

Empirical Evaluation of Data Transformations for Network Infrastructure Applications

Damon Fenacci Björn Franke

Institute for Computing Systems Architecture (ICSA)
School of Informatics
University of Edinburgh
United Kingdom
damon.fenacci@ed.ac.uk, bfranke@inf.ed.ac.uk

Abstract—It is estimated that the amount of data coming out of an optical fibre is doubling every nine months and, thus, the growth rate in network bandwidth by far exceeds that of transistor density stated by Moore’s law. This causes excessive strain on network infrastructure nodes such as routers which need to operate at line rate in order to keep up with the external bandwidth requirements. Consequently, manufacturers of network processors have developed a wide range of technologies including highly parallel and specialised architectures to cope with ever increasing processing demands. Software tool support, however, lags behind and most research in compiling for network processors has focused on improved sequential and parallel *code generation*. In this paper we show that not code, but *data organisation* is the key obstacle to overcome in order to achieve high performance on network infrastructure applications. We evaluate three specialised *data transformations* (structure splitting, array regrouping, and software caching) against the industrial EEMBC networking benchmarks and *real-world* data sets. We demonstrate that speedups of up to 2.62 can be achieved, but at the same time no single solution performs equally well across all network traffic scenarios. This clearly indicates that *adaptive* data transformation schemes are necessary to ensure optimal performance under varying network loads.

Index Terms—Data transformation, Network applications.

I. INTRODUCTION

Since the original launch of the ARPANET in 1969 the Internet has developed at an unprecedented pace that can only be compared to the *microprocessor revolution* itself. Internet traffic has doubled roughly every year since 1997 [1] and there are an estimated 1.73 billion Internet users as of March 2010 [2].

Core routers form the heart of the Internet and must be able to support multiple telecommunications interfaces of the highest speed facilitated in the Internet backbone. At the same time they must be able to forward IP packets at full speed on all of them while also supporting the different routing protocols used in the core Internet. *Edge routers* placed at the edge of an *Internet Service Provider* (ISP) network connect to core routers and typically feature 10 GB uplinks (OC192) and various customer-facing interfaces. Both core and edge routers comprise specialised and highly parallel *network processors* (NPUS) for low latency and high bandwidth processing of independent data packets. Among their many tasks pattern

matching, data manipulation, queue management, and statistics gathering are the most important and timing critical.

No single architectural model for NPUS has emerged and there exists a broad variety of implementations ranging from simple RISC cores with dedicated peripherals, in pipelined and/or parallel organisation, to heterogeneous multiprocessors, based on complex multi-threaded cores with customised instruction sets. While network processor architectures have become ever more sophisticated [3] their programmability and ease of use has suffered [4], [5].

Compilation for NPUS is certainly not a new area and much of the research has focused on improved sequential code generation [6]–[8] and task partitioning and parallel thread management [9]–[13]. In this paper we take a different approach and focus on *data transformations*. We demonstrate that data accesses in network infrastructure applications form a performance bottleneck and evaluate the effectiveness of three, *non-standard* data transformations. *Structure splitting, array regrouping, and software caching* aim at reducing the number of data cache misses which according to a recent performance analysis in [14] limit the overall performance on RISC based NPUS.

We have evaluated the effectiveness of the three data transformations against the industry standard EEMBC [15] Networking benchmarks and two embedded RISC processor cores (INTEL STRONGARM and ARC 750D). These processors are representative for many proprietary RISC cores of which more complex NPUS are composed. In fact, the STRONGARM is utilised in the first generation of INTEL’s IXP network processor line and the ARC 750D can be found in many home networking devices.

Our results confirm that data transformations are key performance enablers for networking applications and speedups of up to 2.62 can be achieved through compiler-directed cache aware data restructuring. Rather than using *synthetic* and *unrealistic* data traces we use *real-world* network traffic and demonstrate that changing traffic scenarios require an *adaptive* approach, i.e. no single data layout and organisation performs equally well for every network traffic situation. This suggests that for the effective operation of NPUS *on-line* data transformations – under the control of the *control plane*

```

struct mbuf {
    struct m_hdr m_hdr;      /* header */
    ...
    union {
        struct m_ext MH_ext; /* link to external payload */
        char MH_databuf[MHLEN]; /* internal payload */
    } M_dat;
    ...
}

```

Fig. 1. Original packet structure in the *IP Reassembly* benchmark.

```

struct mbuf_header {
    struct m_hdr m_hdr;      /* header */
}

struct mbuf_payload {
    ...
    union {
        struct m_ext MH_ext; /* link to external payload */
        char MH_databuf[MHLEN]; /* internal payload */
    } M_dat;
    ...
}

```

Fig. 2. Packet structure definition after *structure splitting*.

processor and performed whenever a change in network traffic patterns is observed – may offer an attractive solution.

A. Motivating Example

Consider the structure definition in the example in figure 1. This structure is an excerpt from the EEMBC *IP Reassembly* benchmark and defines the data structure for packets that carry a payload. Other networking applications feature similar packet data types. Among other fields the `mbuf` structure comprises an `m_hdr` structure for the packet header and an `M_dat` union holding either the internal payload or a link to an external payload.

The *IP Reassembly* application accepts previously split packets, reassembles and arranges them back in the right order which might be different from the order in which the pieces have been received. For this, *IP Reassembly* only needs to access the headers of each received packet, but it does not need to touch the internal payload. Still, when executed on a system with data caches, unused parts of the remaining `mbuf` structure including the internal payload are streamed through the memory hierarchy. This is because the header information used by the application is interleaved with payload data. As valuable cache space is taken up by data not touched by the application the resulting data cache utilisation is low and a high number of data cache misses can be observed.

