Energy Efficient Packet Classification Hardware Accelerator

Alan Kennedy, Xiaojun Wang
HDL Lab, School of Electronic Engineering,
Dublin City University, Dublin 9, Ireland
alan.kennedy@eeng.dcu.ie

Bin Liu
Department of Computer Science and Technology
Tsinghua University, Beijing P.R.China
liub@tsinghua.edu.cn

Abstract

Packet classification is an important function in a router's line-card. Although many excellent solutions have been proposed in the past, implementing high speed packet classification reaching up to OC-192 and even OC-768 with reduced cost and low power consumption remains a challenge. In this paper, the HiCut and HyperCut algorithms are modified making them more energy efficient and better suited for hardware acceleration. The hardware accelerator has been tested on large rulesets containing up to 25,000 rules, classifying up to 77 Million packets per second (Mpps) on a Virtex5SX95T FPGA and 226 Mpps using 65nm ASIC technology. Simulation results show that our hardware accelerator consumes up to 7,773 times less energy compared with the unmodified algorithms running on a StrongARM SA-1100 processor when classifying packets. Simulation results also indicate ASIC implementation of our hardware accelerator can reach OC-768 throughput with less power consumption than TCAM solutions.

1. Introduction

Packet Classification is increasingly being used by networking devices such as routers, switches and firewalls to implement policies like the blocking of unwanted internet traffic. It is also used for services such as giving priority to Voice over IP or IP-TV packets and the billing of traffic based on network usage. As line rate goes up to OC-192 and moves towards OC-768, which corresponds to 31.25 Million packets per second (Mpps) and 125 Mpps in the worst case when minimum sized packets (40 bytes each) arrive back to back, it poses great pressure to the classifier in a router's line-card. Previous studies emphasize how to increase the throughput while reducing the implementation cost, but seldom address the power consumption. In fact, the power control for the classifier is equally important while designing the line-card due to its tight space budget and the power supply.

Due to their large integration scale and high speed, network processors deployed in typical network equipment can consume more power than any other components in the equipment (e.g, the Intel IXP2800 has a peak power consumption of 30W). As a key attached component to the network processor, the classifier is definitely required to be designed power efficient. Analysis in [1] demonstrated that up to 50% of ISP maintenance costs are power related, including the electricity consumed by the routers and the corresponding cooling systems and so on. Research by Gupta and Singh [2] showed that in 2000 the amount of energy used by various networking devices in the U.S. accumulated to nearly the yearly output of a nuclear reactor unit. So when we design a classifier, a multi-dimensional metric should be considered including the power consumption, besides throughput and cost.

Software approaches, for example the Packet Classification algorithms in [3-11], have the advantage of reduced cost but fail to operate at a very high speed due to their low throughput and nondeterministic amount of clock cycles when executing a packet lookup. Our recent research results from [12] show that when Packet Classification algorithms are implemented on devices such as the StrongARM SA-1100 running at 200 Mhz, the maximum achievable throughput from even the best performing algorithms is only around 0.5 Mpps. For this reason hardware methods for implementing Packet Classification are essential to prevent it from becoming a bottleneck.

The popular hardware implementation at present is to employ Ternary Content Addressable Memory (TCAM) due to the fact that it can match the rules in an *O(1)* clock cycle. This is achieved by carrying out parallel comparisons on all the stored rules in one clock cycle plus the use of pipelining. State-of-the-art technology such as the Cypress Ayama 10000 Network Search Engine [13] can perform 133 million 144-bit search key per second. This high lookup rate however comes at a large cost of consuming between 4.86-19.14 watts depending on the TCAM size. Besides the high power consumption, another drawback for TCAM is its poor storage efficiency of rulesets when using rules containing ranges. Research on real world databases in [14] showed that TCAM storage efficiency ranged between 16-53%, with an average of 34%. TCAMs also take up large amounts of die area with one bit requiring 10-12 transistors compared to SRAM which only requires 4-6 transistors per bit. The complexity of TCAMs also determines they can't run at the high clocking speed obtainable by SRAM. A search engine implemented using this approach will require multiple chips including a host ASIC, TCAMs and the corresponding SRAMs.

Rule	*Field0*	*Field1*	*Field2*	*Field3*	*Field4*
R0	128-240	15-15	40-40	180-180	120-140
R1	90-100	0-80	0-200	190-200	130-132
R2	130-255	60-140	0-60	180-180	133-135
R3	90-92	200-200	40-40	180-180	136-138
R4	130-255	60-140	40-40	190-200	60-63
R5	140-150	60-140	0-255	0-255	140-255
R6	160-165	80-80	0-255	0-255	0-80
R7	48-50	0-80	40-40	0-255	0-10
R8	26-36	50-50	40-40	180-180	30-40
R9	40-40	40-70	40-40	0-255	0-60

Table 1: Ruleset containing 10 rules with 5 fields.

In this paper we present a hardware supported one chip solution which can be implemented on an FPGA utilizing the on-chip Block RAM or as an ASIC using on-chip SRAM. The hardware accelerator achieves high lookup rates by using multiple memory blocks in parallel taking full advantage of the flexibility of an FPGAs RAM blocks and the design flexibility of ASICs. High storage efficiency of rulesets is achieved when compared with the memory requirements of some typical Packet Classification algorithms [12]. Compared with software solutions, the hardware accelerator significantly increases the searching speed and greatly reduces power consumption. So our hardware accelerator exhibits an excellent feature when attached to a network processor, either as an on-chip executing unit or as an external component acting as a high speed classifier.

