A Random Linear Network Coding Platform MPSoC Designed in 22nm FDSOI

Mattis Hasler[†], Sebastian Haas[†], Robert Wittig[‡], Stefan Scholze[§],
Andreas Dixius[§], Sebastian Höppner[§], Gerhard Fettweis[†‡], Christian Mayr[§]
[†]Barkhauseninstitut gGmbH Dresden, Germany, Email: name.surname@barkhauseninstitut.org
[‡]Vodafone Chair Mobile Communications Systems TU Dresden, Germany, Email: name.surname@tu-dresden.de
[§]Chair of Highly-Parallel VLSI Systems and Neuro-Microelectronics TU Dresden, Email: name.surname@tu-dresden.de

***Abstract*—Random linear network coding (RLNC) has great potential to improve security, reliability, energy efficiency and throughput of many applications in networking and storage applications. The high computation costs and power consumption caused a reduction of interest in RLNC research more than ten years ago. We present a distributed parallel computation platform aiming at making RLNC affordable and scalable enough to be deployed in real-life-sized applications. As key component of this platform, an MPSoC was developed, produced and measured in our lab. The design aims at high energy efficiency and utilizes a hierarchical communication system for scalability to reach data rates needed by real-life applications with a reasonable power budget. For example, our platform would suffice to equip a 36 Gb/s backplane, 20 W Ethernet switch with an RLNC accelerator on a power budget of 2.4 W, showing an energy efficiency of 37 pJ/b.**

***Index Terms*—MPSoC, RLNC, NoC, memory management, low power, network coding**

I. Introduction

Random linear network coding (RLNC) is a powerful tool with a diverse set of applications. It can be used to improve security, reliability, energy efficiency or throughput of digital networks as well as storage systems [1]–[4]. For example, gains on throughput of 50% [5] or buffer size reduction of 40% [6] are possible in certain real-life Ethernet applications. In RLNC, randomly combining (i.e. encoding) data into arbitrary many variants can be used to synthesize reliability or security properties of the encoded data. Definitely, encoding, decoding, and recoding has to be done extensively. For networking applications, for example, each data stream has to be recoded in each networking node (i.e. switch). Unfortunately, the (en/de/re)coding is expensive, consuming a lot of computing time and energy. So, although being a promising technology, research in this direction started declining almost ten years ago. We propose an RLNC platform with an energy efficiency that makes the development of RLNC enabled real-life applications feasible, by taking on the two main problems of RLNC implementation.

The first problem is that a general-purpose CPU does not reach sufficient data rates in RLNC coding. In 2014 RLNC was optimized on a, for that time, recent CPU, reaching around $9.5\,\mathrm{Gb/s}$ on a $1.7\,\mathrm{GHz}$ x86 CPU utilizing different SSE extensions [7]. Using this solution for RLNC-enabled applications would require a great amount of additional hardware. For example, an application that stores encoded data on hard drives, would constantly require around $50\,\%$ of CPU to cope with each $4.8\,\mathrm{Gib/s}$ SATA3 data stream. Similarly, a $10\,\mathrm{Gib/s}$ Ethernet port would require a full CPU. This does not only increase product costs extremely but also leads to the second problem: high energy consumption. The desktop CPU used in [7] has a thermal design power (TDP) of $15\,\mathrm{W}$ [8], so that it runs RLNC at a power efficiency of $1.63\,\mathrm{nJ/b}$. Based on this power consumption an estimate can be made for the costs of equipping a simple $36\,\mathrm{Gib/s}$ backplane Ethernet switch [9] with RLNC. To match the transfer rate of the switch $1.6\,\mathrm{nJ/b} \times 36\,\mathrm{Gib/s} = 57\,\mathrm{W}$ would be consumed for RLNC, an increase of almost $300\,\%$ compared to the switches basic $20\,\mathrm{W}$ TDP. It becomes obvious that using a desktop x86 CPU is not the right approach when integrating RLNC into an Ethernet switch.