Structure splitting is a data transformation that decomposes a structure into its components and modifies the use sites accordingly. The result of structure splitting applied to the packet data structure from the *IP Reassembly* benchmark is shown in figure 2. The original `mbuf` structure has been split into two structures `mbuf_header` and `mbuf_payload` that contain the header and payload data, respectively. The resulting code leads to fewer data cache misses as header and payload information are not interleaved any more and payload data does not further pollute the data caches.

For the transformed *IP Reassembly* code the data cache hit rate increases from 69% to 78% for the INTEL STRONGARM platform and from 92% to 94% for the ARC 750D. This results in overall speedups of 1.74 and 1.78, respectively.

B. Contributions and Overview

Among the contributions of this paper are:

- 1) Conclusive evidence that poor data cache utilisation limits the performance of networking infrastructure applications,

- 2) the introduction of three data transformations particularly suitable for networking applications, and
- 3) an extensive evaluation of the effectiveness of the proposed data transformations against the industry standard EEMBC benchmarks, two embedded RISC platforms and *real-world* network traffic data.

The remainder of this paper is structured as follows. In section II we show that poor utilisation of data caches causes a major performance bottleneck in networking applications and introduce three specialised data transformations that aim at improving data cache efficiency. This is followed by an extensive empirical evaluation in section III. We discuss related work in section IV before we summarise and conclude in section V.

II. DATA TRANSFORMATIONS

In this section we initially motivate our work on data transformations by showing that applications from the networking domain suffer from an disproportionately higher number of data cache misses in comparison to other embedded application domains. We then introduce three, non-standard data transformations that we have selected for evaluation in section III.

A. Low Data Cache Efficiency

Before we set out to explain the data transformations evaluated in this work we motivate our approach and the specific choice of transformations. While much of the work on compiler optimisation for NPUs has focused on low-level code generation issues [6]–[8] the architecture community repeatedly revisited the problem of low data cache efficiency in NPUs, e.g. in [16]–[20]. We have validated these latter findings to be in line with other work on embedded workload characterisation such as [14], [21]. Consider the diagrams in figure 3 where the average data cache miss rates across the five application domains (automotive, consumer, networking, office, telecom) represented in the industry standard EEMBC benchmarks are shown for three popular embedded processors. For each of the three platforms (INTEL STRONGARM, ARC 750D, and FREESCALE MPC7410) networking applications show four to seven times higher data cache miss rates than for consumer or office applications. Automotive and telecom applications, on the other hand, exhibit the lowest data cache miss rates across the three platforms. This observation cannot only be explained by the lack of temporal locality in packet

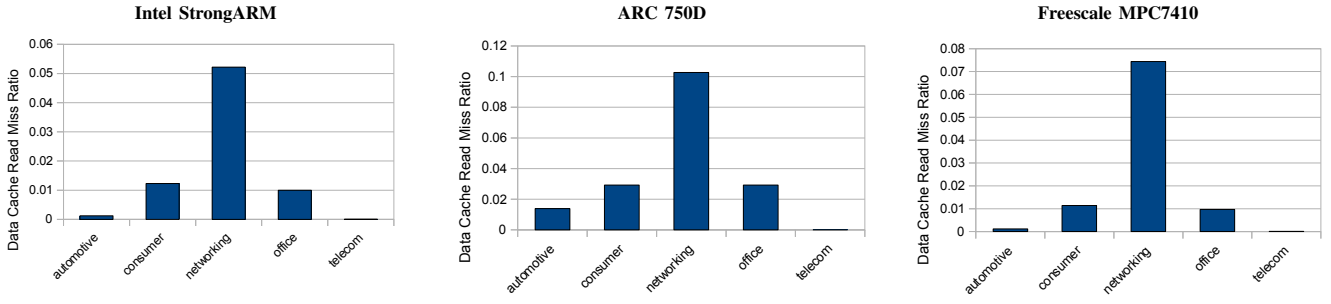


Fig. 3. Average data cache miss rates across the five application domains represented by the EEMBC benchmarks and three different platforms.

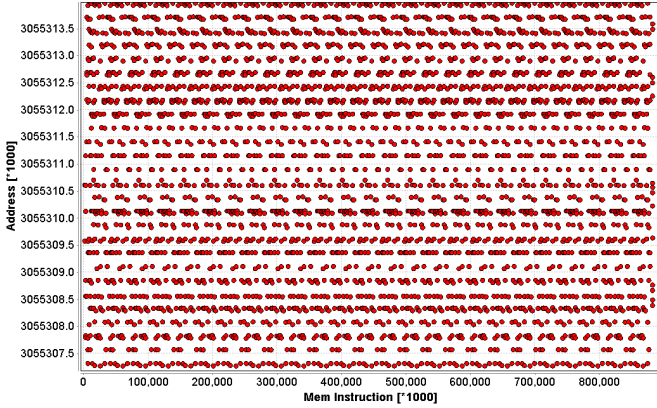


Fig. 4. Trace graph for memory reads for *IP reassembly*.

processing benchmarks, but additional spatial locality issues contribute to the poor overall performance.

B. Cache-Aware Data Transformations

We now introduce three *non-standard* data transformations considered in this paper.

