Performance Analysis of Multi-dimensional Packet Classification on Programmable Network Processors

Deepa Srinivasan
IBM Corporation
RTP, North Carolina, USA
deepas@us.ibm.com

Wu-chang Feng
Portland State University
Portland, Oregon, USA
wuchang@cs.pdx.edu

Abstract

Multi-field packet classification is frequently performed by network devices such as edge routers and firewalls – such devices can utilize programmable network processors to perform this compute-intensive task at nearly line speeds. The architectures of programmable network processors are typically highly parallel and a single algorithm can be mapped in different ways onto the hardware. In this paper, we study the performance of two different design mappings of the Bit Vector packet classification algorithm on the Intel® IXP1200 network processor. We show that: (i) Overall, the parallel mapping has better packet processing rate (25% more) than the pipelined mapping; (ii) In the parallel mapping, a processing element's utilization can be considerably affected by code complexity, in terms of branching, because of significant time wasted (as much as 40% more) due to aborting instruction execution pipelines; (iii) In the pipelined mapping, multiple memory reads per packet can lower the overall performance.

1. Introduction

Network devices such as firewalls, intrusion detection systems and edge routers utilize packet classification based on multiple fields to detect anomalous traffic, determine attack patterns and provide differentiated services. There are several algorithms that can be utilized for multi-dimensional packet classification. Many of the best ones are based on the Bit Vector [1], a highly parallel classification algorithm that was originally implemented using a custom ASIC.

Programmable network processors are emerging platforms that aim to offer sophisticated packet processing capabilities for use in high-speed networks. Such network processors (NPs) can be utilized by network devices such as edge routers, firewalls, etc. to perform compute-intensive tasks such as packet classifications at nearly line speeds. The architecture of NPs typically consists of multiple processing elements that can execute in parallel to facilitate fast-path packet processing [12, 13, 14]. Each processing element (PE) has multiple hardware thread contexts that enable thread context switches that have zero or minimal overhead.

The binary image that is executed on a particular PE is pre-determined at compile/load time and we can map a single algorithm in different ways onto the PEs. This mapping needs to be determined prior to implementation, i.e. at design time and we call it a *design mapping*. Since different mappings could result in different performance/packet-processing speed, it is imperative that given an algorithm, the best possible mapping is chosen so that the network device that utilizes this does not become a bottleneck.

In this paper, we examine the impact that different design mappings (parallel and pipelined) of the Bit Vector have on performance, while implementing on the Intel® IXP1200 network processor [12]. Our preliminary results show that the parallel design mapping has better packet processing rate than the pipelined mapping, primarily due to the multiple memory reads required per packet in the latter.

The remainder of this paper is organized as follows: Section 2 presents the background concepts of the Bit Vector algorithm and IXP1200 processor. Section 3 describes the two design mappings of the algorithm and implementation details. The experiments, detailed results and analysis are presented in Section 4. We discuss related and future work in Section 5. Section 6 presents the summary and conclusion.

2. Background

In this section, we describe background material relevant to the study presented in this paper. We first describe the hardware architecture of the IXP1200 and follow with a description of the Bit Vector algorithm.

2.1. Intel® IXP1200

The Intel® Internet Exchange Architecture (IXA) network processor family [15] is provided to universi-

ties through the Intel® IXA University program [18], along with the required development environment, for use in research projects. Hence, we choose the IXP1200, which is part of the IXA family, as our platform for study. Complete description of the hardware architecture is available from other sources [10, 11] and we present here details that are relevant to our study. The IXP1200 is an integrated network processor, comprised of a single StrongARM processor, six microengines (individual processing elements), standard memory interfaces and high-speed bus interfaces. Figure 1 shows the block diagram of the IXP1200 architecture [20].

On the microengines, instructions are executed in a five-stage pipeline. Each microengine has four hardware-assisted threads of execution. All threads in a particular microengine execute code from the same instruction store on that microengine. Communication between threads in a microengine is done using registers; communication across microengines is done using the shared SRAM and SDRAM.

