Towards Optimized Packet Classification Algorithms for Multi-Core Network Processors

Yaxuan Qi[1, 3], Bo Xu[1, 2], Fei He[1, 2], Xin Zhou[1, 2], Jianming Yu[1, 2] and Jun Li[1, 3]
[1]*Research Institute of Information Technology, Tsinghua University*
[2]*Department of Automation, Tsinghua University*
[3]*Tsinghua National Lab for Information Science and Technology*
yaxuan@ tsinghua.edu.cn

Abstract

In this paper, a novel packet classification scheme optimized for multi-core network processors is proposed. The algorithm, Explicit Cuttings (ExpCuts), adopts a hierarchical space aggregation technique to significantly reduce the memory usage. Consequently, without burst of memory usages, the time-consuming linear search in the conventional decision-tree based packet classification algorithms is eliminated, and an explicit worst-case search time is achieved. To evaluate the performance of ExpCuts, we implement the algorithm, as well as HiCuts and HSM, on the Intel IXP2850 network processor. Experimental results show that ExpCuts outperforms the existing best-known algorithms in terms of memory usage and classification speed.

1. Introduction

To reach multi-Gbps packet classification speed, there are currently two major types of implementations on commercialized products: Software-based implementation on general-purpose processor (such as CPU) and hardware-based implementation on application specific integrated circuits (ASIC). However, both of these two types of implementations have inherent limitations:

♦ **Software-based implementation:** Software-based algorithmic solutions embrace the practice of leveraging the statistical structure of classification rule sets to improve average performance. While current algorithms for packet classification constantly improve the search speed and reduce the memory usage, their performance in software-based implementation is not adequate for practical high-end deployment. The chief problem is, due to the diversity of incoming packet headers, most memory accesses occur to different memory locations. So the probability of CPU cache hit is not high, and thus access to main memory becomes the bottleneck [1].

♦ **Hardware-based implementation:** Although traditional routers and switches based on ASIC can perform packet classification at multi-Gbps speed, these devices are limited in backbone networks. This is because most hardware-based solutions trade off the programmability for processing speed, and the usage of special memory chips, such as Ternary CAMs, requires too much power and board area to support large number of rules. Therefore, hardware-based solutions usually mean higher production cost, longer time-to-market, and more difficulties in upgrade to support new applications. These drawbacks left the hardware-based packet classification products only to the backbones (10Gbps ~ 40Gbps) [2].

Thus, the challenge of combining intelligent software-based algorithms and hardware-based architectures to minimize the unfavorable characteristics of existing solutions motivates the research today. With the advent of powerful network processors (NPs) in the market, many computation intensive tasks can be accomplished more easily in network services. As an emerging class of programmable processors highly optimized for fast packet processing operations, network processors deliver hardware-level performance to software-programmable systems. In this paper, we propose an NP-optimized packet classification algorithm to achieve near line-rate classification speed.

Main contributions of this paper are:

♦ **An algorithm optimized for multi-core network processor:** The algorithm proposed in this paper, Explicit Cuttings (ExpCuts), adopts a hierarchical space aggregation technique to significantly reduce the memory usage. Thus, without burst of memory usage, the time-consuming linear search in common decision-tree algorithms is eliminated, and an explicit worst-case search time is achieved. When multi-channel memory is provided, ExpCuts can be further optimized by populating different level of tree-nodes on multiple memory channels.

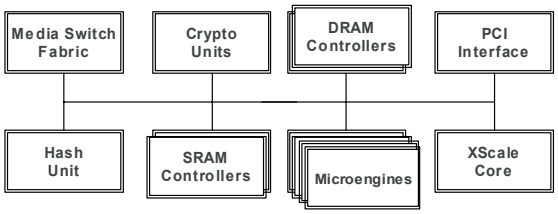

Figure 1. Block diagram of the IXP2850.

♦ **Performance evaluation on Intel IXP2850 NP:** To objectively evaluate the performance of ExpCuts, we built not only a packet classification building block, but also a whole packet processing application on the Intel IXP2850. ExpCuts, as well as previous work HiCuts and HSM are implemented with Intel microcode assembly, and evaluated both on software simulator and hardware platform. Experimental results show that ExpCuts outperforms these existing best-known algorithms in terms of both packet classification speed and memory usage.

