This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2007 proceedings.

On the Extreme Parallelism Inside Next-Generation Network Processors

Lei Shi, Yue Zhang, Jianming Yu, Bo Xu, Bin Liu, Jun Li

***Abstract*--Next-generation high-end Network Processors (NP) must address demands from both diversified applications and ever-increasing traffic pressure. One major challenge is to design an extraordinary scalable architecture. In this paper, it is argued that such an objective can only be sufficed by introducing highly paralleled structure, namely the Paralleled Processing-engine Cluster (PPC). We demonstrate this point from the trade-off among aspects such as performance, programmability and flexibility. However, PPC natively suffers from several critical issues on load-balancing, intra-flow packet ordering and memory contention. After investigating several existing approaches, we present novel solutions for each issue according to the balance between performance and coast. Through intensive analysis and comprehensive simulations, it is shown that the Shortest Queue First scheduling with Class-based prediction (SQF-C) performs nearly optimally, while the hardware based per-flow ordering mechanism resolves packet out-of-order independently with the load-balancing issue, inducting little throughput degradation. Implementing the unified solution, it is capable to design a PPC supporting up to OC-768c line rate. Real implementation is also carried out in our THNPU-1 prototype to verify the conclusions.**

***Keywords*—Network Processor, Parallelism, Load-balancing.**

I. INTRODUCTION

Traditional IP routers rely on Application Specific Integrated Circuit (ASIC) to deal with the expeditiously growing packet processing requirements. At the same time, as the emergence of modern Internet applications, such as triple-plays [1] and P2P [2], more flexibility beyond ASIC is also extremely desired. To answer that, Network Processor (NP) is introduced to combine ASIC's high-speed superiority and General Purpose Processor's (GPP) outstanding flexibility. Due to its enhanced programmability, the time-to-market cycle of NP based products is greatly shortened. Again the same reason lengthens their time-in-market.

To build such a high-end NP, a pragmatic challenge is the development of highly scalable internal architecture, lying on which, engenders the main problems studied in this paper:

Lei Shi, Yue Zhang and Bin Liu are with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. (email: shijim @mails.tsinghua.edu.cn, zhang-yue@mails.tsinghua.edu.cn, liub@tsinghua. edu.cn.) Jianming Yu and Bo Xu are with the Department of Automation, and Research Institute of Information Technology, Tsinghua University, Beijing, China. (email: yujm03@mails.tsinghua.edu.cn, xb00@mails.tsinghua.edu.cn) Jun Li is with the Research Institute of Information Technology, Tsinghua University, and Tsinghua National Laboratory for Information Science and Technology, Beijing, China.(email: junl@tsinghua.edu.cn)

This work is supported by NSFC (No. 60373007, 60573121 and 60625201), the Cultivation Fund of the Key Scientific and Technical Innovation Project, Ministry of Education of China (No. 705003), the Specialized Research Fund for the Doctoral Program of Higher Education of China (No. 20040003048 and 20060003058), China/Ireland Science and Technology Collaboration Research Fund (2006DFA11170), and Tsinghua Basic Research Foundation (JCpy2005054).

1) *In what topology should high-end NP be constructed?*
2) *By which mechanisms can such a NP maximize its utilization?*

Previously, NP vendors have developed quite a lot of solutions. Cisco Systems Inc. fabricated its 40Gbps Silicon Packet Processor (SPP) [3] with highly paralleled cluster structure. Bay Microsystems just disclosed a new chip, called Chesapeake, designed to be an integrated NPU/TM device to operate at 40Gbps [4]. Xelerated Inc. designed X10q [5] in pipelined architecture with up to 200 stages. EzChip Ltd. provided NP-1 [6] in a hybrid topology, i.e., both paralleled and pipelined techniques are employed. Intel Corp. even designed NP [7] in a flexibly reconfigurable scheme. These vendors adopt diversified NP architecture in orientation to separate markets and demands. By now, it is still hazy which design is the best.

Recent studies in academia also cannot come to consensus. Nearly all possibilities within NP's design space [8] are exploited. M. Gries et al. show in [9] paralleled model is the best in dealing with IP forwarding application. At the same time, H. Liu argues that it cannot scale much due to the limited packet-level parallelism within Internet traffics [10]. N. Weng and T. Wolf adopt a random algorithm to map applications to NP's topology. They prefer the pipelined structure based on their simulation results [11]. Facing to such a multifarious research, it is strongly asked to have a thorough study extensively combining theoretical models with practical implementations on NP's architecture design.

In this paper, based on trade-off among NP's performance, programmability, flexibility, scalability, as well as power saving capability and system robustness, we derive a conservative sequitur on its architecture design that an extremely paralleled structure, namely Paralleled Processing-engine Cluster (PPC), is a must, at least functioning as one stage of high-end NP. This argument is validated through experiments on Intel IXP workbench [12].

However, due to the intrinsic nature of parallelism, PPC cannot maximize performance by straightforwardly stacking Processing Engines (PE) together. We focus on three major issues resulting in that: load-balancing, intra-flow packet ordering, and memory contention. We survey several existing methods, carry out analysis to evaluate their performance, and then present novel solution for each issue according to the trade-off between performance metrics and implementation cost. Our main contribution in this paper lies in the following:

1) We propose the Shortest Queue First scheduling with Class-based prediction algorithm (SQF-C), the per-flow ordering mechanism, as well as a distributed PPC memory hierarchy. Due to their decoupled properties, we are able to

0743-166X/07/$25.00 ©2007 IEEE

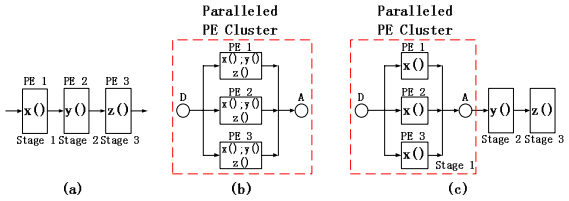

Fig. 1: (a) Pipelined model (b) Paralleled model (c) Hybrid model

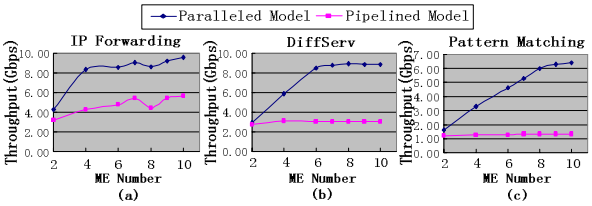

Fig. 2. Performance comparisons of paralleled model with pipelined model

combine them into a unified solution naturally.

2) Through performance analysis, we prove that PPC in our solution guarantees nearly 100% throughput, if only a speedup of 1.5 is provided at the aggregating unit. Furthermore, if the paralleled PEs further permit a speedup of 1.11, a delay bound at a hundred microseconds can be obtained under practical implementation parameters, no matter how many PEs are integrated. Trace-driven simulation results show that our unified solution performs load-balancing nearly optimally while inducting little throughput degradation from the packet ordering mechanism. This significantly overtops previous methods.

