Exploring the VLIW Architecture Space for Network Applications

Mostafa E. Salehi*, Ali Torabi, Abolfazl Salarian

Islamic Azad University, Qazvin Branch, Qazvin, Iran

Received 23 September 2009; revised 9 January 2010; accepted 22 January 2010

Abstract

The increasing diversity in packet-processing applications together with the rapid increase in channel bandwidth has brought about greater complexity in communication protocols. Also influenced by these factors is the computational load for packet-processing engines, demanding high performance microprocessor designs as an indispensable solution. This paper reports on extensive simulation experiments carried out for exploring the performance of instruction-level parallel Very Long Instruction Word (VLIW) processors executing packet-processing applications. On the grounds of the experimental results, a design space exploration has been used to derive an efficient application-specific VLIW processor architecture based on the VEX instruction set architecture. The VEX simulator toolset has been used for design space exploration, and a number of networking applications have been chosen to serve in guiding the architectural exploration. The optimization measures achieve up to 60% improvement in performance for the most representative packet-processing applications.

Keywords Design Space Exploration, VLIW Architecture, Packet-processing Applications.

1. Introduction

As packet processing applications become more complex, the computational demand on packet processing engines keeps increasing. High performance network processors must employ the same performance improvement methods used in general purpose microprocessors, such as Instruction level parallelism (ILP) exploitation. The very long instruction word (VLIW) architectures are useful to accelerate the processing speed, and are used in many high-performance computing platforms, either in the uni-processors or as processing cores in chip multiprocessors. The routing switch processor of *Agere PayloadPlus* [1] draws on programmable VLIW compute-engines to process protocol data units. The Cisco Toaster2 [2] processing engine is a VLIW core with two four-stage instruction pipelines. C-5e network processor [3] combines 17 programmable RISC cores for packet and cell forwarding along with 32 VLIW engines, referred to as serial data processors, for processing data streams. Imagine [4] is a processor designed to support media processing applications and contains eight VLIW clusters controlled by a microcontroller. *Tensilica* [5] along with *Improv* Systems are considered pioneers in the field of configurable processors. While *Improv* Systems is based on the VLIW approach in companion with a tightly controlled toolset, *Tensilica's* current and prior products work on the basis of the RISC model, essentially enabling customers to have their own customized processor. *Tensilica*, however, has adopted the flexible-length instruction extension (FLIX) as its new VLIW architecture.

These VLIW architectures achieve a better performance by exploiting the instruction-level parallelism existing in a program to execute multiple instructions concurrently. The maximum number of executed instructions depends on the dependencies among the instructions and the available functional units. The concurrent execution of multiple instructions in parallel boosts up the number of instructions per cycle which is a key metric for improving the processor's performance. The design space exploration (DSE) of VLIW architecture is associated with different conflicting criteria such as the chip's area, speed, power consumption or on-chip memory requirements. The output is a set of different architectures associated with different trade-offs. While significant research has been conducted on the analysis of VLIW's performance in general processing domain [6-10], very few works have been directed to more specialized domains such as network processors.

* Corresponding Author. Email: m.e.salehi@qiau.ac.ir

This paper proposes a design space exploration for an embedded VLIW processor which obtains a performance-efficient architecture meeting the requirements of common network applications including both data- and control-plane packet-processing tasks. In this work, the VLIW configuration parameters are varied to find an optimal configuration for some popular network benchmarks. The focus of this paper is on packet-processing applications which differ from scientific computing applications that have traditionally been the focus of parallelism prediction and extraction. Network processors are mostly characterized as systems having dynamic workloads, requiring irregular data patterns, in need of switching between varieties of different tasks, and featuring less complex control behaviour. This work introduces a DSE to find out the best VLIW architecture using the VEX system [11]. Section II begins with a brief description of the VLIW architectures. We then go on to describe the VEX system; in particular, the instruction set architecture (ISA), the VEX compiler and the simulation environment. Section III elaborates on our DSE methodology, giving details on its architecture and parameter space definition, and specifying the values used for each iteration of the exploration in the proposed DSE scheme. In Sections IV and V, some packet-processing applications are introduced and analyzed. Experimental results are presented in Section VI, and finally Section VII concludes the paper.