2. Existing Work

Existing algorithmic solutions for packet classification can be categorized based on two classification strategies [11]:

♦ **Field-independent search:** Algorithms like RFC [7] and HSM [8] perform independent parallel searches on indexed tables; the results of the table searches are combined in multiple phases to yield the final classification result. All the entries of a lookup table are stored consecutively in memory. The indices of a table are obtained by space mapping and each entry corresponds to a particular sub-space and stores the search result at current stage. Algorithms using parallel search are very fast in term of classification speed while they may require comparatively large memories to store the big cross-producting tables.

♦ **Field-dependent search:** HiCuts [3] and HyperCuts [9] are examples of algorithms employing field-dependent searches, i.e., the results of fields that have already been searched influence the way in which subsequent fields will be searched. The main advantage of this approach is that the intelligent and relatively simple decision-tree classifier can be used. Although in most cases, decision-tree algorithms require less memory than field-independent search algorithms [12], they tend to result in implicit worst-case search time and thus cannot ensure a stable worst-case classification speed.

Because field-independent search algorithms often need tens of megabytes memory to store the large cross-producting table [4, 12], the large memory requirement sometimes can hardly be satisfied with current SRAM chips [13]. In comparison, algorithms using field-dependent searches are commonly more flexible in terms of memory and speed tradeoffs, and hence are more flexible to be optimized for network processor implementations.

In this paper, the proposed algorithm is based on the well-known field-dependent search algorithm HiCuts. By employing an effective hierarchical space compression technique, the proposed algorithm eliminates the time-consuming linear search in HiCuts and provides with an explicit worst-case bound for search to guarantee near line-speed packet classification speed.

Because all the optimizations are towards multi-core network processors, before describing the algorithm, we give a brief introduction to the architecture and programming challenges of a typical multi-core network processor in the next section.

3. Intel IXP2850 Network Processor

Network processors are typically characterized as distributed, multi-processor, multi-threaded architectures designed for hiding memory latencies in order to scale up to very high data rates. This section gives a brief overview of the hardware architecture of Intel IXP2850.

3.1. Architecture of IXP2850

The architecture of IXP2850 is motivated by the need to provide a building block for multi-Gbps packet processing applications. A simplified block diagram and its description of the Intel IXP2850 are shown in Figure 1 and Table 1 respectively. Details of IXP2850 can be found in [17-20].

Table 1: Hardware overview of IXP2850

Intel XScale core:	Each IXP2850 includes an XScale core. The Intel XScale core is a general purpose 32-bit RISC processor.
Multithreaded microengines	The IXP2850 network processor has 16 MEs working in parallel on the fast packet-processing path, running at 1.4 GHz clock frequency.
Memory hierarchy	IXP2850 has 4 channels of QDR SRAM running at 233 MHz and 3 channels of RDRAM running at 127.3 MHz.
Build-in media interfaces	IXP2850 has flexible 32-bit media switch interfaces. Each interface is configurable as media standard SPI-4 or CSIX-L1 interfaces.

3.2. Programming Challenges

Network processing applications are targeted at specific data rates. In order to meet these throughput requirements, a NP must complete the packet processing tasks on a given packet before another packet arrives. To keep up with the back-to-back

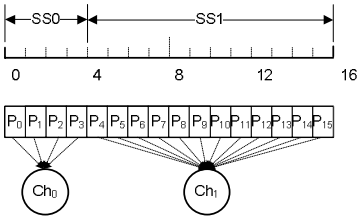

Figure 2: HiCuts space aggregation

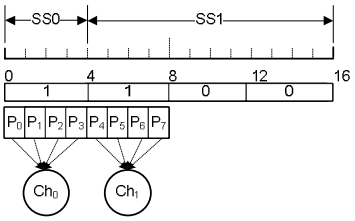

Figure 3: ExpCuts space aggregation

arrival of minimum size packets at line rate, the NP faces the following programming challenges [21]:

♦ **Achieving a deterministic bound on packet processing operation:** Due to the line rate constraint, we need to design the network algorithms in such a way that the number of clock cycles to process the packet on each microengine (ME) does not exceed an upper bound. The key issue is to design network algorithms using the right kind of data structures, and limiting the total number of memory accesses.

