Revisiting the Cache Effect on Multicore Multithreaded Network Processors

Zhen Liu[1], Jia Yu[2], Xiaojun Wang[1], Bin Liu[3], and Laxmi Bhuyan[2]

[1]Dublin City University {liuzhen, wangx}@eeng.dcu.ie

[2]University of California, Riverside {jiayu, bhuyan}@cs.ucr.edu

[3]Tsinghua University liub@tsinghua.edu.cn

Abstract

Caching mechanism has achieved great success in general purpose processor; however, its deployment in Network Processor (NP) raises questions over its effectiveness under the new context. In this study, we thoroughly evaluate the performance of caches in NP with architectural features like multicore, multithread, and integrated packet interface. Our major findings include: (1) In general, a sufficiently large cache effectively reduces the number of memory requests and improves the utilization of the NP computation power. (2) The lower efficiency of private caches caused by duplicate information deteriorates the NP performance under certain circumstances. (3) The appropriate cache block size is constrained by the low spatial locality of network applications. (4) For workloads involving large amount of data movement, increasing cache size cannot bring more benefits when the bottleneck in interconnection bus is reached. In short, caching mechanism in NP can be helpful under appropriate usage.

1. Introduction

Network processor (NP) has emerged as a successful platform to meet the simultaneous demands of high packet forwarding performance and great programming flexibility for network equipments. This is achieved by the employment of architectural features, such as multicore, multithread, optimized instruction set, and integrated network interface, that are adapted to the characteristics of packet processing.

Compared with general purpose processor (GPP), the optimization of memory access latency in NP has a higher priority than bandwidth. Integrated memory controllers that support devices with shorter access latency are often integrated. However, long access latency still exists due to the contention caused by multiple working threads and co-processors reference. In Intel IXP2800, reading 16-byte of data from off-chip DRAM takes up to 295 cycles at a 1.4 GHz working frequency [1]. This long latency will severely hurt the NP performance if it cannot be effectively reduced or hidden.

Instead of caching mechanism, traditional NP often employs multithreading as the latency hiding technique. The main reasons include: (1) People often believe that there is no sufficient locality in network applications. (2) Caches reduce the average access delay at the expense of deteriorated worst-case performance. The unpredictable nature of caching cannot meet the stringent timing requirement of network processing [2].

However, compared with multithreading, caches demand much less memory bandwidth and reduce the pressure on interconnection bus. Moreover, cache is transparent to programmer which reduces the software complexity. Therefore, a growing number of NPs begin to incorporate caching into their design. For example, in Cisco Toaster2, a prefetchable written-delayed small cache is equipped for each of the 16 processing elements (PEs). And AMCC nP3700 has a nonblocking cache shared among all of the PEs [3].

Note that the memory access patterns of network applications are quite different from common programs. Even if caching has received increasing attentions in NP design, systematic analysis of its effectiveness under the context of packet processing is still lacking. Previous works either focus on specific applications or based on the architecture of GPP. This leads to the need of revisiting its effectiveness in a multicore multithread NP for a larger range of network applications. In this research, we strive to give answers to the following two questions:

- How does caching mechanism interact with other architectural features such as multicore and multithreading? Is caching beneficial for NP?
- How to tune the parameters of cache to achieve best performance for NP?

In this paper, we carry out a detailed performance analysis of caching mechanism in NPs. The simulation is based on Intel IXP1200, which includes all the prevailing architectural features such as optimized

978-0-7695-3277-6/08 $25.00 © 2008 IEEE
DOI 10.1109/DSD.2008.41

instruction set, multiple PEs, multithread, multiple types of memory devices and enhanced packet interface. The workloads are carefully selected to represent different types of applications, ranging from basic router functions, network security, to sophisticated switching. They also cover computation-intensive and memory-intensive applications. Real-life packet traces are employed as input to acquire more accurate locality information in network traffic of different link speed.

2. Related works

Previous study typically focuses on basic IP forwarding functions such as route lookup and packet classification, which are only a small subset of programs to be executed on network processors. Tzi-cker Chiuch, et al. proposed several strategies to reduce miss rate in a specially designed route cache [4]. Kaushik Rajan, et al. suggested an effective scheme for caching of Level Compressed trie (LC-trie) nodes to speedup route lookup [5]. Jun Xu, et al. discussed the hardware design and policy selection for the caching of packet classification results [6].

Jayaram Mudigonda, et al. extends their work to a wider range of network applications and their work is most related with ours [7]. But their experiments were based on SimpleScalar tools which simulate GPP [8]. Although they modify the source code to support more than one thread, the contentions between memory requests from several PEs, and the interaction among PEs, different types of memory devices, and packet interface can not be simulated. Moreover, their major evaluation metric is hit ratio of caching, which is not equal to the performance of network processor.

