Unité Mixte de Recherche 5104 CNRS - INPG - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

P-Ware: Performance-Aware Transaction-Level Simulation for Network Processor Applications

Ismail Assayad, Sergio Yovine

Verimag Research Report n^o TR-2006-7

June 2006

P-Ware: Performance-Aware Transaction-Level Simulation for Network Processor Applications

Ismail Assayad, Sergio Yovine

June 2006

Abstract

Platform-based design is an approach to cope with increasing costs in developing complex embedded systems. In order to support performance analysis at system-platform level, this report presents a methodology and tool which provide a joint SW/HW component-based modelling and simulation framework. Our framework allows for specifying variable transaction latencies, and separates functional and timed behavior of components. We apply the framework for analyzing several implementations of an IPv4 forwarder application on an Intel's dual IXP2800. The analysis allows evaluating both SW and HW performance such as packets throughput, threads utilization, bus bandwidth and channels conflicts.
Keywords: Simulation, Performance, IPv4, IXP2800

Keywords:

Reviewers:

How to cite this report:

```
@techreport { ,
title = { P-WARE: Performance-Aware Transaction-Level Simulation for Network
Processor Applications},
authors = { Ismail Assayad, Sergio Yovine},
institution = {  Verimag Research Report },
number = {TR-2006-7},
year = { },
note = { }
}
```

1 Introduction

Many embedded applications such as video compression, HDTV and packet routing require higher and higher performance. The HW solution is to ressort to multiprocessor chips, such as the Intel's IXP2XXX NP family [2] and the Philips Wasabi chip [4]. However, due to the complexity of developing SW for complex multiprocessor architectures, developers are often unable to fully optimize the performance of their programs on these advanced chips [1]. This motivates the need for the definition of performance-aware models for building predictable systems, in terms of resource usage, timing behavior and bandwith usage for embedded SW and HW.

Current practices for system-level design and analysis do not provide satisfactory solutions for coping with the increasing complexity of embedded systems. System-design approaches can be classified according to their modelling scope, i.e., the defined articulation points and layers in the design flow. Three categories can be distinguished: SW-based design (SBD), HW-based design (HBD) and platform-based design (PBD).

On one hand, SBD relies on a high-level representation of communication between components which is then optimized and resolved through middlewares. A number of parallel programming models have been presented for multi-processor chips [16, 8, 6, 12]. SBD is not concerned with HW performance.

On the other hand, HBD does not consider application programming as part of the design and thus no model articulation point at SW level is defined. The SW running on HW components, usually written in assembly, is independent of the structure of the application. This approach focuses on architectural and system-level modelling. For instance, [5] provides flexible wrappers between functional code and HW-specific I/O interfaces. The tool introduced in [13] generates a SystemC-based multiprocessor SoC representation from a library of parameterized communication components specific for packet-switching networks. HBD approaches reduce design time for HW/SW interface refinement and component integration, but are not concerned with SW performance.

A key concept in PBD [17] is the notion of system-platform which provides the abstraction layer allowing for composing SW- and HW-level views using the "meet-in-the-middle" principle. The framework proposed in [18] is specific to clock-driven concurrent HW components with message-passing communication. The methodology presented in [14] is specific to SW models given as acyclic task graphs. The behavior of a task is abstracted to the best- and worst-case execution times and input/output data packet communication. Such model does not allow for obtaining intrinsic HW performance. The framework proposed in [15] does not provide any component model for clearly articulating SW and HW interactions. [3] proposed a methodology and tool dedicated to MIMD architectures using Kahn Process Networks for modelling SW. [9] describes an approach centered around a single communication component model (COM), which abstracts away the overall system interconnection. Therefore, only coarse grain COM-level performance can be observed. In order to handle performance in PBD, we have developed P-Ware, a component-based methodolgy and tool for performance-aware modelling and analysis at system-platform level. P-Ware combines several timed transaction-level HW components and user-level SW components. P-Ware allows for specifying components having variable transaction latencies, and separates their functional and timed behaviors. P-Ware allows for performance simulation of components in isolation (e.g. SW components tasks execution times, HW components idle time).

