# Reducing the dynamic FPGA reconfiguration overhead with loop transformations

Tom Degryse[*,1], Karel Bruneel[*,1],
Harald Devos[*,1], Dirk Stroobandt[*,1]

[*] *ELIS, Ghent University, Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium*

**ABSTRACT**

**Dynamic hardware generation reduces the number of FPGA resources needed and speeds up the application by optimizing the configuration for the exact problem at hand at run-time. If the problem changes, the system needs to be reconfigured. When this occurs too often, the total reconfiguration overhead is too high and the benefit of using dynamic hardware generation vanishes. Hence, it is important to minimize the number of reconfigurations.**

**We propose a novell technique to reduce the number of reconfigurations by using loop transformations. Our approach is similar to the temporal data locality optimizations. By applying our technique, we can drastically reduce the number of reconfigurations, as indicated by the matrix-vector multiplication example. After applying the loop transformations, the number of reconfigurations decreases by an order of magnitude. This speeds up the application and improves the usefulness of dynamic hardware generation techniques.**

KEYWORDS: FPGA; dynamic hardware generation; loop transformations

## 1 Introduction

One of the major advantages of FPGAs (Field Programmable Gate Arrays) is their reconfigurability. However, the majority of todays FPGA-based systems do not exploit this benefit or only exploit it on a very large time scale. In a traditional system, the FPGA loads its configuration bits from a local memory when the system starts up. Once the system has booted, the functionality of the FPGA remains the same during the entire run time of the system. The FPGA is only reconfigured after a firmware upgrade, but this situation occurs very infrequently.

[1]E-mail: {Tom.Degryse, Karel.Bruneel, Harald.Devos, Dirk.Stroobandt}@elis.UGent.be

Dynamic hardware generation uses this reconfigurability on a much shorter time scale by exploiting run-time knowledge of the exact problem at hand. At run-time, a specialized hardware circuit is generated, which is substantially smaller and faster than the generic counterpart. If the problem at hand changes, the dynamic hardware generation tool creates a new specialized FPGA configuration and reconfigures the FPGA. When the problem at hand does not change too often, the total execution time, including the reconfiguration overhead, is less than the execution time of the generic circuit.

In this work, we make use of so-called parameterizable configurations [BS08]. These are FPGA configurations, generated off-line, in which some of the configuration bits are expressed as closed form Boolean functions, the tuning functions, of some of the inputs, called the parameters. On-line specialization of a parameterizable configuration means evaluating these tuning functions. This technique has a substantially lower on-line hardware generation overhead than conventional hardware generation methods, which makes it much more useful in practice.

Because of the overhead of the hardware generation and reconfiguration, it is important to minimize the number of reconfiguration steps during program execution. This can be done by maximizing the reuse of the input parameter values, which is very similar to optimization of the temporal data locality in a system with a memory hierarchy. The application of loop transformations is one well known technique to optimize the data locality, both in hardware and software. In the next section, we will investigate how the temporal data locality optimizations can be used to minimize the number of reconfigurations.

Section 3 continues with the matrix-vector multiplication example we used to evaluate our technique. This example shows that our technique drastically reduces the number of reconfigurations and hence reduces the total execution time significantly. This further improves the usefulness of dynamic hardware generation techniques. We end this paper with some concluding remarks in section 4.

## 2 Minimizing the number of reconfigurations

Our target, minimizing the number of reconfigurations or equivalantly, maximizing the hardware reuse, is very similar to maximizing the reuse of data elements in a local memory. Hence, the same techniques that are used to optimize the temporal data locality, can be used to reduce the number of reconfigurations. Loop transformations are a well known set of transformations to optimize both spatial and temporal data locality and introduce parallelism [GVB+06].

However, there are some differences between the temporal data locality optimization and the minimization of the number of reconfigurations. First of all, there is no configuration cache in an FPGA. So, if we want to reuse a certain parameter value, the different usages of this value should be placed right after eachother in time, or elsewise stated, the reuse distance must be equal to one. This situation could be improved by using an FPGA architecture with multiple configuration memories.

Next to this, there is also a different cost function associated with a miss. First of all, after a parameter has changed, a new configuration must be generated. The cost of this operation depends on the number of boolean functions that must be evaluated and on the complexity of the boolean functions. Both aspects can be different for every parametric input, but are known after the parameterizable circuit has been generated off-line.

**for** $i = 0 \ldots N$
  **for** $j = 0 \ldots N$ by $B$
    $reg_0 = A(i,j) \times x(j)$
    $reg_1 = A(i,j+1) \times x(j+1)$
    $\ldots$
    $reg_{B-1} = A(i,j+B-1) \times x(j+B-1)$
    $y(i) = y(i) + reg_0 + \ldots + reg_{B-1}$

(a)

**for** $j = 0 \ldots N$ by $B$
  **for** $i = 0 \ldots N$
    $reg_0 = A(i,j) \times x(j)$
    $reg_1 = A(i,j+1) \times x(j+1)$
    $\ldots$
    $reg_{B-1} = A(i,j+B-1) \times x(j+B-1)$
    $y(i) = y(i) + reg_0 + \ldots + reg_{B-1}$

(b)

Figure 1: Matrix-vector multiplication example $A \times x = y$ : parallel code (a) and the optimized version (b). Subvector $x[j \ldots j+B-1]$ is the parametric input.