3. Experiment methodology

3.1. Simulator and parameters

We use NePSim 1.0 as the base simulator, which has been validated to Intel IXP1200 network processor [9]. We extended the simulator to support cache mechanism. Coherence among private caches is maintained by software. In IXP1200, sending and receiving packets are implemented by software. These operations are nontrivial and consume large amount of instructions. In order to focus on packet processing, we added an automatic packet loader into the Media Interface to relieve the effect of these routine I/O tasks on the NP performance.

The modified architecture of the simulated network processor is shown in Figure 1, in which MicroEngine (ME) is a terminology for PE used in Intel IXP series

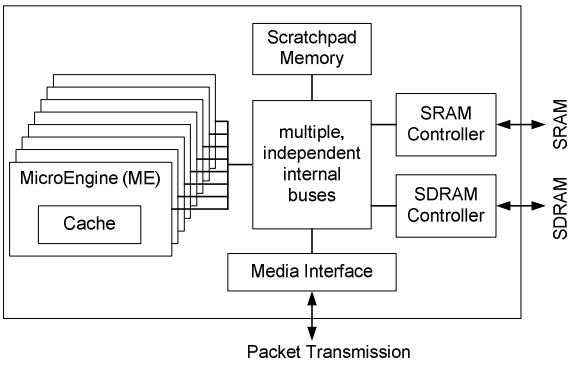

Figure 1. Architecture of the simulated NP.

Table 1. Simulator parameters.

ME frequency	696 MHz
Pipeline depth	5
Data cache	direct-mapped, 1-cycle latency, write through, no refilling
SRAM bus bandwidth	116 MHz × 32 bit
SDRAM bus bandwidth	116 MHz × 64 bit
SDRAM bus DDR?	No
SDRAM banks	4
SDRAM access latency	8.62 ns row access
	8.62 ns column access
	17.24 ns precharge
Media Interface bandwidth	80 MHz × 64 bit

products. Table 1 presents the major parameters. If not specifically mentioned, the cache block size is 8-byte and the thread number per ME is four. In our experiments, the improved hit rate caused by cache set associative only slightly affects the NP performance. Therefore, only direct-mapped cache is discussed here.

3.2. Workloads

The workloads used in our study are ported from the benchmarks provided by NePSim 1.0. They are four representative network applications:

1) Internet Protocol Forwarding (ipfwdr). It validates IP header and performs a longest prefix match (LPM) algorithm using trie lookup. Four bits are taken from the destination IP address at a time.

2) Network Address Translation (nat). It uses the source IP address to compute an index, which serves in a hash table lookup to retrieve the replacement address and port.

3) Message Digest algorithm (md4). It takes the packet as input and produces as output a 128-bit "fingerprint". This program is intended for digital signature applications such as Secure Sockets Layer (SSL) or firewall.

4) Uniform Resource Locator routing (url). It performs a string-matching algorithm on the packet payload against a number of string patterns stored in

SRAM. This program is the base of content-switching applications.

Figure 2 illustrates the instruction mix profile and raw code size of the selected programs. Table 2 gives the percentage of instructions executed from each of the major instruction categories. We can see that these programs differ in size and instruction distribution. ALU operations are the most frequently used (54.86% on average) for all of the four applications. But their memory operations exhibit significant difference. In ipfwdr and nat, only several fields in packet header are needed while in md4 and url, the whole packet should be loaded from SDRAM. For ipfwdr and nat, the major memory operations are table searches, which are not needed in md4. In url, the pattern table and auxiliary data structure are stored in SRAM.

3.3. Packet traces

We utilize four packet traces collected by National Laboratory for Applied Network Research (NLANR) from Internet exchange points [10]. BWY is located at the edge of the network and has an ATM OC3c link connected to its Internet Service Provider (ISP). UFL monitors all the information entering and leaving the campus of University of Florida. It employs a Packet Over SONET (POS) OC12c link. Abilene-I and Abilene-III (short for a-i and a-iii in Table 3) are OC48c and OC192c backbone traces.

The source and destination IP addresses in the packet traces published by NLANR are renumbered to maintain anonymity. This process retains flow pattern; but the renumbered IP addresses cannot be found in either the route table or nat table. To solve this problem, for each unique IP address, we randomly generate a new IP address according to the prefixes in route table or nat table. Another problem with the packet traces do not contain packet payload that are needed for md4 and url. Hence, we add random content to the packet and extend them to the length specified in packet header.

