

A SCALABLE PIPELINE ARCHITECTURE FOR LINE RATE PACKET CLASSIFICATION ON FPGAS

Jeffrey M. Wagner *

Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089, USA
email: jmwagner6@live.com

Weirong Jiang and Viktor K. Prasanna

Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089, USA
email: {weirongj, prasanna}@usc.edu

ABSTRACT

Multi-dimensional packet classification is a key function for networking applications in high-speed routers. Although a multitude of research has explored this area, efficient packet classification that supports large rule sets at line rate remains challenging. This paper presents a scalable pipeline architecture, named BiConOLP, for line rate packet classification on FPGAs. We study the problem of balancing memory distribution across pipeline stages while keeping overall resource usage low using a multitude of pre and post mapping waste elimination techniques, algorithm optimizations, and customized hardware implementations. Our experimental results show that our architecture can store 10K unique, 5-field rules in a single Xilinx Virtex-5 FPGA. Our architecture can also sustain above 45 Gbps throughput for minimum size (40 bytes) packets with only 45.6% memory resource usage, making our design competitive with many state-of-the-art FPGA-based packet classification engines. To the best of our knowledge, our work is the first to achieve perfectly balanced memory distribution (to within 10% overall) over both decision tree and rule list pipeline stages and support large rule sets at line rate with efficient resource utilization.

KEY WORDS

FPGA, Network Performance, Reconfigurable Architecture, Packet Classification, Pipeline, SRAM

1 Introduction

Next-generation high-speed routers are critical to the future development of networking systems and the Internet. In order for such routers to keep up with existing state-of-the-art technologies, they must support a variety of network applications, such as virtual private networking, firewall processing, policy routing, traffic billing, Quality of Service (QoS) differentiation, and other value added services. The common task shared between these applications is *multi-dimensional packet classification*, which categorizes multiple field values within the packet header using a set of predefined rules. However, due to the rapid growth of the rule set size and the current link rate having been pushed

beyond the OC-768 rate, i.e. 40 Gbps, multi-dimensional packet classification has consequently become one of the fundamental challenges of designing high-speed routers. Additionally, such throughput is impossible using existing software-based solutions [5].

Over the past several years, the packet classification functionality has been developed primarily using two technologies: ternary content addressable memories (TCAMs) [11, 16, 18], and static random addressable memories (SRAMs) [7, 8, 14]. TCAMs have significant disadvantages in terms of clock rate, power consumption, and circuit area, compared to SRAMs [7]. Conversely, mapping decision-tree-based packet classification algorithms [4, 15] onto SRAM-based pipeline architectures appears to be a promising alternative [1]. This allows high throughput to be achieved, in addition to increased flexibility and low cost by using SRAM-based FPGA solutions.

Decision-tree-based packet classification can be implemented using tree traversal and rule lookup / comparison. Each packet traverses a search tree in the memory, and retrieves a matched entry when it arrives at a tree leaf. The decision tree can be constructed such that a matched entry identifies a finite set of rules. Searching these rule sets requires multiple memory accesses, which can be pipelined to achieve high throughput. A simple pipelining approach is to map each level of a decision tree onto a separate stage, thus allowing one packet to be processed every clock cycle. However, this approach results in high resource usage, excessive memory wastage, and unbalanced memory distribution over the pipeline stages, which all significantly impact the ability to support large rule sets on-chip.

Although more sophisticated pipelining approaches can combat some of these issues, unbalanced memory distribution still remains dominant for all pipelined architectures [2]. In an unbalanced pipeline, the “fattest” stage, which stores the largest number of tree nodes / rules, becomes a bottleneck thus adversely affecting overall pipeline performance in the following ways:

1. More time is needed to access the pipeline stages containing larger local memory, leading to a reduction in the global clock rate.
2. Since it is unclear at design time which pipeline memory stage will be the fattest, we need to allocate mem-

*The author is now employed with Northrop Grumman Corporation.

ory with the maximum size for each stage. Such over-provisioning results in memory wastage [1] and excessive power consumption.

To address the above challenges, we propose a scalable pipeline architecture, named BiConOLP, for line rate packet classification. We make following contributions:

- To the best of our knowledge, this work is the first one to achieve perfectly balanced memory distribution (to within 10% overall) over both decision tree and rule list pipeline stages for multi-dimensional packet classification.
- Two self-optimizing, fine-grained mapping schemes are proposed to realize the above goal. We introduce the notion of inversion factor and the largest-leaf heuristic to invert subtrees for decision tree memory balancing, and customization of memory port and width configurations to optimize rule packing for rule list memory balancing.
- A novel bidirectional-configurable optimized linear pipeline architecture (BiConOLP) using dual-port high-speed Block RAMs are presented to enable the above mappings. The architecture balances memory distribution and provides system-wide optimization and memory waste reduction while achieving one packet per clock throughput.
- Implementation results show that our architecture can store 10K unique, 5-field rules in a single Xilinx Virtex-5 FPGA with only 45.6% memory resource utilization. Our architecture can also sustain above 45 Gbps throughput for minimum size (40 bytes) packets, which exceeds the OC-768 rate. To the best of our knowledge, our design is among the first FPGA implementations able to perform multi-dimensional packet classification at line rate while supporting a large rule set with abundant additional resources on-chip.