We apply the framework for analyzing the performance of several implementations of an IPv4 forwarder application on an embedded multiprocessor HW architecture, namely Intel's dual IXP2800. Several network processor models have been introduced to support IPv4 forwarding. The model presented in [7] is a high level timing abstraction of HW. Although sufficient for SW performance measurements, it does not support evaluation of any HW metric. [11] proposed an analytical model for both IPv4 and IXP1200 architecture used for evaluating different design alternatives. The model however does not allow for capturing components with dynamic effects such as history-dependent memory bank latencies. [10] developed a coloured Petri nets model for IXP2400 where packet arrivals and memory conflicts parameters are inputs given as exponential distributions and probabilities, respectively. In contrast, P-Ware allows evaluating both appli-

cation and HW performance such as packets throughput, threads utilization, bus bandwidth and channels conflicts.

2 HW view

2.1 Component model

The HW view is composed of a connection of several builiding components whose general model is depicted on figure 1. Time is taken into account using a clock characterized by its frequency. Behaviors are described by an automaton where Y.xxx denotes the state of automaton Y, [xxx] denotes a condition on the transition, ?xxx and !xxx denote synchronous communications.

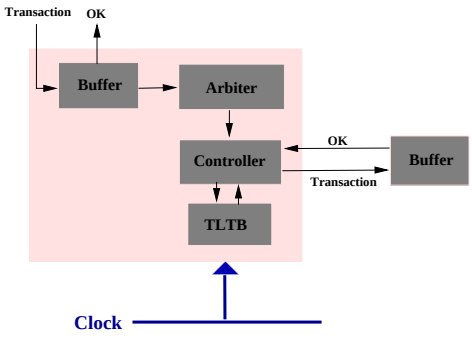
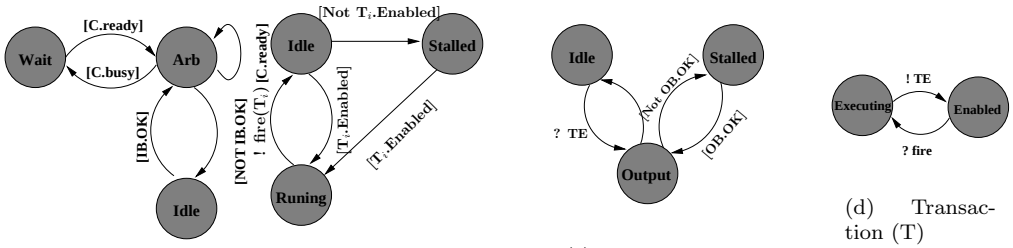


Figure 1: HW component model

TLTB is the set of timed-level behaviors of component's transactions. A transaction is a set of sequences of operations to be executed by the component for a given request. Several transactions can be executing at a given time inside a component. Two transactions T_1 and T_2 which are about to perform operations o_1 and o_2 may be in conflict if o_1 and o_2 use a shared resource r. We call the set of operations which are in conflict a "conflict group". Formally, a transaction is a pair $\langle T, (C; L)\rangle$ where T is the automaton shown in figure 2(d), C is either a chain or a multi-chains operations graph with one conditional source node prolonged with at least two chains, and L is a function giving for each operation the corresponding latency. TLTB is a pair $(\mathcal{T}; \mathcal{R})$ composed of $\mathcal{T}$, the set of transactions, and of $\mathcal{R}$, the set of conflict groups. An example is given later. Hereinafter, we use the term transaction to denote both a transaction request and a TLTB transaction when there is no risk of confusion.



(a) Arbiter (A)

(b) TLTB control (C)

(c) Output control (C)

(d) Transaction (T)

Figure 2: Component behaviors

Buffer (IB) stores transactions and signals if the component accepts or refuses new requests. It is characterized by its storage capacity, the number of internal queues, back pressure thresholds and latency. Sending a transaction to a buffer is done by giving the destination queue and the input when OK is true, i.e., when the buffer accepts inputs. Receiving a transaction is operated in a similar way.

Arbiter (A) is characterized by an arbitration function, which computes the buffer source queue, and a latency. Figure 2(a) shows the functional-behavior model of the arbiter where IB denotes the input buffer.

