Modeling and Analysis of Core-Centric Network Processors

YI-NENG LIN and YING-DAR LIN
National Chiao Tung University
YUAN-CHENG LAI
National Taiwan University of Science and Technology
and
KUO-KUN TSENG
National Chiao Tung University

Network processors can be categorized into two types, the coprocessors-centric model in which data-plane is handled by coprocessors, and the core-centric model in which the core processes most of the data-plane packets yet offloading some tasks to coprocessors. While the former has been properly explored over various applications, research regarding the latter remain limited. Based on the previous experience of prototyping the virtual private network (VPN) over the IXP425 network processor, this work aims to derive design implications for the core-centric model performing computational intensive applications. From system and IC vendors' perspectives, the continuous-time Markov chain and Petri net simulations are adopted to explore this architecture. Analytical results prove to be quite inline with those of the simulation and implementation. With subsequent investigation, we find that appropriate process run lengths can improve the effective core utilization by 2.26 times, and by offloading the throughput boosts 7.5 times. The results also suggest single-process programming, since context-switch overhead impacts considerably on the performance.

Categories and Subject Descriptors: C.3 [**Special Purpose and Application-Based Systems**]: Real-time and Embedded Systems, Microprocessor/Microcomputer Applications; C.4 [**Performance of Systems**]: Design Studies, Modeling Techniques

General Terms: Performance, Verification

Additional Key Words and Phrases: Network processor, embedded system, modeling, core-centric, simulation

This work was supported in part by the Program for Promoting Academic Excellence of Universities of Taiwan National Science Council, and in part by Intel and Chung-Hua Telecom.

Authors' addresses: Y.-N. Lin, Y.-D. Lin, and K.-K. Tseng, National Chiao Tung University, Hsinchu, Taiwan; email: ynlin@cs.nctu.edu.tw. Y. C. Lai, National Taiwan University of Science and Technology, Taipei, Taiwan.

ACM Reference Format:
Lin, Y.-N., Lin, Y.-D., Lai, Y.-C., and Tseng, K.-K. 2008. Modeling and analysis of core-centric network processors. ACM Trans. Embedd. Comput. Syst. 7, 4, Article 41 (July 2008), 15 pages. DOI = 10.1145/1376804.1376809 http://doi.acm.org/10.1145/1376804.1376809

1. INTRODUCTION

Networking applications offering extra security and content-aware processing features demand much more powerful hardware platforms to achieve high performance. For computational intensive applications such as the virtual private network (VPN) [Braun et al. 1999], general-purpose processors are often adopted; however, the cost is considerable while the throughput is not satisfactory because of the heavy cryptographic operations. Rather, the application-specific integrated circuits (ASICs) [Smith 1997] can meet the performance requirement with a circuitry designed for both networking and cryptographic processing. Nonetheless, the lack of adaptability makes it less appealing.

Network processors [Lekkas 2003] have been embraced as an alternative to tackle the above-mentioned problems for their *core-processor/coprocessors*-based architecture on which control and data-plane processing are separated for efficiency and their reprogrammability for functional adaptations. The core processor can perform complicated operations and is thus responsible for control messages, while a number of multithreaded coprocessors, having specifically designed instructions for networking purposes, are employed for mass data-plane processing. This kind of architecture, referred to as the *coprocessors-centric* model, is frequently applied as a core device, which requires moderate configurability, but high scalability [Lin et al. 2003; Clark et al. 2004; Comer and Martynov 2006; Lin et al. 2007; Tan et al. 2004]. When implementing an edge device that deals with relatively mild traffic volume, both control and data-plane packets are processed by the core processor. This is referred to as the *core-centric* model. Nonetheless, routinelike and computational intensive tasks such as receiving, transmission, and en-/de-cryption can still be offloaded to certain application-specific coprocessors [Lin et al. 2005].

Several studies have acknowledged the feasibility of adopting these models in packet processing for applications such as DiffServ, VPN, Cryptographic algorithms, and intrusion detection and prevention (IDP). In addition to evaluation through implementing both models to discover system bottlenecks [Lin et al. 2003; Clark et al. 2004; Comer and Martynov 2006; Lin et al. 2007; Tan et al. 2004; Lin et al. 2005], mathematical modeling [Crowley and Bear 2002; Wolf and Franklin 2006; Lu and Wang 2006] is favored for the coprocessors-centric model in order to unveil design implications that are unlikely to be observed through real benchmarking. However, analytical resort for the emerging core-centric model is yet unattempted.