However, the research conducted on RLNC ten years ago was mainly done on general-purpose hardware. This seems unproblematic at first, because although there is no hardware specifically designed for RLNC in an x86 CPU, with the use of SSE extensions the computation can be done at astonishing data rates, performing multiple operations per clock cycle on average. However, the high power consumption of an x86 CPU results in very poor energy efficiency. This is partly because of the immense functional range of x86 causing a large chip area, of which only a fraction is used for RLNC, and partly by the high clock frequency of desktop CPUs. While the clock frequency naturally relates proportionally to the data rate, it overproportionally relates to power. Increasing the clock frequency will, despite increasing the data rate, lower the energy efficiency.

Because increasing clock frequency or per-cycle data rate of a single processor are not viable options, our approach is to increase energy efficiency by keeping frequency and processor complexity low and instead parallelize computation at the thread level. This of course means, that multiple threads are running in parallel on physically different processors. Following the well-known idea of the application-specific instruction-set processor (ASIP), a small RISC processor is extended to match the per cycle throughput of the x86-SSE approach but with much less overhead of area and power consumption.

This research is co-financed by public funding of the state of Saxony/Germany.

Additionally, the design is targeted at a lower clock frequency to improve efficiency. To compensate for the reduced data rate, multiple processors are coupled to a distributed computation platform. Its communication framework makes it scalable enough to easily reach rate rates in the above $10\,\mathrm{Gib/s}$.

Although not commonly used ten years ago, nowadays specialized hardware blocks are more common. Advancements in tooling allow exploring different solutions. ASIPs are often based on very small and efficient processors making them the perfect target for power-efficient implementations of specific algorithms. The ASIP approach maintains the flexibility of the processor it is based on while reaching per-cycle performance close to an ASIC implementation, or the SSE approach in this case. The hardware (i.e. chip area) and power consumption costs, however, are similarly low to an ASIC implementation. We present a hierarchical parallel computations platform, that includes multiple layers of parallel computation. At the lowest level, the ASIP uses its ISA extensions to perform SIMD style operations on multiple words in parallel. Multiple processors working in parallel on one chip are the next layer, which extends to a third, consisting of chips coupled with direct chip-to-chip links. We present an MPSoC that implements this platform, showing superior energy efficiency and the ability to scale throughput to double-digit gigabit per second data rates. As an example application, we estimate the feasibility to use our platform as an RLNC accelerator for an Ethernet switch.

The remainder of this work briefly examines previous RLNC hardware implementations in Section II, then cover the basics of RLNC and its computation hotspots in Section III. In Section IV the MPSoC is presented, especially the components crucial for efficiency. The lab measurement, comparison with other works and the evaluation of a potential Ethernet switch RLNC accelerator are addressed in Section V.

II. State of the Art

A data rate of RLNC computation of at least single-digit $\mathrm{Gb/s}$, coupled with a power consumption of a microcontroller, is a gap that has not been filled until now, best to our knowledge. The use cases would be manifold, reaching every corner of consumer electronics processing data, like network interfaces, network switches, storage controllers, DMA controllers, etc.

When the research on RLNC peaked out, mainly general-purpose hardware was used [10]–[12]. As already mentioned, these approaches are limited in energy efficiency. However, the idea of accelerating RLNC in hardware has hardly been picked up. In 2014 an implementation [7] on a general purpose CPU did make use of the available vector extension (e.g. SSE2, etc.). Although this was not hardware developed for the specific case, the per-cycle performance is close to a theoretical bound. A full custom design was presented in [13]. The implementation is part of an IoT chip and focuses on very efficient computation and a data rate around $1\,\mathrm{Mb/s}$. Another work implementing RLNC in hardware is [14] and does so using FPGAs. It reaches data rates of $65\,\mathrm{Mb/s}$. An ASIP implementation has been done in [15], reaching $1.2\,\mathrm{Gib/s}$ on a small Cadence Tensilica LX5 RISC processor in simulation, assuming a clock frequency of $300\,\mathrm{MHz}$. None of the found hardware implementations say anything about scalability. So, although one may be superior in energy efficiency and another outperforms our ASIP in pure data rate, none of these works can truly be compared to our platform, merely serving as reference points.