2. Architecture Model and Simulation Environment

2.1. The Concept of VLIW Architecture

Some high performance processors exploit the instruction-level parallelism to meet the performance demands of embedded applications. The very long instruction word or VLIW is an ILP approach to helping processor designers exploit the high levels of ILP by executing multiple concurrent operations in the form of a long instruction word. In VLIW architecture, the creation of an instruction word of simultaneously-issued operations is completely done at compile time. In fact, the VLIW architecture owes much of its efficiency and high speed to the fact that it makes no run-time decisions concerning the instruction issue and scheduling. In other words, in VLIW architecture, the dynamic runtime ILP extraction of SuperScalars is shifted to the compile time, allowing for more flexibility. The code optimizations introduced by the compiler try to maximize the ILP to exploit the VLIW's parallelism. The inter-dependence of the compiler and the VLIW architecture makes the design space exploration more difficult compared to exploration in other architectures.

In a VLIW machine, the data path consists of multiple functional units which can be independently controlled through dedicated fields in the instruction word. The distinctive feature of the VLIW architectures is that its long instructions are in fact the machine instructions. There is no additional layer of interpretation for expanding the machine instructions into micro-instructions. While complex resource or field conflicts often exist between functionally independent operations in a horizontal micro-code engine, a VLIW machine generally has an orthogonal instruction set and a higher degree of parallelism.

2.2. The VEX System: A VLIW Example

VEX [11] models a parametric platform for designing embedded processors based on the VLIW architecture and allows for variation in a common set of applications and system resources such as issue width, number of functional units and registers, and processor's instructions set. The VEX development system, referred to as the VEX toolchain [12], includes the set of tools that allows C programs compiled for a VEX target to be simulated on a host workstation. The VEX toolchain is mainly intended for architecture exploration, application development, and benchmarking. It includes a very fast architectural simulator that uses a form of binary translation to convert the VEX assembly language files to native binaries running on the host workstation. The translator annotates the binaries to collect execution statistics and includes a cache simulator to collect D-cache and I-cache data. Figure 1 shows the overall structure of the VEX toolchain. The Vex system can be characterized in terms of the following three components.

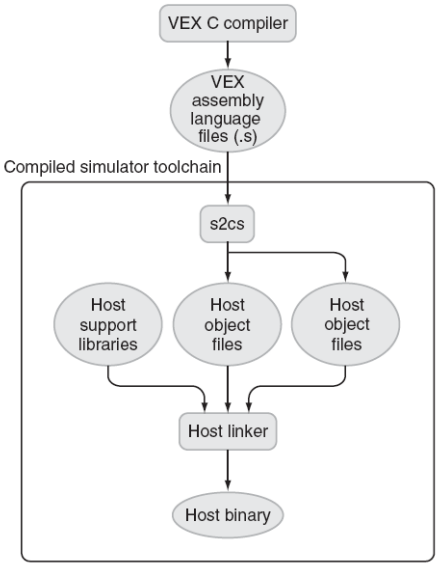


Fig.1. Structure of the VEX toolchain [11].

2.2.1. The VEX Instruction Set Architecture: VEX ISA

The VEX architecture is a 32-bit clustered VLIW that is parametric and customizable to application-specific domains. The VEX ISA is based on the ISA of the HP/ST Lx family of VLIW embedded cores [13]. Its scaling

property allows for changing the number of clusters, execution units, registers, and latencies while its customizability enables adding special-purpose instructions to the base instruction set. The basic structure of each cluster has been specified in [11]. VEX includes a complete experience of all architecture latencies and resource constraints: Parallel execution units including multiple ALUs and multipliers, parallel memory pipelines, a large visible register set, and an efficient branch architecture. The basic unit of execution in VEX is an operation, which is similar to the notion of instruction in typical RISC processors. Syllable is the encoded operation, and a collection of syllables issued in a single package and executed in VLIW architecture as an atomic unit is called an instruction word.