In this work, we analyze the untapped core-centric network processors by modeling the IXP425 performing VPN. The IXP425 [Intel IXP425] employs an XScale core processor in charge of general packet processing and coprocessors executing receiving, transmission, and cryptographic operations. Analytical models of two schemes, the *busy-waiting* (BW) and *interrupt-driven* (ID) scheme, are developed using the continuous-time Markov chain, a method widely adopted for modeling complex systems. In the BW scheme, the core hands over the intermediate results to the coprocessor for application-specific processing and waits for the coprocessor to finish. This primitive approach in which the core wastes a significant time waiting is used by some operating systems, for instance, NetBSD, to cooperate with coprocessors. The BW scheme is then revised and extended to the ID scheme. In this scheme, the multiprocess mechanism is incorporated and the core switches, with overhead in terms of delay, to another process when offloading a certain task to a coprocessor or after a certain process run length. This technique is realized in NetBSD by enabling the OCF (open crypto framework) option. Though busy-waiting overhead is significantly alleviated, efficiency of the core in this scheme is subject to not only the *context-switch overhead* but also *process run lengths* among multiple processes. The inappropriate process run lengths could lead to poor allocation of computational resource.

Besides the analytical models, a simulation framework is developed for inspecting internal characteristics of the system, which oftentimes cannot be obtained from real implementations because of the limitation of the monitoring facilities and from mathematical analysis as a result of enormous state space. With these established analytical and simulation models, we try to reveal some design implications for future core-centric NPs. Since the core is frequently found to be the bottleneck [Lin et al. 2005], the main track of the investigation focuses on mechanisms to improve (1) the core efficiency and (2) overall performance. We study techniques for the former by analyzing the process run length and context-switch overhead and, for the latter, through exploring the benefits from offloading. The modeling framework and design implications are believed to be useful in rapid system prototyping.

This article is organized as follows. Section 2 briefs the overview of the core-centric IXP425 network processor system and describes architectural assumptions. We elaborate the analytical models and simulation design in Sections 3 and 4, respectively. Section 5 presents the results and observations. Some conclusive remarks of this article are made in Section 6.

2. BACKGROUND

2.1 Performance Model Overview

The core of Intel IXP425 is an XScale processor handling system initialization and software objects execution. Three buses interconnected by two bridges provide the connectivity among components on IXP425. To assist the XScale core in processing networking packets, two coprocessors called network processor engines (NPEs) are used for providing functions such as MAC filtering and CRC checking/generation, in cooperation with an encryption/decryption coprocessor. Our analytical models for the processing flow are based on the implementation of VPN over the IXP425 network processor. As shown in Figure 1, the processing flow can be summarized in five processing stages namely (1) receiving, (2) IPSec preprocessing, (3) en-/decryption, (4) IP processing, and, finally,

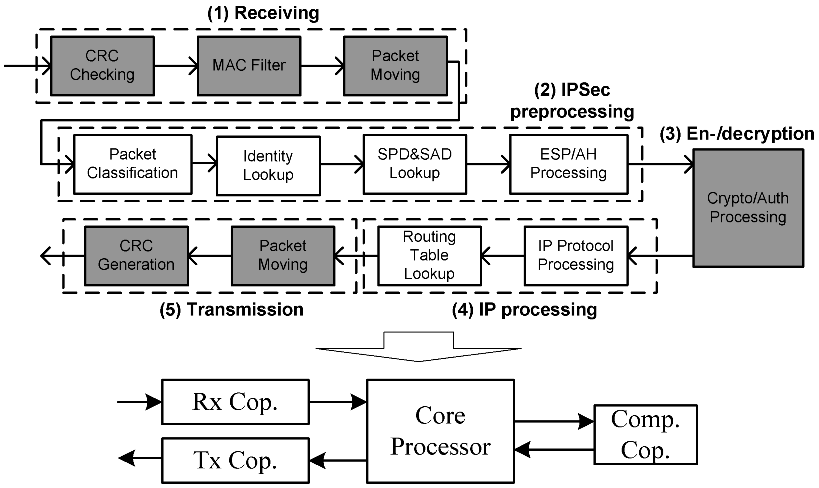

Fig. 1. Processing flow and task allocation of the VPN application over IXP425: physical and logical views.

(5) transmission. The shadowed blocks, i.e. tasks 1, 3 and 5, are offloaded to corresponding coprocessors, namely, the receiving coprocessor, computational coprocessor, and transmission coprocessor, respectively, whereas tasks 2 and 4 are handled by the core through multiprogramming and context switching.

2.2 Architectural Assumptions

Some coprocessors may incorporate multiple hardware threads [Intel IXP2400] to alleviate memory access latency by switching out the processor control from a thread to another when issuing a memory access. Nevertheless, hardware multithreading requires duplicate register sets, which increase the cost and is helpful only for memory-access intensive applications, such as DiffServ and IDP. Therefore, in this work we assume single thread in each coprocessor since VPN is computational intensive, rather than memory-access intensive. Buffer space for each processing stage is also encompassed, except for the BW scheme, which needs no buffer between the core and the computational coprocessor.