♦ **Masking memory latency through multi-threading:** Even if the data structures are designed to complete packet processing within a definite time interval, it is not sufficient to meet the line rate processing requirements because memory latencies are typically much higher than the amount of processing budget. Therefore, the second important challenge is to utilize the multiple hardware threads effectively to mask memory latencies.

♦ **Maintaining packet ordering in spite of parallel processing:** Another significant challenge in programming the NPs is to maintain packet ordering. This is extremely critical for applications like media gateways and traffic management. Packet ordering can be guaranteed using sequence numbers and/or strict thread ordering.

In the next section, we will propose an NP-optimized packet classification algorithm according to the IXP2850 architecture and the programming challenges.

4. Algorithm Optimization

4.1. The HiCuts Algorithm

The proposed algorithm is based on one of the well-known packet classification algorithm Hierarchical Intelligent Cuttings (HiCuts). HiCuts preprocesses the packet classification rules to build a decision-tree for field-dependent search, and in each leaf-node of the decision-tree, a small number of rules bounded by a threshold (*binth* in [3]) are stored for linear search. Packet header fields are used to traverse the decision-tree until a leaf-node is reached. The rules stored in that leaf are then linearly searched for a match.

Geometrically, HiCuts decomposes the multi-dimensional search space by heuristics that exploit the characteristic of real-life rule sets. At each internal-node, the current search space is segmented into certain number of equal-sized sub-spaces along a particular dimension. The number of cuttings and the dimension to cut is determined by heuristics [3]. The sub-spaces obtained on each fields are intersected and each intersection generates a child node. To link the current node with its children, HiCuts stores a pointer array at each internal-node. Each pointer in the array corresponds to a sub-space and is sequentially stored according to the order of the sub-spaces.

Because different child-nodes may share the same sub-ruleset, multiple pointers can be **aggregated** to point to a single child node. Figure 2 shows how to aggregate sub-spaces using pointer array: Sub-spaces 0 through 3 are aggregated to SS0 by pointers P_0~P_3. Sub-spaces 4 through 16 are aggregated to SS1 by pointers P_4~P_{15}. SS0 and SS1 are the search space for child-node Ch_0 and Ch_1 respectively.

4.2. Explicit Cuttings

4.2.1 Motivation

Although HiCuts has good time/space tradeoffs and works well for real-life rule sets, direct implementation of HiCuts without NP-aware optimization on multi-core network processors may suffer from:

♦ **Non-deterministic worst-case search time:** Because the number of cuttings varies at different tree nodes, the decision-tree does not have deterministic worst-case depth. Thus, although worst-case search time is the most important performance metric in packet classification applications, HiCuts does not have an explicit bound for search.

♦ **Excessive memory access by linear search:** Experimental results (in section 6.6) show that linear search is very time-consuming. Although the number of rules for linear search is limited to *binth* (4~16), it still requires tens of memory accesses to off-chip SRAM chips. Such an amount of memory accesses will result in a serious system bottleneck and thus significantly impair the performance of HiCuts.

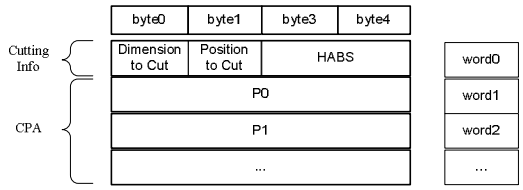

Figure 4: Data-structure of ExpCuts

Thus the design objectives of packet classification algorithm optimized for NPs requires an explicit worst-case bound for search and the elimination of the linear search at leaf-nodes.

Motivations to design such an algorithm are:

♦ **Fix the number of cuttings at internal-nodes:** If the number of cuttings is fixed to 2^w (w is a constant referred as *stride*), the current search space is then always segmented into 2^w sub-spaces at each internal-node. This guarantees a worst-case bound of O(W/w), where W is the bit-width of the packet header.

♦ **Eliminate linear search at leaf-nodes:** Linear search can be eliminated if we "keep cutting" until every sub-space is full-covered by a certain set of rules. The rule with the highest priority in the set is then the final match.

4.2.2 Optimization

Because both motivations tend to result in memory burst due to the fixed *stride* and the minimized *binth* (Elimination of linear search is equivalent to set *binth*=1 in [3]), the main optimization task therefore becomes how to effectively reduce the memory usage.