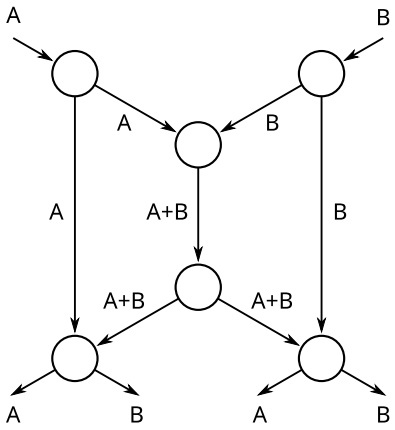

Fig. 1. RLNC example in butterfly network. Two sources transmit data A and B to two sinks with the help of an encoded packet in the middle link.

III. Network Coding

RLNC originates from a work conducted in 1978 [16] on satellite broadcasting the combination of two data streams, which could be decoded by the receivers with the knowledge about the respective other stream. Modern RLNC extends this idea to the point that a sender can generate an arbitrary number of random linear combinations of source data blocks. The receiver can recover the original data if —and only if— it can gather a minimum number of any of those encoded blocks, given they are linear independent. This does not only have use cases connected to transmission but also storage and distribution of data. On the one hand, the loss of any encoded block can be coped with by producing an additional encoded block, probably in advance. On the other hand, distributing encoded blocks to multiple potentially untrusted data handlers (i.e. network infrastructure or storage providers) prevents each handler to decode the data on its own, thus securing the data's secrecy.

RLNC has the potential to improve the costs for resilience, increase bandwidth and reduce transmission delays in complex, chaotic, or lossy communication networks. In Fig. 1 a networking example shows two senders injecting the data A and B into the network. Both receivers at the bottom want to receive both A and B. Without RLNC this would only be possible by sending A and B sequentially over the middle link. But with RLNC a combination of both is transmitted allowing both receivers to recover the missing datum with the help of the respective other datum. Notably, it is assumed that the network nodes can combine packets by encoding them on the fly.

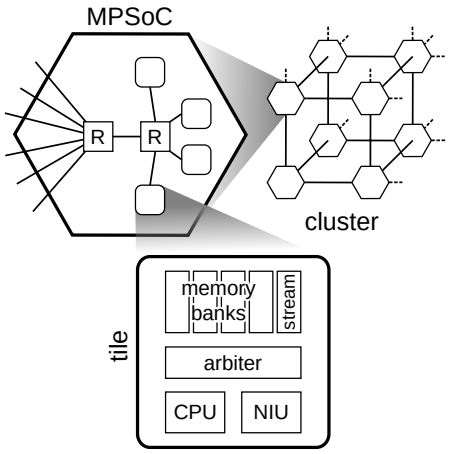

Fig. 2. Hierarchical overview of the proposed MPSoC with the levels: cluster of chips; chip of tiles; computing tile

The coding technic needed by the nodes depends on the multiplication and inversion of matrices with values from Galois Fields ($GF(q)$). To generate a number of coded packets into a matrix X a random coefficient matrix C is multiplied with a source matrix M containing a set of source packets.

$$X = C \times M$$

Similarly, recoding (re)encodes already coded packets without decoding them first, by combining data and coefficients of the source packets. For decoding, the receiver gathers coded packets filling the receiving coefficient matrix $\bar{C}$ and the data matrix $\bar{X}$. When a sufficient number of packets are available the original packets are recovered by solving [17]:

$$M = \bar{C}^{-1} \times \bar{X}$$

As described in [17] a network router (packet forwarder) mostly has to perform packet coding or recoding. Thus, the acceleration technics presented in this work focus on the expensive matrix multiplication in $GF(q)$. When targeting Ethernet, a packet size of 1500 bytes, 16 coefficients and a Galois Field size of $GF(2^8)$ are a reasonable parameters set [15].

