P4TE

Analyzing P4 Code to RMT Hardware Mapping

Debobroto Das Robin
Kent State University

Introduction

This document discusses the feasibility of P4TE into existing PISA hardware. Commonly available PISA contains 32 match-actions stages. Each stage having 16 2Kx40n wide TCAM. And these 32 stages are shared by both ingress and egress stage of a program. Therefore for both ingress and egress stage of the program can use 32 stages. The most unoptimized and trivial intermediate representation for the V1model.p4 is generated by the open source P4 compiler for V1model.p4. After compiling the data plane program for P4TE with this compiler we found that our program for the leaf switches needs 23 tables (in sequential order. Actual embedding can be mapped parallely) in both ingress and egress stage. The spine switches need even less number of tables. These intermediate representations can be found at https://github.com/drobinkent/P4TE/tree/main/p4src/Build.

Please remember that, in this intermediate representation, the open source compiler represents a action as a table. Which is a very much unoptimized way . in real life a a table and corresponding action (at least one independent action) can be mapped in a single stage. So our program with 23 tables required in both ingress and egress stage can safely be mapped to RMT hardware with 32 stages.

Next comes the question of header size. In this document look at the “Header Size” calculation section. (the header field definition can be found at https://github.com/drobinkent/P4TE/blob/main/p4src/src/headers.p4). Here we have shown the total header fields required for P4TE is 2854 bits. Which is well below then the 4000 bits header vector limit. Moreover, while developing the system we have not focused on optimizing the header size. For example, to store the ingress color field of a traffic we have allocated 32 bits, where only 2 bits are enough. So a significant reduction in header size is possible. In future we plan to do this optimization.

We analyze how we can achieve further optimization in required resources, in the next part of the document.

We analyzed the intermediate representations of P4TE’s code generated by the P4 compiler. It reveals that P4TE needs a total of 9 tables in leaf switches and 8 switches in spine switch. In our paper, we have discussed that the space required for each of the tables is really small. None of the tables actually need to be mapped to multiple stages of the RMT switch. But due to the if-else statements and other dependencies they may need extra stages.

Our discussion is divided into 3 sections:

1. The Ingress Control Bock
2. The Egress Control Bock
3. An example of how to convert a dangling If-Else statement to a TCAM based implementation

We discuss each of the 3 blocks with their code along the critical path of the whole program graph. When 2 tables are independent of each other and can be executed parallel we marked them in green color. This indicates they can be mapped to the same stage in PISA hardware.

As we have discussed P4TE's stage requirements along the critical path a summation of total stages required indicated the total number of states required for P4TE in RMT hardware.

The Ingress Control Block

Here we will discuss only the P4 program for leaf switch. The P4 program for Spine switch is also similar and less complex.

Ingress Code Block	Note	Stage Required
`if(hdr.p2p_feedback.isValid()){ standard_metadata.egress_spec = CPU_PORT; local_metadata.flag_hdr.do_l3_l2 = false; exit; }else if (hdr.packet_out.isValid()) { standard_metadata.egress_spec = hdr.packet_out.egress_port; hdr.packet_out.setInvalid(); exit; }else if (hdr.packet_in.isValid() && IS_RECIRCULATED(standard_metadata)) { local_metadata.flag_hdr.do_l3_l2 = false;` egress_queue_rate_value_map.write((bit<32>)hdr.packet_in.path_delay_event_port, (bit<48>)local_metadata.egress_rate_event_hdr.egress_traffic_color ); `mark_to_drop(standard_metadata); }else{`	1	1
`init_pkt();`		1
`ingress_delay_processor_control_block.apply(hdr, local_metadata, standard_metadata);`		**2**
`ingress_rate_monitor_control_block.apply(hdr, local_metadata, standard_metadata); }`		**2 Previous one nd this one are parallel**
`if ((hdr.icmpv6.type == ICMP6_TYPE_NS ) && (hdr.icmpv6.type == ICMP6_TYPE_NS)){ ndp_processing_control_block.apply(hdr, local_metadata, standard_metadata);` exit; `}`		1
`if (local_metadata.flag_hdr.do_l3_l2) { l2_ternary_processing_control_block.apply(hdr, local_metadata, standard_metadata);`	2	1
`my_station_processing_control_block.apply(hdr, local_metadata, standard_metadata);`		1
`if (hdr.ipv6.isValid() && local_metadata.flag_hdr.my_station_table_hit) { downstream_routing_control_clock.apply(hdr, local_metadata, standard_metadata);`	3	1
`if(local_metadata.flag_hdr.downstream_routing_table_hit){ local_metadata.flag_hdr.is_pkt_toward_host = true; if(hdr.ipv6.hop_limit == 0) { mark_to_drop(standard_metadata); } } else{`		
`local_metadata.flag_hdr.is_pkt_toward_host = false; local_metadata.flag_hdr.found_multi_criteria_paths = true; #ifdef DP_ALGO_ECMP upstream_ecmp_routing_control_block.apply(hdr, local_metadata, standard_metadata); #endif`	4	1