The remainder of this paper is organized as follows. Section 2 reviews background and related works. Section 3 discusses the memory balancing and waste reduction over pipeline stages and proposes two self-optimizing, fine-grained mapping schemes. Section 4 proposes a corresponding bidirectional-configurable optimized linear pipeline architecture. Section 5 analyzes experimental results to evaluate the performance of our approaches. Section 6 concludes the paper.

2 Background

2.1 Packet classification overview

An IP packet is usually classified based on the five fields in the packet header: the 32-bit source/destination IP addresses (denoted **SA/DA**), 16-bit source/destination port numbers (denoted **SP/DP**), and 8-bit transport layer protocol. Individual entries for classifying a packet are called *rules*. Each rule can have one or more fields, a priority,

and an action to be taken if matched. Different fields in a rule require different kinds of matches: prefix match for SA/DA, range match for SP/DP, and exact match for the protocol field. A packet is considered matching a rule only if it matches all the fields within that rule. When a packet matches a rule, an action is assigned to that packet. Additionally, a packet can match multiple rules, but only the rule with the highest priority is used to take action.

2.2 FPGA related work

The FPGA has become an attractive option for implementing real-time network processing engines. Although multi-dimensional packet classification is a saturated area of research, little work has been done using FPGAs.

Lakshman et al. [10] propose Parallel Bit Vector (BV) decomposition-based algorithm. The algorithm performs parallel lookups on individual fields and uses returned bit vectors to check for a match. This approach provides high throughput at the cost of low memory efficiency, with the memory requirement being $O(N^2)$ where N is the number of rules. An expansion to this concept by Song et al. [16] adds TCAMs to the BV algorithm for multi-match packet classification. This approach utilizes a TCAM to perform prefix or exact match and a multi-bit trie implemented in Tree Bitmap [3] for source or destination port lookup. The authors do not report actual FPGA implementation results, however they predict their design can achieve 10 Gbps after pipelining on advanced FPGAs, using less than 10% of the available logic and fewer than 20% of the available Block RAMs of a Xilinx XCV2000E FPGA for 222 rules.

Two recent works [7, 9] discuss several issues on implementing decision-tree-based packet classification algorithms on FPGAs, with different motivations. Kennedy et al. [9] implement a simplified HyperCuts algorithm on an Altera Cyclone 3 FPGA that stores hundreds of rules in each leaf node and matches them in parallel. Although they claim to achieve power efficiency, this approach results in low clock frequency (i.e. 32 MHz reported [9]). Jiang et al. [7] propose a two-dimensional linear dual-pipeline architecture that supports on-the-fly update. Although they claim to store 10K real-life rules in on-chip memory of a single Xilinx Virtex-5 FPGA while sustaining 80 Gbps throughput for minimum size (40 bytes) packets [7], their memory resource usage almost reaches the maximum limit thus affecting larger rule set implementations.

2.3 Memory-balanced pipelines

Pipelining can dramatically improve the throughput of tree traversal. However, as discussed earlier, this simple pipelining scheme results in unbalanced memory distribution, leading to low throughput, inefficient memory allocation, and memory waste.

Baboescu et al. [1] propose a Ring pipeline architecture for tree-based search engines. The pipeline stages are

configured in a circular, multi-point access pipeline. Although an almost balanced pipeline is achieved through the use of many small, equally-sized subtrees mapped to different stages, significant limitations exist. Such limitations include the use of content addressable memories (CAMs) as index tables for incoming IP packets, which may result in lower speed, and the need for all packets to traverse the pipeline twice to complete the tree traversal, which results in a throughput of 0.5 packet per clock cycle (PPC).

Jiang et al. [7] adopt a two-dimensional linear dual-pipeline architecture to achieve a throughput of 2 PPC. However, their mapping algorithm requires both decision tree and rule list pipelines to be optimized at the same time. This limits the ability to maximize memory-balancing as the single worst-case memory imbalance of either pipeline becomes the bottleneck for the entire system.

3 Memory balancing and waste reduction

State-of-the-art techniques cannot achieve perfectly balanced memory distribution, due to several constraints they place on mapping:

1. They require the tree nodes on the same level be mapped onto the same stage.
2. Their mapping scheme is uni-directional: the subtrees partitioned from the original tree must be mapped in the same direction (either from the root, or from the leaves).