3. ANALYTICAL MODEL

3.1 The Busy-Waiting Scheme

In this scheme, no buffer is present between the core and computational coprocessor. Therefore, the core has to wait on the signal of completion before it can resume execution. For example, when the core finishes the IPSec preprocessing, the result is passed (offloaded) to the computational coprocessor for en-/decryption and is thereafter again handed over to the core for IP processing. In this regards, the core and the computational coprocessor can be seen as different processes in a logical CORE processor, since only one of them can be active at any time. The scheme can further be simplified as a series of queues consisting of three servicing stations, as shown in Figure 2. In this abstraction, all stations

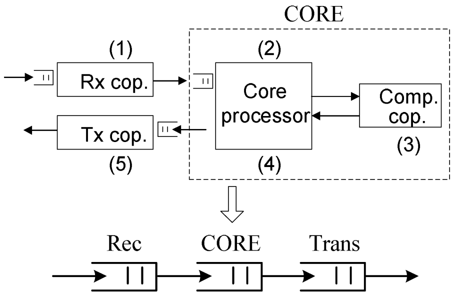

Fig. 2. The BW scheme.

are independent $M/M/1/\infty$ models and the departure-time distribution from a queue is identical to the interarrival-time distribution of another. The utilizations of the receiving and transmission coprocessors are trivial, whereas, for CORE, it can be derived as

$$U_{CORE} = \frac{\mu_R}{(T_{Core\_A} + T_{cop} + T_{Core\_B})^{-1}}, \tag{1}$$

where μ_R denotes the mean arrival rate at the CORE which is the same with the mean departure rate from the receiving coprocessor. T_{Core_A}, T_{cop} and T_{Core_B} represent the processing time for IPSec preprocessing, en-/decryption and IP processing, respectively. Finally, we can obtain the utilizations for core and computational coprocessor as

$$U_{Core} = U_{CORE} \times \frac{T_{Core\_A} + T_{Core\_B}}{T_{Core\_A} + T_{Cop} + T_{Core\_B}}, \tag{2}$$

and

$$U_{Cop} = U_{CORE} - U_{Core}, \text{ respectively,} \tag{3}$$

since the CORE consists of both the core processor and computational coprocessor.

3.2 The Interrupt-Driven Scheme

Contrasted with BW, in the ID scheme, the core passes the results from IPSec preprocessing to the computational coprocessor and resumes without being blocked. To realize this concept, two processes need to be forked in the core for IPSec preprocessing and IP processing, respectively, and buffer is required between the core and coprocessor. When the IPSec preprocessing is done and the packet is passed to the coprocessor's buffer, the core switches, with a switching delay T_D, to the other process for performing IP-related operations so that the core is not stalled; context switches also occur after a certain period of process run length. To reflect this enhancement, a *processor-control switch* referred to as PCS is adopted to capture activities of the two processes. According to the above descriptions we can formally define a state of the system as

$$ST = (R, A, C, B, T, S),$$

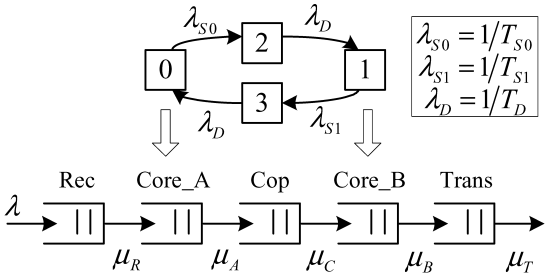

Fig. 3. The interrupt-driven model.

Table I. Notations for the Analytical Models

• λ	denotes mean packet arrival rate.
• T_{S0}	denotes the mean run length of PCS at Core_A.
• T_{S1}	denotes the mean run length of PCS at Core_B.
• λ_{S0}	denotes the mean switching rate of PCS from 0 to 1. $\lambda_{S0} = 1/T_{S0}$.
• λ_{S1}	denotes the mean switching rate of PCS from 1 to 0. $\lambda_{S1} = 1/T_{S1}$.
• T_D	denotes the mean context switch delay.
• λ_D	denotes $1/T_D$
• μ_X	denotes the mean service rate of processing stage X.
• buf_X	denotes the buffer size of processing stage X.

where *R, A, C, B and T* denote the queue lengths for the five processing stages, namely receiving, IPSec preprocessing referred to as Core_A, en/de –cryption referred to as Cop, IP processing referred to as Core_B, and transmission, while S denotes the PCS. As shown in Figure 3, $S = 0/S = 1$ means the core is processing packets at Core_A/Core_B; it then switches to an intermediate state $S = 2/S = 3$ after a run length of T_{S0}/T_{S1}, and waits for T_D to finish the context switch. Parameters used in the analytical model are described in Table I.

Even with the multiprocess mechanism, the core could still be busy-waiting for (1) packet arrivals from its predecessor and (2) available buffer slots in its successor to which the processing result is passed. In this way, the *effective core utilization*, which is derived by subtracting the busy-waiting overhead from *overall core utilization*, suffers. Therefore, the PCS should be manipulated well to avoid these situations by (1) setting appropriate run lengths T_{S0} and T_{S1} so that the processing resource is reasonably distributed, and (2) characterizing appropriate transition conditions so as to ensure that context switches are performed upon those conditions. All possible transitions are listed in Table II and explained as follows.