In this network processor simulator, thread can only retrieve packets received by a particular port in Media Interface. Thus, we have to manually allocate these packets to different port. In our experiment, we define flow as a set of packets that have the same source and destination IP addresses and packets belonging to the same flow are allocated to the same port. In this case, the cache hit rate of each ME can be improved and the packet reordering problem is also naturally solved.

4. Performance analysis

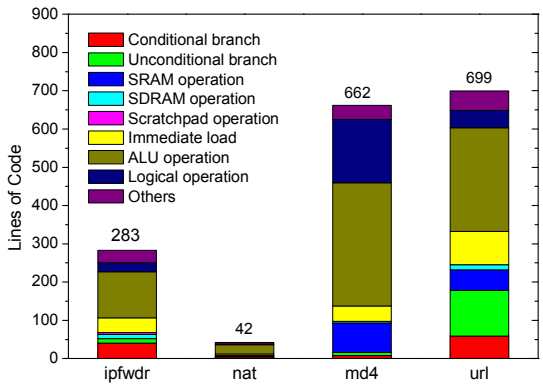

Figure 2. Raw code sizes and instruction mix.

Table 2. Percentage of executed instructions.

Prog.	ALU	Branch	Immed. Load	Memory	Others
ipfwdr	47.2	12.7	11.9	5.00	23.23
nat	56.3	29.7	1.56	7.81	4.69
md4	60.7	1.24	3.29	12.8	21.97
url	55.3	15.6	4.03	17.5	6.58

4.1. Effectiveness of caching for single ME

In this subsection, we report and analyze the NP performance of only one ME with different sizes of data caches. The data structures used in the four workloads can be divided into the three types: (1) Packet data and auxiliary data structures such as packet descriptors. They are private for each packet and have no temporal localities among different packets. (2) Application data such as tables and constants. They are shared by all packets and typically cannot be modified during packet processing. (3) Global variables such as the number of exception packets. This is another type of shared data that can be changed by packet processing.

Table 3 lists the cache miss rate of the four workloads with different packet traces. In ipfwdr and nat, packet data are directly loaded into the register file of ME from the buffer of Media Interface, without being cached. Only the route table or nat table are stored in cache. It can be seen that the cache miss rate decreases sharply with the cache size. Since nat table is much smaller than the route table, the cache miss rate of nat is also much lower than ipfwdr.

In md4, except for some constants, no application data are needed in the fingerprint calculation. The data locality comes mostly from the accesses of packet itself and is much lower than other applications. Therefore, its cache miss rate is not relevant with cache size and the speed of packet traces. As can be demonstrated in the columns for md4 in Table 3, for all traces, there is almost no change in the miss rate with the cache size.

Table 3. Cache miss rate for single ME (%).

Size (KB)	ipfwdr				nat				md4				url			
	bwy	ufl	a-i	a-iii	bwy	ufl	a-i	a-iii	bwy	ufl	a-i	a-iii	bwy	ufl	a-i	a-iii
4	19.6	31.5	25.3	29.3	8.08	21.2	17.6	23.4	12.8	12.3	12.3	12.4	29.5	29.4	29.5	29.0
8	15.4	23.5	18.8	21.9	4.30	12.4	9.12	15.7	12.8	12.3	12.3	12.4	25.8	25.8	25.9	25.4
16	2.53	7.52	4.92	6.45	0.63	5.34	4.41	8.21	12.0	12.1	12.1	12.1	19.1	19.2	19.2	18.6
32	1.51	4.60	2.94	4.19	0.61	1.03	1.20	0.28	11.8	11.9	11.9	11.9	10.4	10.2	10.3	9.85
64	1.19	2.44	1.73	2.05	0.61	1.03	1.20	0.28	11.6	11.7	11.7	11.7	9.52	9.43	9.44	9.47
128	0.69	1.48	1.36	1.42	0.73	1.18	1.42	0.56	11.7	11.7	11.7	11.7	9.54	9.43	9.45	9.46