The layout of the rest of this paper is as follows. Section 2 briefly explains two software algorithms HiCut and HyperCut. Section 3 details the changes made to the algorithms in order to make them better suited to hardware acceleration and more energy efficient when building the search structure. Section 4 presents detailed implementation of the hardware accelerator while Section 5 gives the simulation results and clarifies the parameters used to obtain them. Section 6 concludes the paper.

2. Packet Classification Algorithms

In order to help better understand our hardware oriented modification to the original software algorithms, in this section we will explain two typical packet classification algorithms, namely HiCuts and HyperCuts respectively.

2.1 Hierarchical Intelligent Cuttings (HiCuts)

HiCuts by Gupta and McKeown [5] is a decision based tree algorithm, which allows incremental updates to a ruleset. It takes a geometric view of packet classification by considering each rule in a ruleset as a hypercube in hyperspace defined by the *F* fields of a packets header. The algorithm constructs the decision tree by recursively cutting the hyperspace one dimension at a time into sub regions. These sub regions will contain the rules whose hypercube overlap. Each cut along a dimension will increase the number of sub regions with each sub region containing fewer rules. The algorithm will keep cutting into the hyperspace until none of the sub regions exceed a predetermined number called *binth*.

Figure 1 shows a decision tree built from the ruleset in Table 1. The more cuts performed to an internal node (represented by an ellipse in Figure 1), the fatter and shorter the decision tree. Too many cuts however will result in an unacceptable amount of memory needed to store the decision tree. For that reason the number of cuts *np* which can be performed on a dimension at an internal node *i* is limited using a predefined variable known as *spfac*. Each cut will create child nodes, with the number of cuts always starting with 2 and doubling each time the following equation is satisfied:

$$spfac*number\ of\ rules\ at\ i \le \sum rules\ at\ each\ child\ of\ i + np \qquad (1)$$

The algorithm has many heuristics for choosing which dimension should be cut. The method we chose is to record the largest number of rules contained in a child after cutting each dimension and pick the dimension which returns the smallest number. Each time a packet arrives, the tree is traversed from the root node until a leaf node (represented by a rectangle in Figure 1) is found, which stores a small number of rules limited by the *binth* value. Once a leaf node is reached a small linear search of the rules contained within it is performed to find the matching rule.

Figure 2 shows the cuts performed to the decision tree in Figure 1. Field 0 is selected to cut the root node in 4. This results in 4 child nodes of which 1 exceeds the *binth* value. The node exceeding *binth* is split in 2 using Field 4 with both child nodes equaling the predetermined *binth* value.

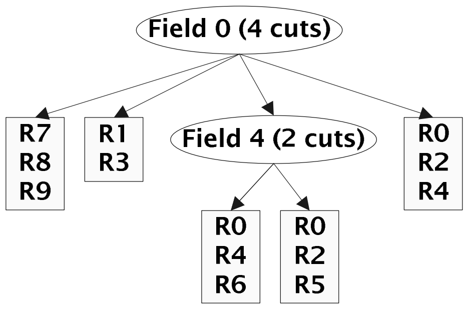

Figure 1: HiCuts Decision tree (binth 3).

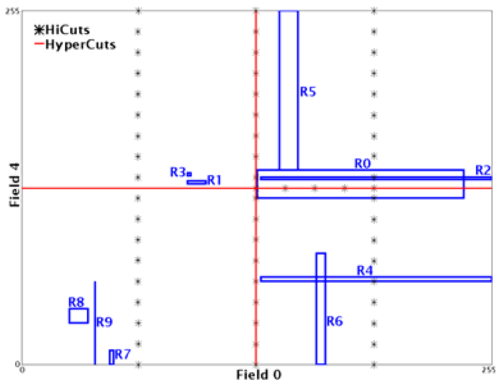

Figure 2: Cuts Performed by HiCuts and HyperCuts.

2.2 Multidimensional Cutting (HyperCuts)

HyperCuts by Singh et al [6] is a modification of the HiCuts algorithm, which also allows incremental updates. The main difference from HiCuts is that HyperCuts recursively cuts the hyperspace into sub regions by performing cuts on multiple dimensions at a time. Figure 3 is an example of such a decision tree built from the ruleset shown in Table 1. The algorithm takes a simple approach to deciding which dimensions should be considered for cutting. The number of distinct range specifications for each dimension is calculated and the dimensions with a number of distinct range specifications greater than or equal to the mean number of range specifications is considered. HyperCuts acts like HiCuts if only 1 dimension is chosen for cutting. The algorithm also limits the number of cuts which can be performed to an internal node *i* (represented by an ellipse in Figure 3) using a space measure function in order to prevent memory explosion. The maximum number of child nodes created by the combination of cuts between the chosen dimensions is bound by the following condition:

$$\textit{max child nodes at } i \leq \textit{spfac*sqrt(number of rules at } i) \qquad (2)$$

The paper by Singh et al never made it clear how to choose the best combination of cuts among the chosen dimensions. Here we choose the combination which resulted in the smallest number of max rules stored in a child node. HyperCuts also takes advantage of extra heuristics, which exploit the structure of the classifier such as region compaction and pushing common rule subsets upwards. Region compaction allows for more efficient cutting of a dimension as it only cuts the region covered by the rules rather than the full region. Pushing common rule subsets upwards will reduce the replicated storage of rules by storing rules common to all child nodes in their parent node. If this option is chosen a linear search of rules stored in internal nodes may need to be carried out as the decision tree is traversed. HyperCuts and HiCuts reduce storage further by merging child nodes which have associated with them the same set of rules and removing child nodes which contain no rules.