The default VEX cluster contains two register files, four integer ALUs, two 16 × 32-bit multiplication units, and a data cache port as shown in Figure 2. Up to four operations per instruction can be issued in each cluster. The register set consists of 64 general-purpose registers (GRs) that are 32-bits wide, and there are eight 1-bit branch registers (BRs). In addition, the default VEX cluster also contains a control or branch unit. Without changing the instruction set, the cluster could be significantly altered with no impact on the software. Also, the functionality of the cluster could be expanded with additional special-purpose instructions. The default VEX clusters contain memory, integer, and branch functional units. The memory units perform operations such as load, store, and prefetch. Each memory unit is associated with an access to the memory system. The integer units execute the common set of integer operations on registers or immediate operands; viz. compare, shift, and select. The branch units execute control operations based on the conditions stored in the branch registers, such as conditional branches, unconditional jumps, direct and indirect calls, and returns. Given the restriction that only four syllables may be used to encode the operations for each cluster, at most four operations may be issued in a single instruction.

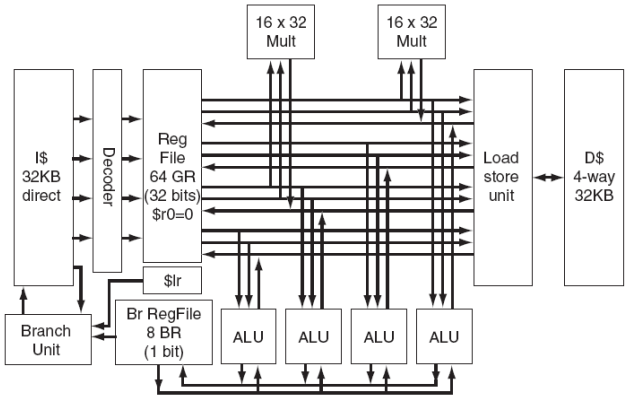


Fig.2. Structure of the default VEX cluster. The default cluster includes 4 integer units, 2 multipliers, a load-store unit, and a control unit [11].

2.2.2. The VEX C Compiler

The VEX C compiler is at the core of the VEX toolchain, and is derived from the Lx/ST200 C compiler, which is, in turn, derived from the Multiflow C compiler. This compiler draws on trace scheduling, and is a robust ISO/C89 compiler. A very flexible parametric machine model determines the target architecture. VEX permits architecture space exploration by changing the number of clusters, execution units, the issue width, and the operation latencies without recompiling the compiler. In the VLIW architecture, the compiler plays an important role in scheduling multiple concurrent operations and in exploiting the maximum amount of ILP.

2.2.3. The VEX Simulation system

The VEX simulator is a functional simulator that is based on the notion of the so-called compiled simulation technology. The compiled simulator (CS) converts the VEX binary to the host computer's binary by first converting VEX to C and then producing the host executable by invoking the host's C compiler. In addition to the standard semantics of the instructions, CS introduces a profiler to count the execution cycles and other interesting statistics and also performs a simple cache simulation. CS operates on each of the individual VEX assembly language files and translates these files back to C. The CS-generated C files are then compiled with the host platform's C compiler and linked with the support libraries that deal with the instrumentation. The simulation system also comes with a fairly complete set of POSIX-like *libc* and *libm* libraries based on the GNU *newlib* libraries.

3. Experimental Setup

With the VEX simulation environment, the source code of each algorithm has its own MAKEFILE which is modified to use the VEX compiling tools. In the VEX compiler, several types of architectural customizations are possible; for instance, one may augment the VEX ISA with custom instructions for functions which are computationally intensive. It is also possible to consider different processor and memory architectures in which the customizations are supplied at the compile time. In this experiment, we have used the memory customizations which are defined in a cfg file. For the parameters not defined within the custom memory architecture, the default CS configuration file (i.e. vex.cfg) is used. Before running the VEX compiler, since there is no default value for the machine configuration, it must be defined within an mm file.

For design space exploration, we have changed the value of different parameters when generating the machine configuration files. Our intention is to experiment with the VEX architectural characteristics whilst running on different architectures and to obtain the best cost/performance trade-off for our application of interest. Table 1 shows the ranges of parameter values possible to be

Table 1
The ranges of parameter values for the machine architecture configuration file.

Parameter	Values
Issue width	1-32
MemLoad	1-32
MemStore	1-32
ALU	1-32
Multiply	1-32
General Registers	4-64
Branch register	2-8

varied for the design space. Details on the definition and description of each parameter are given in [11].