Both of these constraints are unnecessary; the only constraint we must obey is:

Constraint: If node A is an ancestor of node B in a tree, then A must be mapped to a stage preceding the stage to which B is mapped.

This section studies the problem of balancing memory distribution across pipeline stages while keeping overall resource utilization low thus minimizing memory wastage. Additionally, we describe both pre and post mapping waste elimination techniques, and propose two self-optimizing, fine-grained mapping schemes to solve these problems.

3.1 Decision tree construction

Our work is mainly based on the HyperCuts algorithm [15], which is considered to be the most scalable decision-tree-based algorithm for multi-dimensional packet classification [17]. This decision tree construction algorithm employs several heuristics to cut the space recursively into smaller subspaces. Each subspace ends up with fewer rules, which to a point allows a low-cost linear search to find the best matching rule. The HyperCuts algorithm also allows cutting on multiple fields per step, resulting in a fatter and shorter decision tree.

Although the HyperCuts algorithm suffers from memory explosion due to rule duplication, our pre-mapping

waste elimination technique is to apply two optimizations called *rule overlap reduction* and *push common rules up* [12]. After applying these optimizations, rule duplication is dramatically reduced and the memory requirement becomes linear with the number of rules.

3.2 Tree partitioning and subtree inversion

Figure 1(A) and Figure 1(B) shows both tree partitioning and subtree inversion of an example decision tree. Tree partitioning is necessary to allow subtree inversion for optimal use of the bidirectional pipeline.

In a tree, there are few nodes at the top levels while there are a lot of nodes at the leaf level. Hence, we can invert some subtrees so that their leaf nodes are mapped onto the first several pipeline stages, thus facilitating uniform memory distribution. We propose a *Largest leaf* heuristic (The subtree with the most number of leaves is preferred) to select the subtrees to be inverted in order to fill the first stage to the ideal capacity.

Algorithm 1 finds the subtrees to be inverted, where S denotes a subtree, N the total nodesize of all decision tree nodes, H the number of pipeline stages, CAP the *ideal stage capacity*, and IFR the *inversion factor*. A larger inversion factor results in more subtrees to be inverted. When the inversion factor is 0 (or close to the number of pipeline stages), none (or all) of the subtrees are inverted respectively. The complexity of this algorithm is $O(K)$ where K denotes the total number of subtrees.

Algorithm 1 Select subtrees to be inverted

Require: K subtrees.

Ensure: V subtrees to be inverted.

- 1: $V = 0$, $W =$ the total nodesize of all subtree root nodes.
 - 2: $CAP = N/H$
 - 3: **while** $V < K$ AND $W < (IFR \times CAP)$ **do**
 - 4: Based on *Largestleaf* heuristic, select S with the most number of leaves from those not inverted.
 - 5: $V = V + 1$, $W = W -$ (nodesize of S root node) $+(\#$ of leaves of $S)$
 - 6: **end while**
-

3.3 Bidirectional subtree-to-pipeline mapping

Figure 1(C) shows how the bidirectional mapping algorithm is used for decision tree mapping. Decision tree mapping utilizes the two sets of subtrees created from subtree inversion. Those subtrees which are mapped from roots are called the *forward subtrees*, while the others are called the *reverse subtrees*. We use a bidirectional mapping algorithm (Algorithm 2) to map both sets of subtrees to the decision tree pipeline, where i denotes the pipeline stage, R_n the number of remaining total nodesize, R_h the number of remaining stages, and M_i the memory address pointer for stage i . The bidirectional mapping algorithm