IV. MPSoC Architecture

In Fig. 2 a flexible distributed computation architecture is shown, that implements the hierarchical communication structure mentioned in the Introduction (Section I). Similar to [18], the platform is a cluster of MPSoCs connected by chip-to-chip links. Inside each MPSoC a network-on-chip (NoC) implements the connectivity of an arbitrary number of computing tiles. Each tile represents a dedicated computation unit featuring a processor with local memory, a data streaming controller, and a network interface.

Following the defined architecture in this work, an MPSoC is proposed that has been successfully designed manufactured, and measured in the lab. It uses four application-specific processors (ASIP), with an extended instruction set, each being able to process $645\,\mathrm{Mb/s}$ of data. The data management is implemented by a closely coupled memory management of 15 $4\,\mathrm{KiB}$ memory banks, a shared memory streaming controller, a network on chip (NoC) with $64\,\mathrm{Gb/s}$ links, and six $6\,\mathrm{Gb/s}$ chip-to-chip LVDS links.

A. RLNC Accelerator

As examined in [19] an application-specific instruction-set processor (ASIP) yields a good balance between the power efficiency of an ASIC implementation of a certain algorithm and the flexibility of a general-purpose CPU. When implementing an algorithm with the ASIP approach, the most demanding operations are fixed and optimized in hardware while the majority of the algorithm —representing only a fraction of computation effort— can be written in software, allowing for run-time adaptation. For RLNC Galois Field multiplication has been identified as the single most important hotspot. Within a tile, a configurable and extendable Tensilica LX6 RISC core is used as the main processor. Similar to the solution of [15] the LX6 is extended by Galois Field ($GF(2^8)$) multiplication instructions.

Custom instructions are available to perform 16x 8-bit $GF(2^8)$ —or 8x 16-bit $GF(2^{16})$— multiplications at once. In the processor, the VLIW support option is selected to allow the folding of load/store and multiplication instructions. Fully utilized, the processor pipeline can process 128-bit data each cycle. In a single cycle, it loads 16 new source values (128-bit) into the register file, multiplies another set with an internal coefficient, and stores 128-bit data over the other memory port. The flow is occasionally disturbed by the necessity to load new coefficients which harms the throughput. Still, the availability of input data to the processor is crucial when maximizing throughput. This is why the memory subsystem is the first thing to be designed to serve these data rates.

B. Memory System

Each processor is connected to a private closely coupled memory of 15x $4\,\mathrm{KiB}$ memory banks. Because double-port memory consumes almost twice the power and chip area compared to single-port memory [20]–[22], the latter are used. To efficiently share the single-port memories, offline arbitration is used [23]. The memory arbiter connects memory masters (i.e. processor memory ports) to memory banks. It keeps master-to-bank assignments in a register set, only reprogramming it when necessary. A reprogramming happens when a memory master starts requesting addresses from a different bank than before. This way the bank selection is cut from the processor–memory critical path, resulting in fast memory accesses for consecutive transactions to the same bank while paying/waiting two cycles when switching the memory bank. Multiple masters accessing different physical memory banks do not interfere with each other.

C. Network on Chip

Although the NoC allows arbitrary topologies —especially enabling the potential for bigger MPSoCs— in this instance,

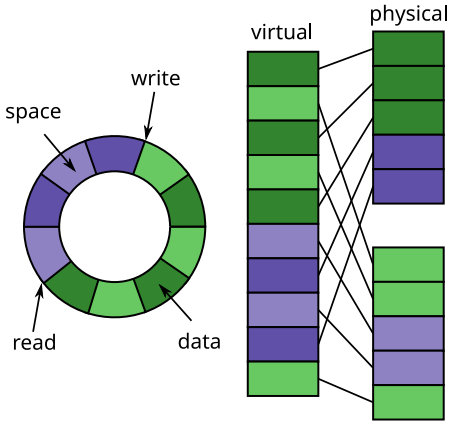

Fig. 3. Concept of a shared memory-based FIFO controller. Slices of the FIFO's ring buffer are mapped to two physically independent memory banks. Because of that, memory access is evenly distributed between the two banks.