As presented in Table II, all transitions for the five processing stages can be classified into (1) arrival, in which packets are received from the network or predecessor, and (2) departure, in which packets are passed to the successor or transmitted. The only stage having an arrival rate is the receiver (λ), while others do not because arrivals are activated by their predecessors. To hand over the processed result to the next thread, it is necessary that the successor has available buffer space $(X < buf_R)$ to accommodate the processed results. Besides the buffer space requirement in the successor, when acting as a predecessor, the Core_A and Core_B have to additionally acquire the processor control, namely

Table II. State Transitions in the ID Scheme

Component	Transition Type	Condition(s)	Rate
Receiving	Arrival	$R < buf_R$	λ
	Departure	$A < buf_A$	μ_R
Core_A	Arrival	follower of Rec	n/a
	Departure	$PCS = 0, C < buf_C$	μ_A
Cop	Arrival	follower of Core_A	n/a
	Departure	$B < buf_B$	μ_C
Core_B	Arrival	follower of Cop	n/a
	Departure	$PCS = 1, T < buf_T$	μ_B
Transmission	Arrival	follower of Core_B	n/a
	Departure	n/a	μ_T
PCS	0 => 2	$A > 0, B > 0$	λ_{S1}
		$(A = 0, B > 0)$ or $C = buf_C$	∞
		$(B = 0)$ or $T = buf_T$	0
	2 => 1	n/a	λ_D
	1 => 3	$A > 0, B > 0$	λ_{S2}
		$(A > 0, B = 0)$ or $T = buf_T$	∞
		$(A = 0)$ or $C = buf_C$	0
	3 => 0	n/a	λ_D

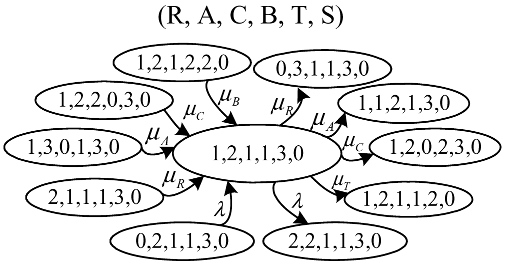

Fig. 4. Example state transitions of the interrupt-driven model.

the PCS, in order to perform packet processing. A PCS value of 0/1 means Core_A/Core_B controls the processor. The PCS starts to switch from 0 or 1 if (1) the run length has been reached (T_{S0} for Core_A and T_{S1} for Core_B), or (2) the corresponding active stage (either Core_A or Core_B in which the PCS resides) does not have any packet but the other does. No switching will occur if (1) the target stage to which PCS is going to switch has no packets or (2) the successor of the target stage has no buffer left. When the PCS switches from 0 to 1 or from 1 to 0, a context switch delay T_D is counted.

Figure 4 depicts an example state whose buffer size equals to 3 for each stage and the corresponding transitions in which five are transiting into the state and others are transiting out from the state. While the inbound and outbound transitions are comprehensible, special note needs to be taken for the PCS, namely the PCS in the example state never transits from 0 to 1. This is because the Core_B cannot pass the processed packet to the transmission stage, which is already full. The transition of PCS from 0 to 1 simply contributes to the unnecessary overhead and, therefore, is considered an invalid transition.

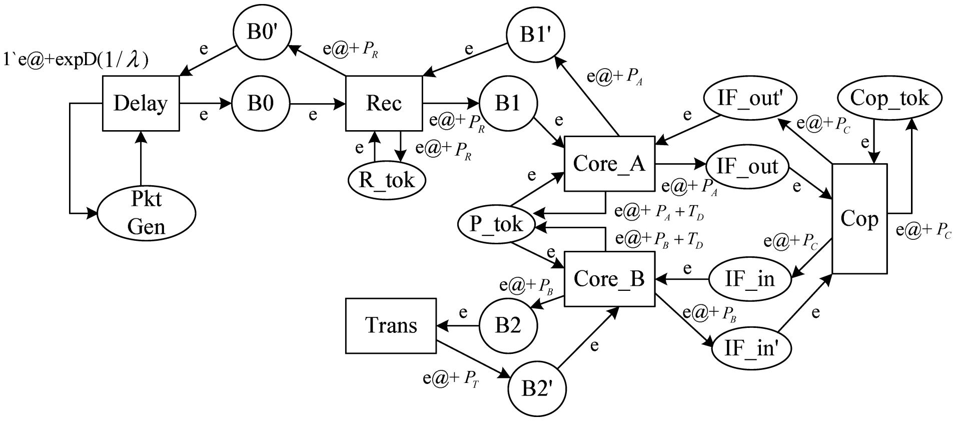

Fig. 5. Petri Net for the ID scheme.