Note that in HiCuts, in order to maximize the reuse of child nodes, the sub-spaces with identical rules are aggregated by employing pointer arrays to lead the way for search. However, the use of pointer arrays will dramatically increase the memory storage because the size of each array is considerably large when the number of cuttings is fixed. For example, if w=8 is fixed, each internal-node must store 256 pointers for search. If a decision-tree contains tens of thousands internal-nodes, the total memory usage to store the pointer arrays may exceed tens of mega-bytes, which is too large for current SRAM chips [18].

To effectively compress the size of these pointer arrays, some employ bit-string technique to aggregate consecutive pointers [11, 13, 22]: First, an *Aggregation Bit String* (ABS) is used to track the appearance of unique elements in the pointer array, and then compress a sequence of consecutively identical pointers as one element in a *Compressed Pointer Array* (CPA). More specifically, each bit of an ABS corresponds to an entry in the original pointer array, with the least significant bit corresponding to the first entry. A bit in an ABS is set to '1' if its corresponding entry in the original pointer array is different from its previous one, i.e. bit set in an ABS indicates that a different sequence of consecutively identical pointer starts at the corresponding position. Whenever a bit is set, its corresponding unique pointer is appended in the CPA. Accordingly, the n-th pointer in the original point array can be found by: add the first n bits in the ABS to get an index, and then use the index to load the pointer from CPA.

Ideally, all the pointer arrays should be compressed using ABS and CPA. However, loading such a bit string also may cause excessive memory accesses. Fortunately, in our experiments on various real-life rule sets, we found that the number of child nodes of a certain internal tree node is commonly very small: with 256 cuttings at each internal-node, the average number of child nodes is less than 10. This observation is very consistent to the results reported earlier [3, 9, 10]. Such small number of child nodes indicates that the pointer array is very sparse, i.e. the number of bits set in ABS is also sparse. Thus, this observation motivates us to further compress the ABS to effectively reduce the number of memory accesses.

Figure 3 illustrates the *Hierarchical Aggregation Bit-String* (HABS) we proposed to further compress the data structure: The 4-bit HABS is set to "1100" because sub-spaces 4~7, 8~11 and 12~15 all belongs to SS1. If a packet falls in sub-space 9, the corresponding child-node pointer can be located by (1+1+0)<<2+(9&0x11)=5, i.e., P_5.

Define the size of HABS as 2^v, the number of pointers as 2^w, and u=w-v. To compress the 2^w pointers: First, divide the 2^w pointers into 2^v sub-arrays. Then set the bits in HABS to '1' if the 2^u consecutive pointers in its corresponding sub-array are different from the pointers in previous sub-array, i.e. a bit set in an HABS indicates that a different sequence of consecutively identical sub-array of pointers starts at the corresponding position. At the same time, whenever a bit is set, its corresponding sub-array of pointers is appended in the CPA. According to this scheme, the nth pointer in the original point array can be located by: 1) extract the higher v bits of n to get a v-bit value m; 2) extract the lower u bits of n to form a u-bit value j; 3) add 0~m bits of the HABS to get an sub-array index i; 4) use ((i<<u)+j) as the index to load the corresponding pointer from CPA.

In the implementation of ExpCuts, the size of HABS is set to be 16, and HABS is stored together with the cutting information within a single 32-bit long-word (see Figure 4). Such a data-structure can be effectively loaded by the word-oriented SRAM controller on IXP2850 without any excessive memory accesses. Implementation and evaluation of ExpCuts on IXP2850 will be discussed in the next two sections.

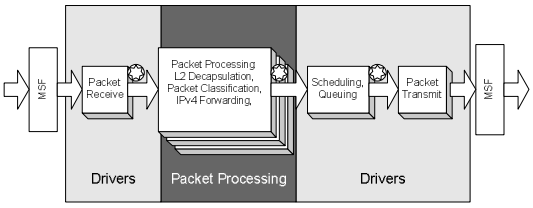

Figure 5: Application Mappings

5 Implementation of Packet Classification on IXP2850

Previous study about packet classification on network processor is either on limited number of rules [23] or just by software simulation [13]. In this paper, an entire packet processing application is implemented on the Intel IXP2850 and the packet classification algorithms added to this application can support large real-life rule sets. The overall application has been tested and evaluated not only by software simulator but also on a hardware platform.