4. Domain-Specific Applications and Optimization Strategy

According to the IETF (Internet Engineering Task Force), the operations of network applications can be functionally categorized into data-plane and control-plane functions [14]. There are a large variety of NP applications that contain a wide range of different data-plane and control-plane processing tasks. To properly evaluate network-specific processing tasks, it is necessary to specify a workload that is typical of this context. *Commbench* [15] is composed of eight data-plane programs that are categorized into four packet-header processing and four packet-payload processing tasks. Within a similar line of work, *NetBench* [16] contains nine applications that are representative of commercial applications for network processors featuring both small low-level code fragments as well as large application-level programs. Both *CommBench* and *NetBench* present data-plane applications. *NpBench* [17], on the other hand, targets both control-plane and data-plane workloads. Ramaswamy and Wolf et. al. [18] have presented a tool, called *PacketBench*, which provides a framework for implementing network-processing applications and extracting workload characteristics. The Embedded Microprocessor Benchmarking Consortium (*EEMBC*) [19] has also developed a networking benchmark suite to reflect the performance of client and server systems (*TCPmark*), and functions mostly carried out in infrastructure equipment (*IPmark*).

With the representative benchmark applications for IPv4 protocol header- and payload-processing at hand, we have performed a simulation-based design space exploration for packet-processing applications based on the VEX architecture. The selected applications are IPv4-radix and IPv4-trie as RFC1812-compliant look-up and forwarding algorithms [20], a packet-classification algorithm called Flow-Class [18], internet protocol security (IPSec) [21] and the message-digest algorithm 5 (MD5) as payload-processing applications. The measurements carried out with reference to these network applications reveal the performance challenges of different programs. The presented measurements also account for dynamics, in that the frequency of a measured event is weighed by the number of times the event occurs during execution of the application.

Each pipeline stage of our simulation model is parameterized by the width of the stage and the sizes of the specialized memory structures within the stage. All these parameters must be tuned in a way to maximize the processor performance for the selected applications. In this paper, a heuristic optimization scheme has been used for design space exploration. The heuristic is intended to vary a broad range of processor parameters to obtain an optimum processor configuration in terms of performance.

5. The Analysis of Packet-Processing Applications

A common approach to lower the application execution time is reducing the number of clock cycles per instruction or instead increasing the number of instructions per clock (IPC). The instruction-level parallelism (ILP) techniques try to increase the IPC. The utilization of this technique requires a detailed analysis of the application characteristics for making a proper architectural decision. The superscalar and VLIW processors exploit concurrent functional units and execute as many instructions as possible in parallel. Therefore, an architecture designer needs to know the optimum number of parallel functional units promising the best cost/performance trade-off. This metric depends strictly on the consequent logic or arithmetic instructions that would be executed on the concurrent functional units. The most frequent chains of consequent logic or arithmetic instructions in our applications are summarized in Figure 3. As can be seen in the figure, the most frequent chains of logic/arithmetic sequences in header-processing applications (i.e IPV4-radix, IPV4-trie, and Flow-class) are one or two instructions. However, the payload-processing applications (i.e IPSec and MD5) are more computationally intensive