Figure 2 shows the cuts performed to the decision tree in Figure 3. The root node is split in 4 by performing 2 cuts to both field 0 and field 4. None of the child nodes created exceed the *binth* value so no more cuts need to be performed.

3. Algorithmic Changes Towards Hardware Accelerating

In order to make the algorithms better suited to hardware acceleration and consume less power during the building of the search structure some modifications were made. The first modification was to remove the region compaction and push common rule subsets upwards heuristics from the HyperCuts algorithm. The region compaction heuristic was removed as it needed large amounts of hardware resources in order to carry out the floating point division required when calculating which path

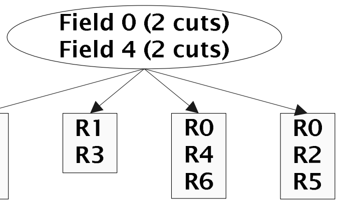

Figure 3: HyperCuts Decision tree (binth 3).

to follow when traversing the decision tree. Floating point division would also consume extra power. Pushing common rule subsets upwards was removed as it meant the searching of rules would have to be carried out while traversing the decision tree, slowing down the hardware accelerator.

The number of cuts allowed to internal nodes for both the HighCut and HyperCut algorithms is limited to 32, 64, 128 or 256 cuts. It was found through the testing of different size and shaped rulesets generated using ClassBench [15] that 32 cuts is a much better starting position than 2 as it leads to a significant decrease in computation and makes an insignificant increase to memory consumption. It was also found that by capping the number of cuts to 256, savings are made in memory consumption and computation with little decrease in throughput. Reducing the amount of computation will lead to power savings as less time is spent building the search structure. For HiCuts the number of cuts to an internal node starts at 32 and doubles each time the following condition is met:

$$(\textit{spfac*number of rules at } i \leq \textstyle\sum \textit{ rules at each child of } i + np) \\ \&(np<129) \qquad (3)$$

HyperCuts considers dimensions for cutting with a number of distinct range specifications greater than or equal to the mean number of distinct range specifications for all the five dimensions. All combination of cuts between the chosen dimensions is considered if they obey the following condition where *spfac* can be 1, 2, 3 or 4:

$$(np \leq 2^{(4+spfac)})\&(np \geq 32) \qquad (4)$$

Capping the number of cuts to 256 also makes the algorithms better suited to hardware acceleration as all the information needed for an internal node can fit fully in one memory word, which can be accessed in a single clock cycle. The hardware accelerator uses 4800-bit wide memory words. Each of the cuts to an internal node requires 1 bit for indicating if the resulting child is an internal or leaf node, up to 12 bits (depending on number of memory words) for the memory location of the node in the search structure and 5 bits for indicating the starting position of the node at the resulting memory location.

In order to calculate which cut the packet should traverse to, the internal node stores 8-bit mask and shift values for each dimension. The masks indicate how many cuts are to be made to each dimension while the shift values indicate each dimensions weight. The cut to be chosen is calculated by ANDing the mask values with the corresponding 8 most significant bits from each of the packets 5 dimensions. The resulting values for each dimension are shifted by the shift values with the results added together giving the cut to be selected.

No. Rules	*Software*		*Hardware*	
	HiCuts	*HyperCuts*	*HiCuts*	*HyperCuts*
60	2,200	1,745	3,000	3,000
150	6,200	5,382	6,000	5,400
500	28,776	13,372	24,000	15,600
1000	43,020	25,592	35,400	28,800
1600	79,444	43,298	69,600	46,800
2191	110,704	56,161	97,200	61,800

Table 2: Memory needed for the search structure and ruleset (bytes), spfac=4, speed=1.

Another modification made to the algorithms is to store the actual rule in the leaf node rather than a pointer. This was found during testing of the many rulesets created using ClassBench to have only a small increase in memory consumption for a large increase in throughput as data is presented to the hardware accelerator one clock cycle earlier. Each saved rule uses 160 bits of memory. The Destination and Source Ports use 32 bits each with 16 bits used for the min and max range values. The Source and Destination IP addresses use 35 bits each with 32 bits used to store the address and 3 bits for the mask. The storage requirement for the mask has been reduced from 6 to 3 bits by encoding the mask and storing 3 bits of the encoded mask value in the 3 least significant bits of the IP address when the mask is 0-27. The protocol number uses 9 bits with 8 bits used to store the number and 1 bit for the mask. The number of the stored rule uses 16 bits. Each 4800-bit memory word can hold up to 30 rules, and it is possible to perform a parallel search of these rules in one clock cycle.