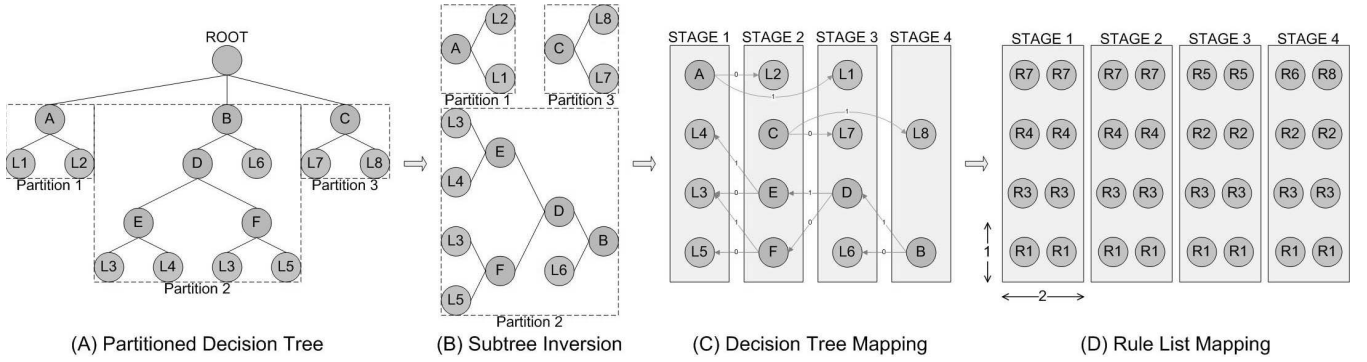


Figure 1. Mapping a decision tree and rule lists onto pipeline stages

is a self-optimizing, fine-grained mapping scheme that focuses around two main ideas:

1. **Fine-grained mapping:** allows two tree nodes on the same tree level to be mapped onto different pipeline stages.
2. **Bidirectional mapping:** allows two subtrees to be mapped onto different directions.

Upon completion, the bidirectional mapping algorithm finds the minimum number of pipeline stages, ensures optimized packing of nodes in all stages for memory balancing, and satisfies the ancestor constraint mentioned in Section 3.

3.4 Configurable rule list mapping

Figure 1(D) shows how the configurable mapping algorithm is used for rule list mapping. Using the optimized, fine-grained decision tree mapping from the bidirectional subtree-to-pipeline mapping stage as a reference, the configurable mapping algorithm maps the rule lists to a second independent pipeline. The purpose of two independent pipelines is to allow more efficient overall resource utilization by exploiting the unique features of the decision tree and rule lists.

We use a configurable mapping algorithm (Algorithm 3) to map all rules to the rule list pipeline, where D denotes the memory port configuration, W the memory width configuration, L the maximum decision tree stage capacity, R_l number of remaining total rule lists, and R_h number of remaining pipeline stages. The configurable mapping algorithm is a self-optimizing, fine-grained mapping scheme that focuses around three main ideas:

1. **Fine-grained mapping:** allows two rule lists from different leaf nodes to be mapped onto different pipeline stages with the same address pointer.
2. **Dual-port mapping:** allows a rule list to map its rules vertically within a single pipeline stage memory by occupying two consecutive memory addresses.

Algorithm 2 Bidirectional fine-grained mapping

Require: K subtrees.

Ensure: H stages with mapped nodes.

- 1: Initialize: $H = G$.
 - 2: **while** $R_n \neq \phi$ **do**
 - 3: Initialize: $ReadyList = \phi$, $NextReadyList = \phi$, $R_n = N$, $R_h = H$.
 - 4: Perform subtree inversion algorithm.
 - 5: Push the roots of K forward subtrees and the leaves of V reverse subtrees into $ReadyList$.
 - 6: **if** $(\lfloor \log_2(CAP) \rfloor / \log_2(CAP)) < 1$ **then**
 - 7: $CAP = 2^{(\log_2(CAP)+1)}$
 - 8: **end if**
 - 9: **for** $i = 1$ to H **do**
 - 10: $M_i = 0$
 - 11: Sort the nodes in $ReadyList$ by first *iscritical*, then in decreasing order of *nodesize*.
 - 12: **while** $M_i < CAP$ and $Readylist \neq \phi$ **do**
 - 13: Pop node from $ReadyList$ and map to locations M_i through $M_i + (nodesize - 1)$.
 - 14: $M_i = M_i + nodesize$
 - 15: **if** The node is in forward subtrees **then**
 - 16: The popped node's children are pushed into $NextReadyList$.
 - 17: **else if** All children of the popped node's parent have been mapped **then**
 - 18: The popped node's parent is pushed into $NextReadyList$.
 - 19: **end if**
 - 20: **end while**
 - 21: $R_n = R_n - M_i$, $R_h = R_h - 1$
 - 22: Merge the $NextReadyList$ to the $ReadyList$.
 - 23: **end for**
 - 24: $H = H + 1$
 - 25: **end while**
-

3. **Variable-width mapping**¹: allows a rule list to map its rules horizontally within a single pipeline stage

¹The maximum data width remains limited by the target FPGA hardware.

memory by allowing the data width of a stage to be any power of 2.

Upon completion, the configurable mapping algorithm finds the minimum number of pipeline stages and ensures optimized packing of nodes in all stages for memory balancing.

Algorithm 3 Configurable fine-grained mapping

Require: R rule lists.