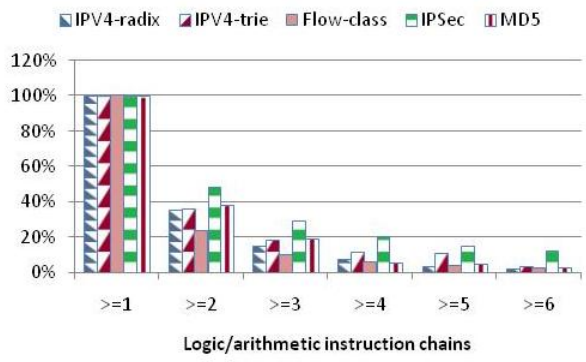


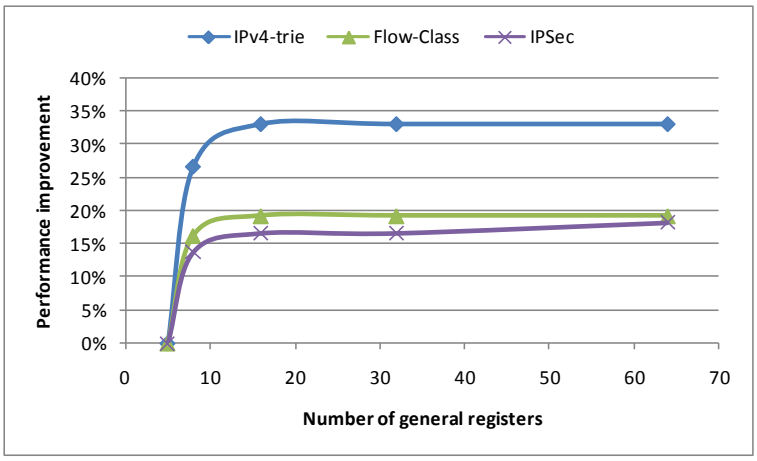
Fig. 3. The occurrence of consequent logic/arithmetic instruction chains according to the length of the logic/arithmetic chain.

and have sequences of one, two, and three instructions. Therefore a superscalar or VLIW processor can exploit concurrent functional units and improve the ILP of the selected applications. What we have investigated here will be further elaborated in our discussion on experimental results.

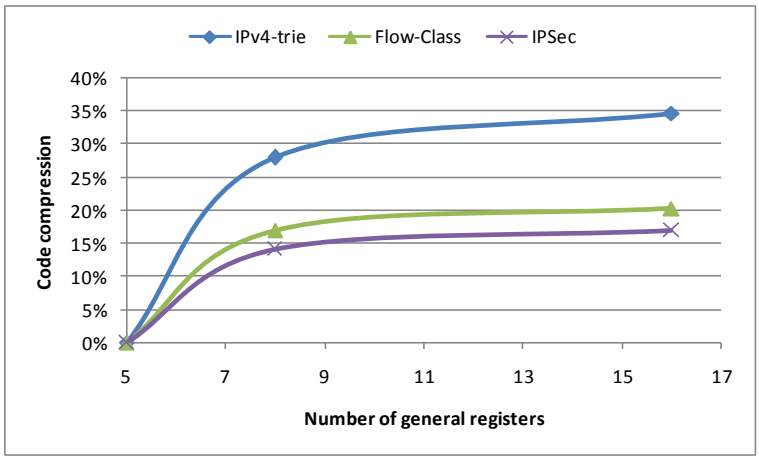
6. Results and Discussion

We have exhaustively evaluated many VLIW configurations and explored the architecture design space for obtaining the optimum performance using representative packet-processing applications. The performance has been calculated in terms of the dynamic execution cycles of each application. An important parameter to explore is the size of the register file. Large register files provide enough registers for the compiler, such that the register allocation algorithm can use as many registers as required and generate a more optimum code. On the other hand, when the size of the register file is small, the local and temporary variables should be placed in memory; consequently, in our load-store RISC architecture, extra instructions are required for reading variables from the memory to the register. Therefore, when the number of registers is relaxed, the final object code has fewer memory accesses, and hence, would be more optimized. However, large register files occupy a large fraction of the chip area and also have more power consumption. There is, thus, a trade-off between the chip area on the one hand and the achieved performance on the other, in terms of the size of the register file.

We have calculated the execution cycles of IPV4-trie, Flow-Class, and IPSec for different sizes of the register file (or general registers in VEX terminology). As shown in Figure 4, the performance improvement almost levels off when the number of registers reaches 16, and the introduction of more registers only increases the area



(a) Performance improvement



(b) Object code compression.

Fig. 4. Performance improvement and code compression of the selected applications for different numbers of general registers compared to the register file consisting of 5 registers: (a) Performance improvement, (b) code compression.

overhead with negligible payoff on performance.

Other important parameters that considerably affect the performance are the issue width and the number of concurrent ALUs. Using the VEX re-targetable compilation and simulation environment, we have explored the effect of increasing the issue width and the number of ALU units on the performance improvement for the representative applications.

The performance improvement results for different issue widths are demonstrated in Figure 5. As shown in the figure, the performance improvements almost level off for issue width = 4. When the issue width is increased to values greater than 4, the area overhead would increase significantly while the performance is improved negligibly. Therefore, the optimum value of the issue width can be 2 or 4, based on the available area budget and the required performance improvement.