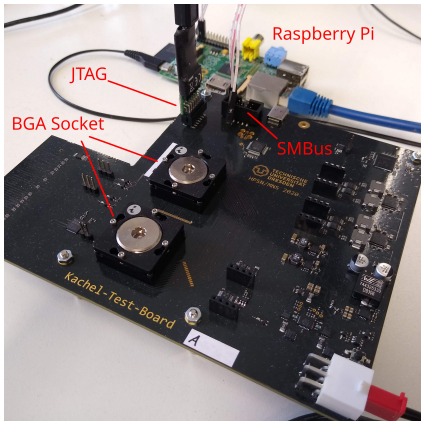

Fig. 4. Testboard with two BGA400 sockets, JTAG, PMBus connections.

only two routers are present (Fig. 2), because it yields the best balance of router size and bandwidth. One of the routers connects the four computation tiles, while the other manages the chip-to-chip links for communication beyond the chip boundary. Each on-chip link provides a full-duplex connection that transports one 131-bit packet each cycle in each direction. A transfer may be composed of one header packet and multiple body packets. The capacity of payload data of header and body packets are 64 and 128 bit. With reasonable sized transfers the data rate achieved by the NoC gets close to 128 bit per cycle ($64\,\mathrm{Gb/s}$ @$500\,\mathrm{MHz}$), which is enough to keep the computation tile busy.

The NIU can handle one NoC packet every cycle, which may contain up to 128 bit of data and forward it to the memory. Parallel to that it may load 128 bit from local memory on the other port and forge NoC packets to be sent. It provides the often on MPSoCs available RDMA protocol [24]–[27] to support basic data transfers between local processor memories and is closely coupled with a custom streaming controller to implement transparent FIFO channels between processors.

D. FIFO channels

The purpose of the hardware FIFO controller is to relieve its users from doing buffer management in software. Additionally, it enables hardware units, like the NIU, to use FIFO channels without implementing complex ring buffer logic themself. Similar to the approach in [28] the FIFO channels are implemented using a hardware shared memory FIFO controller. It allows the definition of FIFO channels using a cyclic ring buffer and a set of read and write pointers to control the filling of the buffer. It calculates pointers and enforces the integrity of the buffers completely in hardware. The pointer calculation logic distributes the buffer over two memory banks and interleaves access to it (Fig. 3). This helps to separate memory transactions of the two users (i.e. reader and writer) into different physical memory banks. Theoretically, the FIFO can transfer a full memory word each cycle, because the reader and writer access different memory banks in parallel. In practice, due to memory bank switch penalties and pointer calculation, transfer speed reaches around 90 % of the theoretic boundary. This drawback is, however, overcompensated by the aforementioned use of single port memories, cutting power consumption and chip area in half, compared to dual port memories.

The NIU uses these hardware-guided FIFO buffers to implement streaming channels between processors on different local tiles or even different chips. To do so, one NIU takes the role of the FIFO reader, sending the data to an NIU on another tile. Here, the NIU acts as the FIFO writer streaming data to the buffer. To prevent buffer overflows in the receiving tile, the receiver NIU notifies the sender NIU about its local buffer filling level. The software on both sides only has to work with the FIFO hardware interface.

V. Evaluation

GlobalFoundries produced the chip in $22\,\mathrm{nm}$ FDSOI, the die has an area of $9\,\mathrm{mm}^2$ and is shown in Fig. 5. The chip has multiple supply voltages applied to different areas. The SRAM macros, for example, need a supply of $0.8\,\mathrm{V}$, but the processors may run between $0.4\,\mathrm{V}$ and $0.7\,\mathrm{V}$ depending on the applied clock frequency to save power.