Controller (C) actions are twofold, TLTB-control actions and output-control actions. C decides when to launch a transaction: if the transaction is enabled it immediately fires it, otherwise, the firing is delayed. C resolves conflicts. When a transaction completes, C stalls until recipient buffer accepts input, then it sends the result as a transaction. The model of the controller is the composition of automata 2(b) and 2(c).
In experiments, we have put special attention on accurately modelling read and write transactions due to the impact they have on HW performance. Figure 3 illustrates a read transaction flow on a HW model whose component blocks are not detailed.

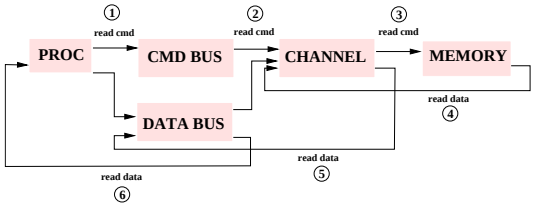


Figure 3: read transaction flow

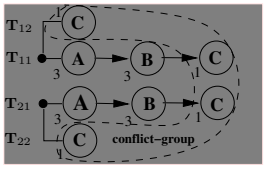
2.2 Performance

Observers collect temporal traces for the buffer, the arbiter, the controller and the TLTB states along components execution. A trace of a block B is a sequence of pairs (c, s), where s is a state of B and c is the clock cycle value when B enters state s.

Profiling consists of maintaining performance counters on observed traces during a predefined observation time interval I. Performance metrics are component available bandwidth, conflicts and output rate. Idle time is the time during which the arbiter, the controller and the TLTB are all idle. Available bandwidth is the amount of transactions that can be handled during the component idle time when considering either minimal or maximal duration of TLTB transactions. The per TLTB-transaction available bandwidth is computed similarly. Component conflicts and output rate are the average input buffer size and number of outgoing transactions, respectively, over I.

2.3 Example

Component. Figure 4 shows the model of a DRAM TLTB. It has a 3 cycle latency arbiter, includes two independent banks b_1 and b_2 operated at 125MHz, with a precharge (A) latency of 3 cycles (24ns), a row access (B) latency of 3 cycles (24ns) and a column access (C) of 1 cycle (8ns). Operations A and B are common to sequential access patterns. Such accesses are reduced to operation C after the first one, and are thus performed with a throughput of one per cycle. TLTB is thus composed of four paths: there are two possible transactions T_{11} and T_{12} for bank b_1 and two other transactions T_{21} and T_{21} for bank b_2 where T_{k1} is composed of operations A, B and C while



125MHZ

Figure 4: TLTB block model of a DRAM

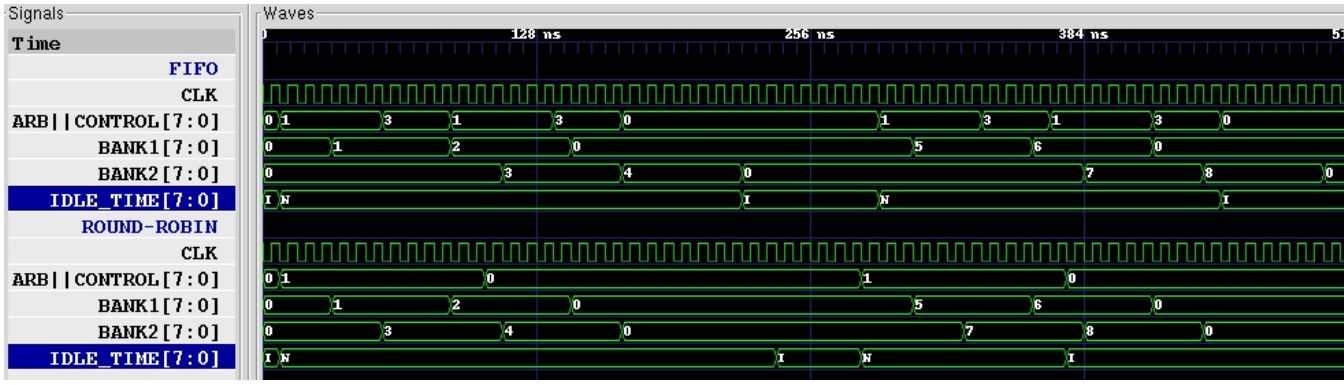


Figure 5: Idle time for the two DRAM configurations

T_{k2} is composed of operation C only (figure 4). All transactions are in conflict at operation C. The controller operates as follows. Upon receiving a transaction request $R = (i, j, k)$ concerning a memory access for data in column i, row j and bank k, the controller tests if transactions T_{k1} and T_{k2} are enabled. If it is the case, it fires either T_{k1} or T_{k2} according to the preceeding transaction request (R_k^{last}) of bank b_k : if $R_k^{last}.j = R.j$ then it fires T_{k2} otherwise it fires T_{k1}.