4. PETRI NET SIMULATION ENVIRONMENT

Tools have been available for simulating architectures similar to NPs [Nussbaum et al. 2004; Davis et al. 2005]. Though accurate, they focus mainly on the low-level configuration such as cache structure and lack flexibility in task allocation. In this section, we describe the construction of the simulation environment based on the timed and colored Petri Nets (CPNs) [Murata 1989; Zuberek et al. 1998] which captures component-level activities well. It is used to validate the analytical model discussed in the previous section as well as to observe possible hints for future NP design.

We adopt the event-driven CPN-Tools [Ratzer 2003] as our simulator. The features it supports, including the colored tokens, stochastic functions, and hierarchical editing, provide efficiency in the construction of timed and CPNs corresponding to our model. As shown in Figure 5, the net of the ID scheme contains five processing stages presented as*transitions* (Rec, Core_A, cop, Core_B, and Trans). Each is associated with a control token indicating the availability of the processing resource and equipped with a *place* representing buffers, namely B0, B1, IF_out, IF_in, and B2. The size of the buffers is configured in other five places, i.e. B0', B1', IF_out', IF_in' and B2', respectively, by marking them with a number of initial tokens. The following description exemplifies a sample processing flow.

When a packet arrives at the receiving coprocessor, B0, with the interarrival time being exponentially distributed with mean $^1/_\lambda$, one token in B0, is consumed, indicating the occupation of a buffer slot. Once the receiving coprocessor is available (namely, the R_tok place contains a token), the packet is processed for P_R μs and then passed to the Core_A stage if there is room $(B1' > 0)$, while the tokens go back to R_tok and B0$'$, respectively. If the token in P_tok is available, meaning Core_B is not executing, Core_A will start to process the packet for P_A μs and then offloads en-/decryption operations to the computational coprocessor which takes P_C μs. Notably, the token returning to P_tok costs an additional T_D μs for context switch overhead. Similar procedures apply to Core_B and the transmission coprocessor which last for P_B and P_T μs,

Table III. Processing Time of the Tasks Evaluated in a Real Implementation[a]

Task	Processing Time (μs/packet)
(1) Receiving (Rec)	27.3
(2) IPSec preprocessing (Core_A)	31
(3) En-/decryption (Cop)	12.6
(4) IP processing (Core_B)	49
(5) Transmission (Trans)	27.3

[a] Process run lengths and context switch are to be configured.

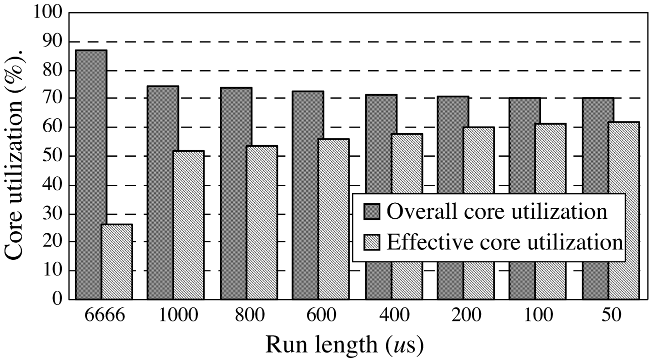

Fig. 6. Run lenth ($T_{S0} = T_{S1}$) versus core utilization under an input load of 35 Mbps.

respectively. The net of the BW scheme can be similarly built except that no buffer is present between Core_A and Cop and between Cop and Core_B, that is, both IF_out′ and IF_in′ are excluded from the net. Although the nets quite resemble the analytical models, some differences exist. First, the processing time of each stage is deterministic in simulation yet exponentially distributed in the analysis. Second, the run lengths of Core_A and Core_B cannot be configured because of the limitation of the tool in simulation.

5. EVALUATION

In this section, we first validate the analytical model against simulations and implementation benchmark. We then evaluate the core-centric network processors of the ID and BW schemes, trying to identify design implications.

5.1 System Parameter Determination

Parameter settings for the analytical model as well as the simulation are listed in Table III. We then try to find the most appropriate process run length for PCS. To rule out the interference of contexts-switch overhead, T_D is configured to a value close to zero. As Figure 6 presents, compared to the normal run length of 6666 μs [Microsoft], when choosing 100 μs we can have 2.26 times improvement on the effective core utilization while consuming 20.5% less computational resource. Busy-waiting overhead, which is the difference of the overall core and the effective core utilization, is significantly alleviated. This is credited to the frequent switching that helps avoid much backlog and, therefore, possible stall, at components.

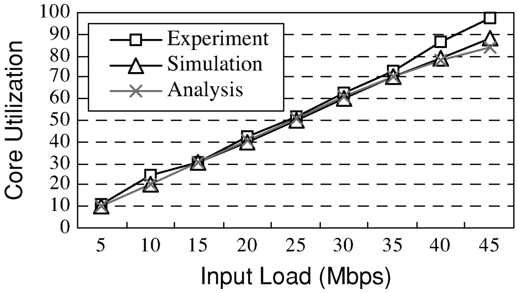

Fig. 7. Analytical model validation against the simulation and real implementation.