1) *Structure Splitting*: We have further investigated the causes of the distinctly higher data cache miss rate for networking applications using the PIN tool [22]. A memory trace resulting from the EEMBC *IP Reassembly* benchmark, and typical for other benchmarks such as *IP Packet Flow*, *IP Packet Check*, *IP Network Address Translator*, *Quality of Service* and *TCP* is shown in figure 4.

Over time *IP Reassembly* processes a number of independent packets represented by a C structure such as the one shown in figure 1. On the y-axis of figure 4 the address range of the accessed data items is plotted. The inspection of the memory trace immediately reveals discrete “horizontal stripes” in the access patterns. The “gaps” between the stripes correspond to packet payloads that are not touched by the algorithm. Still, parts of each data cache line are “polluted” with some of this payload data and the overall available cache size is effectively reduced.

Structure splitting [23]–[25] is a transformation based on *reference affinity* and given a program and a structure type targeted for splitting, the compiler changes the allocation and the access of all objects of the split type (see figures 1 and

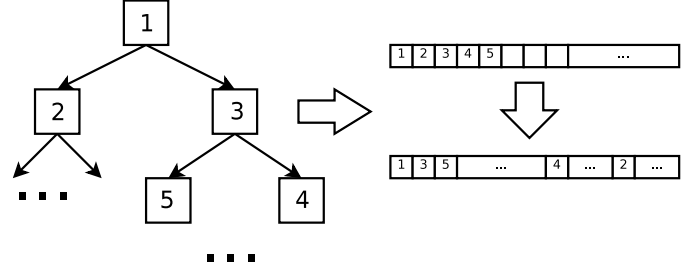


Fig. 5. Example of *array regrouping* of a tree structure.

2) *Structure splitting* de-interleaves differently accessed data fields and, thus, increases overall cache efficiency.

2) *Array regrouping*: This transformation is concerned about the flat memory organisation of dynamically created data structures such as trees or graphs.

Traditionally, tree or graph based data structures are heap allocated and consequently their actual memory layout is not only affected by the order in which nodes are inserted, but the details of the specific implementation of the `malloc` library function and the current state of the heap plays an important role, too. It is therefore little surprising that the resulting flat data organisation is not necessarily optimal with respect to its cache behaviour and, in particular, does not respect dynamic access patterns. Estimated reference affinity costs based on frequency profiling [24] can lead to a better layout. First, if the data structure is *bound* and *quasi-static* it can be embedded in a fixed-size single-dimensional array. This is the case for most lookup structures such as routing tables and an example of this transformation is shown in figure 5. Second, taking into account dynamic access frequencies collected in a profiling pass an element ordering can be determined that reduces overall access cost due to cache misses [24]. For example, if node 5 is repeatedly accessed along the path 1 – 3 – 5 the layout on the bottom half of the right side of figure 5 has better spatial locality than the default layout above.

3) *Software Caching*: This transformation exploits different access frequencies to individual leaf nodes in a tree based lookup data structure and maintains a small, but efficient auxiliary data structure serving the most frequently accessed items. The operation of the auxiliary data structure is comparable to a *cache* and may be implemented using a hash table [26].

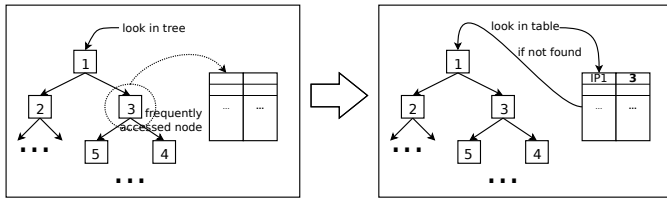


Fig. 6. Example of *software caching* using a hash table supporting a tree structure.

This principle is shown in the diagrams in figure 6. In a tree shaped lookup data structure such as a routing table node 3 may get frequently accessed. Upon determining that node 3 is *hot* it is entered in to the auxiliary hash table introduced next to the original tree structure. Subsequent accesses to the lookup tree are modified such that first the hash table is checked and a full traversal of the tree is only performed if the requested data item cannot be found in the hash table. The benefit of installing an auxiliary hash table arises from the fact that most queries can be served efficiently by the hash table and fewer accesses need to be resolved by the tree. The tree is not entirely replaced with a different data structure and can still support algorithms relying on depth-first or breadth-first order tree traversals.

There are a number of prerequisites before auxiliary lookup data structures can be introduced by the compiler. First, shape analysis [27], [28] is performed to determine the shape of the original data structure. In this work we restrict our transformation to binary trees. Second, in a profiling stage the most frequently accessed nodes and paths are determined. Third, the auxiliary hash table is created and read-only accesses to the original tree are augmented with prior lookups to the newly introduced table. Similarly, write accesses to the tree are augmented with corresponding updates to the table.

III. EMPIRICAL EVALUATION

In this section we present the results of our experimental evaluation of the three data transformations introduced in section II. Before we start, we briefly describe our the target platforms and benchmarks before we demonstrate the need for real-world network traffic data.

A. Benchmarks

EEMBC has defined a set of established benchmarks targeting different embedded application domains. In table I we provide a short description of the relevant EEMBC networking benchmarks that have been used in this work.

Each of the benchmarks comes with one or more *synthetic* data sets. While these data sets are suitable for processor performance evaluation they do not exhibit the temporal locality and variation over time that is characteristic for *real-world* network traffic. Therefore, we have used different, *publicly available* data sets representing *actual* Internet traffic and routing information in our experiments (see section III-C).