Table 2: Multi-processing vs. Context-pipelining

Task partitioning	Advantages	Disadvantages
Multi-processing	Adding more functionality or scaling to higher data rates simply involves using more MEs in parallel. Packet header and descriptors can be read in once, cached in local memory, and used by all the packet-processing tasks.	Access to data structures shared across multiple packets needs to be synchronized across multiple MEs. Every packet-processing ME must contain code for all packet-processing tasks.
Context-pipelining	Access to data structures shared across multiple packets only needs to be synchronized on the threads running on single ME. Each microengine only needs to contain code for the task runs on it.	Adding more functionality or scaling to higher data rates to an existing task may involve restructuring the code to run on multiple MEs. Per-packet state has to be passed from one ME to the other by shared memory or scratch/NN rings.

5.1 Task Partitioning Methods

There are two general ways to partition tasks onto multiple MEs on the Intel IXP2850 [1, 21]: multi-processing and context-pipelining. In the multi-processing approach, each ME executes all the functions of the application, while in the context-pipelining approach, each function is allocated to a different ME. Advantages and disadvantages of these two general methods are shown in Table 2.

5.2. Mapping Packet Processing in a Typical Application

The application on IXP2850 receives Ethernet frames that carry IPv4 packets. The frames are reassembled into packets and the Layer-2 (Ethernet) headers are removed. Then packet classification and forwarding are performed. Finally, packets are segmented into CSIX c-frames and transmitted to the CSIX fabric. Figure 5 and Table 3 show the application design mapped to an IXP2850 network processor.

Table 3: Microengine Allocation

Task	Receive	Processing	Scheduling	Transmit
#MEs	2	1~9	3	2

Table 4: Optimized Memory Allocations

	SRAM#0	SRAM#1	SRAM#2	SRAM#3
Utilization	56%	0%	47%	31%
Headroom	44%	100%	53%	69%
Allocation	level 0~1	level 2~6	level 7~9	level 10~13

5.3. Memory Allocation

As described in section 3, SRAM and DRAM are two types of commonly used NPU memory. Although Intel IXP2850 network processor supports up to 2GB DRAM, which is 8 times larger than the maximum size of SRAM, the latency of DRAM is about twice as long as that of SRAM. In addition, different access granularities must be considered as well as memory size and latency [16]. On the Intel IXP2850, SRAM is word-oriented (optimized for 4-byte access), while DRAM is burst-oriented (optimized for 16-byte access). Thus to effectively speed up the packet classification speed, all the data-structures for packet classification should be stored in SRAM.

There are 4 SRAM controllers on the IXP2850, allowing independent parallel access. To maximize the performance, we propose to distribute different level of the decision-tree nodes on different SRAM channels according to the bandwidth headroom of each channel. Here the bandwidth headroom refers to the memory bandwidth not utilized by the application without adding the packet classification code. For example, if the stride w=8, the tree depth is then 104/8=13. In proportion to the memory bandwidth headroom of each SRAM channel, the 13 levels of tree nodes are allocated in a way described by Table 4.

5.4. Instruction Selection

To compute the sum of the HABS, indeterminist and time-consuming iterations by traditional RISC instructions are required. It usually takes more than 100 RISC instructions (ADD, SHIFT, AND, and BRANCH) to compute the number of bits set in the HABS. Therefore, without hardware support, the computation burden will become a new performance

bottleneck for the proposed ExpCuts algorithm. Fortunately, IXP2850 provides with a hardware instruction named POP_COUNT, which can count the number of '1's in a 32-bit bit-string within only 3 system cycles [24]. Thus, using POP_COUNT instruction with an ALU AND instruction to mask off undesired bits, the total number of cycles required by HABS computation can be reduced by more than 90% compared to other RISC implementations [16]. This is essential for the ExpCuts algorithm to achieve the line rate.