There are two basic programming choices in the Intel® Software Developer Kit – programming in microcode/assembly language using the microACE architecture [19] or programming in Microengine C (also known as microC) [17]. The latter is a C-like language that includes features for programming on the IXP1200. The code that we use in this study implements microblocks (that run on microengines) using microC. This SDK was provided by Intel on a limited basis.

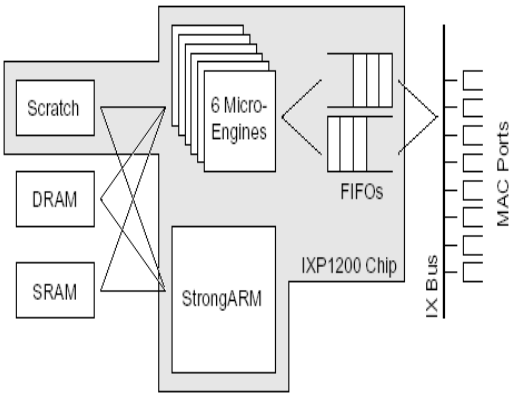

Figure 1 (from [20]): Block diagram of the Intel® IXP1200 network processor

2.2. Bit Vector algorithm

As is typical with NPs, the hardware architecture of the IXP1200 is highly parallel. Hence, an algorithm that is capable of performing the various stages of classifying a packet in parallel is well suited for implementation on the IXP1200. Several packet classification algorithms [4, 5, 7, 8, 9] exist in current literature, each with different space-time tradeoffs [6]. From figures 1 and 2, we note the striking similarity of the architecture of an implementation of this algorithm and that of hardware architecture of a network processor. Thus, it is natural that the Bit Vector algorithm can be mapped onto an NP such as the IXP1200 easily. Hence, we consider the Bit Vector algorithm in this paper. A key feature of the IXP1200 is asynchronous memory access. A microengine thread can issue a memory request and continue processing. The completion of the request can be asynchronously reported back to the microengine thread. This facilitates hiding memory latency – while one thread is waiting for a memory request to complete, another thread on the same microengine can execute.

3. Design Mappings

This section presents the implementation details of the Bit Vector algorithm and the two design mappings studied in this paper. While there are various possible mappings, we limit our study to two that are significantly different from each other in terms of microengine allocation for individual tasks.

3.1. Bit Vector implementation

Packet classification algorithms typically consist of two phases: a pre-processing phase, which computes and builds the data structures in memory from input rules; and a classification phase, in which the data structures are looked up with the packet header values to determine the matching rule. For purposes of this study, we do not consider the performance of the pre-processing phase since this is not done in the fast path, but rather by the slower core processor.

For our implementation, we choose the following data structures. The set of non-overlapping intervals of the range of values in the input rules are represented by binary tries for each dimension. A two-dimensional byte array represents the bit vector where each element contains the matching rules for each range for each dimension. Complete explanation of the algorithm's working can be found from other sources [1, 22]. While it is possible to use different data structures that will produce code with different performance characteristics, for our preliminary analysis, we use the data structures mentioned here and keep them constant in both the design mappings to obtain a relative performance comparison. This is further discussed in section 4.