In order to reduce memory consumption the nodes are rearranged after the search structure has been built. All the internal nodes are stored first followed by the leaf nodes. This modification means that the leaf nodes can be saved contiguously in the search structure improving the storage efficiency of rules. To locate a leaf node the number of the memory word where it is located and the starting position of the leaf node within that memory word is needed.

Both the HiCut and HyperCut algorithms use parameters known as *spfac* and *binth* to trade off speed against memory consumption. A third parameter we use to trade speed against memory consumption is called *speed*. When the *speed* parameter is set to 0 the leaf nodes are stored contiguously. This means the search structure is saved in the most memory efficient way possible but will not result in the highest possible throughput as the number of clock cycles needed to classify a packet will be:

$$\lfloor cycles \rfloor = ((z + pos)/30) + 1 + x \quad \textit{where: } 0 \le pos \le 29, \quad z \ge 0 \qquad (5)$$

The number of internal nodes traversed to reach the leaf node is represented by *x*. The starting position of the leaf node in a memory word is represented by *pos* and *z* is the position of the matching rule in the leaf node. If the *speed* parameter is set to 1 a leaf node is only stored in a memory word with a staring position greater than 0 if:

$$RulesStoredInLeaf + pos \le 30 \qquad (6)$$

This means there may be reduced storage efficiency as the leaf nodes may no longer be stored contiguously.

No. Rules	*Software*		*Hardware*	
	HiCuts	*HyperCuts*	*HiCuts*	*HyperCuts*
60	1.32E-02	9.58E-03	9.94E-03	4.65E-02
150	7.44E-02	1.00E-01	3.94E-02	8.81E-02
500	7.61E-01	2.44E-01	2.89E-01	4.20E-01
1000	2.47E+00	6.66E-01	1.00E+00	7.30E-01
1600	7.46E+00	1.65E+00	2.05E+00	1.34E+00
2191	3.79E+01	2.17E+00	3.20E+00	1.84E+00

Table 3: Energy used to build the search structure (Joules), spfac=4, speed=1.

Database	*HiCuts*		*HyperCuts*	
acl1_rules	***memory***	***cycles***	***memory***	***cycles***
300	7800	2	7800	2
1,200	30,600	2	30,600	2
2,500	63,600	2	63,600	2
5,000	127,200	4	127,200	4
10,000	254,400	4	254,400	4
15,000	384,000	4	384,000	4
20,000	471,600	4	468,600	5
24,920	589,200	5	589,200	5
fw1_rules	***memory***	***cycles***	***memory***	***cycles***
300	7200	2	7200	2
1,200	28,200	2	28,200	2
2,500	59,400	2	59,400	2
5,000	142,200	3	142,200	3
10,000	1,086,600	3	657,600	4
15,000	1,244,400	4	1,226,400	4
20,000	1,931,400	6	2,964,600	6
23,087	3,311,400	8	8,256,000	6
ipc1_rules	***memory***	***cycles***	***memory***	***cycles***
300	7200	2	7200	2
1,200	27,000	2	28,200	2
2,500	64,800	3	61,800	3
5,000	144,000	3	144,000	3
10,000	292,800	3	292,800	3
15,000	379,800	4	379,800	4
20,000	491,400	5	491,400	5
24,274	585,000	5	585,000	5

Table 4: Memory consumption (bytes) and worst case number of clock cycles needed to classify a packet for synthetic filter sets generated using ClassBench, spfac=4, speed=1.

Reduced storage efficiency will however lead to an increase in throughput as the number of cycles needed to classify a packet will now be:

$$\lfloor cycles \rfloor = (z/30) + 1 + x \qquad (7)$$

The hardware accelerator has been designed to handle up to 1024 memory words which are 4800 bits wide. This means that search structures up to 614,400 bytes can be saved in memory. This could easily be doubled to 2048 memory words and implemented on devices such as the Virtex XC5VLX330T which can store up to 1,458,000 bytes. The 4800 bit memory word is spread out over 134 memory blocks. Each memory word can save either 1 internal node or up to 30 rules. A memory word can be accessed by the classifier in 1 clock cycle through a 4800-bit wide data bus.

