Packet Classification with Evolvable Hardware Hash Functions

Harald Widiger, Mathias Handy, Dirk Timmermann
University of Rostock, Institute of Applied Microelectronics and Computer Science
{harald.widiger; mathias.handy; dirk.timmermann@e-technik.uni-rostock.de}

1. Introduction

Bandwidth demands of communication networks are rising permanently. Thus, packet classification is a severe problem in modern communication technology. IP lookups in routers i.e. need to be as fast as possible while the hardware resources are limited. Conventional algorithms as presented in [1] must make a tradeoff between classification speed and memory demands. By the use of a hash function both demands can be met. Ideally, hash functions have a search complexity of O(1). Thus, they are independent of the number of elements searched in. With O(N) the memory need scales only linearly with the number of elements. However, it is problematic to find a sufficient hash function. It has to be both high performance and of low hardware costs. It might be easy to find a good hash function for a specific amount of elements out of a huge search space. But because of changing key sets in packet classifiers a hash function, that used to be sufficient, might be insufficient for a modified key set. A solution for this problem is a permanently adapting and improving hash function. This goal can be obtained by evolutionary computing completely done in hardware. Such an evolvable hardware hash function is proposed here.

2. Evolvable Hardware Hash Function

An evolvable hardware hash function is to be developed. It shall evolve autonomously and be implemented in hardware only. Thus, a complete hardware evolution comes to pass. This is realized by constantly traversing an evolution pipeline comparable with the one in [2]. At the moment the system is implemented as a SystemC software model. A linear collision resolution for the hash functions is used. However, the target architectures are next generation FPGAs.
Different hardware architectures of hash functions are explored to determine their potential. A promising architecture which is high performance and cheap in hardware costs is shown in Figure 1. It consists of a number of multiplexer elements. The multiplexers are controlled by registers. Those registers form the genome of the hash function. For every output bit two multiplexer outputs are connected via an xor function.

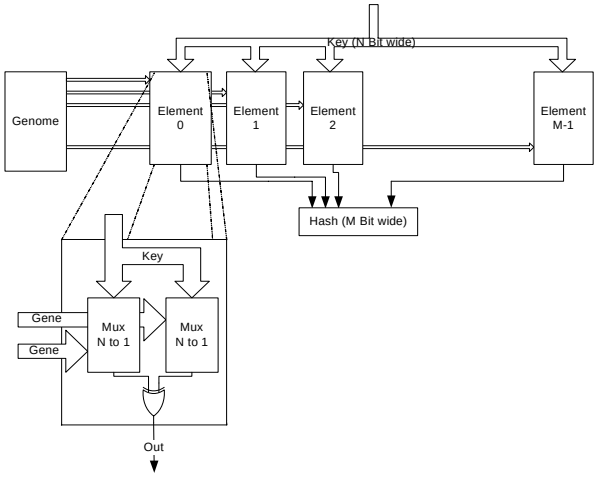

Figure 1. Architecture of an evolvable hash function

Another architecture we consider is presented in [3], where all input signals are connected to a row of 4-input look up tables (LUTs). The LUT outputs are connected to the next stage of LUTs by a connection matrix. The matrix and the logic functions in the LUTs are register controlled. The registers form the genome here as well.

3. Early Simulation Results

The evolvable system comprises the complete evolutionary algorithm. It was implemented as a SystemC model. The model is functional identical to the targeted VHDL implementation. The hash functions were evaluated with different key sets of up to hundred thousand 32 bit wide randomly generated keys. The memory load of the hash memory was set to different levels ranging from 37% to 87%. All hash functions evolved over thousand generations.

In Figure 2, the graphs reveal, that the architectures performed differently. The hash functions build of multiplexers (Hash 1 and Hash 2) showed great

performance.