3) Investigations on hardware complexity of our solution show that it is scalable to cope with PPC running at up to OC-768c line rate.

We have applied PPC solution in a real implementation, THNPU-1, a FPGA based multi-stage NP prototype that fully supports OC-48c line rate IPv4/IPv6/MPLS forwarding.

In rest of this paper, we discuss necessities of PPC in Section II, propose unified solution to handle PPC issues in Section III, carry out performance analysis in Section IV, and present simulation results in Section V. Section VI introduces implementations and finally Section VII concludes the paper.

II. Why Parallelism is a Must in High-end Network Processor Design

A. General NP Models

State-of-the-art NP designs generally follow three styles of topology[1]: pipelined model, paralleled model and hybrid model, depicted in Fig. 1(a)(b)(c). In a pipelined model, the application is partitioned into several sequential micro-tasks and mapped to all the stages of the pipeline. In a paralleled model, multiple PEs are connected in a pool topology. Each PE handles part of arrival traffic. In the hybrid model, the former two are mixed, with a high-level pipelined topology and paralleled sub-topology at each stage. We then compare their efficiencies in building a high-end NP from several critical aspects. Since in hybrid model, each stage follows a paralleled model, we only contrast the paralleled model with the pipelined one, and derive architecture choice for an entire NP or its single stage in case a hybrid model is adopted.

B. Trade-off between Performance and Programmability

The ideal performance of NP in paralleled model, measured by its throughput, increases linearly with PE number. Even accounting the degradation caused by load-balancing and packet ordering issue, the performance curve is only slightly biased referred to our results in Section IV and V. At the same time, users of such NP are transparent to its architecture by simply running the same code on each PE. It provides the best programmability.

On the other hand, users of NP in pipelined model need to divide each application uniformly across all the stages in order to maximize PE utilization, which affects its programmability. Even if application is divided automatically by embedded high-level compilers, it still cannot perform best due to the facts: 1) some atomic tasks are not dividable any more, such as pattern matching application; 2) some task's processing time is not even predictable, such as packet filtering; 3) the impacts of inter-stage communications can not be ignored.

In this sense, paralleled model is the better choice from the trade-off between performance and programmability. Experiments are also carried out on Intel Develop Workbench [12] to validate this point. We configure Intel IXP2800 to be paralleled and pipelined model respectively, both employing 2~10 MEs[2] to execute test applications. (IXP2800 has 16 MEs in total, but 6 out of 16 are fixed for packet Rx/Tx, Queue Management and Scheduling). Three standard applications are tested: IPv4 forwarding (FW), Diffserv (DS) and Pattern Matching (PM) [13]. Each time 1000 fixed-length packets are sent at the input. Packet length is set to 64 bytes for FW/DS and 104 bytes for PM, where 64 byte random payload in each packet is matched by 1000 8-byte-patterns. The task division across stages in pipelined mode is handled by compiler from Intel, which is normally better than a manual one.

Fig. 2 depicts the throughput dynamics when ME number increases. Results show that the paralleled model always performs better than the pipelined one. Under FW application in Fig. 2(a), the throughput in paralleled model increases before 4 MEs are employed, and then stays stable at about 9.5Gbps, while in pipelined model the throughput has never exceeded 6Gbps. Under DS application in Fig. 2(b), the curve of paralleled model is similar, but throughput this time keeps on increasing until 6 MEs are employed. This is because the system bottleneck after that has shifted from ME to memory bandwidth; DS requires less memory transaction but more ME processing compared with FW. Under PM application where computing complexity is the dominating factor, it is clearly shown in Fig. 2(c) that the paralleled model receives continuous throughput climb-up while the pipelined model remains nearly unchanged. This result validates our argument that PM application cannot be divided further into granules. In addition, in all our experiments, pipelined model suffers from throughput degradation also because of high communication complexity between all the consecutive stages.

C. Flexibility and Scalability

Next Generation high-end NP should also adapt with

[1] Multi-pipelining model can be classified to paralleled model, with a pipeline substituting for each paralleled PE.

[2] ME is used interchangeably with PE in Intel IXP.

evolvements of network protocols, processing tasks, traffic patterns, and be scalable to support the ever-increasing link speed. While NP in paralleled model meets these requirements perfectly by updating micro-codes of each PE homogeneously and altering the number of paralleled PEs adaptively, the pipelined NP generally fails to achieve that. The variation of processing task on each stage will not only affect itself, but also force the codes to be repartitioned and remapped across all the stages to guarantee maximal performance. This introduces sizable overhead. In addition, to provide higher processing capacity, solely adding stages is inadequate, repartition of processing task is also a must.

D. Power Saving and Robustness

As NP chip density rises with PE number, the power dissipation also becomes a critical scaling bottleneck. Authors in [14] has developed adaptive power saving techniques for the paralleled NP model by turning off a subset of PEs during light load periods. It brings significant reduction in power consumption (up to 30%) while introduces little impacts on overall throughput. Only a portion of PEs is affected in each power saving operation. However, this technique can not be extended to pipelined model, since the only PE in each stage cannot be turned off. Although frequency adaptation can be employed, all PEs are involved in each operation. When frequency adaptation is not continuously variable, its efficiency is normally worse than that of paralleled model.

The same issue also exists when system robustness is considered. Under failures of some PE, NP in paralleled model can gracefully deal with it by bypassing the broken PE. However, NP in pipelined model needs a great endeavor to achieve the same feature.

E. Paralleled PE Cluster

In respect that the paralleled model gains advantages in nearly all aspects illustrated above, we conclude that such architecture is essential in fabricating next-generation high-end NP. It will perform as entire NP in the pure paralleled model or a single stage in the hybrid model.

We name such structure the Paralleled PE Cluster (PPC). In what follows, we focus on the major scaling issues of PPC, and propose a unified solution to maximize its performance.

III. Exploiting Parallelism in High Speed Processing

A. Problem Statement

The main challenge that affects the PPC to maximize its utilization is the combined problem of load-balancing, intra-flow packet ordering, and memory contention.

1) *Load-balancing*: Packet arrivals at PPC should be distributed to each PE uniformly. Furthermore, this uniformity is not defined on the traffic volume to each PE directly, but on the workload measured by PE instruction cycles. This makes things complicated since the processing time of each packet can not be accurately estimated. In case the load-balancing is non-uniform, packet delay at the heavily loaded PE will be considerable large. At that time, due to the scarceness of on-chip buffer, packet loss is inevitable.