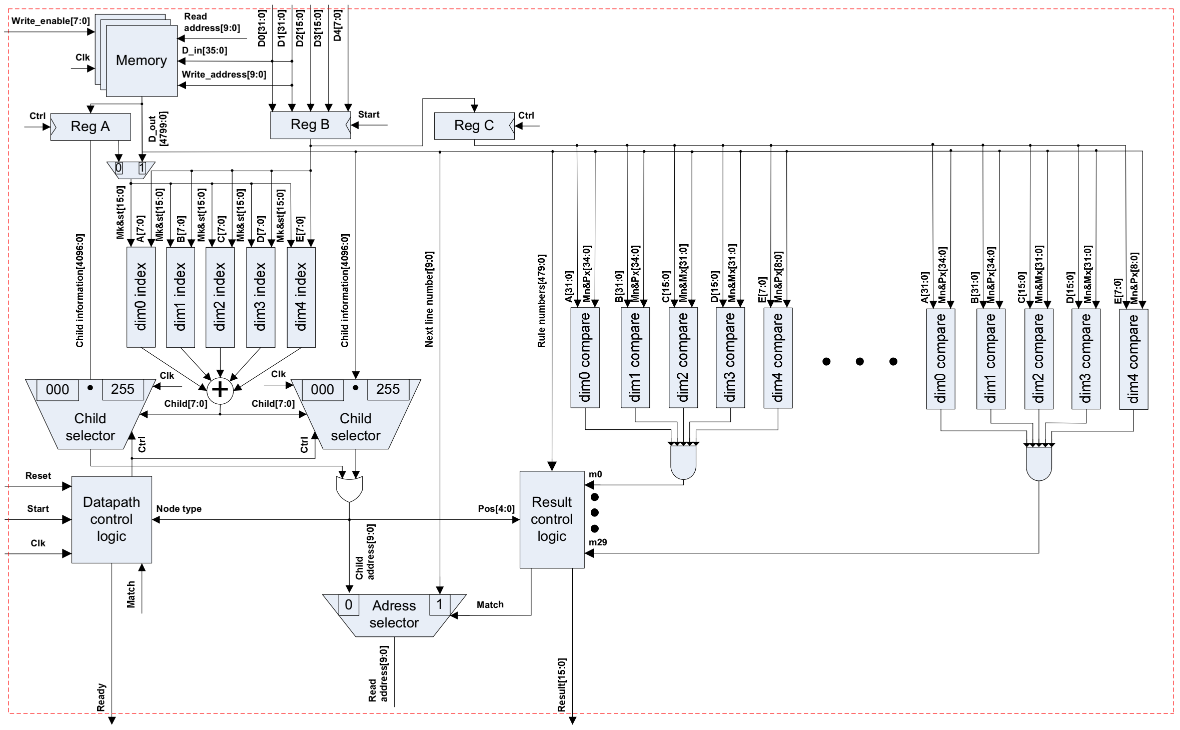

Figure 4: Hardware Accelerator Architecture.

The results in Tables 2,3,5,6,7 and 8 were generated using *acl1* rulesets and traces obtained from [16]. Table 2 shows the memory used by the search structure built for the hardware accelerator using the modified HiCut and HyperCut algorithms as well as the memory used by the software implementation of the original algorithms. Table 3 compares the energy used to build the search structures for the hardware accelerator with that of the search structure implemented in software. The approach adopted by [12] was used to get the energy figures with the algorithms simulated running on a StongARM SA-1100 using Sim-Panalyzer [17]. It can be seen that the storage efficiency of the search structure for the hardware accelerator compares well with that of the software approach, with HiCuts outperforming its software counterpart and HyperCuts having only a slight increase over its software counterpart. In terms of energy used building the search structure the modified algorithms show a large improvement when the rulesets increase in size. The modified HiCuts algorithm uses 11.84 times less energy than that of the unmodified software version when building the search structure for 2191 rules.

Table 4 shows the memory consumption for *acl1*, *fw1* and *ipc1* rulesets generated using ClassBench. It can be seen that the modified algorithms scale well to large rulesets both in terms of memory consumption and worst case number of clock cycles needed to classify a packet. The *fw1* rulesets consume more memory than the *acl1* and *ipc1* rulesets as they contain many wildcard rules. The *fw1* rulesets with over 10,000 rules can still be strored in the FPGAs block RAM by reducing *spfac,* trading off memory against throughput.

4. Hardware Implementation Architecture

The hardware accelerator has been designed to traverse an internal node of the decision tree in 1 clock cycle. It can also do a parallel comparison of up to 30 rules contained within a leaf node in 1 clock cycle. This is possible due to the fact the hardware accelerator can access a 4800-bit memory word every clock cycle. By storing the decision trees root node information in a register separate from main memory it is possible to traverse the root node for an incoming packet while searching a leaf node for the previous packet. Carrying out these tasks in parallel has the effect of reducing the worst case number of clock cycles by 1. This means the hardware accelerator is able to classify a packet every clock cycle if the worst case number of clock cycles needed to classify a packet is 2.

Before any packets can be classified by the hardware accelerator the first step is to save the preprocessed search structure to memory. The hardware accelerator has a shared interface used for both reading in packets and storing the search structure. The memory banks are accessed using a *Write_enable* signal. The *write_address* signal selects which row is to be written to and is incremented after the selected row on all RAM blocks has been filled. Incremental updates to the search structure can be made if a copy of the search structure is kept in off-chip memory for the control plane processor to use when updating the search structure.

Figure 4 shows the architecture of the hardware accelerator. The Flow Chart shown in Figure 5 explains its operation. Once the *Reset* pin is set the hardware accelerator transfers the decision trees root node information from main

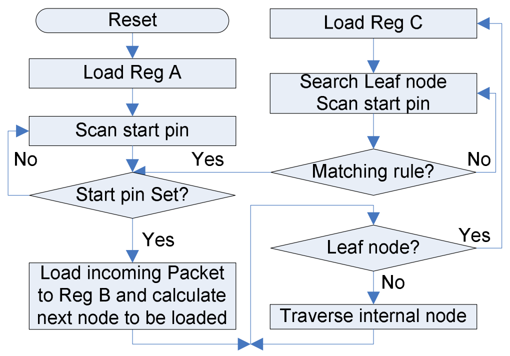

Figure 5: Hardware Accelerator Flow Chart.

memory to Reg A in 1 clock cycle. As explained in section 3 this information includes the starting position, memory location and node type for each of the roots child nodes. It also includes the 8-bit mask and shift values for each dimension used for selecting which child the incoming packet should go to.