`#ifdef DP_ALGO_CP_ASSISTED_POLICY_ROUTING`
`cp_assisted_multicriteria_upstream_routing_control_block.apply(hdr,       local_metadata,standard_metadata);`
`cp_assisted_multicriteria_upstream_policy_routing_control_block.apply(hdr, local_metadata, standard_metadata);`
`#endif`
`}`
`}`
	5	3

Note:

1) All the if-else expression of this code block needs some variable to check. Carefully look at. Except for the following block, variables used in all other if-else block expressions are previously known. So simply we can pass these fields to a TCAM based match action table. This table will be fixed and will only need one TCAM. This saves the use of multiple stages for mapping if-else blocks. Serves the same purpose using TCAM efficiently.

```
else{
    init_pkt();
    ingress_delay_processor_control_block.apply(hdr, local_metadata, standard_metadata);
    ingress_rate_monitor_control_block.apply(hdr, local_metadata, standard_metadata);
}
```

2) Both of these 2 blocks need 2 TCAM
3) 3 nested if-else 3 stages required. `Downstream_routing_control_clock` is a simple MAT and can be mapped in first stage
4) A simple MAT for ECMP routing is required if ECMP is used
5) If we use the P4TE
    a) cp_assisted_multicriteria_upstream_routing_control_block.apply(hdr, local_metadata,standard_metadata) : -- This is simple 3 MAT that can be matched paralley.
    b) cp_assisted_multicriteria_upstream_policy_routing_control_block.apply(hdr, local_metadata, standard_metadata): This looks like a lot of dangling if-else. But Carefully look all of the variable used in if-else are already available in the metadata before the bloc starts executing. And the number of variables and their bitwidth is pretty small. All of them can be used as exact-match field of TCAM and the corresponding action can be selected using TCAM matching action.

    Basically whenever you have some exact match variables to match in if-else expression you can use them in TCAM as exact match. As the number of if-else logic is always small, they can be easily mapped to hardware

```
lookup_flowlet_id_map();
   if (hdr.ipv6.traffic_class == TRAFFIC_CLASS_LOW_DELAY){
       if (local_metadata.flow_inter_packet_gap > FLOWLET_INTER_PACKET_GAP_THRESHOLD){
             bit<48> low_delay_path_rate_status = 0;
             egress_queue_rate_value_map.read(low_delay_path_rate_status,     (bit<32>)
local_metadata.delay_based_path);
            if(low_delay_path_rate_status == (bit<48>)GREEN ){
               use_low_delay_port();
            }else if(low_delay_path_rate_status == (bit<48>)YELLOW ){
                use_low_egress_queue_rate_port();
            }else if((low_delay_path_rate_status == (bit<48>)RED ) ){ // use safe rate port
                use_low_egress_queue_depth_port();
            }
            update_flowlet_id_map();
       }else{
           use_old_port();
           update_flowlet_id_map();
       }
   }else if (hdr.ipv6.traffic_class == TRAFFIC_CLASS_HIGH_THROUGHPUT){
       if (local_metadata.flow_inter_packet_gap > FLOWLET_INTER_PACKET_GAP_THRESHOLD){
            bit<48> low_utilization_path_rate_status = 0;
             egress_queue_rate_value_map.read(low_utilization_path_rate_status,
(bit<32>)local_metadata.egr_queue_based_path);
            if(low_utilization_path_rate_status == (bit<48>)GREEN ){
               use_low_egress_queue_depth_port();
            }else if(low_utilization_path_rate_status == (bit<48>)YELLOW ){
             use_low_egress_queue_rate_port();
            }else if((low_utilization_path_rate_status == (bit<48>)RED ) ){
               use_low_delay_port();
            }
            update_flowlet_id_map();
       }else{
           use_old_port();
           update_flowlet_id_map();
       }
   }else{

             use_low_delay_port(); //for all other traffic try to reduce FCT
   }

}
```

Figure 1: Leaf Switch Ingress Pipeline P4 Program Graph

The Egress Control Block

The code for the egress stage is divided into 2 blocks

a) Block 1: This block is basically used for controlling when to clone or recirculate a packet to generate a new packet. All the if-else expression is based on some already available value in the pipeline. And here we are only using some equality checking. So simply we can replace this whole if-else logic using a exact match based TCAM

Egress Control Block	Note	Stage Required
Block 1		

`if(IS_NORMAL(standard_metadata)){`
 	
`egress_queue_depth_monitor_control_block.apply(hdr, local_metadata, standard_metadata);`
		2