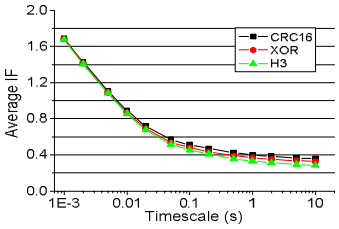

Fig. 3. Average IF using static hashing

2) *Intra-flow packet ordering*: The packets in same flow[3] should depart from PPC in the same order with their arrivals. This requirement comes from the intrinsic demands of upper layer protocols, such as TCP [15].

3) *Memory contention*: PPC should provide sufficient memory bandwidth, as networking applications are mostly data-intensive. Under line rate R, its overall bandwidth request reaches up to $R(2+\rho_p)$. Here ρ_p is ratio of average read/write bytes in each packet processing to the average packet length. Due to the well-known fact that memory technologies can not catch up with the increase of link speed, a proper memory hierarchy should be adopted to enhance system efficiency.

Although the above problems are always incorporated, we try to figure them out in a divide and conquer method.

B. Load-Balancing

Previously, several load-balancing algorithms have been proposed [16-20]. They can be partitioned into two categories: Flow-based Load-Balancing (FLB) and Packet-based Load-Balancing (PLB). FLB dispatches packets of a same flow to an unchanged PE, thus preserves packet order intrinsically; PLB assigns each packet independently, hence achieves fine uniformity. We investigate these two approaches respectively and then present our novel solution.

1) *FLB*

FLB generally adopt hashing techniques to map individual flows into finite PE space. Two classes of FLB can be used: Static Hashing (SH) [20] and Dynamic Hashing (DH) [19].

SH maps each flow by directly hashing its 5-tuples to PE ID. This approach introduces little hardware cost, but also receives limited performance due to the non-uniformity of hashing and the heavy-tailed flow-size distribution in Internet. To reveal this point, we carry out experiments using real trace collected at the GE link connecting Tsinghua University to CERNET [21]. Traffic generated from this trace is load-balanced into 8 PEs by SH with three popular hashing functions (CRC16 [20], XOR, H3 [22]). Denote the traffic volume dispatched to PE l in a time period T by $A^l(T)$, which is assumed to stand for the amount of processing tasks. We define load-Imbalance Factor (IF) in a time period T to be the maximal derivation ratio formulized by

$$IF=\mathrm{Max}\{\ |A^l(T)-\mathrm{avg}[A^l(T)]|/\mathrm{avg}[A^l(T)]\ \} \quad (1)$$

Given a pre-defined timescale T_S, we calculate the average IF of every non-overlapped time periods with length T_S to represent overall load-imbalance degree. Average IFs under timescales from 1ms to 10s are depicted in Fig. 3. As we increase timescale exponentially, it actually does not go asymptotically to zero, but stay stable at 0.2~0.4. It indicates

[3] In this paper, flow is defined by 5-tuple. I.e., IP source address, IP destination address, source port number, destination port number and type of transport protocol.

the universal load-imbalance caused by SH.

DH, another approach, maps each flow on-line based on the load status of all the PEs, as such improves the load-balancing uniformity. However, at least two issues affect its performance and scalability: 1) It needs to maintain per-flow information. Under OC-768c line rate with full load, the simultaneous active flow number may increase beyond 1M if we set flow timeout to 2s. Currently, there is no such memory technology to provide enough capacity for the flow table and fast access speed. 2) It still suffers from non-uniform load-balancing due to the heavy-tailed flow-size distribution, when flow remapping is not employed. On the other hand when active flows are allowed to be remapped, intra-flow packet order, as the penalty, is not strictly preserved.

To sum up, FLB avoids the reordering cost, however, pay a great deal in performance due to their non-uniformities. It is further shown by our simulations in Section V, both SH and DH with no flow remapping receive at least 10% throughput degradation under fix-cycled or variable-cycled applications.

2) *PLB*

PLB load-balances each packet independently to achieve the optimal uniformity. It leaves intra-flow ordering issue handled by extra output resequencing mechanisms. The optimal PLB is derived as below.

We use vector $R=\{p_r, io_b, c_o, m_b\}$ to define per-second demands from arrival traffic on processing capacity, IO bandwidth, co-processor invoking and memory bandwidth[4]. With load-balancing algorithm F, the workload assigned to PE i is defined by $R_i=F_i(R)$. If we denote the capacity of PE i by $C_i=\{p[i], io[i], c[i], m[i]\}$, and performance metrics I_i at PE i by $\{D_i, TP_i\}$, where D_i, TP_i are the average delay and throughput at PE i. I_i is written by $I_i = \{D(R_i, C_i), TP(R_i, C_i)\}$ (2)

The optimal PLB's target is to maximize I, where

$$I = \sum_{i=1}^{k} I_i / k = \sum_{i=1}^{k} \{D[F_i(R), C_i], TP[F_i(R), C_i]\} / k \quad (3)$$

We consider PPC which shares IO/co-processor resources and decouple memory contention issue by assuming it not to be the bottleneck, (3) becomes (4) given fixed io_b and c_o.

$$I = \left\{ \sum_{i=1}^{k} D\left(F_i(p_r), p[i]\right), \sum_{i=1}^{k} TP\left(F_i(p_r), p[i]\right) \right\} / k \quad (4)$$

Since $TP(a,b)=max(b/a,1)$ and $D(a,b)$ is the convex function of a/b, the maximization of overall throughput and minimization of average delay lead to the same solution under admissible traffic input, that $F_i(p_r) = p_r p[i] / \sum_{i=1}^{k} p[i]$ (5)

It demonstrates that the optimal PLB is to dispatch packets by their workloads (packet processing times) in proportion to PE's processing capacity. If identical PEs are used in PPC, (5) becomes $F_i(p_r)=p_r/k$.

Some basic PLB algorithms such as Deficit Round Robin (DRR) [23] and Surplus Round Robin (SRR) [24] are broadly used for their $O(1)$ complexities where packet size is assumed to represent actual packet processing time. To outperform them, several prediction algorithms have been introduced. Some recent solutions adopt flow-based prediction [25, 26]. Packet processing time within flow f is explicitly forecasted by $T=\alpha_f L+\beta_f$, where L is the packet length, parameters α_f and β_f are adaptively adjusted according to actual processing time of recent packets in f. The PE which has been assigned the lowest predicted workloads will receive packet in each load-balancing operation. We call it Shortest Queue First with Flow-based prediction (SQF-F).

Nevertheless, SQF-F may fail to preserve accuracy under current networking environments in respect that: 1) As in FLB algorithms, it is hard to maintain per-flow prediction context when total active flow number increases to as large as 1M under OC-768c line rate. If prediction is carried out at flow-aggregation level, SQF-F will thoroughly fail since processing time of each flow is independent; 2) In handling applications with processing time $T=rand\times\alpha_f' L+\beta_f'$, such as packet filtering where *rand* is random variable uniformly distributed in [0,1], SQF-F losses its accuracy, since α_f is not predictable. In worst case, it actually generates processing time in random, and performs even .worse than the simplest RR scheduling.