On the next rising clock edge the hardware accelerator begins scanning the *Start* pin. A *Ready* pin will go high to indicate a new packet can be classified. When the *Start* pin is set high the incoming packet to be classified is stored in Reg B and the *Ready* pin is put low. An index value for each dimension is created by ANDing the five 8-bit mask values stored in Reg A with the 8 most significant bits from the packets 5 dimensions stored in Reg B. The resulting indexes are shifted using the 8-bit shift values stored in Reg A and then added together to determine which node address should be selected from Reg A. This node address is used to select which memory word should be loaded from main memory on the next rising clock edge. On this edge the hardware accelerator checks if the node to be loaded from main memory is an internal or leaf node.

If the node loaded from main memory is an internal node then the hardware accelerator will use the internal node information loaded to traverse to the next node. The mask values from the internal node loaded are ANDed with the packet values from Reg B. These values are shifted using the shift values from the internal node loaded and then added together. The result is used to determine which child node should be loaded from main memory on the next rising clock edge. If the selected child is still an internal node then the process of traversing the internal nodes is repeated until a leaf node is found. Each internal node to be traversed takes 1 clock cycle.

The packet value in Reg B will transferred to Reg C if the node loaded from main memory on a rising clock edge is a leaf. The accelerator then uses 30 comparator blocks in parallel to compare the packet value in Reg C with the leaf nodes rule information loaded from main memory. While this compare takes place the *Start* pin is again monitored and the *Ready* pin goes high to indicate that a new packet can be classified. If the *Start* pin does go high the new packet information is saved in Reg B and the *Ready* pin is put low. The mask and index values for the root node stored in Reg A are used with the packet value in Reg B to determine which child node should be loaded from main memory once a matching rule has been found for the previous packet.

On the next rising clock edge the hardware accelerator checks if a matching rule has been found. The hardware accelerator will continue searching the leaf node if a matching rule has not been found. If a match has been found then the *Start* pin is checked to see if it has been set. If it has not been set meaning a new packet has not been read in, then the hardware accelerator will continue monitoring the *Start* pin until it is set. If the *Start* pin has been set meaning a new packet has been read in, the hardware accelerator will check if the resulting child node traversed to is an internal node or a leaf node. An internal node will mean repeating the process of searching for a leaf node while a leaf node will mean repeating the process of loading the packet value from Reg B to Reg C, carrying out a parallel compare on the rules stored in the leaf node and putting the *Ready* pin high to indicate a new packet can be read in.

5. Simulation Results

5.1 Simulation Parameters

The hardware accelerator was implemented in VHDL and targeted at two technologies: a 65nm ASIC library by Taiwan Semiconductor Manufacturing Company and a Xilinx Virtex-5 FPGA. For the ASIC solution the hardware accelerator was synthesized using Synopsys. Post place and route timing analysis indicate a maximum theoretical operating frequency of 226 MHz and a gate count of 51,488 for the hardware accelerator. In order to estimate the power consumption the Synopsis Prime Power tool was used to analyze the annotated switching information from a VCD file. Simulations were run for both the HiCut and HyperCut algorithms running on the hardware accelerator at 226 MHz. This meant running 6 simulations for each algorithm using the packet trace files corresponding to the 6 rulesets.

The hardware accelerator was synthesized using Xilinx ISE for the FPGA. Post place and route timing analysis indicate a maximum theoretical operating frequency of 77 MHz. The accelerator uses 3,280 of the FPGAS slices (22%) and the memory banks for the search structure use 134 Block RAMs (54%). Post place and route simulations were carried out on the netlist for subsequent VCD power analysis with the Xilinx XPower FPGA power analysis tool. The same 12 simulations ran for the ASIC were run for the FPGA with the frequency set to 77 MHz.

Table 5 compares the performance figures for the FPGA and ASIC solutions used to implement the hardware accelerator to the StrongARM used to run the software algorithms. The devices are running at different speeds ranging from the ASIC operating at OC-768 speeds down to the SA-1100 operating at less than OC-1. Since the hardware accelerator and SA-1100 processor are implemented in different technologies, a direct comparison of power consumption would be unfair. For this reason we use the approach adopted by [18] to normalize the power

Device	*Virtex5SX95T*	*ASIC*	*SA-1100*
Process [nm]	65	65	180
Voltage [V]	1	1.08	1.8
Frequency [Mhz]	77	226	200
Power [mW]	1811	18.32*	42.45*
Area (Gates)	17,600,998	51,488	
Slices	3,280 (22%)		
Block RAMs	134 (54%)		

Table 5: Device comparison. (normalized power*)

No. Rules	*Software (running on SA-1100)*		*ASIC (65 nm)*		*FPGA (Virtex5SX95T)*	
	HiCuts	*HyperCuts*	*HiCuts*	*HyperCuts*	*HiCuts*	*HyperCuts*
60	4.60E-07	7.82E-07	7.58E-11	7.90E-11	2.39E-08	2.38E-08
150	5.69E-07	1.09E-06	7.32E-11	7.55E-11	2.43E-08	2.41E-08
500	6.72E-07	1.28E-06	1.00E-10	1.21E-10	3.21E-08	3.09E-08
1000	8.62E-07	1.85E-06	1.24E-10	1.19E-10	3.94E-08	3.45E-08
1600	1.09E-06	1.40E-06	1.81E-10	1.42E-10	4.89E-08	3.86E-08
2191	1.09E-06	1.94E-06	2.07E-10	1.46E-10	5.22E-08	3.87E-08

Table 6: Average energy (normalized) needed to classify a packet (Joules) , spfac=4, speed=1.

