Branch Prediction for Network Processors

David Bermingham, Zhen Liu, Xiaojun Wang
Network Innovations Center
School of Electronic Engineering
Dublin City University
Ireland
Email: David.Bermingham@eeng.dcu.ie

Bin Liu
Institute of Networking
Dept of Computer Science and Technology
Tsinghua University
Beijing, P.R China
Email: liub@tsinghua.edu.cn

Abstract— **Meeting the future requirements of higher bandwidth while providing ever more complex functions, future network processors will require a number of methods of improving processing performance. One such method will involve deeper processor pipelines to obtain higher operating frequencies. Mitigation of the penalty costs associated with deeper pipelines have achieved by implementing prediction schemes, with previous execution history used to determine future decisions. In this paper we present an analysis of common branch prediction schemes when applied to network applications. Using widespread network applications, we find that unlike general purpose processing, hit rates in excess of 95% can be obtained in a network processor using a small 256-entry single level predictor. While our research demonstrates the low silicon cost of implementing a branch predictor, the long run times of network applications can leave the majority of the predictor logic idle, increasing static power and reducing device utilization.**

I. Introduction

As the Internet has evolved, the functions required by a modern Network Processor (NP) have developed from simple packet forward to tasks requiring complex data processing such as packet classification and network security, while also meeting the higher bandwidth requirements of an expanding network. Typically, solutions such as increasing parallelization and hardware offloading have been employed as a means of meeting these requirements.

Firstly, parallelization provides a means of massively increasing performance via additional Process Engines (PEs), while also retaining the flexibility vital to network processor architectures. Secondly, hardware acceleration provides another mechanism of reducing latency and increasing processing throughput, with computational intensive tasks such as encryption or deep packet inspection implemented on dedicated hardware. Both solutions present difficulties which must be examined. While increased parallelization provides additional resources, the demand placed on the memory and IO subsystem also scales, along with the difficulty associated with programming a massively parallel system (i.e. task partitioning, load balancing). On the other hand, hardware offloading presents a major challenge to one of the original design considerations of a network processor, namely flexibility. Ultra-high performance accelerators will tend implement optimized versions of a particular algorithm, reducing the flexibility afforded to the programmer as well as making the incorporation of future developments and improvements difficult.

Therefore, it is with this in mind that we examine if some of the future processing requirements can be met by improving the performance of the RISC PEs instead of increasing the number of PEs implemented or implementing hardware acceleration. Previous works within the NP design space [1, 2, 3] have focused on examining the effectiveness of more complex processor design techniques such as superscalar or cache, with commercial NP architectures such as Cavium OCTEON[4] incorporating dual-issue PEs along with a coherent cache memory. Following this trend, it can be expected that more complex design techniques will increasingly be needed to meet future NP requirements. One such method of increasing PE performance is via a deeply pipelined architecture.

Deeper instruction pipeline allows additional performance to be extracted by dividing instructions into smaller and faster tasks which can then operate in parallel. For reasons of the costs associated with the parallel architecture and need for on-chip peripheral components, it is typical to see relatively shallow pipelines within the PEs used in NP designs. By implementing deeper processor pipelines, additional performance to be extracted from a NP design, providing the branch penalty associated deeper processor pipelines can be minimized. This branch penalty occurs when a program flow instruction such as a conditional branch alters the program counter, requiring a delay while the condition is evaluated. While a shallow pipeline can absorb this penalty via pipeline stalls, the performance loss of such solutions would be prohibivately expensive in a deeply pipelined processor. Within general purpose processor, the most common solution to the problem has been to implement some form of branch predictor which attempts to calculate the outcome of a branch without having to insert stall instructions. The objective of this paper is to examine if such prediction techniques can be implemented in network processors as a means of extracting addition NP performance via deeply pipelined.

The rest of this paper is organized as follows; In Section II we present a brief overview of branch prediction. Section III details the simulation framework employed, with Section IV detailing the branch characteristics, predictor performance and limitations of these solutions. Finally, a summary and conclusion are presented in Section V.