Denote the processing time of packet i to be random variable X. When n packets are backlogged before one PE (n is large enough), the total prediction error of RR algorithm (equivalent to prediction with average) is $\sum_{i=1}^{n} X_i - nE(X)$, following normal distribution $N[0,nVar(X)]$. Contrastively in worst case of SQF-F, denote predicted processing time for packet i by random variable Y exhibiting same distribution with X, the total prediction error is calculated by $\sum_{i=1}^{n}(X_i - Y_i)$, follows normal distribution $N[0,2nVar(X)]$. It reveals that SQF-F performs even worse than RR at worst case with total prediction error variation one time larger.

3) *Class-based Prediction*

To overcome the inaccuracy and the complexity of flow-based prediction, we introduce a novel Class-based prediction scheme [26] in SQF based load-balancing, namely SQF-C. We show that this method will always perform better than RR algorithm, while need not to keep per-flow information due to the finite number of traffic classes.

In SQF-C, arrival traffic is partitioned into t classes according to application types. For a packet of length L in class i, we can model its accurate processing time by

$$T_i = \alpha_i + f_i(L) + U_i + \sum_{j=1}^{m} V_i^j g_i^j(L) \quad (6)$$

Here, α_i, $f_i(L)$ and $g_i^j(L)$ are determined off-line by investigating processing micro-codes, U_i and V_i^j are random variables determining processing time variation. For each packet in class i, SQF-C generates prediction value by

$$P_i = E(T_i) = \alpha_i + f_i(L) + E(U_i) + \sum_{j=1}^{m} E(V_i^j) g_i^j(L) \quad (7)$$

We then analyze its performance. Denote the traffic ratio of class i by q_i, and the total class number by l, when n packets are backlogged before one PE (n is large enough), the total prediction error by SQF-C counts to

$$\sum_{i=1}^{n}(X_i - Y_i) = \sum_{i=1}^{l} \sum_{j=1}^{nq_i} [T_i^j - E(T_i)] \quad (8)$$

Here T_i^j denotes the accurate processing time of jth packet in class i. Then the total prediction error follows normal distribution $N\left[0, n\sum_{i=1}^{l} q_i Var(T_i)\right]$. Since $\sum_{i=1}^{l} q_i Var(T_i) < Var(X)$ holds at any time, SQF-C always performs better than RR.

[4] The resource demands at each PE are far from these four, however, from our experience and previous literatures, they take up the most part.

C. Packet Ordering

1) *Symmetrical Round Robin*

One ordering method is to dispatch packets at input in a strict RR manner to all the PEs, and read them out at output in the same order. We name it the Symmetrical RR algorithm. In this approach, the packets processed faster will not be scheduled out until all the precedent packets probably with larger delay have been outputted. This method actually adopts delay equalization technique and increases the delay of each packet to the largest one in recent. When all the packet buffers beside some PE are occupied by packets that are blocked due to the out-of-order issue, this PE will become idle and degrade the overall system throughput.

2) *Tagging*

Another ordering method is the tagging and in-order output scheduling. At the input, each arrival packet is tagged with a sequence number. Then at the output, only the packet with in-ordered sequence number can be scheduled out. The tagging technique can be carried out in global scope or per-flow scope. Global Tagging (GT) is easy to implement, however, it introduces throughput degradation when output resequencing buffer is limited, which is the case in NP with scarce on-chip memory. Comparatively, the per-Flow Tagging (FT) approach keeps packet sequence in flow scope, thus allows all the packets in head of each flow to be scheduled out after processing, considerably alleviating the ordering constraints. Nevertheless, due to the difficulty to maintain per-flow information, FT is often carried out on merely coarse-grained flow aggregation level, whose performance greatly depends on the uniformity of hash function that maps flows to their aggregations. In worst case, FT degenerates to GT.

We carry out experiments to test the performance of Symmetrical RR and GT based on reconfigurable IXP2400 under Intel Develop Workbench. We construct 4 MEs of each IXP2400 as a PPC and realize Symmetrical RR and GT mechanisms by employing Intel's HyperTask Chaining model and Asynchronous Insert Synchronous Remove (AISR) array technology respectively. PPC's performance without ordering constraint is also plotted for comparison. Two standard applications [12] are deployed: one is IPv4 Forwarding (FW) which stands for the fixed-time processing; the other is IPv4 Forwarding plus Pattern Matching (FW+PM), which gives the case of variable-time processing. The input data rate is set to 5Gbps for FW and 2.5Gbps for FW+PM to saturate the MEs.

We depict throughputs of three ordering methods in Fig. 4. Under FW application and real packet length distribution[5] (Fig. 4(a)), both ordering mechanisms achieve nearly same throughput as that of no-ordered one, since in this case no out-of-order actually happens. Under 49 byte fixed-length packet arrivals (Fig. 4(b)), GT suffers from 27.1% throughput degradation. AISR array this time becomes system bottleneck as packet arrival rate increases a great deal. Meanwhile, Symmetrical RR retains the throughput similar to a no-ordered one, since there is still no packet out-of-order and the resulting output blocking under fixed-time processing. However, under FW+PM application with real packet length distribution (Fig. 4(c)), the throughput of Symmetrical RR degrades 64.7%, due

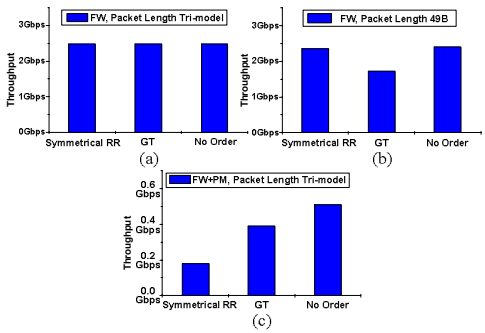

Fig. 4. PPC throughput with packet ordering mechanisms

to the highly biased packet processing time. While the throughput received by GT also decrease 23.5% because of the significant increase in resequencing buffer length.

To sum up, both the ordering mechanisms can not perfectly deal with all the traffic patterns and applications.

3) *Per-flow ordering without per-flow information*

In this paper, we develop a novel per-flow ordering method without maintaining large numbers of per-flow information. Our solution is based on the fact that the number of buffer blocks (each block holds a packet) inside current NP is limited. Therefore, we only store the flow information of packets currently buffered in system, since packets waited in the queue before the load-balancing point or already outputted will not cause out-of-order. This mechanism is implemented by a hardware dispatching and aggregating unit (DA). Compared with previous solutions using programmable PE to manipulate traffic, hardware DA is able to achieve much higher line rate while maintaining per-flow orders.