`egress_rate_monitor_control_block.apply(hdr, local_metadata, standard_metadata);`		2 Prev one and this one is parallel

`#ifdef DP_BASED_RATE_CONTROL_ENABLED`
`leaf_rate_control_processor_control_block.apply(hdr, local_metadata, standard_metadata);`
`#elif  DP_ALGO_ECMP`
`if(standard_metadata.deq_qdepth > ECN_THRESHOLD) hdr.ipv6.ecn = 3; //setting ecm mark`
`#endif`
		4

`if (local_metadata.is_multicast == true ) {`
`    exit;`
`}`
`#ifdef DP_ALGO_CP_ASSISTED_POLICY_ROUTING`
`if(IS_RECIRC_NEEDED(local_metadata)) {`
`    is_recirculation_needed = true;`
`}`
`#endif`
`if(is_recirculation_needed &&   IS_CONTROL_PKT_TO_NEIGHBOUR(local_metadata) && IS_CONTROL_PKT_TO_CP(local_metadata)){`
 	
`    clone3(CloneType.E2E, (bit<32>)(standard_metadata.ingress_port)+ ((bit<32>)MAX_PORTS_IN_SWITCH * 2), {standard_metadata, local_metadata});`
`}else if(  IS_CONTROL_PKT_TO_NEIGHBOUR(local_metadata) && IS_CONTROL_PKT_TO_CP(local_metadata)){`
`    clone3(CloneType.E2E, (bit<32>)(standard_metadata.ingress_port)+ (bit<32>)MAX_PORTS_IN_SWITCH, {standard_metadata,`
	All variables in if-else areknown previous and actions are only modify	1

`local_metadata});`
`   }else if (IS_CONTROL_PKT_TO_CP(local_metadata)) {`
`        clone3(CloneType.E2E, CPU_CLONE_SESSION_ID, {standard_metadata, local_metadata});`
`   }else if ( IS_CONTROL_PKT_TO_NEIGHBOUR(local_metadata)) {`
`         clone3(CloneType.E2E, (bit<32>)(standard_metadata.ingress_port), {standard_metadata, local_metadata});`
`   }else{`
`        //log_msg("Unhandled logic in cloning control block");`
`   }`
`}`
	ing
	some
	header
	fields	

b) Block 2: The code is shown in the following table. The nested if-else can go up to 8 levels. So apparently seems like we need 8 stages. But All the expressions in ef-else just use the equality operator. So, simply make MAT with 8 fields. And use the TCAM based MAT for executing the actions. The actions involve only copying or modifying some fields from the header vector to another. No memory modification is involved. There are some counters or meters, they are only used to generate some reports for a paper. They have no practical use. So we can just ignore them

   Most importantly our code is not optimized in this block. So in multiple actions, some common header field manipulation is included. So we can easily move them to one independent block. So at worst case even we can conclude that

Egress Control Block	Note	Stage Required
Block 2		
	All values used in if-else Expression is known entering the stage. And needs only one TCAM	1

```
if(IS_NORMAL(standard_metadata)){
   egressPortCounter.count((bit<32>)standard_metadata.egress_port);
   if(local_metadata.flag_hdr.is_pkt_toward_host && hdr.mdn_int.isValid()){
        recirculate<parsed_headers_t>(hdr);
        mark_to_drop(standard_metadata);
   }else if (standard_metadata.egress_port == CPU_PORT) {
       //making some header fields valid/invalid and set value of some header field from metadata
               …………...
   }
   else{
       //making some header fields valid/invalid and set value of some header field from metadata
               …………...
   }
}else{
   if (standard_metadata.egress_port == PORT_ZERO) {
       //making some header fields valid/invalid and set value of some header field from metadata
               …………...
       recirculate<parsed_headers_t>(hdr);
   }else if (standard_metadata.egress_port == CPU_PORT) {
       //making some header fields valid/invalid and set value of some header field from metadata
       ctrlPktToCPCounter.count((bit<32>)standard_metadata.egress_port);
   }else{
       p2pFeedbackCounter.count((bit<32>)standard_metadata.egress_port);
       #ifdef DP_BASED_RATE_CONTROL_ENABLED
       if (hdr.mdn_int.isValid()   && (hdr.mdn_int.rate_control_event  ==
RATE_DECREASE_EVENT_NEED_TO_BE_APPLIED_IN_THIS_SWITCH)){
           if(local_metadata.flag_hdr.is_packet_from_downstream_port == true){
               //making some header fields valid/invalid and set value of some header field from metadata
               …………...
           }else if (local_metadata.flag_hdr.is_packet_from_upstream_port == true){
               //making some header fields valid/invalid and set value of some header field from metadata
               …………...
           }
       }else{
           if(local_metadata.flag_hdr.is_packet_from_downstream_port == true){
               mark_to_drop(standard_metadata);
               …………...
           }else if (local_metadata.flag_hdr.is_packet_from_upstream_port == true){
               //making some header fields valid/invalid and set value of some header field from metadata
               …………...
           }

       }
       if (hdr.mdn_int.isValid()   && (hdr.mdn_int.rate_control_event  ==
RATE_INCREASE_EVENT_NEED_TO_BE_APPLIED_IN_THIS_SWITCH)){
           if(local_metadata.flag_hdr.is_packet_from_downstream_port == true){
               //making some header fields valid/invalid and set value of some header field from metadata
           }else if (local_metadata.flag_hdr.is_packet_from_upstream_port == true){
               //making some header fields valid/invalid and set value of some header field from metadata
               …………...
           }
       }else{
           if(local_metadata.flag_hdr.is_packet_from_downstream_port == true){
               mark_to_drop(standard_metadata);
               …………...
           }else if (local_metadata.flag_hdr.is_packet_from_upstream_port == true){
               //making some header fields valid/invalid and set value of some header field from metadata
               …………...
           }

       }
       #else
       if(local_metadata.flag_hdr.is_packet_from_downstream_port == true){
           mark_to_drop(standard_metadata); //Because we do not want to send the feedback packets to hosts
       }else if (local_metadata.flag_hdr.is_packet_from_upstream_port == true){
           build_p2p_feedback_only();
       }
       #endif
```

```
    }
}
```