According to the experimental results, other parameters such as the number of concurrent memory ports for load and stores as well as the number of branch registers have no significant impact on performance improvement. Figure 6 and Table 2 demonstrate the effect of various parameters in 13 different configurations on performance improvement. As shown in the figure, the sixth configuration proves the most superior considering the area-aware performance improvement. Although there are

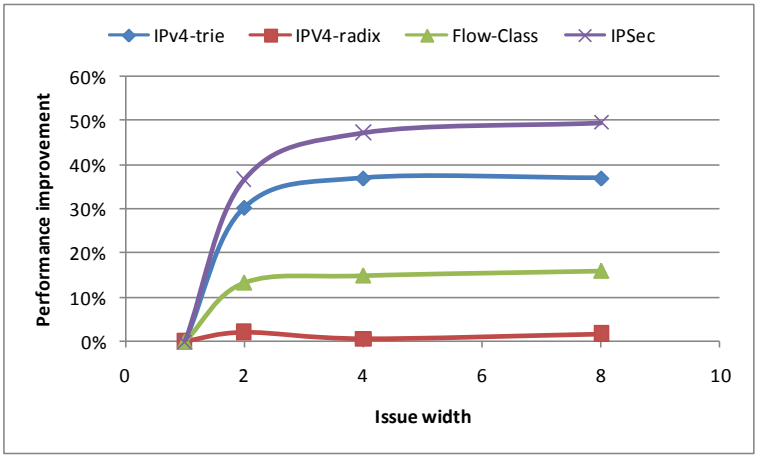


Fig. 5. Performance improvement of the selected applications for different issue widths.

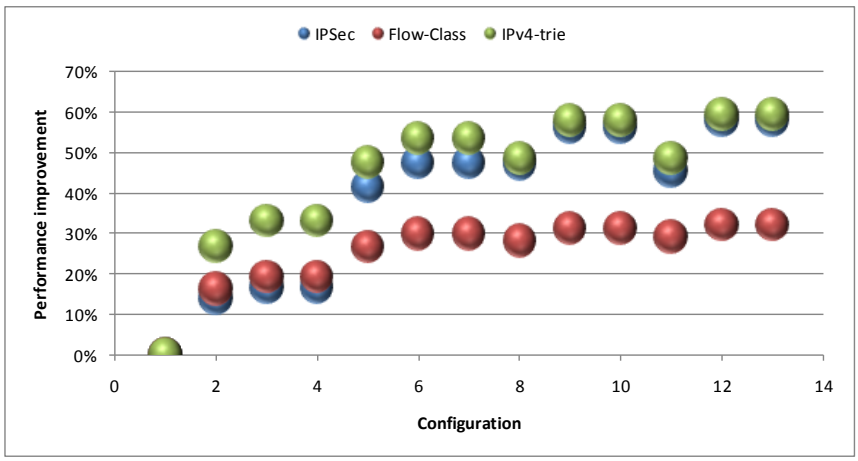


Fig. 6. Performance improvement of the representative applications according to the Table 2.

Table 2
13 different processor configurations

Configuration	1	2	3	4	5	6	7	8	9	10	11	12	13
Issue width	1	1	1	1	2	2	2	4	4	4	8	8	8
ALU	4	4	4	4	4	4	4	4	4	4	8	8	8
General registers	5	8	16	32	8	16	32	8	16	32	8	16	32
Memory ports	1	1	1	1	1	1	1	1	1	1	1	1	1
Branch registers	2	2	2	2	2	2	2	2	2	2	2	2	2

some configurations that lead to higher performance improvements, they impose considerable area overhead.

On the grounds of the experimental results, we have proposed the optimum values for each parameter to bring about the best performance improvement relative to the imposed area overhead. The optimum parameter values are summarized in Table 3.

Table 3

Optimum configuration in terms of performance improvement with reference to the starting point.