The details of DA unit are given in Fig. 5. Totally three register-based tables are maintained: *memory table*, *flow table* and *thread table*. Memory table stores status of each memory block in system and flow information of the corresponding packet in the block. By recording a *next block* field for each packet, it builds a link list for each active flow in system as packet arrival sequence. Flow table stores the information of each active flow in DA, including the *head*/*tail* of its link list, and *flowID* defined by 5-tuples. Thread table keeps busy/idle status of each PE thread in NP and the index of associated memory block.

The packet ordering mechanism works as below. When a packet arrives at ingress, we first search its 5-tuples in flow table to judge whether it belongs to a new flow or an active flow. In former case, a new flow entry is inserted to the flow table and this arrival packet's entry in memory table is set accordingly: *head* field is written to 1 to indicate it is the first packet of a flow. In latter case, the entry representing this active flow in flow table is updated: this packet's index in memory table is stored in *tail* field of flow table, and the old tail packet's *next block* field in memory table is updated to restore the active flow link list. On the other hand at the egress direction, only the packets with *head* field in memory table being 1 can be scheduled out. In this manner, per-flow packet order is preserved. After scheduling each packet out, the three tables are updated to the correct value.

[5] We use a Tri-model (64 bytes, 576 bytes, 1500 bytes, 50%, 30%) to mimic current packet length distribution in Internet. I.e., 64 byte packet accounts for 50% packets, 576 byte 30%, the others are of 1500 byte length.

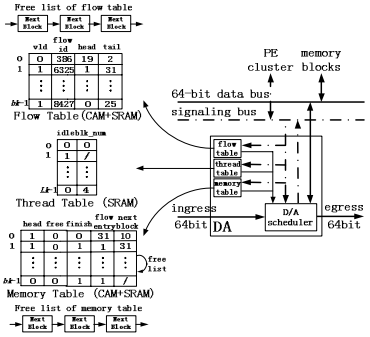

Fig. 5. Block diagram of DA

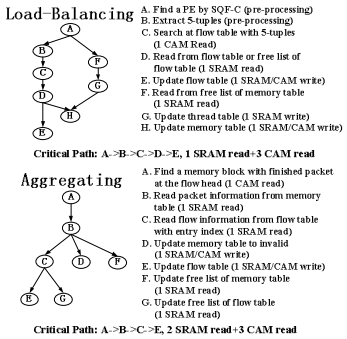

Fig. 6. Evaluation of DA's timing complexity

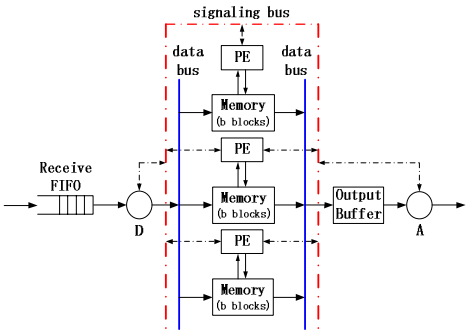

Fig. 7. Distributed memory hierarchy

We further quantitate the complexity of our approach. The mechanisms in each load-balancing and aggregating operation are described in Fig. 6. For each packet, in maximal 2 SRAM reads and 3 CAM reads are necessary. (1 CAM write cycle approximately equals to 2 CAM read cycles). In a 16-PE PPC with thread size of 16 and total memory size of 256 blocks, which is enough to guarantee system performance according to our results in Section IV, the sizes of three tables are about 4.1kB (flow table), 1.28kB (memory table) and 0.51KB (thread table). Running frequency estimation tool CACTI 4.2 [27], we obtain that the access time of on-chip CAM and SRAM with such sizes reach as small as 1.05ns and 0.41ns under 65nm technology. Consequently, the maximal line rate supported by DA unit reaches 251.9M packets per second, corresponding to 80Gbps even under 40 byte packet arrivals. This fully supports OC-768c line speed processing.

D. Memory Contention

Traditional memory hierarchies in NP mostly adopt shared-memory approach, where bandwidth request up to $R(2+\rho_p)$ stresses on the single memory. To resolve this bottleneck, we propose a distributed memory hierarchy for PPC. It reduces the bandwidth demands on each memory down to $2R$, which is in theory the minimum value.

Figure 7 depicts our distributed memory solution. We implement local memory to each PE. Different from data caches in shared-memory approach, the entire packet is buffered in the local memory, not only the handles. In this way, the processing on the packet payload can be directly carried out between PE and its local memory, while shared-memory approach needs to access the bottlenecked memory in each payload processing. Moreover, the hardware complexity of our approach is acceptable. The local memory buffering up to b packets is sufficient, where b is the number of threads in each PE. As such, a 16-PE PPC with each PE containing 16 threads only requires a total local memory size of 3Mb. It is feasible under current on-chip SRAM technology.

E. Our Unified Solution

Summarily, in this section we have proposed novel solutions for the load-balancing, intra-flow packet ordering and memory bottleneck issues. Our approach for each of them outperforms previous ones from the trade-off between performance and cost. Importantly, these solutions are decoupled with each other, thus allows us to integrate them into a unified solution to finally solve the combined problem stated in Subsection III.A. In the rest of this paper, we carry out analysis and simulations to study the PPC performance using our unified solution.

IV. Performance Analysis

A. Models

We depict the equivalent model of PPC in Fig. 8. Packet arrivals are first buffered in Receive FIFO (RF) waiting to be dispatched. After that by SQF-C, they are associated with one thread at a selected PE and stored in its local memory. Each PE processes packets in a sequential order, i.e., it only switches thread when one packet processing task is completed, so as to keep PE busy while fetching the next packet. Hence, each local memory is modeled by a Thread FIFO (TF) with b memory blocks. (Each block holds one packet.) The newly dispatched packet is placed at the tail of corresponding TF. After processing, packets are sent out to Output Buffer (OB) for multiplexing, which has totally l memory blocks. Here, only head packet of an active flow can be scheduled out, therefore guarantees intra-flow packet orders.

Packet arrivals at ingress are assumed to be of fixed-length P_L and represent workload W_L measured by PE cycles which follows exponential distribution with average $1/\mu$. (The cases of non-exponential processing time and variable-length packet arrival are examined in simulation section.) We denote the time dispatching one packet before TF by $1/\lambda$, aggregating one packet after OB by $1/\beta$, and let $k\mu<\lambda\approx\beta$ to make sure both dispatching and aggregating unit are not system bottlenecks.

B. Throughput Guarantee

To investigate the maximal throughput of PPC, we assume RF is always backlogged. Then applying SQF-C algorithm at the dispatch unit, TF before each PE is always saturated. PPC throughput will be maximized if only there is at least one unprocessed packet at each TF. This is the case when OB's length does not increase to l since when TF is not blocked.