Performance. Let us consider the following sequence of DRAM memory transaction requests over an observation interval of 100 cycles: $(0,0,0)$, $(0,1,0)$, $(0,0,1)$, $(0,1,1)$, $30*$, $(1,0,0)$, $(1,1,0)$, $(1,0,1)$, $(1,1,1)$, $62*$. For convenience we denoted $n*$ an absence of transaction request during n cycles. We consider two configurations of DRAM. The first configuration includes a buffer with two queues storing b_1 and b_2 transaction requests. The arbiter uses a ROUND-ROBIN arbitration policy for selecting the source queue. The second configuration includes an in-order FIFO buffer composed of one internal queue.

A simulation trace is shown in figure 5. Controller states are coded as follows: idle 0, runing 1, waiting for T_{1k} to be enabled is coded by 2 (i.e. waiting for bank b_1) and waiting for T_{2k} to be enabled is coded by 3 (i.e. waiting for bank b_2). Bank state 0 codes the enabled state while k, with $k \geq 1$, indicates that the bank is in the not-enabled state because of the execution of the k-th transaction request of the input sequence. The idle time deduced from these states corresponds to time when arbiter, controller and banks are all idle, i.e., at state 0. Idle time is denoted by I.

Simulation shows that DRAM bandwidth availability is 31% higher with ROUND-ROBIN arbitration (59 cycles) than with FIFO one (45 cycles) for the same input sequence over I.

3 SW view

3.1 Component

The SW-view is a set of components whose model is described in figure 6. It is composed of a hierarchical task graph, HTG, describing application logic, a scheduler, and a controller of HTG tasks execution.

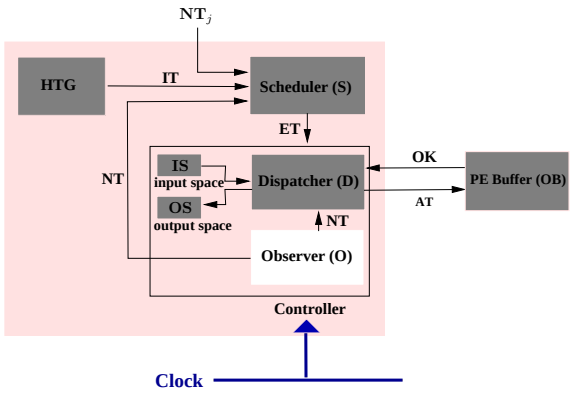


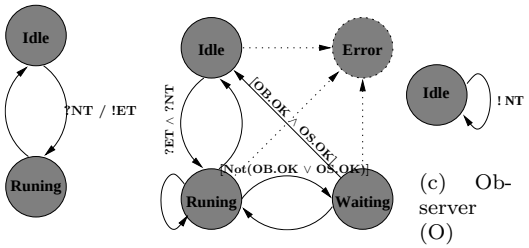
Figure 6: SW component model

Scheduler (S) computes and sends the set of enabled tasks (ET) to the controller. IT is the set of HTG tasks. NT$_j$ is the set of tasks of component j whose end-time were notified. The behavior of S is depicted in Fig. 7(a).

Controller (D and O) is composed of a dispatcher (D) (7(b)) and an observer (O) (7(c)). Every task in ET evolves through three states: "waiting", "executing" and "completed". The behavior of D is described by Fig. 7(b). "waiting" state indicates that D is waiting for an input (IS) or output (OS) memory space, or a processor. "runing" state corresponds to the ordering and mapping operation. The behavior of O consists of updating NT with completed tasks and sending it to D and S.

3.2 Peformance

Performance Observer (PO). After reception of a message from either D or O, say s for a task T, PO stores a pair (c, s) indicating that T enters state s at cycle c. Stored data are used for computing SW performance, i.e., tasks waiting times, tasks execution times and memories occupation times.