II. Branch Prediction

With branch operations comprising a large amount of executed instruction [6], the number of stall cycles required constitutes a sizable lost processing time. Indeed, the work in [6] found for the MIPS architecture it was shown branch induced NOPs comprised 8% of the total instruction executed. In general, there are two types of branch prediction mechanism available. The first, static prediction [7], attempts to utilize a heuristic approach at compile time as a means of determining if a branch will be taken or not, e.g assuming forwarding branches not are taken while backward branches are taken. More complicated branch predictions schemes attempt to gather run-time information when making decisions.

Dynamic predictors retain a history of previous branch outcomes which are then used to determine if a future branch prediction will likely be taken. Ideally, the histories of *n* previous branches are maintained in an array of *n* * 2-bit sequential saturating counters. Together, these counters form the Pattern History Table (PHT). Using the result of previous branch evaluations, the saturating counters count from strongly not-taken to strongly taken. Addressing the PHT is achieved using either the branch address (B*imodal*) or via a global Branch History Table (BHT) (*Gag* [8]), or a combination of both *Gshare* [9]. An example of a *Gshare* based dynamic predictor is shown in figure 1, with the branch and program counter address used to create an XOR'ed index into the pattern table. Since different branches may map to the same entry in the PHT, a number of solutions have been proposed which attempt to solve this interference issue, such as the *Gap* predictor which implements *m* PHTs in parallel. In [10], a number of highly parallel architectures where proposed, with the *PAg* scheme implementing a per-address BHT and the *PAp* predictor implementing both a per-address BHT along with *m* PHTs. Along with these schemes, a combining approach [9] can be used. However, when compared to the small cache-less area cost of a PE these three solutions require a large amount of transistors to implement. Further information on branch prediction schemes can be found in [11, 12].

Within network processing design space, work presented [2] has briefly examined on the topic of branch prediction for network processors. It was demonstrated how performance increases of up to 15.7 % could be achieved. However the predictors examined in this work are prohibitively expensive for used in a PE. Although the exact silicon cost will vary from one technology to another, an approximate cost for implementing a branch predictor on a RISC-type PE is shown in appendix A.

III. Experiment Methodology

Using the ARM execution unit found in the Simplescalar toolset [14], we implement a simulation framework which attempts to more closely model the demands of a network processor. Shown in figure 2, the simulation model removes operations which would not be seen in real network applications such as file IO or system calls. Packets buffered within the interface unit before being moved to packet memory. Once processing is complete, packets are transferred back to the interface unit for egress. By removing such operations from the simulated applications, we ensure that only those branches core to application functionality are included in any simulation results.

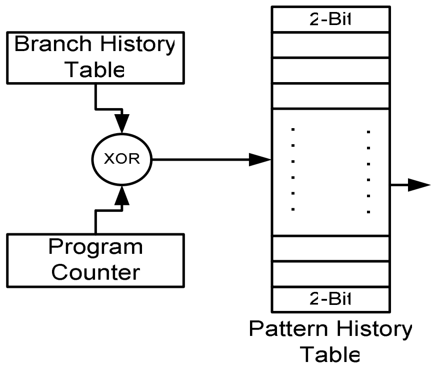

Fig.1. Gshare type branch predictor

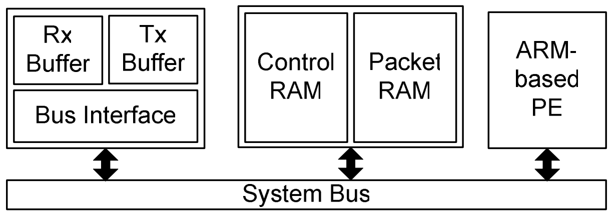

Fig.2. Simulation Framework

TABLE I
Target Applications

Applications	Algorithm
Forwarding	LC-Trie, Radix, Hash
Classify	RFC, EGT-WPC, Hicuts
Queue	DRR
Metering	TBM, TrTCM
IPsec-Encryption	AES-CBC, CAST-CBC
IPsec-Authentication	SHA-1, MD5
Error	CRC32, Reed-Solomon

In all, 16 network applications are evaluated. The applications simulated are summarized in Table I. Broadly speaking, network applications are divided into Header Processing Applications (HPA) and Payload Processing Applications (PPA). While header applications such as IP forwarding will tend to use data such as addresses or packet length during conditional operations, payload processing tasks such as IP encryption will tend to function using only the payload length. A detailed overview of these applications and algorithms can be found in [1 – 3, 5].