5.2 Validation of the Analytical Model

In order to make the behavior of the simulation inline the analytical model and implementation, the context switch delay, T_D, has been gradually adjusted. We finally find that, with T_D being very close to 0 the analytical results are mostly within 1% of the simulation, as presented in Figure 7. The discrepancy comes from different assumptions between the analysis and simulation. The former assumes exponential distribution for the packet interarrival time and instruction-processing time, while the latter uses deterministic ones in order to be realistic. It is implied that the context-switch delay is minor in the implementation, which is quite unreasonable, indicating that *only one process in the core is employed practically for both IPSec preprocessing and IP processing*. The utilization of the implementation is slightly higher (3–4%) than the analytical model because of the operating system overhead. The discrepancy noticeably increases when being overloaded. It is also surprising that the limited buffer size, which is configured to 3 in the analytical model, does not influence the model accuracy when the load is under system capacity. As the analytical model closely matches with the simulation and is configurable for parameters such as process run lengths, we conduct the investigation mostly through the former. The simulation is adopted only for large buffer sizes, which are frequently not feasible in the analytical model because of the state–space explosion.

5.3 Differentiated Run Lengths

Run lengths have been shown to influence the system performance. Rather than having the same run length for Core_A and Core_B, whose processing times are different, it is sensible to differentiate them so as to distribute the computational resource according to load. Figure 8 presents the results of the differentiation, in which T_{S0} is configured as 100μs found appropriate previously. It is observed that the system performance improves as T_{S1} increases. While the largest advance occurs at $T_{S1} = 200$, minor improvement continues even when $T_{S1} = 6666$. This is credited to the conditions that force the PCS to switch from 1. For example, the PCS switches immediately if the transmission stage is full or if the Core_B stage contains no packets, so that busy-waiting can be avoided.

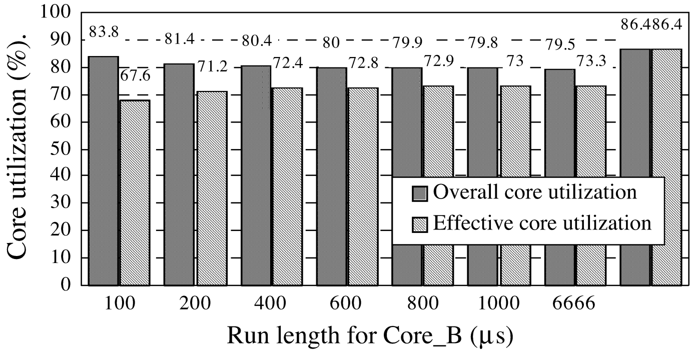

Fig. 8. Benefits from differentiated run lengths for Core_A and Core_B. T_{S0} is configured as 100 μs.

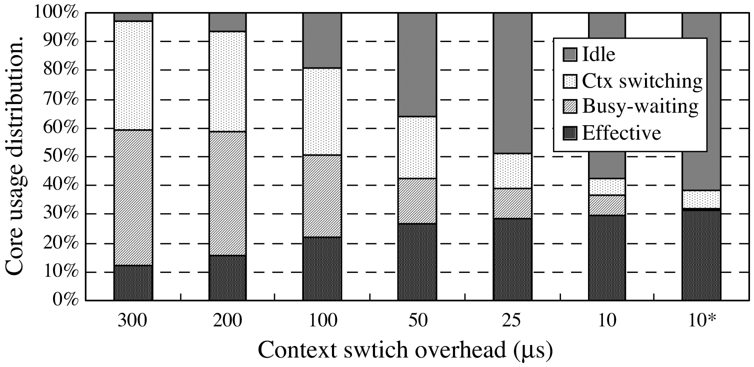

Fig. 9. Core usage distribution for different context-switch delays. The asterisk means T_{S0} and T_{S1} are configured to 100 and 200 μs.

The BW scheme is also involved in this comparison, in which single process is implemented in the Core, namely, no context switch occurs during execution. Surprisingly, it outperforms the ID scheme because (1) context-switch overhead is not present because of the single-process mechanism and (2) the Cop is efficient enough so that the Core spends limited time waiting.