B. Target Platforms

We have evaluated the data transformations against two embedded RISC processor cores (INTEL STRONGARM and

Benchmark	Description
IP Packet Flow	IP packet store&forward
IP Packet Check	IP packet header check
IP Reassembly	IP fragmentation and reassembly
IP Network Address Transl. (NAT)	Network address translation
Route Lookup	IP route lookup
Open Shortest Path First (OSPF)	Dijkstra shortest path first alg.
Quality of Service (QoS)	Bandwidth and traffic shaping
TCP	TCP data transfer

TABLE I
EEMBC 1.1 & 2.0 NETWORKING BENCHMARKS.

Platform	StrongARM SA-1110	ARC 750D
Processor Core		
• Pipeline	single, 5-stage	single, 7-stage (interlocked)
• Execution Order	In-Order	In-Order
• FP Support	No (SW)	Yes
• Branch Prediction	No	Yes
Memory System		
• Level 1 Cache	16k(I)/8k(D)	8k(I)/8k(D)
• Level 2 Cache	-/-	-/-
• MMU	Yes	Yes
• Main Memory	Simple RAM model	DDR-RAM model
• Bus	native	native
Simulation		
Simulator	SimIt-ARM v2.1	ARC simulator
• Options	FP library	Default
• I/O	Emulated	Emulated
SW Dev. Tools		
• Compiler	GCC 3.3.2	GCC 4.2.1
• Compiler Options	-O3	-O3

TABLE II
DETAILS OF THE INTEL STRONGARM SA-1110 AND ARC 750D PLATFORMS.

ARC 750D). The details of the processors and their memory system, their simulators and the software development tool chains are shown in table II. These platforms are representative for many proprietary RISC cores of which more complex NPUs are composed. The INTEL STRONGARM is utilised in the first generation of INTEL's IXP network processor line and the ARC 750D can be found in many home networking devices.

C. Real-World Data Sets

For general processor benchmarking the *synthetic*, default data sets provided with the EEMBC benchmarks are convenient, however, they are less suitable for the evaluation of networking applications in a realistic environment. This is mainly due to two reasons: (a) the static data such as routing tables are very small and fit entirely into even small data caches, and

Source	Time	Description
CAIDA	31/03/2009 – 07:01	1 min traffic traces from the <i>equinix-chicago</i> backbone router [29]
	31/03/2009 – 12:28	IP Prefix to Autonomous System map (routing table of backbone routers)
	13/09/2009	Topology of the IPv4 autonomous systems [30]
UoE	12/01/2010 – 03.11.16:00	3 x 1 min traffic traces from the edge router of the Informatics School
	18/09/2009	Routing tables of the edge router of the Informatics School
	16/02/2010	Topology of the network of the Informatics School

TABLE III
NEW SOURCES OF INPUT DATA FOR EEMBC NETWORKING APPLICATIONS.