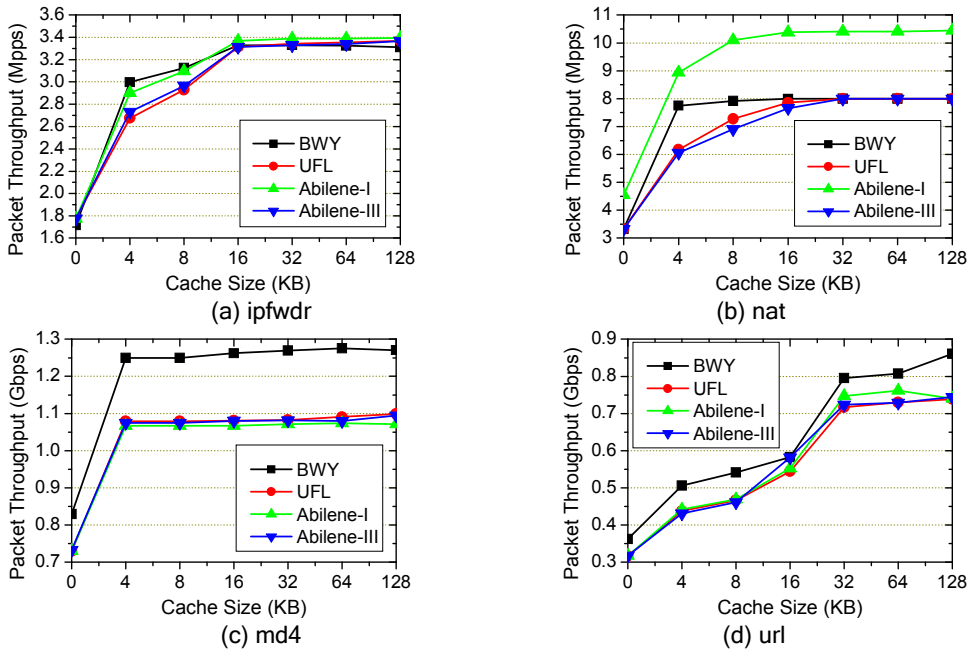

Figure 3. Packet throughput for single ME.

In url, both the pattern table and packet payload are loaded into cache. The miss rate is higher than ipfwdr and nat, but it still decreases with the increase of cache size. The cache miss rate stops falling at about 9%, which is attributed to the low temporal locality in payload accessing.

Figure 3 presents the packet throughput of the four workloads with only one ME. In ipfwdr and nat, the metric is million packets per second (Mpps) because the two programs are executed on a per-packet basis and are independent of packet size. While in md4 and url, the metric is gigabit per second (Gbps) because they are performed on the entire packet.

It can be found that, with only one ME, adding cache to NP effectively increases the traffic throughput. Even with url that achieves the smallest improvement, a 4 KB cache brings a performance gain of about 35.96%. Although memory operations only account for a small percentage of total instructions executed, their large access latency (over 50 ME cycles for SRAM and 200 for SDRAM) still makes them a dominate factor. Even with multiple threads, this long time to recover from the suspended state increases the

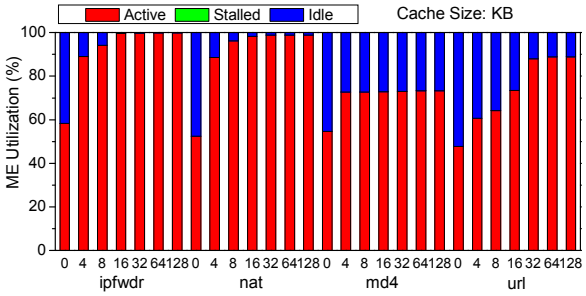

Figure 4. Internal statistics of ME for Abilene-I.

chance of turning all threads into sleep, which make the ME idle and wastes the processing power. The introduction of cache greatly reduces this possibility and improves the utilization rate of ME.

Figure 4 shows the internal statistics of ME utilization for Abilene-I. The "stalled" state happens if at least one command queue for memory operations in the NP becomes full. With only one ME, the number of memory requests is far from overwhelming a command queue. Therefore, only "active" and "idle" state can be observed in this figure.

Table 4. Average cache miss rate for Abilene-I (%)

Size (KB)	ipfwdr			nat			md4			url		
	2-ME	4-ME	8-ME	2-ME	4-ME	8-ME	2-ME	4-ME	8-ME	2-ME	4-ME	8-ME
4	33.1	38.3	51.7	23.34	24.2	28.4	12.2	11.8	11.5	37.7	41.2	44.7
8	23.2	28.7	36.3	16.68	18.1	22.8	12.2	11.8	11.5	28.6	36.8	41.0
16	16.7	22.0	28.0	11.19	12.5	14.1	12.2	11.8	11.5	25.7	28.1	36.2
32	4.10	15.8	20.4	5.96	9.24	9.79	12.2	11.8	11.5	19.1	24.6	27.7
64	2.67	3.41	14.5	0.61	4.35	8.27	11.1	11.5	11.5	10.7	18.5	23.3
128	1.45	2.08	4.46	0.91	1.28	5.98	10.8	10.4	11.2	8.12	11.7	17.8