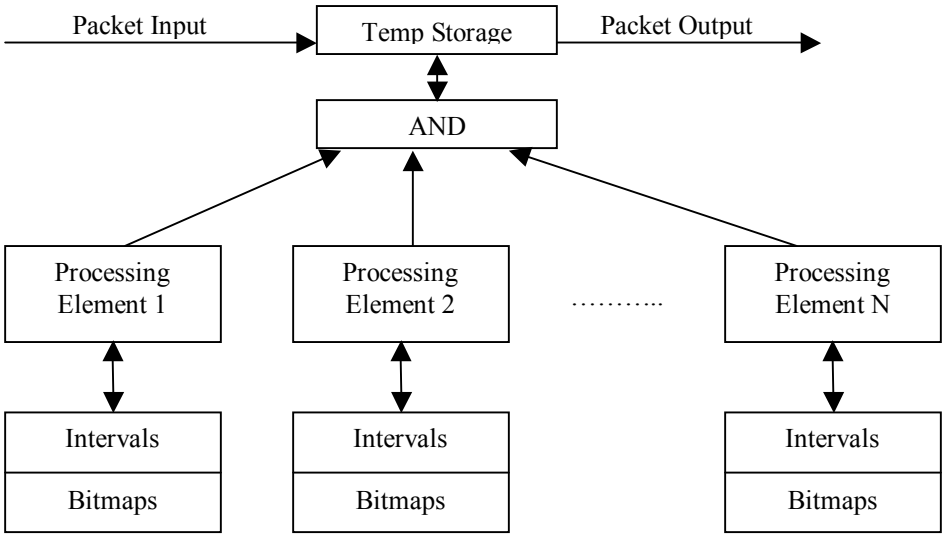

Figure 2: Block diagram of a parallel implementation of the Bit Vector algorithm (from [1])

3.2. Mapping the algorithm to the IXP1200

Recall that the code executed by each microengine is pre-determined at compile and link time. Hence, we need to partition the responsibilities prior to implementing and running the code on the IXP1200. The two standard functions that will be required are receiving and transmitting packets. In all our implementations, we allocate microengine 0 and microengine 5 for receiving and transmitting packets respectively. That gives us four microengines to use in the classification phase. The implementation of the classification phase can be done in different ways. The following sections list two such mappings. We use the following terminology: microengines that perform classification, receive and transmit functions are called *cls*, *rcv* and *xmit* microengines respectively.

3.2.1. Parallel Design Mapping. In this approach, all the classification steps for a single packet are performed by a single hardware thread in one microengine, as illustrated in Figure 3. We call this the parallel mapping or simply *parallel.*

The detailed division of responsibility and inter-microengine communication is as follows. Microengines 0 and 5 receive and transmit packets respectively. We reuse the code from the microC microACE sample for these, with minor modifications. Each of the four hardware threads on microengine 0 receives packets from a single port and queues them for use by microengines 1 through 4. The queues used for this are circular and are placed in SRAM. Since there are four *cls* engines and all four perform the full classification for a packet, we create a queue for each of the *cls* hardware threads. Each of the four *rcv* threads rotates through the four queue numbers sequentially. There are 128 entries in each queue and each entry occupies 2 words or 8 bytes of SRAM memory. The threads in the *cls* microengines wait for a new entry in their respective queue. Once an entry is available, it reads the appropriate packet headers, performs the classification and queues it for transmission by one of the *xmit* threads. Similar to the *rcv* threads, there are 4 *xmit* threads that service the 16 *cls* threads. Hence, each *xmit* thread rotates through transmitting packets from the 4 queues that are allocated to it.

3.2.2. Pipelined Design Mapping. This design mapping is illustrated in Figure 4. The first step (lookup in the *P-set*) of classification for a packet is done by multiple microengines.

Each microengine performs the lookup for one particular dimension. For example, microengine 1 determines the range in the *P-set* for dimension 1; microengine 2 determines the same for dimension 2 and so on. At any given time, a single *cls* engine can perform a 1-dimension *P-set* lookup for 4 packets. The results of these lookups are sent to a different microengine, which then retrieves the appropriate bit vectors and performs the logical AND operation. We call this the pipelined mapping or simply *pipelined.*

3.3. Verification of the implementations

We first implemented and verified the algorithm in C and then ported it to microC, applying the two di-