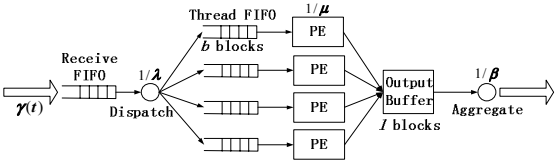

Fig. 8. Equivalent analytic model of PPU module

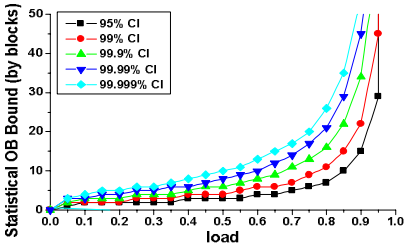

Fig. 9. Statistical backlog bound at OB

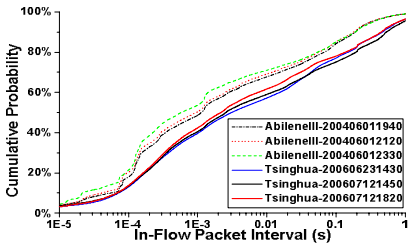

Fig. 10. Intra-flow packet interval distributions

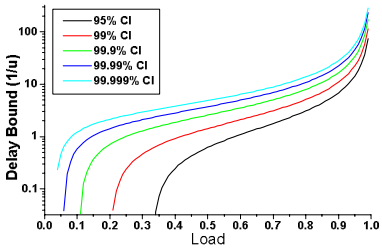

Fig. 11. Statistical delay bound at RF

There are two reasons when OB's length increases to l, causing PPC throughput to degrade: a) traffic burst at OB's input; b) OB's output blocking because of packet out-of-order issue. Below we investigate their probabilities respectively.

a) As the exponential packet processing time at each PE, packet arrivals at OB follow Poisson process with arrival rate $k\mu$ when each PE is saturated. Then OB behaves as M/D/1 queue. Solving it by *Pollaczek-Khintchine formula* [28], we derive numeric results on statistical backlog bounds at OB, as shown in Fig. 9 in different *Confidence Intervals* (CI). When OB's size is fixed to l, the minimal speedup at aggregating unit to constrain OB's length below l can be derived inversely. In an implementation with totally 16 blocks in OB, a speedup of 1.5 is sufficient to limit this probability below 10^{-5}.

b) Investigate packet P_2 blocked at OB because of out-of-order issue. There should exist at least one older packet in the same flow, denoted as P_1, which is still in PPC system. Denote the delay of P_1, P_2 at PPC by $D(P_2)$, $D(P_1)$ and their arrival interval at ingress by I_{12}. This blocking probability can be sized by

$$\mathrm{P}\{[D(P_1)\text{-}D(P_2)]\geq I_{12}\}<\mathrm{P}[D(P_1)\geq I_{12}] \tag{9}$$

We analyze some packet traces from NLANR [29] and DragonLab [21], and depict intra-flow packet interval distributions in Fig. 10. We observe

$$\mathrm{P}[I_{12}<90\mu\mathrm{s}]<0.2 \tag{10}$$

On the other hand, from Subsection IV.C, given maximal traffic burst c=0, we have

$$\mathrm{P}[D(P)>90\mu\mathrm{s}]\leq 2\times 10^{-5} \tag{11}$$

Thus, the probability that all the packets in OB encounter out-of-order is computed by

$$\begin{aligned}&\left(\mathrm{P}\{[D(P_1)-D(P_2)]\geq I_{12}\}\right)^l < \left(\mathrm{P}[D(P_1)\geq I_{12}]\right)^l\\ &\leq \left\{\mathrm{P}[D(P_2)\geq 90\mu s]+\mathrm{P}[D(P_2)<90\mu s]\mathrm{P}[I_{12}<90\mu s]\right\}^l\\ &\leq (0.00001+0.99998\times 0.2)^l \approx 0.2^l \to 0\end{aligned} \tag{12}$$

Summarize a) and b), 99.999% throughput is guaranteed for PPC, if only a small speedup of 1.5 is provided at the aggregating unit.

C. Delay Guarantee

In this part, we assume traffic arrival $\gamma(t)$ during any time length t is leaky-bucket constrained by arrival curve $\gamma(t)=Rt+c$ ($c<\infty$). The packet delay at PPC is introduced at RF, TF and OB, sequentially. At RF, packet is backlogged due to the traffic burst. If we assume the arriving traffic to be strictly admissible (c=0), we can model the delay at RF by the waiting time in a D/M/k queue, and further safely approximated by that in an M/M/k queue, with arriving rate R/P_L and departure rate $k\mu$. Then, the statistical delay bound at RF is derived by famous *Erlang C formula* [28]. The case of k=4 is shown in Fig. 11. When offered load ρ is below 0.9, delay is bounded by about $29/\mu$ in 99.999% CI. Under bursty traffic arrival ($c\neq 0$), this bound increases to $D_R=29/\mu+c/4P_L\mu$. At the case of $k>4$, the delay distribution will remain stable according to the characteristics of the M/M/k queue.

At TF with fixed size b, it is easy to size its strict delay bound by $D_T=b/U$, where $1/U$ is the maximal processing time for each packet. Lastly at OB, it is modeled by an M/D/1 queue at worst case when all PEs are saturated. In Subsection IV.B, it is proved that OB's length will not exceed 16 blocks with probability 0.99999 when DA operates at a speedup of 1.5. Thus, statistical delay bound at OB with 99.999% CI is calculated by $D_F=(1/1.5k\mu)\times 16=2.67/\mu$ when k=4, and even smaller when $k>4$.

Totally, the delay bound at PPC with 99.998% CI sizes to

$$\begin{aligned}D=D_R+D_T+D_F &= 29/\mu+c/4P_L\mu+b/U+2.67/\mu\\ &= 31.67\bar{I}/f+bI_{\max}/f+c\bar{I}/4fP_L\\ &= (31.67+b(I_{\max}/\bar{I}))(\bar{I}/f)+c\bar{I}/4fP_L\end{aligned} \tag{13}$$

Here $\bar{I}$, I_{max} are the average/maximal PE cycles executed on each packet, f is PE frequency. In a implementation with b=16, $\bar{I}=300$, I_{max}=6000, f=1.2GHz, P_L=500Byte, we size D=(87.92+15.625c)μs, where c is in unit of M bit. This is to say that if we slightly degrade the utility of PEs to 90% (allow a speedup of 1.11) and adopt a high speed aggregating unit working at speedup of 1.5, the packet delay at PPC under strictly admissible traffic pattern will be considerably small, say below 90μs in 99.998% CI. This result holds no matter how many PEs are configured in PPC.