Ensure: H stages with mapped rules.

```

1: Initialize:  $ReadyList = R, R_l = R, R_h = H.$ 
2:  $CAP = R/H$ 
3: if  $(\lfloor \log_2(CAP) \rfloor / \log_2(CAP)) < 1$  then
4:    $CAP = 2^{(\log_2(CAP)+1)}$ 
5: end if
6: if  $CAP > L$  then
7:    $CAP = L$ 
8: end if
9: for  $i = 1$  to  $H$  do
10:   $M_i = 0$ 
11:  Sort the rule lists in  $ReadyList$  in decreasing order of  $ruleddepth.$ 
12:  while  $M_i < CAP$  and  $Readylist \neq \phi$  do
13:    if  $D = single - port$  then
14:      if  $M_i$  is empty then
15:        for  $j = 0$  to  $(\frac{ruleddepth}{W} - 1)$  do
16:          Pop rule from  $ReadyList$  and map rules to location  $M_{i+j}$  in Stage  $i + j.$ 
17:        end for
18:      end if
19:       $M_i = M_i + 1$ 
20:       $R_l = R_l - ruleddepth$ 
21:    else if  $D = dual - port$  then
22:      if  $M_i$  and  $M_i + 1$  are empty then
23:        for  $j = 0$  to  $(\frac{ruleddepth}{2 \times W} - 1)$  do
24:          Pop rule from  $ReadyList$  and map rules to locations  $M_{i+j}$  and  $M_{i+j} + 1$  in Stage  $i + j.$ 
25:        end for
26:      end if
27:       $M_i = M_i + 2$ 
28:       $R_l = R_l - ruleddepth$ 
29:    end if
30:  end while
31:   $R_h = R_h - 1$ 
32: end for

```

Figure 2 shows how utilizing the memory port and width configurations optimizes packing of rules in all pipeline stages. In the non-customized case, each rule list is mapped across the available pipe stages, with one rule per stage. This causes the number of pipe stages to be equal to the largest rule list, thus creating waste when smaller rule lists are mapped. In the customized case, the number of rules per stage is no longer restricted to one, allowing the number of rules per stage to be defined by the memory port

and width configurations. Therefore, each pipe stage can be filled based on the configuration of all the rule lists to be mapped as opposed to only the largest rule list.

3.5 Occupied stage capacity optimization

Although both bidirectional and configurable mapping algorithms adjust stage capacity for memory compatibility and waste reduction potential, memory wastage is still created during mapping if pipeline stages are not fully filled. Eliminating this wastage after mapping optimizes each stage memory further. When complemented with the pre-mapping refinements discussed in Section 3.1, memory wastage due to over-provisioning is reduced to almost zero across all pipeline stages and memory utilization is maximized at the pipeline stage level.

4 Hardware architecture implementation

This paper proposes a bidirectional-configurable optimized linear pipeline architecture (BiConOLP) using dual-port high-speed Block RAMs. The bidirectional pipeline allows two flows from opposite directions to access the local memory in a stage at the same time, balancing memory distribution. The configurable pipeline allows the flexibility to match the design constraints of the optimized bidirectional pipeline to provide system-wide optimization and memory waste reduction. Due to its linear structure, this architecture has many desirable properties:

1. Worst-case throughput of one packet per clock cycle is sustained.
2. Each packet has a constant delay to go through the architecture.
3. Packet input order is maintained.
4. System-wide optimization while supporting large rule sets is maintained.

4.1 Overview

A system block diagram of the basic architecture is shown in Figure 3. The system begins at the *Direction Index Table* (DIT) which stores the relationship between the subtrees and their mapping directions. Any arriving input packet addresses the DIT, which then outputs (1) the distance to the stage where the subtree root is stored in the decision tree pipeline, (2) the memory address of the root in that stage, and (3) the mapping direction which leads the packet to the different entrances of the decision tree pipeline. The DIT output data and packet are then routed to the bidirectional then configurable pipelines, after which the packet exits the system with its respective match and action data.

4.2 Bidirectional pipeline

A block diagram of a single bidirectional pipeline stage is shown in Figure 4.