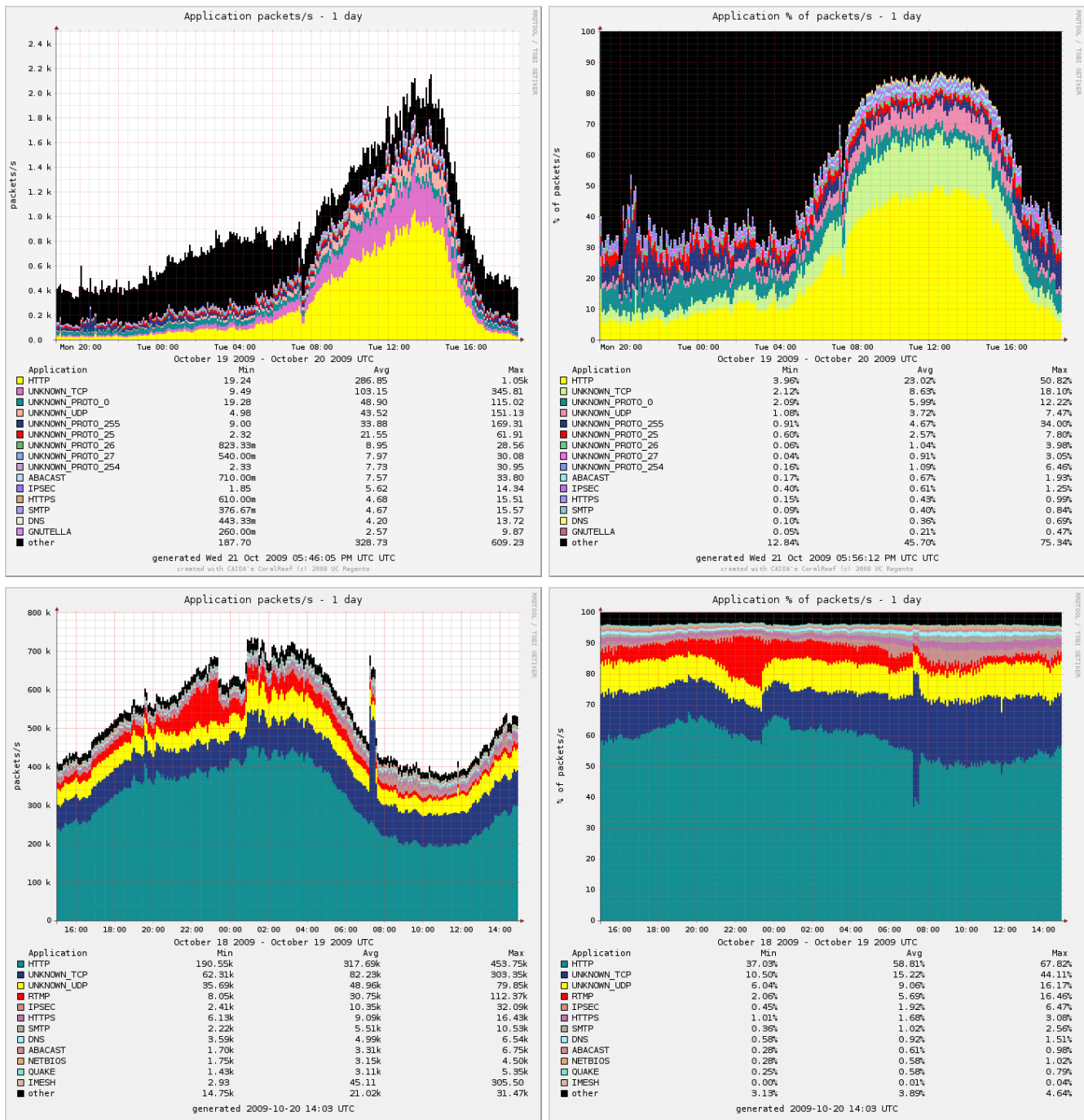
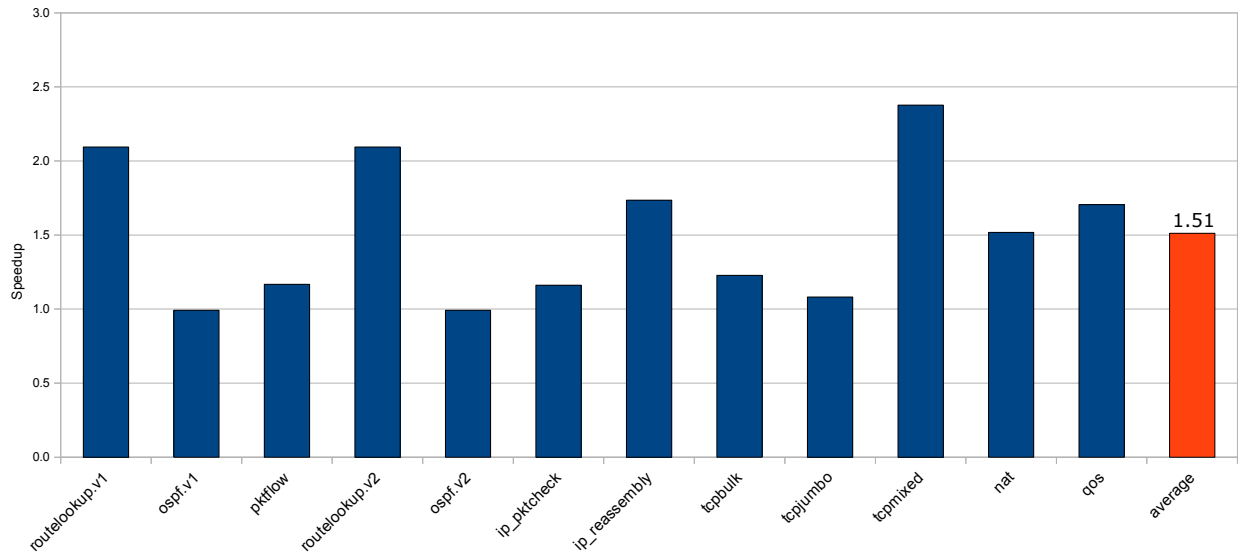


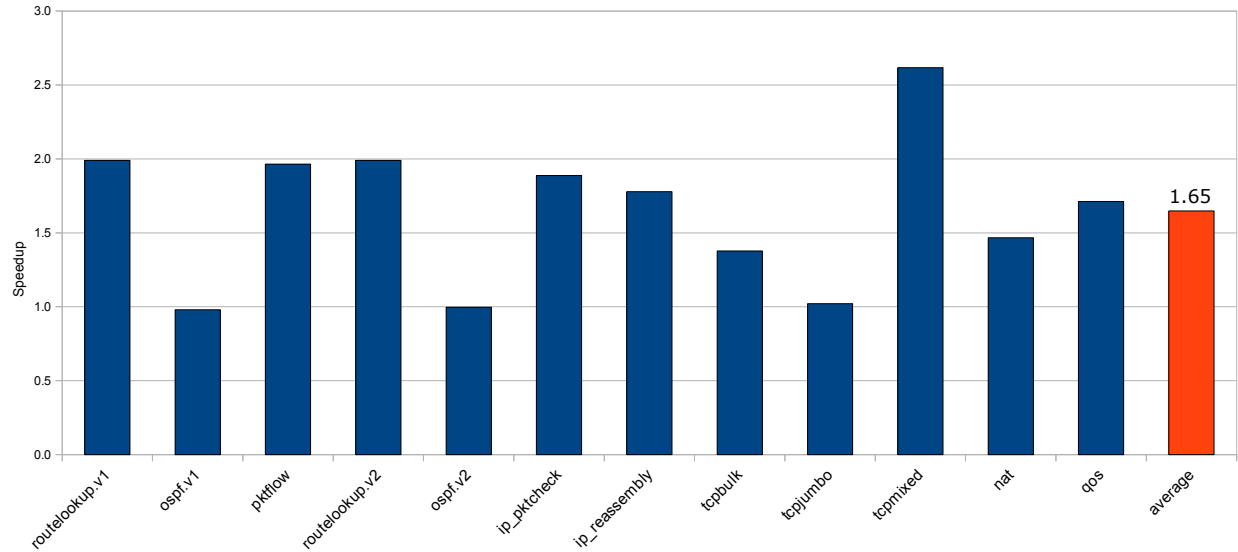
Fig. 7. Variation in traffic volume and type in CAIDA Equinix datacentres in Chicago and San Jose.