Network traces are obtained from [15]. However since these traces must clear sensitive information such as IP address or payload data, we derive semi-synthetic traces from these seeds, with random payload data inserted along with mechanism for rebuilding packets flows. The use of valid random IP address ensures that the branch predictors are tested for the worst case scenario, since a trie-based structure is highly conditional. To ensure a broad analysis of predictor behavior, three different traces are selected which comprise the packet variation seen on IP networks. While the OC-12 AMP trace contains a large proportion of large packets (~60% > 1000 Bytes), the slower OC-3 TXS trace contains almost entirely small packets (~80 < 64 Bytes). Along with the more average OC-192 PSC trace, this length distribution will determine if packet length and predictor performance are related.

IV. Experimental Results

Depending on the underlying target architecture, previous analysis of network processor workloads [1-4] has found that branch instruction comprise between 7.2% and 17% of applications. However, the dual nature of network applications is hidden by an average view, with header applications requiring conditional operations as a means of traversing a trie or decision structure, while payload applications tending to implement conditional operations as a means of processing control loops, e.g. *process while(offset !=packet length)*. By examining the conditional operations executed we can see that for payload tasks, a single branch type will comprise the vast majority of conditional operations. On the other hand, routing or classification tasks require multiple branches to process a single packet this branch behavior translates into higher branch interference.

Using a 512-entry *Gshare* predictor, we see in Table II that header applications such as RADIX can have up to 30% interference, highlighting the fact that about a third of the 512-entries have more than one branch mapped to this location. At the same time, 27.7% of table entries are not used during execution. With NP applications running for long periods without change, an optimized solution should minimize this idle logic. Solution such as better hash indexing may therefore improve performance of a *Gshare* predictor, possibly optimizing the hashing function for small input sequences along with more compact tables. With static power comprising an ever more important component of digital design, the long run-time of network processor applications can result in significant device under-utilization.

A. *Branch Pattern History Table Size*

As was previously discussed in section II, branch prediction schemes for network processors are only viable if the cost of implementing the hardware is significantly lower than the cost associated with the PE. To examine this, we analyze the prediction rates for a simple bimodal predictor as the table size is increased. For space, header (HPA) and payload (PPA) processing applications are averaged together. From table III it can be seen that a small table footprint and good performance can be achieved with a Pattern History Table of either 256 or 512-entries, with a 512-entry PHT providing only 0.46% increase over the 256-entry table. Since these applications will tend to be optimized for processing bandwidth, branch instructions

TABLE II
BRANCH BEHAVIOR

Task	Branch Distribution	Interference	Entries Not Used
AES	74.3%	8.3%	78.7%
MD5	99.8%	0.6%	50.1%
EGT	25.8%	13.6%	59.2%
RADIX	31.2%	30.5%	27.7%
LCTRIE	33.9%	16.4%	50.2%

TABLE III
PREDICTOR PERFORMANCE

Application	Address Hit Rate %		
	HPA	PPA	AVG
Trival (Always Taken)	76.89	68.07	86.97
Bi-256	88.31	94.99	91.06
Bi-512	89.04	94.99	91.49
GAg – 256	93.38	95.05	94.06
GAg – 512	91.50	95.75	93.25
Gshare – 256	94.61	96.23	95.28
Gshare – 512	93.66	97.64	95.30
GAp – 512/8	93.00	94.78	93.73
PAp – 128/256	93.10	96.78	94.62
PAg – 4/4/256	90.10	95.52	92.33

will only represent control operations, such as the loop while counter is less than the packet length shown above. In this case we can see that the random distribution of packet lengths seen in IP traffic does not affect branch prediction performance in network applications. An example of this can be seen in the performance of an application such as the AES algorithm which utilizes as 16-Byte block size. Since the algorithm will have to execute at least three times for every packet (40-byte minimum packet), the prediction counters will tend towards to strong taken, with only the final loop miss-predicting.