(a) Scheduler (S) (b) Dispatcher (D) (c) Observer (O)

Figure 7: Software behaviors

4 Experiments

IXP 2800 includes an XScale processor, multi-threaded microengines (ME), channels, DRAMs, buses and dedicated HW-unit. Details of channels, DRAMs and buses components are depicted in figure 8.

IPv4 is composed of Ingress- and Egress-Data Planes tasks and the Control Plane tasks. Tasks are mapped to a dual IXP2800 system where one NPU handles ingress traffic and the other handles egress traffic [12]. Packet pointers are communicated through cyclic buffers located in DRAM. Figure 9 shows part of the mapping of Ingress Data- and Control-Planes: pkt_rx is the packet receiver, ipv4_fw is the packet forwarder, PEP is the protocol exception packet, PIM is the protocol information manager, and CPDP is the control agent.

Hereinafter, we show the impact of various IXP2800 configurations on both SW- and HW-performance. A configuration denoted by X-YxZ means that IXP uses the X-interleaving mode and that DRAM channels are populated with Y DRAM devices with Z independent banks inside each device. We considered 1, 2 and 3-interleaving modes with 2, 4 or 8 banks for each. Configurations are noted A, B, C, D, E, F, G, H and I, respectively. Over an observation interval of 10E6 ME-cycles, we evaluated (1) packet throughput, i.e., the number of completed packets without any errors; (2) average of processor-threads utilizations; (3) average of the available bandwidth for the PUSH and PULL buses, expressed as a percentage of their respective best-case bandwidth (2.8 GB/s for each bus); (4) DRAM conflicts, i.e., the average number of transactions waiting for DRAM banks of enabled channels. Fig. 10 is a synthesis of SW and HW performance evaluated with P-Ware.

Throughput (TP). With fixed number of channels, throughput increases when channels are populated with more banks (2, 4 and 8). This phenomenon is more contrasted for 1-way than for 3-way interleaving. With 1 channel, throughput practically doubles when passing from 2 to 4 and 4 to 8 banks. With 2 channels, however, throughput increase is smaller and turns around 5% when passing from 4 to 8 banks. For fixed number of banks throughput increases when using 1,2 and 3 channels. Nevertheless, performance gain is slightly more important when increasing number of banks rather than channels. F, H and I provide the highest throughput.

Utilization (U). Average per ME-thread utilization remains under a boundary of 20% for all configurations due to communications with DRAM.

Bandwidth (PUSH and PULL). Available bandwidth is under 1.12 GB/s. PUSH bus is more available than PULL bus due to the fact that IPv4 generates more read- than write-transactions.

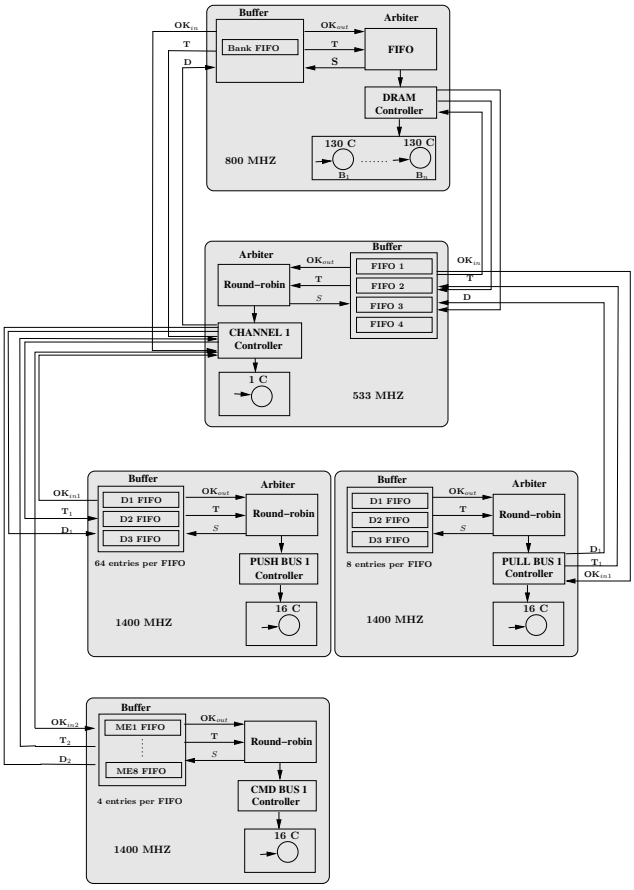


Figure 8: Part of the IXP2800 Components

When augmenting the number of banks or channels, available bandwidth decreases for the internal buses. Furthermore, for the highest-throughput configurations, F, H and I, bandwidth reaches its smaller values for PULL bus while PUSH bus is fully used. Compared to throughput performance, we notice that the lowest-throughput configurations A, B and D are the highest available bandwidth configurations.