No. Rules	*Software (running on SA-1100)*		*ASIC (65 nm)*		*FPGA (Virtex5SX95T)*	
	HiCuts	*HyperCuts*	*HiCuts*	*HyperCuts*	*HiCuts*	*HyperCuts*
60	88,125	51,794	226,000,000	226,000,000	77,000,000	77,000,000
150	71,181	37,323	221,919,129	226,000,000	75,609,614	77,000,000
500	60,245	31,721	164,389,580	171,530,362	56,008,839	58,441,760
1000	47,544	22,249	135,333,231	155,475,310	46,109,109	52,971,676
1600	37,760	29,201	105,444,530	161,201,374	35,925,791	46,663,555
2191	37,399	21,168	99,498,019	136,131,129	33,899,767	46,380,959

Table 7: Total number of packets classified in 1 second, spfac=4, speed=1.

No. Rules	*Software*		*Hardware*	
	HiCuts	*HyperCuts*	*HiCuts*	*HyperCuts*
60	17	22	2	2
150	27	38	3	2
500	29	52	3	3
1000	46	103	4	4
1600	58	70	5	4
2191	58	114	5	4

Table 8: Worst case number of memory acesses, spfac=4, speed=1.

figures for the hardware accelerator and processor so a fair comparison can be made. The power has been normalized so all devices are compared using 65nm technology with a core voltage of 1V. The normalized power P′ (indicated by an asterisk in Table 5) is calculated using the following equation where *S* is the scaling factor of the process technologies and *U* is the scaling factor of the voltage:

$$P' = P * S^2 * U \qquad (8)$$

The ASIC and StrongARM only consider the power consumption and area of the datapath logic whilst the FPGA figures include the power consumption and area for both the datapath logic and memory.

5.2 Throughput

The results in Table 8 show the worst case number of memory accesses needed to classify a packet. For the hardware accelerator this result also represents the worst case number of clock cycles needed to classify a packet. This means that the minimum bandwidth for a given ruleset can be guaranteed under worst case operating conditions. For the rulesets and packet traces used in Table 7 the hardware accelerator can classify up to 546 times more packets when implemented as an ASIC, than the best performing software algorithm RFC tested in [12], when running on a StrongARM SA-1100 processor. When compared to the best performing software algorithm which also supports incremental updates HiCuts, the hardware accelerator can classify up to 4,269 times more packets. The results show that after modification HyperCuts is now the best performing algorithm in terms of both memory usage and throughput. The reason for the increase in throughput is that the modified HyperCuts algorithm allows more cuts to internal nodes than the unmodified version meaning a shorter linear search of leaf nodes is needed.

The area used by the accelerator is equivalent to the area of 51,488 2-input NAND gates which means it could compete with even the most basic RISC type processing engines using no data or instruction cache. This means it would make sense to implement the proposed approach as a hardware accelerator attached on-chip or on-board network processors to remove the burden of packet classification from the network processors processing engines allowing it achieve line speeds of up to OC-768. The figures also show that OC-192 line speeds are obtainable if the proposed approach is implemented on an FPGA.

5.3 Power Consumption

Table 6 compares the average normalized energy needed to classify a packet for the two unmodified packet classification algorithms running on a StrongARM SA-1100 processor to that of the hardware accelerator implemented using ASIC and FPGA technology. The energy figures for the FPGA include the energy used by both the memory and datapath logic whilst the ASIC and RISC solutions only include the energy used by the datapath logic. For this reason it is fairer to compare the energy used by the StrongARM SA-1100 with the energy used by the ASIC.

When the power consumption of the hardware accelerator is compared with HiCuts, the most energy efficient software algorithm tested in [12] which supports incremental ruleset updates, the hardware accelerator shows energy savings of up to 7,773 times on the rulesets tested in Table 6. This massive energy saving shows the hardware accelerator is ideally suited to low power packet classification. The average power consumption of the hardware accelerator when implemented on an FPGA with 614,400 bytes of memory is 1.8W when running at 77 MHz. This shows a large power saving over one of the most energy efficient commercial TCAM solutions the Cypress Ayama 10128 Network Search Engine which consumes 2.9W when running at 77 MHz with 576,000 bytes of memory [13].

The Cypress Ayama 10512 Network Search Engine can classify at most 133 Mpps when running at its top speed of 133 MHz with 2.304 MB of memory. At this speed it consumes 19.14 watts [13]. When implemented as an ASIC the hardware accelerator consumes 11.65mW when running at 133 MHz. This shows massive power savings are possible when you consider that the CY7C1381D 2.304 MB SRAM chip from Cypress consumes 693mW of power when running at 133MHz with a core voltage of 3.3V [19]. When running at 226 MHz the hardware accelerator consumes 19.79mW. The CY7C1370DV25 2.304 MB SRAM chip from Cypress consumes 875mW of power when running at 250 MHz with a core voltage of 2.5V [20]. This shows its possible for the hardware accelerator to classify packets at higher speeds than TCAMs while using less power.