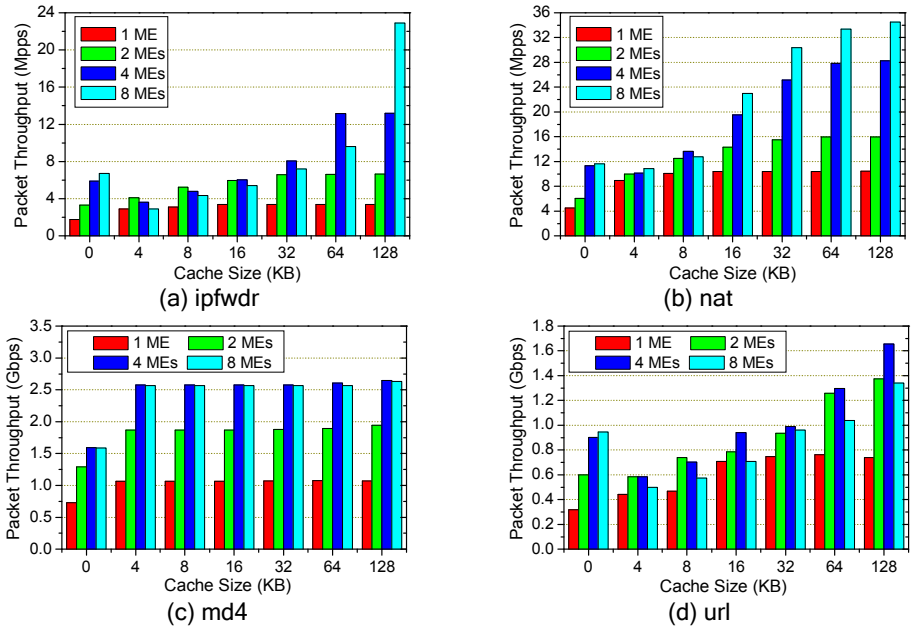

Figure 5. Packet throughput for Abilene-I with different number of MEs.

Even with four threads, a network processor without caching mechanism remains idle for over 50% of time. For ipfwdr and nat, a small cache can save a large percentage of computation power of ME. For a 16 KB cache, the ME keeps busy for 99.64% of time in ipfwdr and 98.33% of time in nat. At this point, adding more entries to cache can not bring more benefit. On one hand, the cache miss rate is hard to improve; on the other hand, almost all the available computation power of ME has been exploited.

Similar phenomenon can also be observed in md4 and url. However, the improvement of ME utilization is not as great as in ipfwdr and nat. The percentage of active time increases from 54.73% to about 73% under all sizes of cache in md4. In url, the maximum increment of ME utilization is from 47.78% to about 89% with a cache larger than 32 KB. This is because in these two applications, ME spends a lot of time on the packet moving among Media Interface, SDRAM, and SRAM. Since this is not relevant with caching, a higher cache hit rate can not turn the ME into active. Providing more bandwidth on interconnection bus is one of the solutions to this problem.

4.2. Effectiveness of caching for multiple MEs

In this subsection, we analyze the effectiveness of caching for multiple MEs. If we do not take Non-Uniform Cache Access (NUCA) into consideration, then one cache shared among all MEs has higher hit rate than private caches for each ME with the same total size. But shared cache has much larger access latency. In Intel IXP1200, the latency of the shared on-chip SRAM is comparable to external SRAM due to the contentions on interconnection bus. Hence, we only consider private caches and the sizes mentioned hereafter refer to the total size of private caches. In our workloads, chances of modifying shared information such as the number of exception packets are small. Therefore, the coherence among local caches is easy to be maintained.

Table 4 lists the average cache miss rate for Abilene-I with different number of MEs. Since all of the four traces exhibit similar patterns, the results of only one of them are presented and discussed due to space limit. It can be seen that except for md4, the cache miss rate falls sharply with the increasing ME number. The reason is that in these applications, the

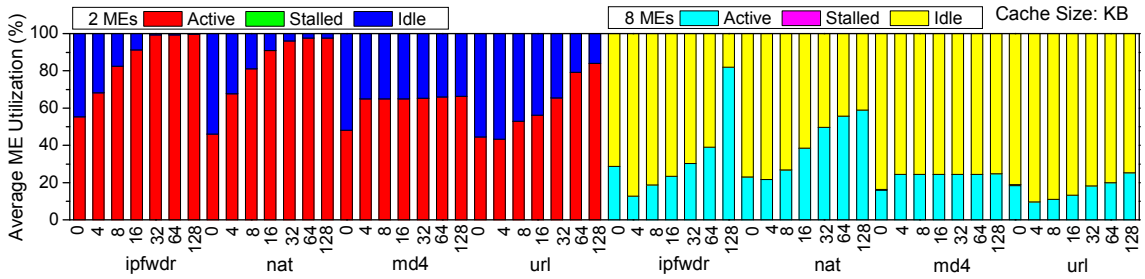

Figure 6. Average ME utilization with 2 MEs and 8 MEs for Abilene-I.