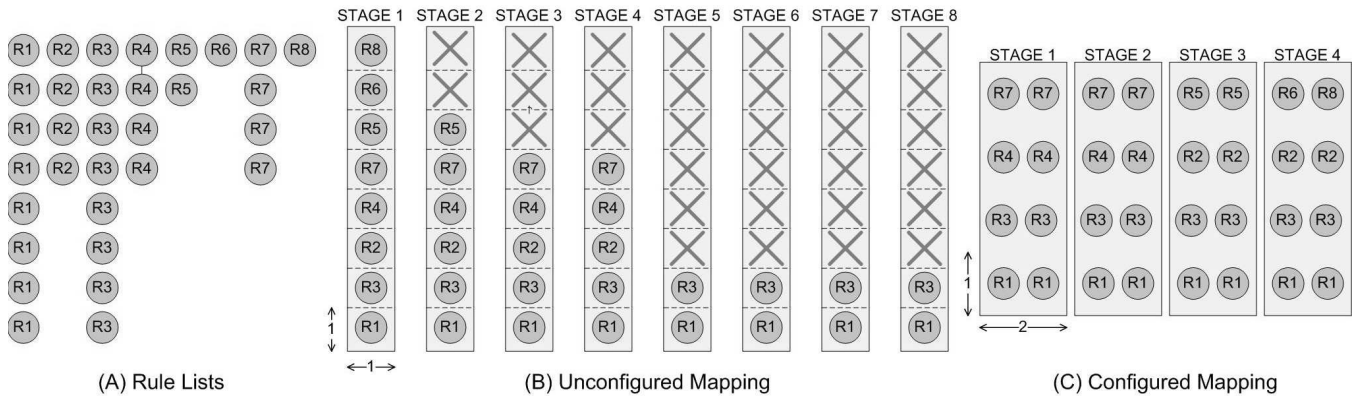


Figure 2. Benefits of customized vs. uncategorized mapping

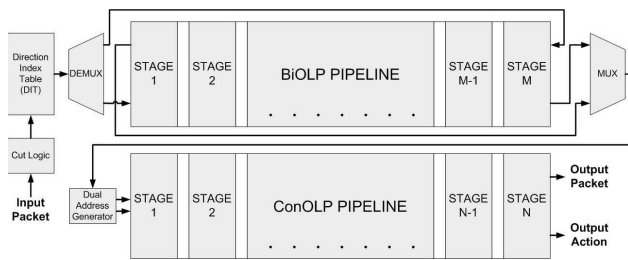


Figure 3. System block diagram

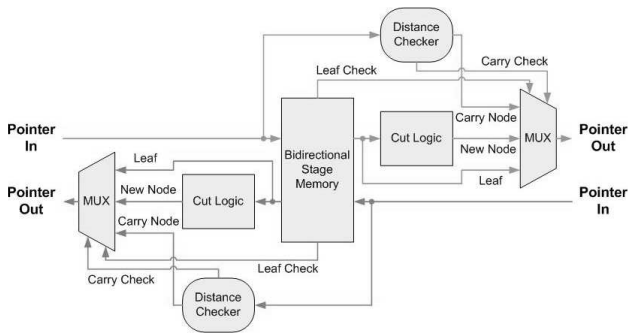


Figure 4. Bidirectional pipeline stage diagram

4.2.1 Bidirectional stage memory

The decision tree pipeline stage memory is constructed using a dual-port synchronous read RAM. This is the backbone of the bidirectional pipeline, allowing two independent accesses to the same pipeline stage.

4.2.2 Cut logic

Different internal nodes in a decision tree may have different numbers of cuts, which can come from different fields. We propose a circuit design using left and right shifters. To generate the next node pointer, the MSB bits of the packet fields, number of cuts on each field, and a base address are

used. The base address points to the address space (range of memory addresses) of a child node within the destination pipeline stage, and the MSB bits of the packet fields point to the specific memory address within that address space as determined by the cuts on each field. First, The number of cuts on each field are used to determine how many packet field MSB bits are used. Second, the packet field MSB bits are aligned to make up the LSB bits of the next node pointer. Third, the base address is shifted to make up the MSB bits of the next node pointer. Fourth, an OR operation merges the remaining base address and the packet field bits to make the complete next node pointer.

4.2.3 Distance checker

Our bidirectional mapping algorithm allows two nodes on the same tree level to be mapped to different stages. We implement this feature by storing the distance to the pipeline stage where the child node is stored in the local pipeline stage memory. When a packet is passed through the pipeline, the distance value is decremented by 1 when it goes through a stage. When the distance value becomes 0, the child node's address is used to access the memory in that stage. If the child node is a leaf node, the distance value stored allows the child node to propagate out of the pipeline without any additional accesses. Data flow muxes are used to route both leaf and node pointers from the current or previous pipeline stage to the pointer output depending on the distance check and leaf check bit values.

4.3 Configurable pipeline

A block diagram of a single configurable pipeline stage is shown in Figure 5.

4.3.1 Configurable stage memory

The rule list pipeline stage memory is configurable based on memory port and width configurations. The memory port configuration determines whether a single or dual-port

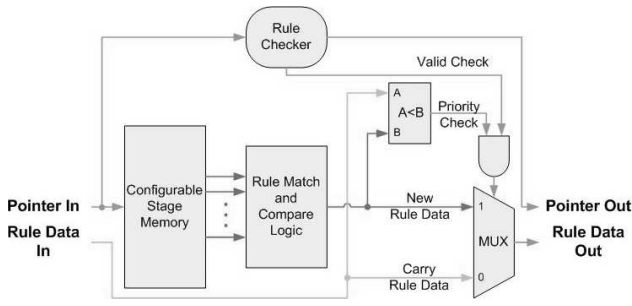


Figure 5. Configurable pipeline stage diagram

synchronous RAM is used. Rules are stored in ordered pairs of 1 or 2 per address when using single-port or dual-port rams respectively. The memory width configuration determines the rule data width for a single pipeline stage.