Channels conflicts (Ci-Bj). Average amount of DRAM requests in the whole channels queues remains under a boundary of 60% of channel buffering capacity (256). DRAM load is distributed among all used banks and channels. By using more channels or banks, the per-channel and the per-bank load is also diminished. This is explained by the fact that load distribution results in a smaller number of waiting requests. It is around 150 requests for configurations A, B and D and falls by 50% to around 75 requests for configurations F, H and I. We denote Ci-Bj conflicts of bank j in channel i.

5 Conclusion

We have presented the P-Ware framework allowing for analyzing both SW and HW performance using flexible and variable transaction-level latency components. Experimental results on a real-life application show that P-Ware is able to perform fast system-platform simulation for early evaluating quantitative impact (in terms of throughput, processor-utilization, bus bandwith and

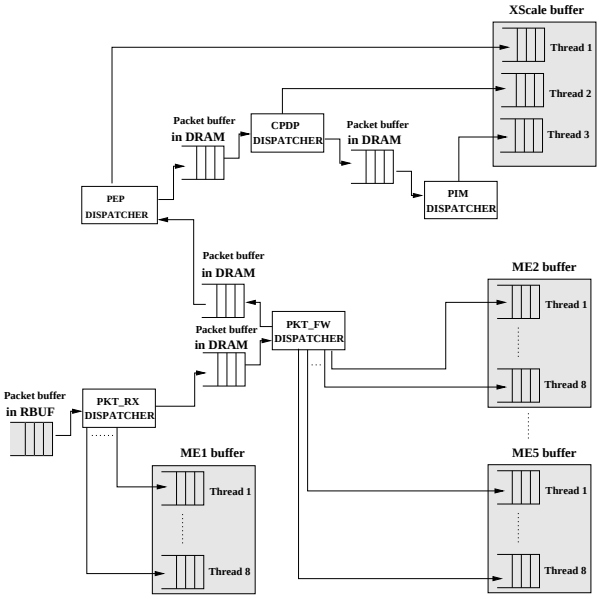


Figure 9: Mapping of INGRESS IPv4.

memory conflicts) of implementing embedded applications on various configurations of network processors.

The P-WARE tool implementation takes advantage from the separation of functional and timed behavior. A part of simulation code computes the processing result by fast classical software approach (without taking into account the internal structure of the component), while another part is devoted to the control of input/output and timed behavior of transactions.

References

[1] Teja tech. www.teja.com/content/teja_backgrounder.pdf. 1

[2] M. Adiletta, M. Rosenbluth, D. Bernstein, G. Wolrich, and H. Wilkinson. The next generation of intel ixp network processors. In *INTEL tech. journal*, volume 6, 2002. 1

[3] I. Augé, F. Petrot, F. Donnet, and P. Gomez. Platform-based design from parallel c specifications. In *IEEE Trans on CADICS*, volume 24, Dec 2005. 1

[4] D. Borodin, A. Terechko, B. Juurlink, and P. Stravers. Optimisation of multimedia applications for the philips wasabi multiprocessor system. In *ProRISC'05 Workshop*. 1

[5] W. Cesrio, A. Baghdadi, L. Gauthier, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. Jerraya, and M. Diaz-Nava. Component-based design approach for multicore socs. In *DAC '02*. ACM. 1

[6] R. Cornea, N. Dutt, R. Gupta, I. Krueger, A. Nicolau, D. Schmidt, and S. Shukla. Forge: A framework for optimization of distributed embedded systems software. In *IPDPS '03*. IEEE CS. 1

[7] J. Fu and O. Hagsand. Designing and evaluating network processor applications. In *HPSR'05*. 1

[8] F. Gharsalli, S. Meftali, F. Rousseau, and A. A. Jerraya. Automatic generation of embedded memory wrapper for multiprocessor soc. In *DAC '02*. ACM. 1