Table 5. Average cache miss rate with 64 KB caches of fifferent block sizes for Abilene-I (%)

Blk	1-ME				2-ME				4-ME				8-ME			
(B)	ipfwdf	nat	md4	url	ipfwdr	nat	md4	url	ipfwdr	nat	md4	url	ipfwdr	nat	md4	url
4	1.45	1.54	15.2	10.2	1.94	1.00	15.1	11.2	2.24	4.87	16.0	18.7	11.8	10.0	16.5	22.9
8	1.73	1.20	11.7	9.44	2.67	0.61	11.1	10.7	3.41	4.35	11.5	18.5	14.5	8.27	11.5	23.3
16	2.36	0.86	10.0	9.14	3.58	0.55	9.46	11.0	4.61	2.78	9.47	19.0	19.3	4.95	17.6	21.5
32	3.36	0.42	14.5	10.7	4.81	0.28	14.4	13.9	6.35	1.41	13.9	23.0	26.3	2.44	13.7	25.5

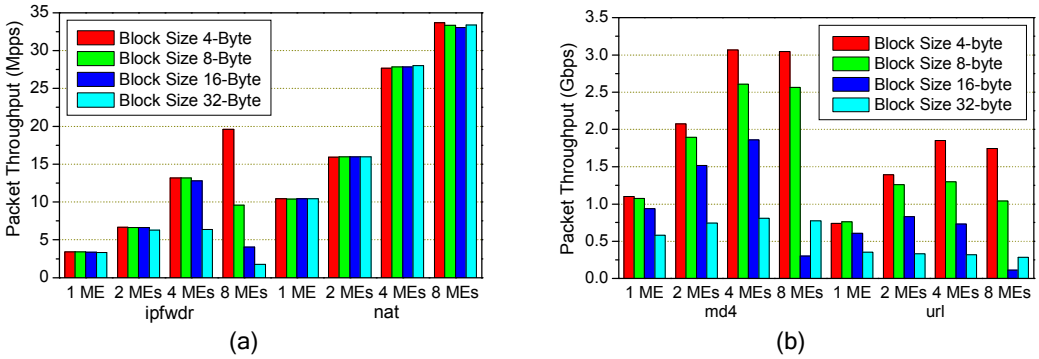

Figure 7. Packet throughput for Abilene-I with 64 KB caches of different block sizes.

major data structure is tables shared by all packets. Even if packets of the same flow are allocated to the same threads, packets from other flows may access the same entries such as the nodes of the route lookup trie. When these packets are allocated to different MEs, the duplicated table entries in local caches reduce the effective cache space. Since md4 only uses some constants and privately owned packet data, dividing a big cache into several small ones only slightly affects the miss rate.

Figure 5 presents the packet throughput for Abilene-I with different number of MEs. Not all schemes yield performance improvement over the baseline of no caching mechanism. Especially when the cache size is small, the packet throughput decreases dramatically. For example, with 8 MEs and a total cache size of 4 KB, the packet throughput drops from 6.72 Mpps to 2.90 Mpps in ipfwdr, and from 0.945 Gbps to 0.498 Gbps in url.

The deteriorated performance comes from the high cache miss rates. Note that cache block is typically larger than the size of the data actually needed. For example, each cache block can hold two route lookup trie nodes. If the additional entry loaded during a cache miss will not be used by successive table searches, it wastes the bandwidth of the already crowded memory bus that is shared among MEs.

This can be further demonstrated in Figure 6, which shows the average ME utilization for Abilene-I with 2 MEs and 8 MEs. Compared with Figure 4, the average utilization of 2 MEs only slightly declines (less than 20%). Since the computation power is doubled, the packet throughput still achieves a considerable increase. As the ME number increases to 8, a small amount of "stalled" state appears due to the contentions in command queue. Although the stalled state is hardly seen with the introduction of caches, the percentage of active state drops dramatically if the cache is not large enough. In the right part of Figure 6, the percentage of active state with 8 KB private caches is only 18.75, 26.83, 24.47, and 11.11 in the four applications. This is in accordance with the performance loss in Figure 5.