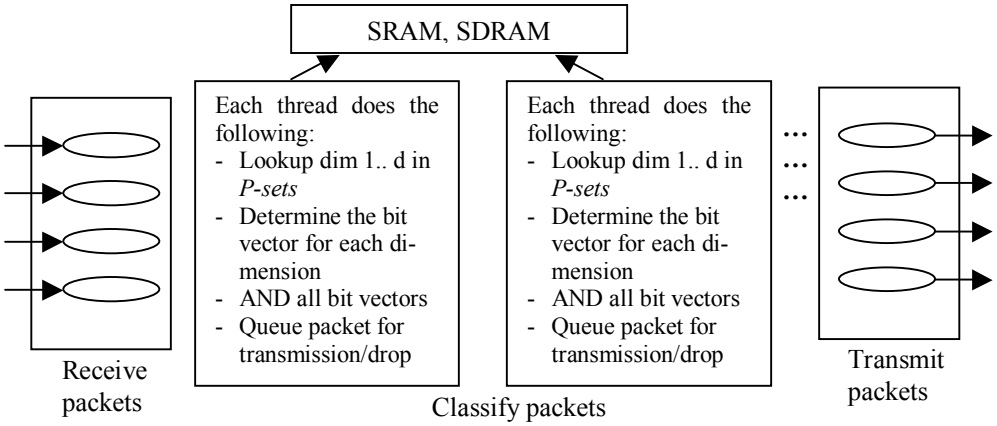

Figure 3: *Parallel* design mapping of the Bit Vector algorithm

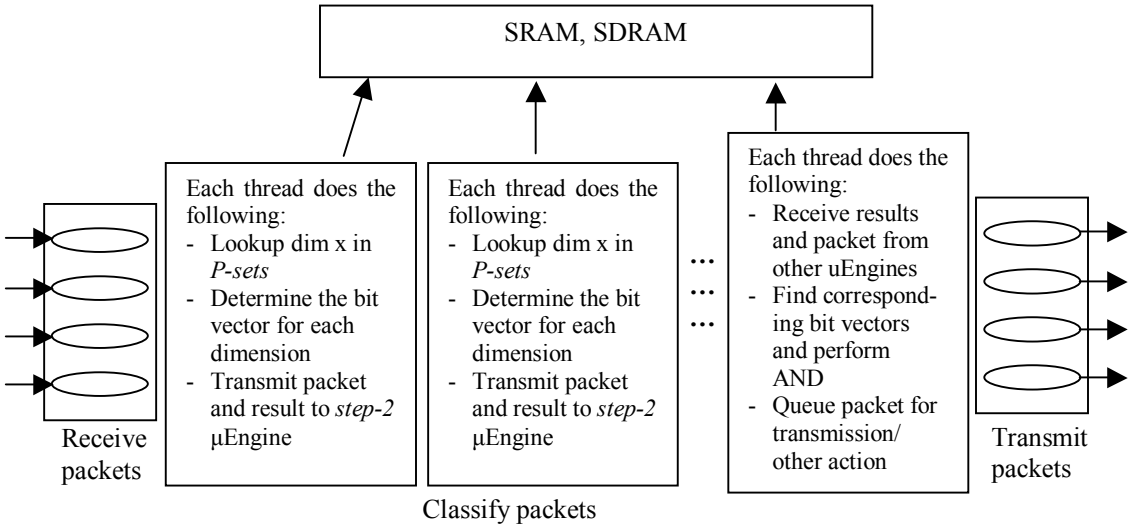

Figure 4: *Pipelined* design mapping of the Bit Vector algorithm

-fferent design mappings. To verify correctness of the implementation on the IXP1200, we gradually increased the number of threads and microengines executing the code until all the microengines were being utilized. We tested the code with sample rulesets and packets and verified the correctness of the output packets using the logging facility provided by the IXP1200 simulator. The code was then run continuously in the simulator for 8 hours and we verified that the simulator did not crash and that packets were received and transmitted at steady rates.

3.4. Other considerations

3.4.1. Management application. In this paper, we study only the performance of the algorithm on the microengines which are the main components in the IXP1200 that offer the capability to perform wire-speed packet processing. When implementing for real-world use, there is the existence of a management application that runs on the main processor of the host system. The presence of such an application has implications for the overall performance of the IXP1200 system since there will be contention to shared data structures in memory. This paper does not consider the performance implications due to the introduction of a

management application or any other source of updates to data structures in memory.

3.4.2. Simulator. The algorithm is implemented in microC and is tested and run in the IXP1200 Developer Workbench which offers a cycle-accurate simulator of the IXP1200. This environment provides access to several performance metrics that reflect the actual IXP1200 hardware. The code implemented for this study can be made to run on the actual IXP1200 hardware. However, running microC microACE code on the hardware was not supported by Intel at the time of this study and hence we limit our study to using the simulator.