A custom BGA400 package houses the die. It is composed of two PCB boards, one implementing the 3D connection network from the bond pads to the balls and the other providing the frame for the filling. The solution is intended for low-volume chip production as usually needed by scientific projects.

A test board hosts two BGA400 sockets. The power management is controllable via a PMBus breakout on the board. For general debug access, a socket provides a connection to the chip's JTAG taps. The two BGA sockets are connected to each other to allow testing of the chip-to-chip links. A photo of the test setup in Fig. 4 shows the two BGA sockets, a USB-converter connected to the JTAG socket, and a Raspberry Pi connected to the PMBus with its I^2C port.

Targeting a network router, the main workload of the proposed system will be data recoding. As already mentioned, the

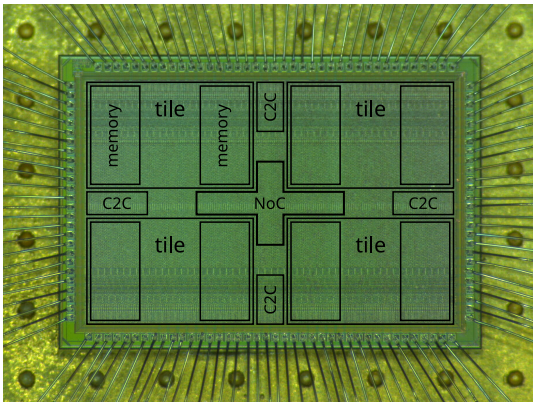

Fig. 5. Die Photo of the proposed chip. Computation tiles; Network-on-Chip; Chip-to-Chip links (C2C)

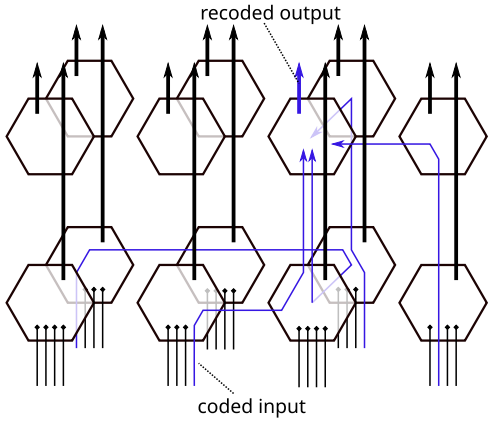

Fig. 6. Cluster of 14 chips, each producing a recoded output stream from mixing multiple input streams. Highlighted is the dataflow of one recoded stream.

TABLE I
SPLITUP OF THE CHIPS POWER CONSUMPTION AND SUPPLY VOLTAGES

name	tile power [mW]	chip power [mW]	supply [V]
LX6 RLNC	19	76	0.7
chip-to-chip link	15	90	1.6
PLL and power	–	20	0.6
mem idle	–	0.3	0.8
mem RLNC	3.1	12	0.8

computational complexity of recoding comes almost entirely from matrix multiplication in $GF(2^8)$. Hence, in the evaluation, the matrix multiplication rate is used as the performance measure. To resemble the computations that appear in RLNC for Ethernet, the data and coefficient vectors are of length 1500 and 16, respectively.

Using these parameters, a single CPU handles $645\,\mathrm{Mb/s}$ of matrix multiplication with a clock frequency of $500\,\mathrm{MHz}$. Together, the four CPUs on one chip process $2.58\,\mathrm{Gb/s}$. Adding a certain number of chips to the cluster of chips increases the possible RLNC data rate and the power consumption accordingly.