When cache size is sufficiently large, i.e. the concurrent memory requests do not exceed the bandwidth memory busses, the utilization of MEs can

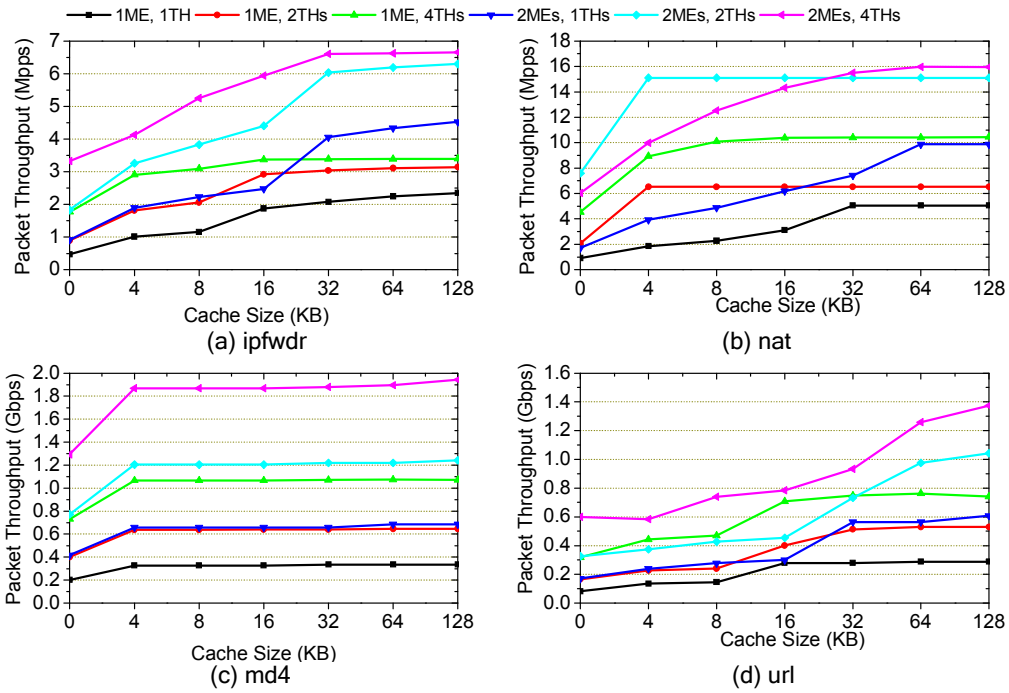

Figure 8. Packet throughput with different number of threads and MEs for Abilene-I.

be greatly improved. For example, in ipfwdr, the packet throughputs for 1 ME, 2 MEs, 4MEs, and 8 MEs with 128 KB caches exhibit an almost linear increase of 3.40 Mpps, 6.66 Mpps, 13.20 Mpps and 22.91 Mpps.

4.3. Impact of cache organizations

In this subsection, we discuss the impact of cache organizations on NP performance. Table 5 lists the average cache miss rate for Abilene-I with 64 KB caches of different block sizes. In ipfwdr, trie node is of 4-byte and it is unlikely to access successive trie nodes in one route lookup. Therefore, the cache miss rate is lowest when the block size is 4-byte. In nat, the hash table entry is organized in a unit of 64-byte and the table search is often completed within one entry. So the cache hit rate gets higher with the block size.

For md4 and url, the variation of cache miss rate is a little complicated. In this implementation, the dominate memory operations are SRAM accesses. In md4, packet payload is loaded into SRAM 16-byte at a time. Although the pattern table entry is organized as 32-byte, typically only part of the information of one entry are needed. Hence, the lowest miss rate occurs at 8-byte or 16-byte according to the ME number.

Table 5 reveals the spatial locality of the network applications, which typically exists within one table entry in these applications. Although the access pattern among different entries depends on the table organization and the implementation of searching algorithms, it can be inferred that the spatial localities of these network applications are much lower than common programs.

Figure 7 shows the packet throughput for Abilene-I with 64 KB caches of different block sizes. When the ME number is small, the sight difference in cache miss rates does not affect the packet throughput too much for ipfwdr and nat. But if the cache miss rate is large, even minor variations can greatly change the performance. In url, the average cache miss rate for 4 MEs with 64 KB local caches ranges around 20%, while the packet throughputs for the cache block sizes of 4-byte, 8-byte, 16-byte and 32-byte are 1.85 Gbps, 1.30 Gbps, 0.82 Gbps, and 0.33 Gbps. This is because big block size increases the miss penalty. When the reduced cache miss rate can not compensate the enlarged average memory access latency, the network processor performance is eventually deteriorated.

4.4. Impact of thread number

This subsection discusses the relationships between cache and thread number. Figure 8 compares the packet throughput with different number of threads and MEs for Abilene-I. Figure 9 presents the internal statistics of single ME with one and two threads for Abilene-I.