After the new configuration has been generated, the FPGA must be reconfigured. There is a fixed cost that comes with every reconfiguration, due to for instance the overhead to stop and restart the calculation. The cost to transfer the configuration bits to the FPGA depends heavily on the FPGA architecture and on the available bandwidth to the configuration memory. In an FPGA with a RAM-like configuration memory, where every configuration bit can be addressed individually, the configuration cost is equal to the total number of bits divided by the configuration memory bandwidth.

Current Xilinx FPGAs only support partial reconfiguration, where the finest level of granularity is the frame. To calculate the reconfiguration overhead, we must count the number of frames that change. At the time of writing, this work is not yet fully completed. Altera does not support partial reconfiguration at the moment, so every time the FPGA needs to be reconfigured, the whole bitfile must be transfered to the FPGA. In this case, the reconfiguration overhead is fixed.

In our approach, we use the polyhedral model [GVB+06] as an intermediate program representation, because this facilitates loop transformations. Within this model, loop transformations are executed by performing simple matrix operations. In the future, we plan to integrate our novell cost model with the polyhedral program representation, so we can automatically search for an optimal sequence of loop transformations.

# 3 Example

To illustrate our proposed approach, we use the matrix-vector multiplication as an example. Matrix operations serve as basic building blocks for many numerical linear algebra applications that are heavily used in scientific computing, including the solution of linear systems of equations, linear least square problems, eigenvalue problems and singular value problems [PTVF07]. Speeding up these operations thus accelerates a large class of algorithms found in scientific computing.

The matrix-vector multiplication $A \times x = y$ consists of a simple two-dimensinal loop. To parallelize the algorithm, the inner loop is partially unrolled by a factor B, which is determined by the size of the FPGA or by the available memory bandwidth. Fig. 1(a) shows the parallelized source code of the algorithm. All B multiplications can be done in parallel. We have transformed this parallel matrix-vector multiplication into a parameterizable configuration using TMAP [BS08], with input vector $x$ as our parametric input. As a result our

prototype becomes more than $50\%$ smaller ($896$ instead of $1920$ LUTs for an unroll factor $B = 16$ and an input bit width of 8). However, the FPGA needs to be reconfigured after every iteration of the $j$-loop, which results in $N \times \frac{N}{B}$ reconfigurations.

If we interchange both $i$ and $j$-loops, the same values of $x$ are used and hence the same configuration can be reused for all iterations of the $i$-loop, as indicated in Fig. 1(b). Due to this simple loop optimization, the number of reconfigurations is reduced to $\frac{N}{B}$, an improvement by a factor $N$ compared to the code in Fig. 1(a). The number of bits that change and thus the overhead of one reconfiguration is the same for both solutions.

However, there is also a drawback associated with this optimization. In Fig. 1(a), we only need to store one element of the resulting array $y$, while in Fig. 1(b) the whole array must to be stored in on-chip memory or the number of accesses to the external memory increases drastically. To reduce the memory footprint, the $i$-loop could be tiled, resulting in an increased number of reconfigurations. In general, we can say that there will be a trade off between parallelism, data locality and the number of reconfigurations.

# 4 Conclusions and future work

We have presented a technique to reduce the number of hardware generation and FPGA reconfiguration steps in a dynamic hardware generation environment by means of loop transformations. Our technique is very similar to temporal data locality optimization. The matrix-vector multiplication example shows that the number of reconfigurations can drastically be reduced by transforming the loop nests in a program. This speeds up the application and thus further improves the usefulness of dynamic hardware generation techniques.

This work is far from finished, so there is plenty of room for further improvements. In the future, we will further elaborate on our cost model to adapt it to a real-world Xilinx FPGA. Furthermore, we will combine our cost model with the polyhedral program representation to automatically select a good sequence of loop transformations. Finally, we will investigate the trade off between the increased memory usage and the reduction of the reconfiguration overhead.

# References


[BS08] Karel Bruneel and Dirk Stroobandt. Automatic generation of run-time parameterizable configurations. *FPL*, accepted for publication, 2008.

[GVB+06] Sylvain Girbal, Nicolas Vasilache, Cédric Bastoul, Albert Cohen, David Parello, Marc Sigler, and Olivier Temam. Semi-automatic composition of loop transformations for deep parallelism and memory hierarchies. *International Journal of Parallel Programming*, 34(3):261–317, 2006.

[PTVF07] William H Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 2007.