3.4.3. Scalability of the Bit Vector algorithm. The size of rulesets used for packet classification varies depending on its purpose. For example, corporate intranets have approximately 150 rules, whereas large ISPs may have around 2400 rules [5]. The Bit Vector algorithm is well-suited for medium sized rulesets [1], with around 512 rules. From preliminary study of the basic Bit Vector algorithm, it is evident that it does not scale well for large rulesets due to the large memory requirement for such rulesets. Several optimizations have been proposed [1, 7] to the basic algorithm that enable more efficient use of the data structures in memory. This paper deals only with the basic Bit Vector algorithm since it studies the behavior and the relative performance of different design mappings of a particular algorithm. Also, the same fundamental idea of parallelism is present in the extended versions of the algorithm. Since this does not affect the performance evaluation in this study, we do not attempt to modify the basic algorithm to achieve better scalability.

4. Experiments and Results

This section presents the performance data, collected using the IXP1200 Developer Workbench, from executing the two design mappings of the Bit Vector algorithm.

4.1. Ruleset

We use a ruleset (shown in Table 1) with 4 rules, each rule specifies a 3-dimensional criteria and the width of each dimension is 4 bits. All the rules have the action set to "Allow", to measure worst-case performance. (For real world rulesets, the number of dimensions ranges from 1 to 5; number of rules ranges from 100s to 1000s; the width of the field takes values 4 (for port numbers) and 128 (for IP addresses)).

Table 1: Ruleset used in this study

Rule	Field 1 (source IP address)	Field 2 (destination IP address)	Field 3 (destination TCP port)	Action
R_1	(10, 11)	(2, 4)	(8, 11)	Allow
R_2	(4, 6)	(8, 11)	(1, 4)	Allow
R_3	(9, 11)	(5, 7)	(12, 14)	Allow
R_4	(6, 8)	(1, 3)	(5, 9)	Allow

While the performance of the algorithm will vary depending on the size and characteristics of the ruleset and the traffic being classified, this paper studies the comparison of two design mappings, given that these factors are constant. The results presented here (the performance of the Bit Vector algorithm itself) cannot be directly generalized and further study will be needed to observe the performance of the Bit Vector algorithm for a large input ruleset against various traffic patterns. This would require implementing and studying Bit Vector algorithm using incremental reads [1], which scales better for large rulesets.

4.2. Simulator configuration

The IXP1200 Developer Workbench allows the user to specify different system configuration parameters that are used by the simulator. For the experiments presented here, we use the basic configuration available – an IXP1200 chip with 1K microstore that has a core speed frequency of 165.890 MHz. We can also specify configuration settings for the IX Bus Device simulator which controls how packets are sent and received from the simulator. For our use, we choose a device with 8 ports, each with a data rate of 100 Mbps and receive and transmit buffer sizes of 256 each. Since we have only one microengine (4 hardware threads) performing the receive operation, we support only four ports. Hence, we configure the simulator to send packet streams to only ports 0 through 3 of the device. We use 4 independent streams of 64-byte TCP/IP packets for the experiment. Each of these streams has packet header values that match one of the rules in the input ruleset. To compare the performance of the different mappings, we run each of the implementations in the simulator, until 75000 packets have been received by the IXP1200 from the bus. We then record the various performance metrics and use them for our analysis.

4.3. Performance results

We collected and observed several metrics such as total number of microengine cycles and total number of IX bus cycles spent to process all the packets, total throughput of the IXP1200, individual microengine utilization (% time executing, aborted, idle) and the memory access rates. Figures 8 through 10 show the comparison of key metrics between the two design mappings.

4.4. Analysis

While comparing the performances of the two design approaches, it is important to keep in mind the allocation of the microengines in each:

- In both mappings, microengines 0 and 5 perform the receive and transmit functions respectively.
- In *parallel*, microengines 1, 2, 3 and 4 perform the full classification functions.
- In *pipelined*, microengines 1 and 2 perform lookup for IP addresses; microengine 3 performs lookup for the transport layer port number (or protocol); microengine 4 performs the step 2 of the algorithm – it combines the results from the previous lookups to determine the matching rule.