Parameter	Optimum configuration
Issue width	2
MemLoad	1
MemStore	1
ALU	2
General Registers	16
Branch register	2

8. Conclusion

We have presented a high-level simulation-based design flow to explore the design space for the VLIW architecture. The adoption of the VEX re-targetable compilation and simulation framework makes the design flow suitable for a wide variety of applications. We have used the VEX exploration framework for experimenting with domain-specific tasks in packet-processing applications. In the light of the exploration results, we have proposed the optimum architecture based on both the estimated hardware cost and the achieved performance improvement. In our exploration, we have determined the optimum values for the issue width, the memory ports, the number of functional units, the number of branch registers, and the size of the general register file. With the help of these results, one can implement an efficient VLIW architecture for a given application domain.

References

[1] B. Klein and J. Garza, Agere systems – communications optimized payload plus network processor architecture, in Network Processor Design: Issues and Practices. Morgan Kaufmann, San Francisco, California, vol. 1, pp. 219–233, 2002.

[2] J. Marshall, Cisco systems – Toaster2, in network processor design: issues and practices. Morgan Kaufmann, San Francisco, California, vol. 1, pp. 235–248, 2002.

[3] P. Lekkas, Network processors architectures, Protocols and Platforms, McGraw Hill, 2003.

[4] B. Khailany, W. J. Dally, et al., Imagine: Media processing with streams, IEEE Micro, pp. 35–46, March/April 2001.

[5] Tensilica - Tensilica: Customizable Processor Cores for the Dataplane, Available online: http://www.tensilica.com/

[6] R. R. Hoare, A. K. Jones, D. Kusic, J. Fazekas, J. Foster, S. Tung, and M. McCloud, Rapid VLIW processor customization for signal processing applications using combinational hardware functions, EURASIP Journal on Applied Signal Processing, article ID. 46472, pp. 1-23, 2006.

[7] G. Ascia, V. Catania, M. Palesi, D. Patti, Multi-objective optimization of a parameterized VLIW architecture, In. Proc. of the NASA/DoD Conf. on Evolution Hardware, 2004.

[8] E. Salami, M. Valero, Initial evaluation of multimedia extensions on VLIW architectures, SAMOS 2004, LNCS 3133, pp. 403-412, 2004.

[9] A. K. Jones, R. Hoare, D. Kusic, An FPGA-based VLIW processor with custom hardware execution, In Proc. of FPGA'05, pp. 107-117, February 2005.

[10] D.byo Saptono, V. Brost, F. Yang, and E. Prasetyo, Design space exploration for a custom VLIW architecture: direct photo printer hardware setting using VEX compiler, In Proc. of IEEE Int. Conf. on Signal Image Technology and Internet Based Systems, pp. 416-421, 2008.

[11] J. A. Fisher, P. Faraboschi, C. Young, Embedded computing a VLIW approach to architecture, Compilers and Tools, Elsevier Inc, 2005.

[12] Hewlett-Packard Laboratories. Vex toolchain. http://www.hpl.hp.com/downloads/vex.

[13] P. Faraboschi, G. Brown, J. A. Fisher, G. Desoli, and F. Homewood, Lx: A technology platform for customizable VLIW embedded processing, In 27th Int. Symp. on Computer Architecture (ISCA), pages 203-213, 2000.

[14] IETF Available from: http://www.ietf.org/

[15] T. Wolf and M. A. Franklin, CommBench a telecommunications benchmark for network processors, in proc. of IEEE Int. Symp. on Performance Analysis of Systems and Software (ISPASS), pp. 154-162, April 2000,.

[16] G. Memik, W. H. Mangione-Smith, and W. Hu, NetBench: A benchmarking suite for network processors, in Proc. Of IEEE/ACM Int. Conf. on Computer-Aided Design, pp. 39-42, November 2001.

[17] B. K. Lee and L. K. John, NpBench: A benchmark suite for control plane and data plane applications for network processors, in proc. of IEEE Int. Conf. on Computer Design (ICCD 03), pp. 226-233, October 2003.

[18] R. Ramaswamy and T. Wolf, PacketBench: A tool for workload characterization of network processing, in proc. of IEEE Int. Workshop on Workload Characterization, pp. 42-50, October 2003.

[19] EEMBC, The embedded microprocessor benchmark consortium, Available from: http://www.eembc.org/home.php.

[20] F. Baker. Requirements for IP version 4 routers. RFC 1812, Network Working Group, June 1995.

[21] S. Kent and R. Atkinson. Security architecture for the internet protocol. RFC 2401, Network Working Group, November 1998.