(b) the traffic data is randomly distributed and lacks variation and temporal locality present in real network traffic. For this reason we have decided to conduct our experiments using publicly available *real-world* data (see table III). Large routing tables, network traffic variations over time and recurrent traffic patterns have been captured by The Cooperative Association for Internet Data Analysis (CAIDA) [31] for two Internet backbone routers. In addition, we used routing tables and network traffic captured at three different times at one of the

fringe routers installed at the School of Informatics at the University of Edinburgh (UoE). Together, this allows us to accurately model variations in network traffic over different times of the day and for different locations on the Internet. Example traffic provided by CAIDA [31] is shown in figure 7. It is clearly visible that the amount and type of traffic varies over time and between different Internet routers. Fast transients are superimposed on underlying, slowly changing patterns representing different protocols.



(a) INTEL STRONGARM



(b) ARC 750D

Fig. 8. Overall performance results for all three data transformations and both platforms (INTEL STRONGARM and ARC 750D)

D. Results

Our overall performance results for both target platforms and all three data transformations are shown in figure 8 where each bar corresponds to the maximum speedup attained through either of the transformations. On average, data transformations contribute to average speedups of 1.51 and 1.65 for the INTEL STRONGARM and ARC 750D platforms, respectively. These results clearly demonstrate the potential of data restructuring over the most aggressive code optimisations (-O3) performed by the default software development tool chains. In fact, on the *tcpmixed* benchmarks speedups of up to 2.37 and 2.61 for the INTEL STRONGARM and ARC 750D, respectively, have been observed.

In table IV we show where transformations are applicable. *Structure splitting* is generally applicable to all packet

Benchmark	Structure Splitting	Array Regrouping	Software Caching
routelookup v1&2		✓	✓
ospf v1&2		✓	
pktflow	✓		
ip_pktcheck	✓		
ip_reassembly	✓		
tcp	✓		
nat	✓		
qos	✓		

TABLE IV
APPLICABILITY OF DATA TRANSFORMATION TO EEMBC NETWORKING BENCHMARKS.

processing benchmarks contained in the EEMBC suite. *Array regrouping* and *software caching* are only applicable to few benchmarks, however, *route lookup* and *shortest path (OSPF)* are among the most frequently performed operations in a network processor and, hence, any improvement of these operations increases overall system efficiency.

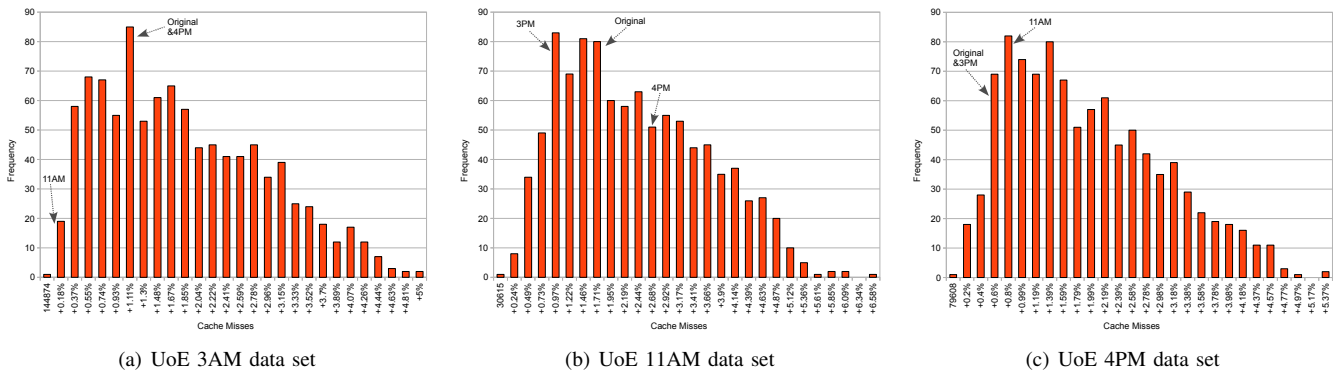


Fig. 9. Data cache miss frequencies of the EEMBC *routelookup* benchmark on the INTEL STRONGARM for the UoE datasets at three different times (3AM, 11AM and 4PM). The arrows represent the performance of the original data layout and the optimal layouts for the other two data sets (obtained from the runs with fewest cache misses in the other two frequency diagrams). The optimal data layout for one traffic scenario does not result in an optimal data layout when applied to traffic captured at the same router at a different time of the day.

We now discuss each individual data transformation in turn. *Structure splitting* results in good speedups ranging from 1.02 to 2.61 across the applications where it is applicable and both platforms. While there is a noticeable correlation of the performance improvements between the two processors some benchmarks such as *ip_pktcheck* result in greatly different performance levels. This is mainly due to the particular data cache sizes and organisations (e.g. associativity) of the two platforms where a specific configuration aligns well with the structure sizes of the benchmark.

