are exploited. Several issues need to be addressed in the future: Existing graph transformations need to be optimized and more transformations will be added; The clustering algorithm will be investigated; Loops and branches should be included in the clustering, scheduling and resource allocation phase.

Acknowledgements

This research is conducted within the CHAMELEON project (TES.5004) and GECKO project (612.064.103) supported by the PROGram for Research on Embedded Systems & Software (PROGRESS) of the Dutch organization for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the technology foundation STW.

References

[1] Thijs Krol and Bert-Steffen Visser: "High-level Synthesis based on Transformational Design", *Internal report, University of Twente*, Enschede, The Netherlands.

[2] Paul M. Heysters, Henri Bouma, Jaap Smit, Gerard J.M. Smit, Paul J.M. Havinga: "A Reconfigurable function array architecture for 3G and 4G wireless terminals" In press: *2002 International Conference On Third Generation Wireless and Beyond*, May 2002.

[3] G.J.M. Smit, P.J.M. Havinga, L.T. Smit, P.M. Heysters, M.A.J. Rosien, "Dynamic Reconfiguration in Mobile Systems", *Proceeding FPL2002* Montpellier France, pp 171-181, September 2002.

[4] Vivek Sarkar. *Clustering and Scheduling Parallel Programs for Multiprocessors*. Research Monographs in Parallel and Distributed Computing. MIT Press, Cambridge, Massachusetts, 1989.

[5] Yuanqing Guo, Gerard Smit, "Mapping and Scheduling of Directed Acyclic Graphs on An FPFA Tile", *Proceedings PROGRESS 2002 workshop*, Utrecht, the Netherlands, 2002.