Overall Analysis

Figure 6 shows the receive and transmit rates of the IXP1200 for the two design mappings; Figure 7 shows the packets sent/received ratio. In *pipelined*, we split the various steps in processing a packet across microengines, since it seems to be an ideal mapping for the algorithm. But the packet processing speed is reduced by 25% in *pipelined* than in *parallel* (seen in Figure 7). This is primarily because: (i) In *parallel*, the SDRAM access to read the packet header for classification occurs only once, by a single hardware thread of the microengine that is performing the entire classification for that packet; (ii) In *pipelined*, splitting the lookups in step 1 of the algorithm across microengines for a single packet, causes three hardware threads on different microengines (1, 2 and 3) to access the packet header in SDRAM for that packet, thus increasing the memory access time required to process one packet by three times. In network processor architectures that have alternate faster mechanisms for inter-processing element communication (such as next-neighbor registers in the IXP2xxx family [12]); these can be utilized to avoid the multiple memory reads.

As mentioned in Section 3.1, it is possible to use different data structures while implementing the Bit Vector algorithm. For example, instead of using binary tries for the *P-sets*, we could use multibit tries. The impact of such a data structure will depend on the characteristics of the input ruleset. We expect that there will be some extent of branching in the code, although it may be lower than when using binary tries. Although multibit tries reduce the number of memory lookups required, they increase the overall memory required. On a network processor platform, fast memory is scarce and large memory available is much slower. Hence, such a data structure may not be suitable for use, depending on the size of the input ruleset.

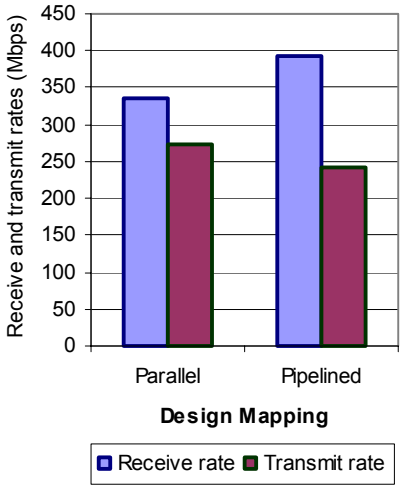

Figure 6: Receive and transmit rates

Microengine Utilization

We focus here on one aspect of the microengine utilization: the microengine aborted time. This is the percentage of a microengine's total time that was wasted due to instructions in its pipeline being aborted.

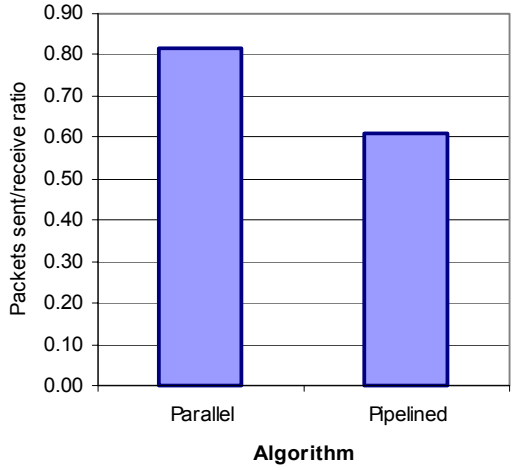

Figure 7: Packets sent/receive ratio

From Figure 8, we see that the aborted time for the classification engines is lower in *pipelined* than that of *parallel*. It was observed from the simulator that microengine aborted time is typically due to branch instructions. This is reflected in the results. In particular, the aborted time for microengine 4 (which performs the full classification in *parallel* and only step 2 of the classification in *pipelined*) in *pipelined* is approximately 60% that of *parallel*. This is because in step 2 of the algorithm, we perform simple operations such as reading the bit vector from memory and performing an AND operation.

Thus, choosing simpler algorithms or designing algorithms that result in lower number of branch instructions is important in improving microengine utilization. While programs are typically optimized to hide memory latency (facilitated by the underlying hardware) to increase microengine utilization, this is an additional factor to consider while implementing on network processors.