4.3.2 Rule match and compare logic

When a rule is retrieved in a stage, it is compared to the packet header fields to find a match. We implement different match types for different fields of a rule, similar to [7]. As part of the rule matching process, both rule and packet are checked using their respective valid bits to prevent false-positive matches. When a match is found in the current pipeline stage, the packet will carry the corresponding action and priority information along the remaining rule list pipeline stages until it finds another match with a higher priority. If memory port and width configurations are utilized, multiple rules can be retrieved in a single stage. This requires the same action, priority, and match information to be compared as before.

4.3.3 Rule checker

Our configurable pipeline rule checker functions similar to the bidirectional pipeline distance checker with one key addition: the rule count field. The rule count field stores the distance to the pipeline stage where the end of the rule list is. When the distance value becomes 0, the rule count field begins to decrement. As with the distance value, the rule count value is decremented by 1 when it goes through a stage. When the rule count value becomes 0, the rule list has finished its linear search. Data flow muxes are used to route action, priority, and match information either from the current or previous pipeline stage to the rule data output depending on the valid check (from rule checker) and priority check bit values.

5 Experimental results

5.1 Hardware specifications

We evaluated the effectiveness of our optimized, fine-grained mapping algorithms by running several test cases

on the ACL_10K rule set (9603 total rules). The detailed results of each test case are not presented, due to space limitation. We implemented our design in Verilog using Xilinx ISE 10.1 development tools. We added additional hardware-based customizations to maximize performance, area savings, and resource utilization reductions, including custom-capacity pipeline stage memories, custom cut logic for the decision tree pipeline stages, and fine-grained pipelining. The target device was Xilinx Virtex-5 XC5VFX200T with -2 speed grade. Post place and route results show that our design can achieve a clock frequency of 143.4 MHz, allowing our architecture to exceed the OC-768 rate, i.e. 40 Gbps, by sustaining 45.88 Gbps throughput for minimum size, i.e. 40 bytes, packets. Post place and route resource utilization results from a related state-of-the-art FPGA-based packet classification engine [7] are compared to our work in Table 1. Additionally, due to our balanced memory distribution and memory waste reduction techniques, our resource utilization improved by 12.0% in occupied slices and 43.6% in Block RAM usage.

Table 2 compares our design with several state-of-the-art FPGA-based packet classification engines. For fair comparison, the results of the compared work were scaled to Xilinx Virtex-5 platforms based on the maximum clock frequency². The values in parentheses were the original data reported in those papers. We used a new performance metric, named *Efficiency*, defined as throughput divided by the average memory size per rule, to evaluate the time-space trade-off. Our design outperformed the others with respect to throughput and efficiency, excluding the Two-dimensional Linear Dual-Pipeline [7]. However, when comparing total memory required with the two-dimensional Linear Dual-Pipeline [7], our design shows an improvement of 20.8%.

5.2 Memory balancing and utilization

It can be seen in Figure 6 that a perfectly balanced memory distribution to within 10% overall is achieved (the decision tree and rule list pipelines overall balancing are 90% and 96% respectively)³.

6 Conclusions

This paper presented two self-optimizing, fine-grained mapping schemes and a novel bidirectional-configurable optimized linear pipeline architecture for scalable line rate packet classification on FPGAs. Additionally, future work on our design can achieve even greater performance and improved capabilities by implementing only minor modifications, such as implementing a larger cache-based system, adding rule updating, and supporting external SRAMs

²The BV-TCAM paper [16] does not present the implementation result about the throughput. We use the predicted value given in [16].

³100% perfect balanced memory requires a rule set that is customized to the constraints of physical hardware. Therefore, it is expected that some stages will have unavoidable empty space near mapping completion.

Table 1. FPGA resource utilization

	State-of-the-art [7]			Our Work		
	Used	Available	Utilization	Used	Available	Utilization
Number of Slices	10,307	30,720	33.6%	6,611	30,720	21.5%
Number of bonded IOBs	223	960	23.2%	228	960	23.8%
Number of Block RAMs	407	456	89.3%	208	456	45.6%

Table 2. Performance comparison

Approaches	Number of rules	Total memory (Kbytes)	Throughput (Gbps)	Efficiency (Gbps/KB)
Our approach	9603	485	45.88	908.4
Two-dimensional Linear Dual-Pipeline [7]	9603	612	80.23	1258.9
Simplified HyperCuts [9]	10000	286	7.22 (3.41)	252.5
BV-TCAM [16]	222	16	10 (N/A)	138.8
2sBFCE [13]	4000	178	2.06 (1.88)	46.3
Memory-based DCFL [6]	128	221	24 (16)	13.9