Array regrouping is most useful for the *route lookup* benchmark where speedups of 1.36 for the INTEL STRONGARM and 1.08 for the ARC 750D have been observed. For *OSPF* array regrouping is less beneficial and, in fact, has shown a slowdown. This is due to the failure of the *reference affinity* algorithm to accurately capture the common access patterns. For this reason we have conducted a further experiment investigating the impact of the data layout on the data cache miss rate. We have simulated 1000 runs of the *route lookup* benchmark with randomly created data layouts and plotted the resulting distribution of cache miss frequencies as shown in figure 9(a). The default layout highlighted in the diagram is at the peak of the resulting distribution. The bars to the left of this peak correspond to data layouts that result in fewer cache misses whereas the bars to the right related to data layouts with an increased number of cache misses. It becomes apparent that a significant reduction of cache misses over this baseline is possible if a suitable data organisation can be found. At the same time less-than-optimal data layouts can lead to a performance degradation. This is true for each of the three data sets evaluated in figures 9(a), 9(b) and 9(c).

We have now applied the best possible layout for each of the three data sets to the remaining two. For example, if the optimal data layouts for the UoE 3AM and 4PM data sets are applied to the 11AM network traffic data as shown in figure 9(b) neither of the previously optimal layouts comes close to the optimal layout for the 11AM data set. Whereas the 3AM data layout still improves slightly on the 11AM baseline the optimal layout for the 4PM data results in an increased

number of cache misses when applied to the 11AM traffic. These results demonstrate that no single data layout is optimal across different network traffic patterns, but optimality can only be achieved with respect to a particular traffic scenario. In our future work we will address this issue and investigate dynamic data layout reorganisation to reflect changing access patterns.

The introduction of *software caching* shows a significant benefit for the *route lookup* algorithm for both architectures. However, the achievable performance improvements due to the caching in the auxiliary hash table vary across data sets. Speedups range from 1.11 for the 4PM UoE data set to 2.09 for the 11AM UoE network traffic data. This effect is due to the probabilistic nature of the processed network traffic where successful hits to the software cache depend on the specific traffic history. In our current work we have used a very small hash table comprising only four elements. With more diverse traffic patterns such as the ones in the 4PM UoE data set larger sizes of the hash table might improve overall performance, but at the same time it remains important to trade off the potential benefits with the increased cost for a miss to the software cache. In our future work we will investigate different sizes and organisations of these structures and allow for dynamic adaption.

Our results for both *array regrouping* and the introduction of *software caching* show a strong dependence on the specific data set. For networking infrastructure applications the data sets are generally not known in advance, but we can rely on temporal locality and slowly changing patterns of the network traffic (see figure 7). It is therefore conceivable to perform online profiling and to use the information for periodic data restructuring to adapt to changing network traffic scenarios.

IV. RELATED WORK

Compilation for NPUs is an active research area [6]–[12] and a comprehensive summary of the programming challenges in network processor deployment are documented in [4].

The design and implementation of C compilers for NPUs is subject of [7] and [8]. In [6] a code generation technique

targeting NPU-specific bit-packing instructions is presented.

Cache design for NPUs has been repeatedly discussed in the computer architecture community, e.g. in [16]–[18], [20]. With this work we share the view that data accesses in an NPU form a performance bottleneck, however, we propose a more flexible software solution orthogonal to existing hardware approaches.

Reference affinity in data transformations has been investigated in [23]–[25], however, these works are primarily concerned with general-purpose workloads rather than networking applications.

V. SUMMARY AND CONCLUSIONS

In this paper we have shown that data cache misses form a performance bottleneck for network infrastructure applications running on RISC based NPUs. We have evaluated three non-standard data transformations (structure splitting, array re-grouping, and software caching) that aim at improving overall cache efficiency for typical networking applications. For the EEMBC networking benchmarks, *real-world* data sets and two platforms we demonstrate that speedups of up to 2.62 can be achieved. At the same time we show that changing network traffic patterns demand an adaptive approach to performing *online data transformation*. In our future work we will investigate possibilities for continuous performance monitoring and efficient, on-the-fly data restructuring.