For applications that have a relatively high cache hit rate such as ipfwdr and nat, the efficiency of

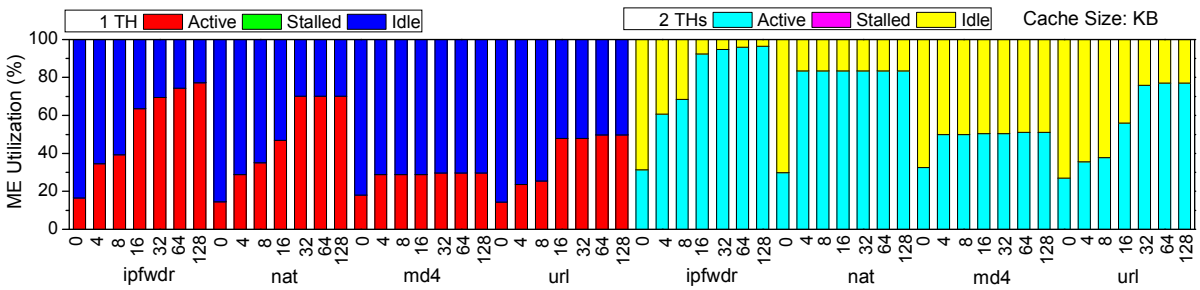

Figure 9. Internal statistics of single ME with one and two threads for Abilene-I.

multithread decreases quickly as more threads are added. In some cases, using a big cache is equivalent to using more threads or MEs. For instance, a ME with a single thread and a 16 KB cache has a comparable packet throughput with a ME with two threads and an 8 KB cache (about 2 Mpps).

Since the cache hit rate is hard to improve for md4 and url, adding more threads is much more effective for reducing the idle state percentage of MEs. In md4, adding more threads doubles the packet throughput easily. In url, the equivalent configurations can also be found and when the cache size is smaller than 32 KB, one 4-thread ME yields more packet throughput than two 2-thread MEs.

5. Conclusions

In this work, we systematically studied the effectiveness of caching mechanism in a multicore and multithreaded NP. We find that caches can release the pressure on shared memory buses and increase the PE utilization rates. But inappropriate cache organization can worsen the performance of NP due to high miss rate or large bus bandwidth consumption. Several hardware/software improvements to current NP design can help achieving higher performance.

- The data structure design should take the cache organization into consideration to fully utilize its characteristics such as the block size.
- The thread allocation policy should be designed to reduce the duplication among private caches if more than one PE is supported.
- PEs can be organized in small clusters and share the same local cache in their clusters to enhance the efficiency of caches.

Future research in this direction will take the cost of cache into consideration, including chip area and power consumption. Code optimization for data/thread allocation will also be made in order to improve cache performance.

6. Acknowledgement

This work is supported by Irish Research Council for Science, Engineering and Technology (IRCSET) Embark Initiative postdoctoral research funding for 2007~2009, and Science Foundation Ireland (SFI) China-Ireland Science and Technology Collaboration Research Fund. This work is also partly supported by National Natural Science Foundation of China (60573121, 60625201), Cultivation Fund of the Key Scientific and Technical Innovation Project, Ministry of Education, China (705003), Specialized Research Fund for the Doctoral Program of Higher Education of China (20060003058), 863 Hi-tech Research and Development Program of China (2007AA01Z216).

7. References

[1] *Intel IXP2800 Network Processor Hardware Reference Manual*, Intel Inc., May 2003.

[2] M. Venkatachalam, P. Chandra, and R. Yavatkar, "A highly flexible, distributed multiprocessor architecture for network processing", *Computer Networks*, 41(5), 2003, pp. 563-586.

[3] P. Crowly, M.A. Franklin, H.Hadimioglu, and P.Z. Onufryk (Ed.), *Network Processor Design: Issues and Practices, volume 1*, Morgan Kaufmann, 2003.

[4] T. Chiueh, and P. Pradhan, "Cache Memory Design for Internet Processors", *IEEE Micro*, 20(1), 2000, pp. 28-33.

[5] K. Rajan, and R. Govindarajan, "A Heterogeneously Segmented Cache Architecture for a Packet Forwarding Engine", *Proceedings of ICS'05*, 2005, pp. 71-80.

[6] J. Xu, M. Singhal and J. Degroat, "A Novel Cache Architecture to Support Layer-Four Packet Classification at Memory Access Speeds", *Proceedings of IEEE INFOCOM'00*, vol. 3, 2000, pp. 1445-1454.

[7] J. Mudigonda, H.M. Vin, and R. Yavatkar, "Managing Memory Access Latency in Packet Processing", *Proceedings of SIGMETRICS 2005*, 2005, pp. 396-397.

[8] http://www.simplescalar.com/.

[9] Y. Luo, J. Yang, L.N. Bhuyan, and L. Zhao, "NePSim: A Network Processor Simulator with a Power Evaluation Framework", *IEEE Micro*, 24(5), 2004, pp. 34-44.

[10] Passive Measurement and Analysis Project, National Laboratory for Applied Network Research. http://moat.nlanr.net/pma.