6. Experiments and Performance Analysis

6.1. Rule Sets and Traffics

Our study focuses on real-life rule sets because experimental results on these rule sets are more convincing than those obtained on synthetic rules [22]. We evaluate all the packet classification algorithms on real-life firewall and core router rule sets. These rule sets are the same as those used in paper [6] and [22]: firewall rule sets are named as FW01, FW02, FW03; core router rule sets are named as CR01, CR02, CR03 and CR04. The largest real-life ruleset (CR04) contains 1945 rules. All rules are 5-dimensional with 32-bit source/destination IP addresses (represented as prefixes), 16-bit source/destination port numbers (represented as ranges) and 8-bit transport layer protocol (represented as discrete values).

6.2. Development Kits

There are two basic programming choices in the Intel Software Developer Kit (Intel SDK): programming in assembly language (Microcode) or programming in C language (MicroC). To make better compatibility with Intel SDK, and to avoid dependency on compiler optimizations, all the application and algorithms are developed using Microcode assembly with the software framework provided by Intel SDK4.0 [25].

To evaluate the performance, the application was tested and run in the IXP2850 Developer Workbench, which offers a *cycle-accurate* simulator of the IXP2850. It provides access to several performance metrics that reflect the actual IXP2850 hardware. The application was also tested on a dual-IXP2850 platform to ensure the code accuracy and compatibility on hardware.

6.3. Space Aggregation Evaluation

The memory requirements with and without space aggregation of the 7 real-life rule sets by ExpCuts are shown in Figure 6. The memory requirement of different rule sets increases along with the increasing number of rules and extent of rule-overlapping. ExpCuts with space aggregation (using HABS and CPT) only requires approximately 15% of the SRAM consumed by ExpCuts without space aggregation. Because the IXP2850 hardware platform only has four 8MB SRAM chips, ExpCuts without space aggregation cannot support CR02, CR03 and CR04 due to their excessive memory usage. In comparison, even for the largest ruleset CR04, ExpCuts with space aggregation only require 11.5MB memory, which can be easily distributed on the four 8MB SRAM chips.

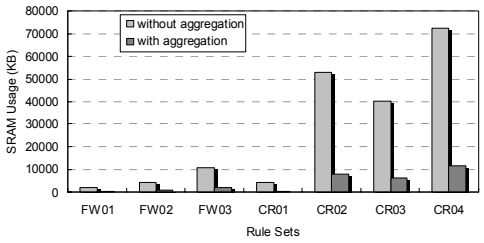

Figure 6: Space aggregation effect

6.4. Relative Speedups

Figure 7 shows the packet classification rates and relative speedups of ExpCuts using the 64-byte TCP packets on the largest ruleset CR04. The results were collected after all optimizations had been applied and all the four channels of SRAM were used to store the decision-tree data-structure. The speedup is almost linear and classification speed reaches up to 7Gbps for 71 parallel threads (9 MEs, each running in 8 thread mode; 1 thread is reserved to handle exceptional packets). Such linear speedups indicate that the memory bandwidths of all the 4 SRAM channels are not full used, i.e., the overall memory accesses required by ExpCuts algorithm are considerably small so that the packet classification rate always increases as more multi-processing threads are added.

6.5. SRAM Channel Impacts

Table 5 shows the SRAM channel impacts of IXP2850. From this table we can see that: 1) even if the overall data-structure is stored in a single channel with 100% bandwidth headroom, the throughput cannot reach 5Gbps. This is because the bandwidth of only one SRAM channel is not enough to support the 13 times of memory accesses; 2) the packet classification speed (throughput) does not increase linearly as more SRAM channels are added. This is not only caused by the difference of SRAM bandwidth headroom (shown in Table 5), but also by the saturation of command request FIFO and SRAM buses [13] [16].