A. Power

The chip runs at a core frequency of $500\,\mathrm{MHz}$ @ $0.7\,\mathrm{V}$. On-chip memory uses a separate power supply of $0.8\,\mathrm{V}$. An additional power supply provides power for the PLL ($0.6\,\mathrm{V}$) and IO-Pads ($1.6\,\mathrm{V}$). Running the $GF(2^8)$ matrix multiplication on all four processors draws a power of $111\,\mathrm{mW}$ with one chip-to-chip link enabled. Power consumption in the chip is split up as displayed in Tab. I. Main power consumers are the LX6 cores with $19\,\mathrm{mW}$ and the chip-to-chip links with $15\,\mathrm{mW}$. On-chip memory consumes $0.3\,\mathrm{mW}$ when idle and increases by $3.1\,\mathrm{mW}$ per active processing tile.

B. Scalability

The data computation throughput of one chip with mentioned parameters accumulates to $2.58\,\mathrm{Gb/s}$, which is approximately a factor of two smaller than the bandwidth of an off-chip link. That means that the data processed by a single chip can comfortably be streamed onto and off the chip over a single link. This has a positive effect on the chip's power consumption since an off-chip link consumes with $15\,\mathrm{mW}$ @ $1.6\,\mathrm{V}$ almost as much power as a whole tile with $19\,\mathrm{mW}$ @ $0.7\,\mathrm{V}$ (cf. Tab. I) due to its higher supply voltage. The availability of multiple links allows for more complex setups.

In order to suffice as a platform to provide RLNC functionality for an off-the-shelf Ethernet router [9] with $36\,\mathrm{Gb/s}$ backplane, a total of 14 chips would be needed to provide enough computation power. In Fig. 6 a setup is sketched that might handle the mentioned data rate. Each of the 14 chips produces a recoded data stream that leaves the cluster at the top. The streams can be dynamically mixed from an arbitrary number of input streams (e.g. 28), that enter the cluster into the bottom layer of chips. The six chip-to-chip links allow an efficient data distribution across the whole cluster. Within the cluster, 60 chip-to-chip links are enabled to provide internal communication and stream data into and out of the cluster. These would consume approximately $60 \times 15\,\mathrm{mW} = 900\,\mathrm{mW}$. With an additional $14 \times 108\,\mathrm{mW} = 1512\,\mathrm{mW}$ for the chips' number crunching, the whole cluster consumes $2.4\,\mathrm{W}$.

C. Comparison

The measured power consumption and computation throughput results in an energy demand of $37\,\mathrm{pJ/b}$ (Tab. I). That compares well (Tab. II) against a software implementation done on a desktop processor in [7], which has an almost four times higher throughput bought with an increase in energy consumption by two orders of magnitude. Another implementation targeting IoT devices [13] achieves energy efficiency three orders of magnitude lower, but only for single-digit $\mathrm{Mb/s}$ data rates and with no scalability options. Similarly, the FPGA implementation from [14] beats the proposed implementation by a factor of two in terms of energy efficiency, but the data rate of $65\,\mathrm{Mb/s}$ lacks the potential to scale the system to data rates above single-digit $\mathrm{Gb/s}$.

VI. CONCLUSION

This work presented a distributed computation platform defining a hierarchical structure as a cluster of chips parti-

TABLE II
DATA RATE AND ENERGY EFFICIENCY COMPARISON

name	throughput [Gb/s]	energy consumption [pJ/b]
this work	2.58	37.23
desktop [7]	9.5	3157
IoT [13]	0.001	0.015
FPGA [14]	0.065	19.04
ASIP [15]	1.3[a]	175[a]

[a] results from RTL simulation and power estimation

tioned into computation tiles. The featured MPSoC implements the platform, targeting an application as an RLNC accelerator. The computation tiles consist of an ASIP core, an optimized closely coupled memory system, a specialized FIFO controller, and an NoC interface. The NoC is extended across chip boundaries with chip-to-chip links allowing the connection of multiple chips to a computation cluster. A single chip handles $2.58\,\mathrm{Gb/s}$ of $GF(2^8)$ matrix multiplication, which resembles an Ethernet RLNC recoding workload. The power consumption of a single chip varies between $110\,\mathrm{mW}$ and $170\,\mathrm{mW}$ depending on the number of used chip-to-chip links, which allows the concept of an $2.5\,\mathrm{W}$ cluster to equip an Ethernet router with an RLNC accelerator. In comparison, the same data rate would need four desktop CPUs [7] consuming roughly $60\,\mathrm{W}$.