We write service curve [30] for such PPC by

$$S(t)=1.1R[t-(31.67+bI_{\max}/\bar{I})(\bar{I}/f)]=1.1R[t-D_S] \tag{14}$$

D. Concatenations

In hybrid NP model composed of several PPCs, the analytic model becomes PPC concatenations. It is clear that if each PPC achieves 99.999% throughput, their concatenations guarantee the same throughput. To size its delay metric, we write service curve of m PPC concatenations from (14) by

$$S_m(t)=1.1R[t-mD_S]=1.1R[t-m(31.67+bI_{\max}/\bar{I})(\bar{I}/f)] \tag{15}$$

Combining it with arrival curve $\gamma(t)=Rt+c$ ($c<\infty$), we obtain the statistical delay bound of m PPS concatenations by (16), which only pays burst once.

$$D_m=m(31.67+bI_{\max}/\bar{I})(\bar{I}/f)+c\bar{I}/4fP_L \tag{16}$$

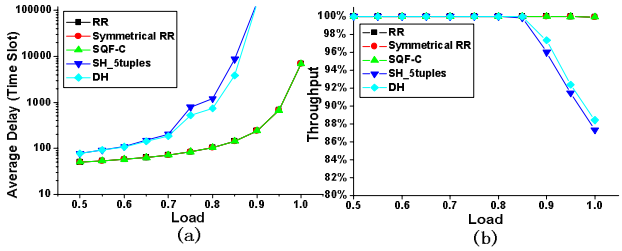

Fig. 12. Performance of LB algorithms under fix-cycled FW application

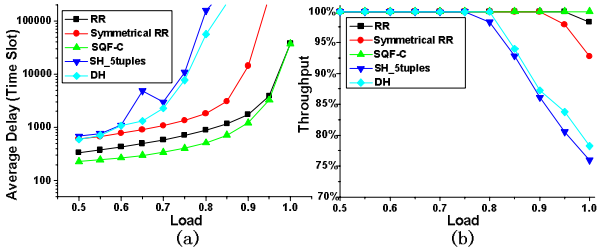

Fig. 13. Performance of LB algorithms under variable-cycled PM application

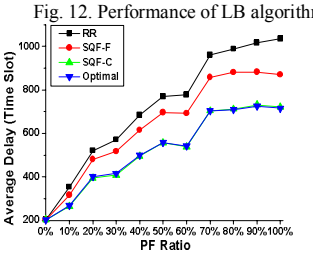

Fig. 14. Prediction algorithms

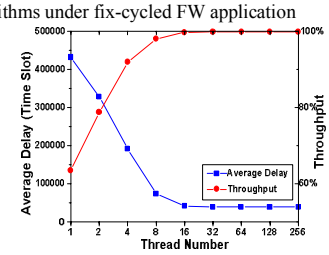

Fig. 15. Impact of thread number

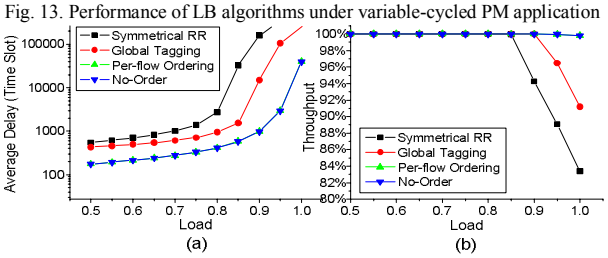

Fig. 16. Performance of ordering mechanisms

V. SIMULATIONS

A. Methodology

We launch simulations on PPC composed of a DA unit and four 16-thread PEs, which also follows the analytical model in Fig. 8. We set TF, OB and RF size to 16, 64 and 30000 blocks. RF has nearly infinite size to accommodate input traffic fluctuation and acquire accurate system throughput. The arriving traffic is generated from real Internet trace [21] at GE port by extracting the timestamp and packet length from continuous packet records. Although this trace only has an average load of 25.2%, we dilute or compress it to obtain other load rates. In each simulation set, about 1M packets are injected to PPC. Time slot is defined as one PE instruction cycle. DA unit works at a speedup of 1.2.

B. Results

1) *PLB/FLB Load-balancing Algorithms*

We first simulate performance of Load-Balancing (LB) algorithms under fix-cycled FW application. They are SQF-C, RR, Symmetrical RR, SH using 5-tuple and DH. Each FW processing lasts 300 PE cycles fixedly. In this case, SQF-C can be seen as the optimal algorithm, since it accurately predicts each packet processing time. Figure 12(a)(b) depicts the average delays and throughput of LB algorithms. Results show that PLB algorithms (RR/Symmetrical RR) perform very close to the optimal SQF-C, while FLB algorithms (SH/DH) receive throughput degradation of 11.6% and 12.6% due to the non-uniformity of hash functions.

We further plot their performance under variable-cycled PM application in Fig. 13(a)(b). Each PM task occupies $4L$ PE cycles, where L is packet length. This time, SQF-C still stands for the optimal algorithm due to the accuracy of its prediction. Results show that RR also receives comparable performance with the optimal SQF-C. Notice that, Symmetrical RR this time can not catch up with the optimal performance, it falls to a throughput of 91.7% due to the cost in handling packet out-of-order. At the same time, FLB algorithms experience even larger delay than in fixed-cycled case and is diminished to a throughput of 75.9%~78.3%.

2) *Processing Time Prediction Algorithms*

We compare SQF algorithms using flow-based and class-based predictions, and the optimal algorithm which is assumed to predict accurately for each packet. RR algorithm is also plotted for comparison. In this simulation set, we apply a mixed application of FW and Packet Filtering (PF) in PPC. PF does similar job with PM, but will finish if only a pattern is found. Thus, its processing time is modeled by $rand \times 4L$, where *rand* is uniformly distributed in [0,1]. Load rate is fixed to 0.9 (similar results are received at load rate of 1.0), and the ratio of packets executing PF increases from 0% to 100%.

Average delays in different algorithms are depicted in Fig. 14, where both prediction algorithms perform better than RR. SQF-C outperforms SQF-F with average delay only 1.0% larger than the optimal one, while SQF-F is 21.5% larger.

3) *Thread Number's Impacts*

We test PPC performance when PE thread number (TF size) increases. The mixed application of FW and PF in set 2 is applied with 20% FW and 80% PF. Load rate is fixed to 1.0 and PPC adopts SQF-C algorithm. Performance metrics given in Fig. 15 demonstrate that as thread number increases beyond 16, the system performance remains stable at the best one.

4) *Ordering Mechanisms*

In this set, we deploy Symmetrical RR, GT and our per-flow ordering mechanisms on PPC. Still, SQF-C scheduling is used. Under mixed applications of 20% FW and 80% PF, we find in Fig. 16 that both ordering methods (SRR, GT) deteriorate PPC performance, with throughput degradation of 16.6% and 8.9% respectively, while our approach receives more than 99.98% throughput of a no-ordered one. This result fits our analysis in Section IV well.