6. Conclusions

With ever increasing line speeds, packet classification has become a bottleneck in wire speed processing for high speed routers. Solutions for packet classification such as software running on the processing engines of network processors can not catch up with the high line rates due to their low throughput. Existing hardware methods for high speed packet classification such as TCAMs have the drawbacks of high power consumption, large board area and poor storage efficiency of rulesets.

In this paper we have introduced an energy efficient packet classification hardware accelerator capable of classifying packets at line rates exceeding OC-768 if implemented using 65nm ASIC technology and at rates in excess of OC-192 if implemented using a Xilinx Virtex5 95T FPGA. The architecture proposed would be ideally suited to implementation as a hardware accelerator attached on-chip or on-board network processors due to its low area footprint, high throughput, low power consumption and high storage efficiency of rulesets.

The hardware accelerator has throughput gains of up to 4,269 times and energy savings of up to 7,773 times when compared with software algorithms implementing packet classification on the processing engines of typical programmable network processors. It also shows the possibility for clock running speed gains of up to 1.7 times and an obvious decrease in power consumption when compared to existing state-of-the-art TCAM technology.

7. Acknowledgments

We would like to thank Kealan McCusker from the Centre for Digital Video Processing, Dublin City University for his help in the ASIC implementation of the hardware accelerator. This work was co-funded by the Irish Research Council for Science, Engineering and Technology, funded by the National Development Plan, the China/Ireland Science and Technology Collaboration Research Fund (2006DFA11170) and the Cultivation Fund of the Key Scientific and Technical Innovation Project, MoE, China (705003).

8. References

[1] Anthony Gallo. Meeting Traffic Demands with Next-Generation Internet Infrastructure. Lightwave, 18(5):118–123, May 2001.
[2] M. Gupta and S. Singh, "Greening of the Internet" in *ACM SIGCOM 2003*, pp. 19-26.
[3] F. Baboescu and G. Varghese, "Scalable packet classification," *IEEE/ACM Trans. Netw*., vol. 13, no. 1 pp. 2-14, 2005.
[4] P. Gupta and N. McKeown, "Packet classification on multiple fields," in *ACM SIGCOMM 1999*, pp.147-160
[5] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings," *IEEE Micro*, vol.20, no. 1, pp. 34-41, 2000.
[6] S. Singh, F. Baboescu, G. Varghese and J. Wang, "Packet Classification Using Multidimensional Cutting" in *ACM SIGCOMM*, 2003, pp.213-214
[7] F. Baboescu, S. Singh, and G. Varghese, "Packet classification for core routers: Is there an alternative to CAMs?" in *IEEE INFOCOM*, 2003, pp. 53-63.
[8] V. Srinivasan, S. Suri, and G. Varghese, "Packet Classification using Tuple Space Search" in *ACM SIGCOMM 1999*, pp. 135-146.
[9] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Network Mag.,* vol. 15, no. 2, pp.24-32, 2001
[10] T. Woo, "A modular approach to packet classification: algorithms and results," in *IEEE INFOCOM*, Mar. 2000, pp. 1213-1222.
[11] P. C. Wang, C. T. Chan, C. L. Lee and H. Y. Chang "Scalable Packet Classification for Enabling Internet Differentiated Services" *IEEE Trans. on Multimedia*, vol. 8, no. 6, pp. 1239-1249, 2006.
[12] A. Kennedy, D. Bermingham, X. Wang and L. Bin, "Power analysis of packet classification on programmable network processors" *IEEE ICSPC*, Nov. 2007, pp. 1231-1234.
[13] Cypress Ayama 10000 Network Search Engine, http://download.cypress.com.edgesuite.net/design_resources/datasheets/contents/cynse10256_8.pdf
[14] E. Spitznagel, D. Taylor, and J. Turner, "Packet Classification Using Extended TCAMs," Proc. 11th Int'l Conf. Network Protocol (ICNP '03), 2003.
[15] D. Hoffman and P. Strooper, "Classbench: A Framework for Automated Class Testing," Software Practice and Experience, vol. 27, no. 5, pp. 573-597, May 1997.
[16] ACL1 RuleSets and Packet traces [Online]. Available: www.arl.wustl.edu/~hs1/PClassEval.html
[17] Sim-Panalyzer, The SimpleScalar-ARM Power Modeling Project. [Online]. Available: www.eecs.umich.edu/~panalyzer/
[18] A. Kinane, "Energy Efficient Hardware Acceleration of Multimedia Processing Tools" *PhD thesis, Dublin City University*, May 2006.
[19] Cypress's high-speed synchronous SRAMs http://download.cypress.com.edgesuite.net/design_resources/datasheets/contents/cy7c1381d_8.pdf
[20] Cypress's high-speed synchronous SRAMs, http://download.cypress.com.edgesuite.net/design_resources/datasheets/contents/cy7c1370dv25_8.pdf