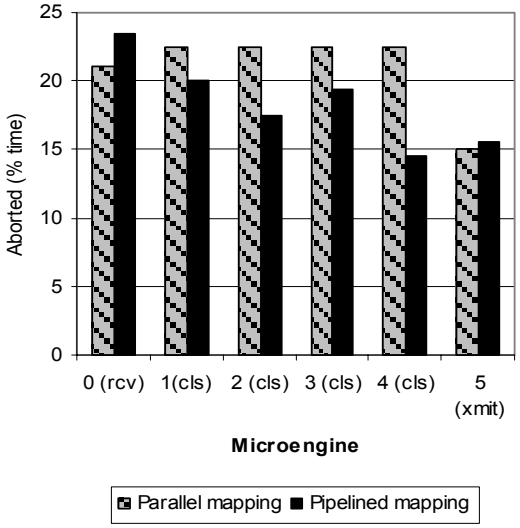

Figure 8: Microengines aborted time

The classification engines in *pipelined* exhibit more idle time than those in *parallel* (Figure 9). In *pipelined*, a new packet cannot be handled by microengines earlier in the pipeline until there are available inter-microengine buffer entries. These entries are freed only when the entire processing for that packet is completed by all microengines and the packet has been queued for transmission by microengine 4.

We summarize our analysis by saying that overall, *parallel* performs better than *pipelined*. Depending on implementation factors in each mapping, the microengines aborted and idle times vary affecting overall microengine utilization.

5. Related Work

There has been a recent study aimed towards designing packet classification algorithms specifically for use on NPs [3]. It presents a detailed analysis of ty-

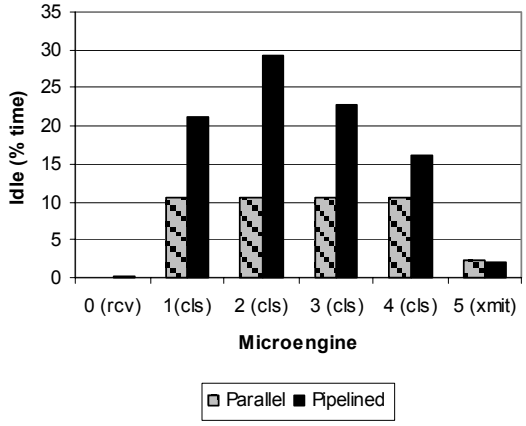

Figure 9: Microengines idle time

-pical rulesets characteristics. Such analyses can b extremely useful for designing the input ruleset used for performance evaluation of an algorithm's implementation. Guidelines for designing classification algorithms for network processors have also been presented [3] and it will be interesting to study the performance of different design mappings of such algorithms and observe the variance.

Other studies [16, 20] have focused on using programmable network processors to implement networking services. NP-Click [21] has been proposed as a programming model for the IXP1200. The effect of design mappings on the programming model is a possible topic for further study.

6. Conclusion

Multi-dimensional packet classification is an important function performed by network devices such as edge routers, firewalls and intrusion detection systems. Such devices can utilize programmable network processors to implement a multi-dimensional classification algorithm and perform this compute-intensive task at line speeds. The fact that a single algorithm can be mapped in different ways onto a network processor yielding different performance results motivates the study presented in this paper. The highly parallel nature of the hardware architecture of the IXP1200 is well suited for a parallel search packet classification algorithm. Hence, we study the performance and behavior of such an algorithm - the Bit Vector algorithm. While actually implementing an algorithm on network processors, we deal with details such as queuing and

memory accesses that vary depending on the design mapping. Thus, choosing the best possible design mapping of an algorithm is critical for achieving optimal performance.

We have presented performance results and analysis from running two different design mappings (parallel and pipelined) of the Bit Vector algorithm on the IXP1200 network processor. Several other design mappings of the algorithm are possible. For example, a single packet can be completely processed within a single microengine, with different functions performed by multiple threads.