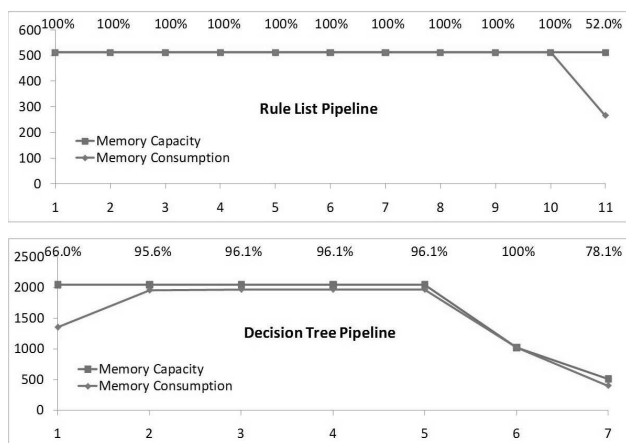


Figure 6. Memory distribution over pipeline stages (the X axis shows the stage IDs, the Y axis the total nodesize, and the memory utilization rate in each stage)

for substantially larger rule sets. Our experimental results show that our architecture can store 10K unique, 5-field rules, and sustain above 45 Gbps throughput for minimum size packets with only 45.6% memory resource utilization. To the best of our knowledge, this work is the first one to achieve perfectly balanced memory distribution (to within 10% overall) over both decision tree and rule list pipeline stages and support large rule sets at line rate with efficient resource utilization.

References

- [1] F. Baboescu, D. M. Tullsen, G. Rosu, and S. Singh. A tree based router search engine architecture with single port memories. In *Proc. ISCA*, pages 123–133, 2005.
- [2] A. Basu and G. Narlikar. Fast incremental updates for pipelined forwarding engines. In *Proc. INFOCOM*, pages 64–74, 2003.
- [3] W. Eatherton, G. Varghese, and Z. Dittia. Tree bitmap: hardware/software IP lookups with incremental updates. *SIGCOMM Comput. Commun. Rev.*, 34(2):97–122, 2004.
- [4] P. Gupta and N. McKeown. Classifying packets with hierarchical intelligent cuttings. *IEEE Micro*, 20(1):34–41, 2000.
- [5] P. Gupta and N. McKeown. Algorithms for packet classification. *IEEE Network*, 15(2):24–32, 2001.
- [6] G. S. Jedhe, A. Ramamoorthy, and K. Varghese. A scalable high throughput firewall in FPGA. In *Proc. FCCM*, 2008.
- [7] W. Jiang and V. K. Prasanna. Large-Scale Wire-Speed Packet Classification on FPGAs. In *Proc. FPGA*, 2009.
- [8] W. Jiang, Q. Wang, and V. K. Prasanna. Beyond TCAMs: An SRAM-based parallel multi-pipeline architecture for terabit IP lookup. In *Proc. INFOCOM*, 2008.
- [9] A. Kennedy, X. Wang, Z. Liu, and B. Liu. Low power architecture for high speed packet classification. In *Proc. ANCS*, 2008.
- [10] T. V. Lakshman and D. Stiliadis. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. In *Proc. SIGCOMM*, pages 203–214, 1998.
- [11] K. Lakshminarayanan, A. Rangarajan, and S. Venkatchary. Algorithms for advanced packet classification with ternary CAMs. In *Proc. SIGCOMM*, pages 193–204, 2005.
- [12] Multidimensional Cuttings (HyperCuts). <http://www.arl.wustl.edu/~hs1/project/hypercuts.htm>.
- [13] A. Nikitakis and I. Papaefstathiou. A memory-efficient FPGA-based classification engine. In *Proc. FCCM*, 2008.
- [14] V. Puš and J. Korenek. Fast and scalable packet classification using perfect hash functions.
- [15] S. Singh, F. Baboescu, G. Varghese, and J. Wang. Packet classification using multidimensional cutting. In *Proc. SIGCOMM*, pages 213–224, 2003.
- [16] H. Song and J. W. Lockwood. Efficient packet classification for network intrusion detection using FPGA. In *Proc. FPGA*, pages 238–245, 2005.
- [17] D. E. Taylor. Survey and taxonomy of packet classification techniques. *ACM Comput. Surv.*, 37(3):238–275, 2005.
- [18] F. Yu, R. H. Katz, and T. V. Lakshman. Efficient multimatch packet classification and lookup with TCAM. *IEEE Micro*, 25(1):50–59, 2005.