Table 5: SRAM Channel Impacts

Num. of Channels	1	2	3	4
Throughput (Gbps)	4963	5357	6483	7261

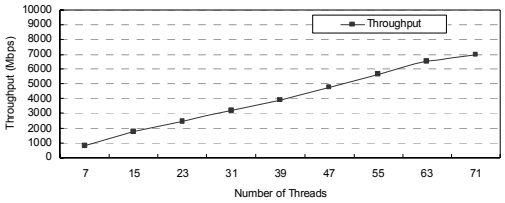

Figure 7: ExpCuts relative speedups

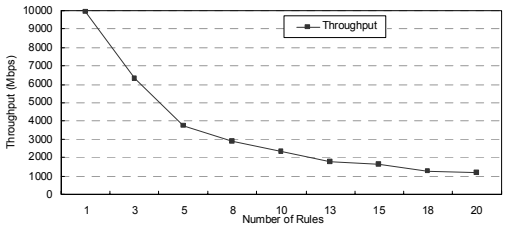

Figure 8: Linear Search Effect

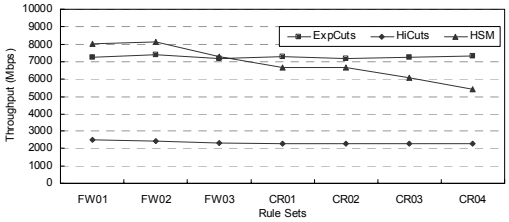

Figure 9: Algorithm Comparison

6.6. Comparison with Other Algorithms

To make objective evaluations, we compare ExpCuts with existing best-known packet classification algorithms: HiCuts and HSM, where HiCuts represents a popular field-dependent search scheme and HSM represents a popular field-independent search algorithm.

As discussed in section 4.1, HiCuts leverages the advantages of both linear search and decision-tree search, reaching considerably fast classification speed with modest memory storage. However, in the worst-case, the linear search done at leaf-nodes needs up to *binth* (in our experiment, *binth*=8) times of memory accesses and each memory access refers to 6 consecutive 32-bits words. Figure 8 illustrates the impact of linear search at leaf-nodes. We can see from this figure that, if the number of rules for linear search is greater than 8, the maximum throughput will be less than 3Gbps.

ExpCuts is also compared with HSM, which is one of the fastest packet classification algorithms [8]. Because HSM only requires $\Theta(\log N)$ memory accesses, and each access only refers to a single 32-bit long-word of SRAM read, the overall memory access is much smaller than HiCuts. Figure 9 compares the throughput of ExpCuts, HiCuts and HSM on the 7 real-life rule sets. We conclude from this figure that: 1) ExpCuts has the best average performance over all rule sets. No matter how large the rule sets are, ExpCuts obtains stable throughput. 2) HSM algorithm is also fast, especially for small number of rules. However, due to the $\Theta(\log N)$ search time, the performance decreases as the number of rules increases. 3) The performance of HiCuts is limited under 3Gbps because of the time-consuming linear search. To obtain practical speed using HiCuts algorithm, the platform might need certain amount of on-chip memory to store all the leaf-nodes for fast data access.

6.7. Summary of Memory Effects

Reducing the effects of memory latency and memory access is critical to the performance of packet classification algorithms on NPs. From the above evaluations, we summarize two performance bottlenecks of NP implementations:

♦ **SRAM Bandwidth:** Memory bandwidth is simply the raw frequency of the memory units measured in Gbps. In packet classification applications, processing one packet needs to read certain amount of memory. For example, linear search requires reading N*6 (N rules, 6-word size) 32-bit words to classify an incoming packet in the worst-case. Because system memory bandwidth is limited, reading fewer amounts of memory words will hence lead to higher performance. In addition, data movement between the SRAM and the Microengines is through the memory push/pull buses, the bandwidth of these buses also limited the maximum performance of NPs.

♦ **I/O Instructions:** The SRAM controller enqueues a command from each command bus in each cycle. If the microengine issues too many I/O operations at a time, the enqueue and dequeue mechanisms in SRAM controller will slow down the I/O operations due to the limitation of its processing capability. Thus, a new system bottleneck yields even if there is still SRAM bandwidth headroom. This means not only the total amount of memory words, but also the number of memory accesses affects the overall system performance.

7. Conclusion

In this paper, we proposed a high-performance packet classification algorithm, ExpCuts, which is based on the well-known HiCuts algorithm but optimized for multi-core network processors. ExpCuts adopts an efficient bit-string aggregation technique to

avoid excessive memory usage. Consequently, without burst of memory usages, the time-consuming linear searches in common decision-tree algorithms are eliminated, and an explicit worst-case search time is achieved. When multi-channel SRAM is provided, ExpCuts can be further optimized by populating different level of tree-node on multiple memory banks.

To objectively evaluate the performance of ExpCuts, we built a complete packet processing application, not just a packet classification building block, on the Intel IXP2850 multi-core network processor. ExpCuts, as well as HiCuts and HSM algorithms are implemented with Intel Microcode assembly, and evaluated both on software simulator and hardware platform. Experimental results show that ExpCuts outperforms the existing best-known algorithms in terms of both packet classification speed and memory usage.