The parallel design mapping has a higher packet processing rate than the pipelined mapping, primarily due to multiple memory reads per packet in the latter. We observe that an important performance metric to be considered is the microengine aborted time, which occurs typically due to branch instructions in the code. One observation is that the aborted time of a microengine which performs simpler operations in the pipelined mapping than the corresponding one in the parallel mapping is 60% lower. This indicates that algorithms which have frequent complex branch decisions will perform worse than those that have simple instructions. These results can be used while designing packet classification algorithms or other class of algorithms for implementation on programmable network processors.

7. Acknowledgments

We are grateful to Erik J. Johnson, Senior Network Software Engineer, Intel Corporation for providing the microengine C code libraries used in this study and for providing technical advice as we progressed in our research.

8. References

[1] T.V Lakshman, D. Stiliadis. "High-speed policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching". Proceedings of ACM Sigcomm, pages 191-202, September 1998.

[2] P. Gupta and N. McKeown. "Algorithms for Packet Classification". IEEE Network Magazine, 2001.

[3] M. Kounavis, A. Kumar, H. Vin, R. Yavatkar and A. Campbell. "Directions in Packet Classification for Network Processors". 9th International Symposium on High-Performance Computer Architecture, February 2003.

[4] V. Srinivasan, G.Varghese, S.Suri and M. Waldvogel. "Fast and Scalable Layer Four Switching". Proceedings of ACM Sigcomm, pages 203-14, September 1998.

[5] M.M. Buddhikot, S. Suri, M. Waldvogel. "Space Decomposition Techniques for Fast Layer-4 Switching". Proceedings of Conference on Protocols for High Speed Networks, pages 25-41, August 1999.

[6] A. Feldmann and S. Muthukrishnan. "Tradeoffs for Packet Classification". Proceedings of Infocom, vol. 3, pages 1193-202, March 2000.

[7] P. Gupta and N. McKeown. "Packet Classification on Multiple Fields". Proc. Sigcomm, Computer Communication Review, vol. 29, no. 4, pp 147-60, September 1999, Harvard University

[8] P. Gupta and N. McKeown. "Packet Classification using Hierarchical Intelligent Cuttings". Proceedings of Hot Interconnects VII, August 99, Stanford. This paper is also available in IEEE Micro, pp 34-41, vol. 20, no. 1, January/February 2000.

[9] F. Baboescu and G. Varghese. "Scalable Packet Classification". Proceedings of ACM Sigcomm, pages 199-210, August, 2001.

[10] IXP1200 Hardware Reference Manual, published by Intel Corporation.

[11] E. Johnson and A. Kunze. "IXP1200 Programming", published by Intel Press, ISBN 0-9702846-7-5.

[12] Intel Corporation. Intel Network Processors product information. http://www.intel.com/design/network/products/npfamily.

[13] IBM Corporation. IBM PowerNP™ product information. http://www-3.ibm.com/chips/products/wired/products/network_processors.html.

[14] Motorola Inc. C-Port™ Network Processors http://www.motorola.com/networkprocessors.

[15] P.N. Glaskowsky. "Intel beefs up networking line". Microprocessor Report, March 2002.

[16] T. Spalink, S. Karlin, L. Peterson. "Evaluating network processors in IP forwarding". Technical report 626-00, Princeton University, November 2000.

17] Intel Corporation. "Intel Microengine C Compiler Language Support Reference Manual", March 2002.

[18] Intel Corporation. "Intel IXA University Program". http://www.ixaedu.com.

[19] Intel Corporation. "Developing ACE for Intel IXP1200 Reference Manual", March 2002.

[20] T. Spalink, S. Karlin, L. Peterson, Y. Gottlieb. "Building a Robust Software-based Router using Network Processors". Proceedings of the 18th ACM Symposium on Operating Systems Principles, pages 216 – 229, October 2001.

[21] N. Shah, W. Plishker, K. Keutzer. "Np-Click: A Programming Model for the Intel IXP1200". 9th International Symposium on High-Performance Computer Architecture, February 2003.

[22] D. Srinivasan. "Performance Analysis of Packet Classification Algorithms on Network Processors". Masters Thesis, Department of Computer Science & Engineering, OHSU, May 2003