To sum up, in all simulation sets, our unified solution always achieves nearly optimal performance and considerably outperforms all the other simulated approaches. Even under highly biased applications, as in Fig. 16, it is able to retain at nearly 100% throughput and provides an average delay of less

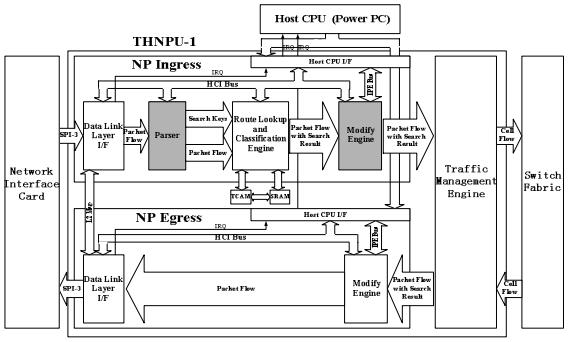

Fig. 17. Block diagram of THNPU-1

than 1000 time slots when offered load is below 0.9. Given 1.2GHz PE, the actual average delay at such PPC is below 1μs.

VI. Implementation

We implement the PPC design in THNPU-1, a FPGA-based NP prototype from Tsinghua University. Fig. 17 illustrates its structure. The parts in grey are two PPC stages. Each contains 4 PEs. The first stage is a packet parser that extracts essential fields from IPv4/IPv6/MPLS headers and as well carries out limited inspection on payload. The second stage is the packet modifier, which handles most IP/MPLS forwarding protocols.

Currently, implementing state-of-the-art TCAM through standard LA-1 interface and advanced DRAM technology such as QDR and RL-DDR, THNPU-1 with each PE working at a frequency of 110MHz, fully supports duplex OC-48c or four duplex GE L2-4 (as well as partly L7) line rate packet processing. Further evolvement to an ASIC chip is expected to meet the requirements of an OC-192 port.

VII. Conclusion

Motivated by the idea that a PPC structure should be introduced in the next-generation high-end NP, we discussed the major bottleneck issues that prevent PPC from reaching maximal performance. Through comprehensive investigations on current approaches, we propose a unified solution that load-balances traffic by SQF scheduling with class-based prediction, orders packets only in per-flow scope, and adopts a distributed memory hierarchy. Both performance analysis and simulation results demonstrate that our solution outperforms previous approaches with nearly 100% throughput guarantee and relatively small average delay. Complexity assessment indicates that PPC solution is scalable to support up to OC-768c line rate.

References

[1] C. Liu, L. Shi, et al., "Utility-based Bandwidth Allocation for Triple-play Services," in *Proc. European Conference on Universal Multiservice Networks*, 2007.
[2] Y. Chawathe, S. Ratnasamy, et al., "Making Gnutella-like P2P Systems Scalable," in *Proc. ACM SIGCOMM*, 2003.
[3] Cisco Systems, Inc., Silicon Packet Processor, http://www.cisco.com/.
[4] Bay Microsystems, Chesapeake, http://www.baymicrosystems.com/.
[5] Xelerated, Inc., X10q, http://www.xelerated.com.
[6] EZchip Technologies Ltd., NP-1, http://www.ezchip.com/.
[7] Intel Corp., Intel IXP2800 Network Processor, http://developer.intel.com /design/network/products/npfamily/ixp2800.htm.
[8] L. Thiele, S. Chakraborty, et al., "Design Space Exploration of Network Processor Architectures," in *Proc. 1st Network Processor Workshop (NP-1) in Conjunction with HPCA-8*, 2002.
[9] M. Gries, C. Kulkarni, et al., "Exploring Trade-offs in Performance and Programmability of Processing Element Topologies for Network Processors," in *Proc. 2nd Network Processor Workshop (NP-2) in Conjunction with HPCA-9*, 2003.
[10] H. Liu, "A Trace Driven Study of Packet Level Parallelism," in *Proc. IEEE ICC*, 2002.
[11] N. Weng and T. Wolf, "Pipelining vs. Multiprocessors - Choosing the Right Network Processor System Topology," in *Proc. ANCHOR 2004 in Conjunction with ISCA 2004*, 2004.
[12] Intel IXP2XXX Product Line Architecture Tool, User Guide, Rev. 002, July 2005.
[13] B. Xu, X. Zhou, et al., "Recursive Shift Indexing: A Fast Multi-Pattern String Matching Algorithm," in *Proc. 4th International Conference on Applied Cryptography and Network Security*, 2006.
[14] Y. Luo, J. Yu, et al., "Low Power Network Processor Design Using Clock Gating," in *Proc. IEEE/ACM DAC*, 2005.
[15] V. Paxson, "End-to-end Internet packet dynamics," *IEEE/ACM Trans. Networking*, vol. 7, no. 3, pp. 277-292, June 1999.
[16] J. Cuo, J. Yao, et al., "An Efficient Packet Scheduling Algorithm in Network Processors," in *Proc. IEEE INFOCOM*, 2005.
[17] L. Kencl and J.-Y. L. Boudec, "Adaptive Load Sharing for Network Processors," in *Proc. IEEE INFOCOM*, 2002.
[18] W. Shi, M. H. MacGregor, et al., "Load Balancing for Parallel Forwarding," *IEEE/ACM Trans. Networking*, vol. 13, no. 4, pp. 790-801, Aug. 2005.
[19] G. Dittmann and A. Herkersdorf, "Network Processor Load Balancing for High-Speed Links," in *Proc. SPECTS*, 2002.
[20] Z. Cao, Z. Wang, et al., "Performance of Hashing-Based Schemes for Internet Load Balancing," in *Proc. IEEE INFOCOM*, 2000.
[21] Tsinghua Egress Traces from DragonLab, http://dragonlab.org/.
[22] L. Carter and M. Wegman, "Universal Classes of Hashing Functions," *J. Computer and System Sciences*, vol. 18, no. 2, pp. 143-154, 1979.
[23] M. Shreedhar and G. Varghese, "Efficient Fair Queuing Using Deficit Round-Robin," *IEEE/ACM Trans. Networking*, vol. 4, no. 3, pp. 375-385, 1996.
[24] H. Adiseshu, G. Parulkar, et al., "A Reliable and Scalable Striping Protocol," in *Proc. ACM SIGCOMM*, 1996.
[25] F. Sabrina and S. Jha, "Scheduling Resources in Programmable and Active Networks based on Adaptive Estimations," in *Proc. IEEE LCN*, 2003.
[26] T. Wolf, P. Pappu, et al., "Predictive Scheduling of Network Processors," *Computer Networks*, vol. 41, no. 5, pp. 601-621, April 2003.
[27] CACTI 4.2, http://quid.hpl.hp.com:9081/cacti/.
[28] R. W. Wolff, *Stochastic Modeling and the Theory of Queues*. Prentice Hall, Oct. 1989.
[29] Abilene-III Traces from NLANR, http://pma.nlanr.net/Special/ipls3.html.
[30] J.-Y. L. Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, New York, 2001.