Example conversion from Dangling if-else to TCAM approach

Consider the following example

```
if(IS_NORMAL(standard_metadata)){
    egressPortCounter.count((bit<32>)standard_metadata.egress_port);
    if(local_metadata.flag_hdr.is_pkt_toward_host){
        if(hdr.p2p_feedback.isValid()  && hdr.mdn_int.isValid()){
```

Here all the variables used inside the if-else expressions are known beforehand. All those variables are already filled in before executing this block. Whenever some value is not filled in earlier, we can not convert the if-else to TCAM based MAT. Here ins this case and most of our cases uses in P4TE, dangling if-else blocks are convertible in TCAM.

Here, in this example, IS_NORMAL is a macro that depends on 4 boolean variables.
Therefor

1) IS_NORMAL needs b bits
2) Local_metadata.flag_hdr.is_pkt_toward_host needs one bit
3) hdr.p2p_feedback.isValid() && hdr.mdn_int.isValid() → needs 2 bit .

Simply, use all those bits as a match field in a TCAM based table. And a number of entries is really small because there will be not too many if-else branching in a system. So we can simply convert them into TCAM based matches.

Header Size

Header Structs	Size in bits
`packet_out_t  packet_out;`	16 bits
`packet_in_t   packet_in;`	520
`ethernet_t    ethernet;`	112
`ipv4_t        ipv4;`	160
`ipv6_t        ipv6;`	320
`mdn_int_t     mdn_int;`	144
`p2p_feedback_t p2p_feedback;`	80
`tcp_t         tcp;`	160
`udp_t         udp;`	64
`icmpv6_t      icmpv6;`	32
`ndp_t         ndp;`	224
`Total`	1832 bits

local_metadata_t	
l4_port_t l4_src_port;	16
l4_port_t l4_dst_port;	16

bool   is_multicast;
bool is_pkt_rcvd_from_downstream;
	2

delay_event_info_t delay_info_hdr;	208

ingress_queue_event_info_t ingress_queue_event_hdr;	80
egress_queue_event_info_t egress_queue_event_hdr;	80
ingress_rate_event_info_t ingress_rate_event_hdr;	104
egress_rate_event_info_t egress_rate_event_hdr;	104
flag_headers_t flag_hdr;	24
mdn_int_t pkt_timestamp;	144
bit<16> flowlet_map_index; bit<16> flowlet_id; bit<48> flow_inter_packet_gap; bit<48> flowlet_last_pkt_seen_time; bit<9> flowlet_last_used_path;	137
bit<10> egr_port_rate_value_range ; bit<10> egr_queue_depth_value_range ; bit<10> delay_value_range; bit<10> minimum_group_members_requirement; bit<9> delay_based_path; bit<9> egr_queue_based_path; bit<9> egr_rate_based_path; bit<32> temp; bit<8> temp_8_bit;	107
Total	1022

In total header field required is 1022 + 1832 = 2854 bits

Result

From The previous discussion, we can see the stage requirement of P4TE along the critical path is 20 stages. Moreover, our code is not optimized. We are working on developing a generalized Traffic engineering framework, hence for flexibility a lot of codes are unoptimized. But still P4TE needs 20 stages where 32 stages are available. Hence we can say P4TE is realizable in currently available P4TE hardwares