8. References

[1] U. R. Naik and P. R. Chandra, "Designing High-performance Networking Applications," Intel Press, 2004.

[2] T. Sherwood, G. Varghese, and B. Calder, "A Pipelined Memory Architecture for High Throughput Network Processors," Proc. the 30th International Symposium on Computer Architecture, 2003.

[3] P. Gupta and N. McKeown, "Packet Classification Using Hierarchical Intelligent Cuttings," Proc. Hot Interconnects, 1999.

[4] P. Gupta and N. McKewon, "Algorithms for Packet Classification," IEEE Network, March/April, 2001.

[5] M. H. Overmars and A. F. van der Stappen, "Range Searching and Point Location among Fat Objects," Journal of Algorithms, vol. 21, no. 3, 1996.

[6] Y. Qi, B. Xu, and J. Li, "Evaluation and Improvement of Packet Classification Algorithms," Proc. the 1st International Conference on Network and Services (ICNS), 2005.

[7] P. Gupta and N. McKeown, "Packet Classification on Multiple Fields," Proc. ACM SIGCOMM, 1999.

[8] B. Xu, D. Jiang, and J. Li, "HSM: A Fast Packet Classification Algorithm," Proc. the 19th International Conference on Advanced Information Networking and Applications (AINA), 2005.

[9] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet Classification Using Multidimensional Cutting," Proc. ACM SIGCOMM, 2003.

[10] M. E. Kounavis, A. Kumar, H. Vin, R. Yavatkar, and A. T. Campbell, "Directions in Packet Classification for Network Processors," Proc. the 2nd Workshop on Network Processors (NP2), 2003.

[11] J. van Lunteren and T. Engbersen, "Dynamic Multi-Field Packet Classification," Proc. IEEE GLOBECOM 2002, vol. 3, 2002.

[12] D. E. Taylor, "Survey & Taxonomy of Packet Classification Techniques," Technical Report, Washington University in Saint-Louis, USA, 2004.

[13] D. Liu, B. Hua, X. Hu, and X. Tang, "High-performance Packet Classification Algorithm for Many-core and Multithreaded Network Processor," Proc. the 6th IEEE International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES), 2006.

[14] P. Piyachon and Y. Luo, "Efficient Memory Utilization on Network Processors for Deep Packet Inspection," ACM Symposium on Architectures for Network and Communications System (ANCS), 2006.

[15] L. Zhao, Y. Luo, L. Bhuyan, and R. Iyer, "SpliceNP: A TCP Splicer using A Network Processor," ACM Symposium on Architectures for Network and Communications System (ANCS), 2005.

[16] X. Hu, X. Tang, and B. Hua, "High-Performance IPv6 Forwarding Algorithm for Multi-core and Multithreaded Network Processor," Proc. ACM SIGPLAN 2006 Symposium on Principles and Practice of Parallel Programming (PPoPP), 2006.

[17] Intel Corporation, "Intel IXP2850 Network Processor Hardware Reference Manual," 2004.

[18] Intel Corporation, "Intel IXDP2850 Advanced Development Platform System User's Manual," 2004.

[19] B. Carlson, "Intel Internet Exchange Architecture and Applications," Intel Press, 2003.

[20] E. J. Johnson and A. R. Kunze, "IXP2400/2850 Programming," Intel Press, 2003.

[21] M. Venkatachalam, P. Chandra, and R. Yavatkar, "A Highly Flexible, Distributed Multiprocessor Architecture for Network Processing," Computer Networks, vol. 41, no. 5, 2003.

[22] Y. Qi and J. Li, "Towards Effective Packet Classification," Proc. IASTED Conference on Communication, Network, and Information Security (CNIS), 2006.

[23] D. Srinivasan and W. Feng, "Performance Analysis of Multi-dimensional Packet Classification on Programmable Network Processors," Proc. the 29th Annual IEEE International Conference on Local Computer Networks (LCN), 2004.

[24] Intel Corporation, "Intel IXP2400 and IXP2800 Network Processor Programmer's Reference Manual," 2004.

[25] http://www.intel.com/design/network/products/npfamily