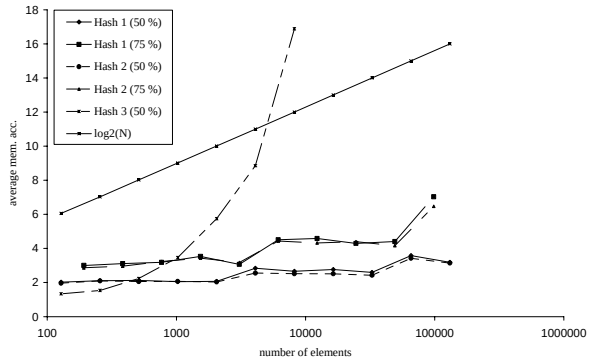

Figure 2. Lookup performance of different hash architectures

The average of required memory accesses over all keys scales only very slightly and remains below five even with 217 keys. Five accesses outperform the 16 memory accesses of a sorted list remarkably. The architecture consisting of LUT rows (Hash 3) showed poorer performance. It scales almost linearly with the size of the key sets. Thus, the architecture is not applicable for implementation in a packet classifier.

However, it has to be mentioned that the theoretical upper bound is still O(N). In the worst case the required memory accesses for some keys are far more than $\log_2(N)$. This is the case at least for the memories with a load of 75% and more. That is why a memory load of 50% should not be exceeded.

The memory demand of such architecture has a complexity of O(N). As the hash function evolves constantly, repetitive rehashing of the keys in the memory is required. To do so there have to be two memory blocks of which one is used in the data path while the other one is rehashed. That means if the memory load is limited to 50%. Exactly 4*N memory locations are required to store keys and classification rules.

4. Intended Hardware Architecture

The evolutionary algorithm especially the computation of the fitness of a hash function is extremely demanding regarding the computation time. During simulation four individuals, which produced an offspring of twelve, were evolved. To evolve 1000 generations with 100.000 keys a computation time of more than a day was needed on a 3.2 GHz machine. That is why we intend to implement the whole system, consisting of the data path and the evolution module on an FPGA. The most time consuming and thus performance critical module is the element evaluating the fitness. To evaluate the fitness of an individual, the module hashes all keys and counts the number of collisions. This process has a complexity of $O(N^2)$. If the initial hash function hashes all keys to the same value, $N^2/2$ collisions can occur. This is the worst case. But as the initial genome of the hash function is always chosen randomly, its quality is always better. Simulations showed that the initial hash functions produce at worst 20 million collisions for 2^{17} keys. Thus, for the above example of 100.000 keys and an offspring of twelve, 240 million clock cycles are needed. On a 125 MHz FPGA the first generation would evolve in two seconds. The evolution rate increases rapidly with the hash function getting fitter. This is without any optimizations. One is the parallel implementation of one fitness evaluation element per offspring.

5. Conclusion and Outlook

The simulations of the model showed that a packet classifier consisting of an evolvable hash function can be very effective. The time complexity is roughly O(1) and the memory demand is O(N), even for very large rule sets. The actual used hash function is always designed for the momentary rule set by the hardware evolution. Evolving constantly, the hash function improves over time and adapts to changes in the rule set. These are excellent characteristics. But the problem of numerous memory accesses for some keys is still to be solved.

A final evolving system will be implemented in a dynamically reconfigurable environment as mentioned in [4]. In such a system the hash functions would not need to consist of register controlled multiplexers. Instead there are just wires from input to output and some combinatorial logic. The wires are simply rerouted to evolve to a new generation. This is achieved by the FPGAs partial reconfigurability.

6. References

[1] P. Gupta, N. McKweon, “Algorithms for Packet Classification”, Proc. of IEEE Network, 2001, pp 24-32.

[2] G. Tufte, P. C. Haddow, “Prototyping a GA Pipeline for Complete Hardware Evolution”, Proc. of EH, 1999, pp 143-150.

[3] E. Damiani, A. G. B. Tettamanzi, “On-Line Evolution of FPGA-Bases Circuits: A Case Study on Hash Functions”, Proc. of EH, 1999, pp 36-33.

[4] S. Kubisch, R. Hecht, D. Timmermann, “Design Flow on a Chip – An Evolvable HW/SW Platform”, Proc. of the 2nd IEEE ICAC, 2005, to be published.

[5] A. Broder, M. Mitzenmacher, “Using Multiple Hash Functions to Improve IP Lookups”, Proc. of IEEE Infocom, 2001, pp 1454-1463.