The small application code associated with network applications results in predictor saturation above 1024 entries, significantly below the 16K-entries found in [12] to be required for general purpose processing. However there remains a sizable performance difference between HPA and PPA tasks, with a similar predictor providing almost 7% less correct predictions when executing RADIX (88.9%) routing compared to any PPA task (>95.8%).

Utilizing a 256-entry PHT, we examine whether a Per-Address scheme such as PAp can provide a means of increasing PPA prediction rates. From figure 4 it can be seen that above 8 BHT entries, the per-address PAp scheme does provide a mechanism for increasing HPA prediction rates. With 128 (n=7) first level entries allowing for the performance difference between HPA and PPA task to be narrowed to ~1%.

Finally, using an optimum PHT table size of between 8 and 9, along with a first level shift register of 128 we analyze the performance of some of the prediction schemes proposed in previous work, along with a trivial 'always taken' static predictor. From table III it can be seen that a Gshare based predictor provides optimum performance, with a 256-entry PHT provide the best performance at the lowest silicon cost.

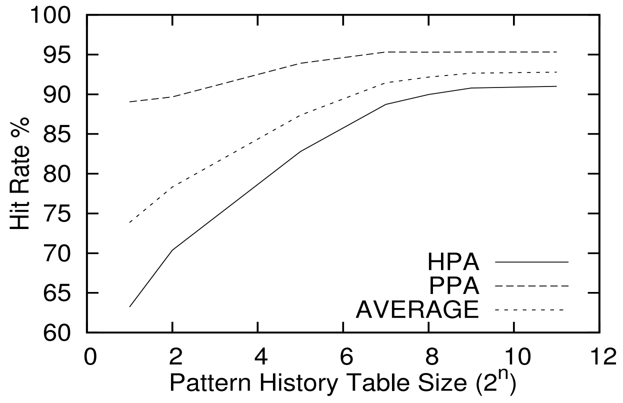

Fig.3. Predictor Hit Rate for varying Pattern History Table Sizes

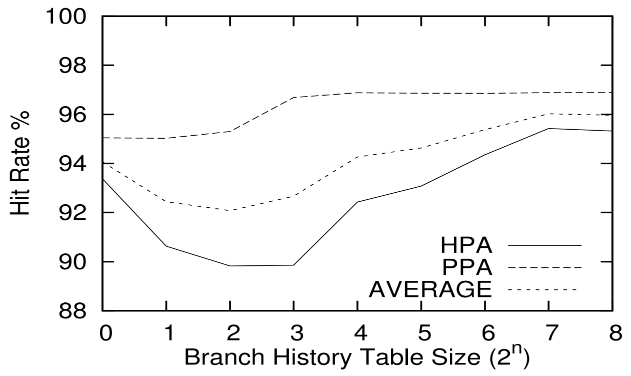

Fig.4. Predictor Hit Rate for varying Branch History Table Sizes

V. Conclusion

With increasing demands placed on network processors, additional performance must be extracted from all aspects of NP design. Implementing deeply pipelined PEs provides one such method, however such processors can result in lower performance when executing highly conditional code. In this paper we have examined which branch prediction schemes are applicable to the small RISC PEs found in Network Processors. Our work finds that unlike general purpose processing, NP applications can provide similar performance with a PHT requirement 64-times smaller. While schemes such as a 256-entry *Gshare* predictor can provide prediction rates of over 95% on average, our research also highlights that although a 256-entry table might be cheap to implement, the static nature of network applications could result in a PHT being severely under-utilized.