5.4 Effect of the Context-Switch Overhead

Though context switching is helpful in alleviating the busy-waiting overhead and hiding memory-access latency, for computational-intensive applications it could jeopardize the performance, as Figure 9 explains. From the figure, we note that a delay of 300 μs leads to low effective utilization (12%), but considerable context switching and busy-waiting burdens (38% and 47%). As the delay reduces, not only the core utilizes effectively but also the overhead is lessened. The burden from busy-waiting can even be annihilated when $T_D = 10$ and T_{S0} and T_{S1} are further configured to 100 and 200 μs, respectively. However, since a context switch delay of 10 μs is quite unrealistic for current XScale core implementation (except for some coprocessors with hardware multithreads [Intel IXP2400]), this result is also advising that system vendors adopt a single

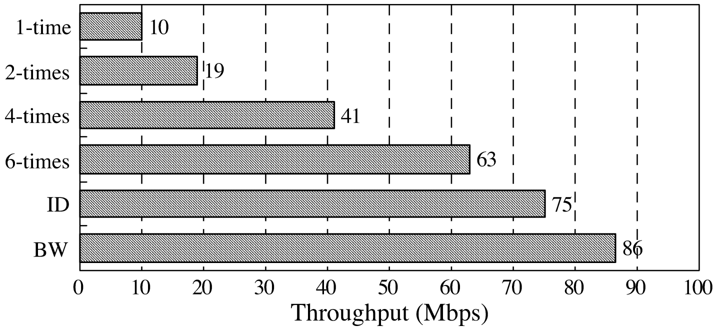

Fig. 10. Throughput of various offloading schemes. The clock rate of the XScale core in the implementation is 533 MHz, as a reference for comparison.

process for handling multiple tasks. Though the process occupies large memory space, the cost is quite affordable for today's memory technology whereas context-switch overhead is substantially alleviated.

5.5 Benefit from Offloading

Offloading complex and routine tasks to specially designed coprocessors has been an alternative to simply speeding up the core processor. However, the benefit from offloading is not well investigated. Figure 10 demonstrates the gain of doing cryptographic operations, which is the most time-consuming task, by (1) multiplying the core clock rate and through (2) offloading to the computational coprocessor. The former includes (1) no speedup and (2) speedup for two, four and six times for the core processor, while the latter involves both ID and BW schemes. As revealed in the figure, the throughput increases in direct proportion to the speedups. Nonetheless, the ID scheme still outperforms the unoffloaded one, equipped with a core of six-time speedup resembling a 3.2 GHz P4 processor. Finally, we can see that the BW scheme scores the highest throughput, as explained previously.

The performance figures can be validated as follows. Let the capability of the core be m cycles/s, and the processing time for Core_A, en-/decryption and Core_B be x, y, and z cycles/Mbits, respectively. We can have

$$\frac{m}{x+y+z} = 10\,(\text{Mbps}), \tag{4}$$

since the throughput of an ordinary core without offloading is 10 Mbps. Moreover, because the core, namely, XScale in the real implementation is the performance bottleneck [Lin et al. 2005], we can also have

$$\frac{m}{x+z} = T\,(\text{Mbps}), \tag{5}$$

where T represents the throughput of the core executing Core_A and Core_B and therefore the throughput of the ID scheme as well. With (4) and (5) we can have

$$\frac{(5)}{(4)} : \frac{z+y+z}{x+z} = \frac{T}{10}. \tag{6}$$

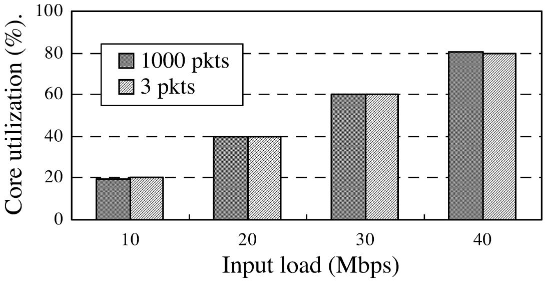

Fig. 11. Core utilization for two buffer sizes under different input loads.

Since $y : (x + z) = (31 + 49) : 12.6 \cong 6.4$, according to Table III, the throughput T can finally be derived as

$$T = 10 \times \frac{1 + 6.4}{1} = 74\,(\text{Mbps}), \tag{7}$$

which is very much close to the one from the analytical model.

5.6 Effect of Buffer Sizes

As pointed out earlier this section, the buffer size does not impact much on the accuracy of the model. This is verified in Figure 11, which compares two significantly different sizes, 3 and 1000 where the latter case is performed through simulation instead of analysis because of large state space. From the figure we can see that the core utilization is the same for both sizes when input load is below the system capability, which is about 45 Mbps. This implies that buffer sizes need not be large as long as the load does not go beyond the system capacity.

6. CONCLUSION AND FUTURE WORK

This work aims at deriving possible design implications for the core-centric network processors through analytical modeling as well as Petri net simulations. The computational-intensive VPN application, which has some complex, but routine, tasks is adopted to explore the benefits from offloading to coprocessors. Two implementation schemes of the Core, namely the busy-waiting and interrupt-driven schemes, are introduced and compared. To date, this work is the first research that practically models this emerging architecture.