REFERENCES

- [1] A. M. Odlyzko, "Internet traffic growth: sources and implications," in *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, ser. Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference, B. B. Dingel, W. Weiershausen, A. K. Dutta, & K.-I. Sato, Ed., vol. 5247, Aug. 2003, pp. 1–15.
- [2] Miniwatts Marketing Group, "World internet users and population stats," Retrieved March 31, 2010, from <http://www.internetworldstats.com/stats.htm>.
- [3] K. Yi and J.-L. Gaudiot, "Features of future network processor architectures," *John Vincent Atanasoff Modern Computing, International Symposium on*, vol. 0, pp. 69–76, 2006.
- [4] C. Kulkarni, M. Gries, C. Sauer, and K. Keutzer, "Programming challenges in network processor deployment," in *CASES '03: Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems*. New York, NY, USA: ACM, 2003, pp. 178–187.
- [5] M. Gries, C. Kulkarni, C. Sauer, and K. Keutzer, "Exploring trade-offs in performance and programmability of processing element topologies for network processors," in *Proceedings of Network Processor Workshop in conjunction with Ninth International Symposium on High Performance Computer Architecture (HPCA-9)*. Morgan Kaufmann, 2003, pp. 75–87.
- [6] J. Wagner and R. Leupers, "Advanced code generation for network processors with bit packet addressing," in *Proceedings of the Workshop on Network Processors (at HPCA8)*, 2002.
- [7] —, "C compiler design for an industrial network processor," in *LCTES '01: Proceedings of the ACM SIGPLAN workshop on Languages, compilers and tools for embedded systems*. New York, NY, USA: ACM, 2001, pp. 155–164.
- [8] J. Kim, S. Jung, Y. Paek, and G.-R. Uh, "Experience with a retargetable compiler for a commercial network processor," in *CASES '02: Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems*. New York, NY, USA: ACM, 2002, pp. 178–187.
- [9] J. Dai, B. Huang, L. Li, and L. Harrison, "Automatically partitioning packet processing applications for pipelined architectures," *SIGPLAN Not.*, vol. 40, no. 6, pp. 237–248, 2005.
- [10] X. Zhuang, "Compiler optimizations for multithreaded multicore network processors," Ph.D. dissertation, Atlanta, GA, USA, 2006, adviser-Pande, Santosh.
- [11] S. Ramakrishna and H. Jamadagni, "Analytical bounds on the threads in ixp1200 network processor," *Digital Systems Design, Euromicro Symposium on*, vol. 0, p. 426, 2003.
- [12] T. Ding and N. Liu, "A parallelizing compiler approach based on ixa," in *ICESS*, ser. Lecture Notes in Computer Science, L. T. Yang, X. Zhou, W. Zhao, Z. Wu, Y. Zhu, and M. Lin, Eds., vol. 3820. Springer, 2005, pp. 720–725.
- [13] Y.-K. Chang and F.-C. Kuo, "Packet processing with blocking for bursty traffic on multi-thread network processor," in *HPSR'09: Proceedings of the 15th international conference on High Performance Switching and Routing*. Piscataway, NJ, USA: IEEE Press, 2009, pp. 160–165.
- [14] D. Fenacci, B. Franke, and J. Thompson, "Automatic identification of tuning opportunities for domain-specific compilers using decision trees for data mining," in *Proceedings of the 13th International Workshop on Software and Compilers for Embedded Systems (SCOPEs 2010)*, 2010.
- [15] The Embedded Microprocessor Benchmark Consortium, "EEMBC benchmarks," <http://www.eembc.org>, 2008.
- [16] T.-c. Chiueh and P. Pradhan, "Cache memory design for internet processors," *IEEE Micro*, vol. 20, no. 1, pp. 28–33, 2000.
- [17] Z. Liu, J. Yu, X. Wang, B. Liu, and L. Bhuyan, "Revisiting the cache effect on multicore multithreaded network processors," in *DSD '08: Proceedings of the 2008 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 317–324.
- [18] K. Gopalan and T.-c. Chiueh, "Improving route lookup performance using network processor cache," in *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2002, pp. 1–10.
- [19] R. Huggahalli, R. Iyer, and S. Tetrick, "Direct cache access for high bandwidth network I/O," in *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 50–59.
- [20] B. Xu, J. Chang, S. Huang, Y. Xue, and J. Li, "Efficiency of cache mechanism for network processors," *Tsinghua Science & Technology*, vol. 14, no. 5, pp. 575 – 585, 2009. [Online]. Available: <http://www.sciencedirect.com/science/article/B7RKT-4XBR35X-5/2/33df60e0%245ef81dbe9622818f184ae2>
- [21] J. Poovey, "Characterization of the EEMBC benchmark suite," <http://www.eembc.org>, 2007.
- [22] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. New York, NY, USA: ACM, 2005, pp. 190–200.
- [23] M. Hagog and C. Tice, "Cache aware data layout reorganization optimization in GCC," in *Proceedings of the GCC Summit*, 2005.
- [24] Y. Zhong, M. Orlovich, X. Shen, and C. Ding, "Array regrouping and structure splitting using whole-program reference affinity," *SIGPLAN Not.*, vol. 39, no. 6, pp. 255–266, 2004.
- [25] R. Hundt, S. Mannarswamy, and D. Chakrabarti, "Practical structure layout optimization and advice," in *CGO '06: Proceedings of the International Symposium on Code Generation and Optimization*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 233–244.
- [26] A. Aggarwal, "Software caching vs. prefetching," in *ISMM '02: Proceedings of the 3rd international symposium on Memory management*. New York, NY, USA: ACM, 2002, pp. 157–162.
- [27] R. Ghiya and L. J. Hendren, "Is it a tree, a dag, or a cyclic graph? a shape analysis for heap-directed pointers in c," in *POPL '96: Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. New York, NY, USA: ACM, 1996, pp. 1–15.
- [28] C. Lattner and V. Adve, "Data Structure Analysis: An Efficient Context-Sensitive Heap Analysis," Computer Science Dept., Univ. of Illinois at Urbana-Champaign, Tech. Report UIUCDCS-R-2003-2340, Apr 2003.
- [29] C. Walsworth, E. Aben, kc claffy, and D. Andersen, "The CAIDA anonymized 2009 internet traces - 31/03/2009," http://www.caida.org/data/passive/passive_2009_dataset.xml, 2009.
- [30] Y. Hyun, B. Huffaker, D. Andersen, E. Aben, C. Shannon, M. Luckie, and kc claffy, "The CAIDA IPv4 routed/24 topology dataset - 13/09/2009," http://www.caida.org/data/active/ipv4_routed_24_topology_dataset.xml, 2009.
- [31] "CAIDA: The cooperative association for internet data analysis," <http://www.caida.org>.