[9] F. Ghenassai. *Trasaction Level Modelling with SystemC. TLM Concepts and Applications for embedded systems*. Springer, 2006. 1

[10] S. Govind and R. Govindarajan. Performance modeling and architecture exploration of network processors. In *QEST'05*. 1

[11] M. Gries, C. Kulkarni, C. Sauer, and K. Keutzer. Comparing analytical modeling with simulation for network processors: A case study. In *DATE'03*. 1

[12] Teja IPv4. www.teja.com/content/4980_Teja_IPv4.pdf. 1, 4

[13] A. Jalabert, S. Murali, L. Benini, and G. De Micheli. Pipescompiler: A tool for instantiating application specific networks on chip. In *DATE*, 2004. 1

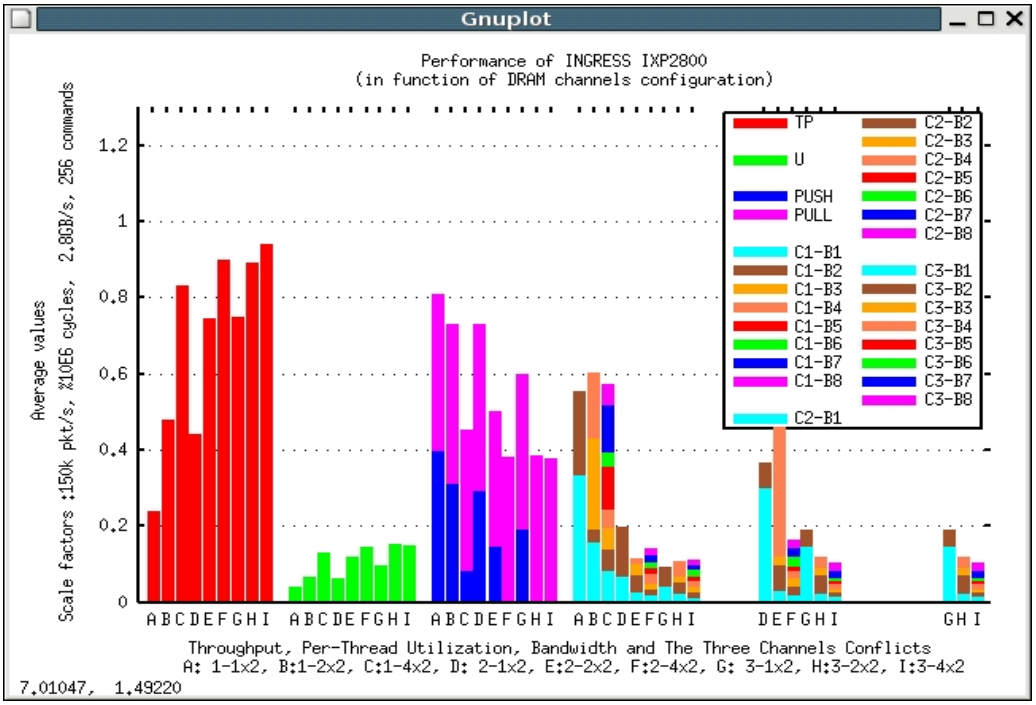


Figure 10: IPV4 and The INGRESS-IXP2800 Performance

[14] S. Mahadevan, M. Storgaard, J. Madsen, and K. M. Virk. ARTS: A system-level framework for modeling mpsoc components and analysis of their causality. In *MASCOTS'05*. 1

[15] J. M. Paul, D. E. Thomas, and A. S. Cassidy. High-level modeling and simulation of single-chip programmable heterogeneous multiprocessors. *ACM Trans. DAES*, 2005. 1

[16] P. G. Paulin, C. Pilkington, M. Langevin, E. Bensoudane, and G. Nicolescu. Parallel programming models for a multi-processor soc platform applied to high-speed traffic management. In *CODES+ISSS '04*. ACM. 1

[17] Alberto Sangiovanni-Vincentelli. Defining platform-based design. *EEdesign, EETimes*, February 2002. 1

[18] X. Zhu, W. Qin, and S. Malik. Modeling operation and microarchitcture concurrency for communication architectures with application to retargetable simulation. In *CODES+ISSS'04*. ACM. 1