Appendix

Following a similar architecture to the ARM 9TDMI processor, the transistor cost of the PE can be estimated at 111,000 [12]. Assuming additional registers are needed for context switching, data transfer, etc, we can determine the cost of a single 'shared-master' 16 * 32-bit register bank as 32 + (16 *32) latches, or ~6500 transistors per bank. With 7 additional banks for context switching, along with a 2 banks for SRAM and DRAM transfers, the total cost of this PE is at least ~182,000 transistors. With 2-bit up/down saturating counter requiring 28 transistors to implement, a 2KB (or 8K-entry) would require over 229,000 transistors to implement. The 2-KB bimodal examined in [12] is therefore too expensive to implement next to a simple RISC PE, while more complex predictors such as a 2-level predictor or combining predictor would occupy significantly more area than the processor. When examining branch predictor performance on network processor architectures, the fundamental question is not whether performance increases can be extracted from such solutions, but if such predictors are justified relative to the small footprint of the PE.

Acknowledgment

This work is co-founded by the SFI China-Ireland Research collaboration Fund and the China/Ireland Science and Technology Collaboration Research Fund (2006DFA11170).

References

[1] B. K. Lee and L. K. John, "Npbench: A benchmark suite for ontrol plane and data plane applications for network processors," in *ICCD '03: Proceedings of the 21st International Conference on Computer Design*. Washington, DC, USA: IEEE Computer Society, 2003, p. 226.
[2] G. Memik and W. H. Mangione-Smith, "Evaluating network processors using netbench," *Trans. on Embedded Computing Sys.*, vol. 5, no. 2, pp. 453–471, 2006
[3] T. Wolf and M. Franklin, "Commbench-a telecommunications benchmark for network processors," in *ISPASS '00: Proceedings of the 2000 IEEE International Symposium on Performance Analysis of Systems and Software*. Washington, DC, USA: IEEE Computer Society, 2000, pp. 154–162.
[4] Cavium Inc, http://www.cavium.com
[5] D. Bermingham, A. Kennedy, X. Wang, and L. Bin, "An analysis of network processor workloads," *proceedings of the 2007 China-Ireland International Conference on ICT (CIICT07)*, pp. 354–361, 2007.
[6] R. F. Cmelik, S. I. Kong, D. R. Ditzel, and E. J. Kelly, "An analysis of mips and sparc instruction set utilization on the spec benchmarks," *ACM SIGOPS*, vol. 25, no. Special Issue, pp. 290–302, 1991.
[7] J. Lee and A. Smith, "Branch prediction strategies and branch target buffer design," *IEEE Computer*, vol. 17, no. 1, pp. 6–22, Jan. 1984.
[8] S.-T. Pan, K. So, and J. T. Rahmeh, "Improving the accuracy of dynamic branch prediction using branch correlation," *ACM SIGPLAN*, vol. 27, no. 9, pp. 76–84, 1992.
[9] S. McFarling, "Combining Branch Predictors, Tech. Rep. TN-36, June 1993. [Online]. Available: http://www.hpl.hp.com/techreports/Compaq-DEC/WRL-TN-36.pdf , last accessed on 21st June 2008.
[10] Tse-Yu Yeh and Yale N. Patt. "Alternative implementations of two-level adaptive branch prediction", *IEEE Computer Society Press Instruction-level parallel processors* – pp 150-160, 1995.
[11] M.-C. Chang and Y.-W. Chou, "Branch prediction using both global and local branch history information," *Computers and Digital Techniques, IEE Proceedings* -, vol. 149, no. 2, pp. 33–38, Mar 2002.
[12] K. Thangarajan, W. Mahmoud, E. Ososanya, and P. Balaji, "Survey of branch prediction schemes for pipelined processors," *System Theory, 2002. Proceedings of the Thirty-Fourth Southeastern Symposium on*, pp. 324–328, 2002.
[13] S. Furber, *ARM System-on-Chip Architecture*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.
[14] D. Burger, T. M. Austin, and S. W. Keckler, "Recent extensions to the simplescalar tool suite," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 4, pp. 4–7, 2004.
[15] National Laboratory for Applied Network Research (NLANR), http://www.nlanr.net/, last accessed on 21st June 21, 2008.