The analytical model is verified to have behaviors closely in line with the simulation (within 1%) and the implementation (within 3–4%). Detailed investigations are then carried out on mechanisms that (1) exploit efficiently the core processor which is usually the bottleneck and (2) improve the system performance. Through both analytical and simulation measures, we observe that

—by adopting appropriate process run lengths, 2.26 times improvement on the effective core utilization and 20.5% less consumption on the computational resource can be achieved; better results can be obtained if run lengths of stages are further differentiated according to the processing time;

—by reducing the context-switch delay from 300 to 10 μs we can have 2.6 times advance on the effective core utilization, and the switching overhead and busy-waiting time can be alleviated by as much as 90%; this observation also strongly suggests the use of (1) a single process for multiple tasks or (2) hardware multithreading, since 10 μs delay is normally unfeasible for today's multiprocess technique;

—the busy waiting scheme may be considered as long as the coprocessor is fast enough;

—incorporating coprocessors for the bottleneck task, namely, the en-/decryption, boosts the throughput for 7.5 times compared to that of single core processor;

—buffer sizes need not to be large as long as the load has not gone beyond the system capacity.

We believe the first three findings are useful for system vendors while the others may interest IC vendors. Discovery and the modeling framework developed in this study should be applicable to network processors of similar architecture and beneficial to rapid system prototyping.

As a future direction, we plan to extend this approach by considering memory-access intensive applications such as IDP (Intrusion Detection and Prevention). In such extension, memory access operations can be offloaded to coprocessors specifically designed with a wide memory bus. To further analyze the potential memory bottleneck, the model can also involve multiple memory modules or multiport memory to accommodate concurrent accesses.

REFERENCES

Braun, T., Günter, M., Kasumi, M., and Khalil, I. 1999. Virtual private network architecture. Technical rep. IAM-99-001, CATI.

Clark, C. et al. 2004. A hardware platform for network intrusion detection and prevention. In *Proceedings of the 3rd Workshop on Network Processors and Applications (NP3)*. Madrid, Spain.

Comer, D. and Martynov, M. 2006. Building experimental virtual routers with network processors. In *Proceedings of the 2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM)*.

Crowley, P. and Baer, J.-L. 2002. A modeling framework for network processor systems. In *Proceedings of the Network Processor Workshop in Conjunction with 8th International Symposium on High Performance Computer Architecture (NP1)*.

Davis, J. D., Fu, C., and Laudon, J. 2005. The RASE (rapid, accurate simulation environment) for Chip Multiprocessors. In *Proceedings of the Workshop on Design, Architecture and Simulation of Chip Multiprocessors (dasCMP'05)*.

Intel IXP425 Network Processor. http://www.intel.com/design/network/products/npfamily/ixp425.htm.

Intel IXP2400 Network Processor. http://www.intel.com/design/network/products/npfamily/ixp2400.htm.

Lekkas, P. C. 2003. *Network Processors: Architectures, Protocols and Platforms (Telecom Engineering)*. McGraw-Hill, New York.

Lin, Y.-D., Lin, Y.-N., Yang, S.-C., and Lin, Y.-S. 2003. DiffServ edge routers over network processors: Implementation and evaluation. *IEEE Netw. 17*, 4, 28–34.

Lin, Y.-N., Lin, C.-H., Lin, Y.-D., and Lai, Y.-C. 2005. VPN Gateways over network processors: Implementation and evaluation. In *Proceedings of the 11th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'05)*. San Francisco, CA.

Lin, Y.-N., Chang, Y.-C., Lin, Y.-D., and Lai, Y.-C. Lai. 2007. Resource allocation in network processors for memory access intensive applications. *J. Syst. Softw. 80*, 7.

Lu, J. and Wang, J. 2006. Analytical performance analysis of network-processor-based application designs. In *Proceedings of the 15th International Conference on Computer Communications and Networks (IC3N'06)*. IEEE Press, Los Alamitos, CA. 33–39.

Microsoft TechNet. http://www.microsoft.com/technet/prodtechnol/windows2000serv/reskit/core/fned_ana_trrf.mspx?mfr=true.

Murata, T. 1989. Petri Nets: Properties, analysis and applications. *Proc. IEEE 77*, 4.

Nussbaum, D., Fedorova, A., and Small, C. 2004. An Overview of the Sam CMT simulator kit. Technical rep., Sun Microsystems.

Ratzer, A.V. et al. 2003. CPN Tools for editing, simulating, and analysing Coloured Petri Nets. In *Proceedings of the International Conference on Applications and Theory of Petri Nets*.

Smith, J. M. S. 1997. *Application-Specific Integrated Circuits*. Addison-Wesley, Reading, MA.

Tan, Z., Lin, C., Yin, H., and Li, B. 2004. Optimization and benchmark of cryptographic algorithms on network processors. *IEEE Micro 24*, 5, 55–69.

Wolf, T. and Franklin, M. K. 2006. Performance models for network processor design. *IEEE Trans. Parall. Distrib. Syst. 17*, 6, 548–561.

Zuberek, W. M., Govindarajan, R., and Suciu, F. 1998. Timed colored Petri net models of distributed memory multithreaded multiprocessors. In *Proceedings of the Workshop on Practical Use of Colored Petri Nets and Design/CPN*.

Received July 2007; accepted March 2008