REFERENCES

[1] C. Gkantsidis, J. Miller, and P. Rodriguez, "Comprehensive view of a live network coding p2p system," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, 2006, pp. 177–188.

[2] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft, "Xors in the air: practical wireless network coding," *IEEE/ACM Transactions on networking*, vol. 16, no. 3, pp. 497–510, 2008.

[3] F. H. Fitzek, T. Toth, A. Szabados, M. V. Pedersen, D. E. Lucani, M. Sipos, H. Charaf, and M. Medard, "Implementation and performance evaluation of distributed cloud storage solutions using random linear network coding," in *2014 IEEE International Conference on Communications Workshops (ICC)*. IEEE, 2014, pp. 249–254.

[4] P. Ostovari, J. Wu, A. Khreishah, and N. B. Shroff, "Scalable video streaming with helper nodes using random linear network coding," *IEEE/ACM Transactions on Networking*, vol. 24, no. 3, pp. 1574–1587, 2015.

[5] K. Miller, T. Biermann, H. Woesner, and H. Karl, "Network coding in passive optical networks," in *2010 IEEE International Symposium on Network Coding (NetCod)*. IEEE, 2010, pp. 1–6.

[6] A. Engelmann, W. Bziuk, A. Jukan, and M. Médard, "Exploiting parallelism with random linear network coding in high-speed ethernet systems," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2829–2842, 2018.

[7] S. M. Günther, M. Riemensberger, and W. Utschick, "Efficient gf arithmetic for linear network coding using hardware simd extensions," in *2014 International Symposium on Network Coding (NetCod)*. IEEE, 2014, pp. 1–6.

[8] I. Inc. (2013) Intel core i3-4010u prozessor. [Online]. Available: https://ark.intel.com/content/www/de/de/ark/products/75107/intel-core-i34010u-processor-3m-cache-1-70-ghz.html

[9] *Cisco Business 220 Series Smart Switches*, Cisco System, 2021.

[10] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *IEEE Transactions on Information Theory*, vol. 52, no. 10, pp. 4413–4430, 2006.

[11] M. Wang and B. Li, "How practical is network coding?" in *200614th IEEE International Workshop on Quality of Service*. IEEE, 2006, pp. 274–278.

[12] X. Chu, K. Zhao, and M. Wang, "Practical random linear network coding on gpus," in *International Conference on Research in Networking*. Springer, 2009, pp. 573–585.

[13] G. Angelopoulos, A. Paidimarri, M. Médard, and A. P. Chandrakasan, "A random linear network coding accelerator in a 2.4 ghz transmitter for iot applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2582–2590, 2017.

[14] S. Kim, W. S. Jeong, W. W. Ro, and J.-L. Gaudiot, "Design and evaluation of random linear network coding accelerators on fpgas," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 1, pp. 1–24, 2013.

[15] J. Acevedo, R. Scheffel, S. Wunderlich, M. Hasler, S. Pandi, J. Cabrera, F. H. Fitzek, G. Fettweis, and M. Reisslein, "Hardware acceleration for rlnc: A case study based on the xtensa processor with the tensilica instruction-set extension," *Electronics*, vol. 7, no. 9, p. 180, 2018.

[16] M. Celebiler and G. Stette, "On increasing the down-link capacity of a regenerative satellite repeater in point-to-point communications," *Proceedings of the IEEE*, vol. 66, no. 1, pp. 98–100, 1978.

[17] D. Gonçalves, S. Signorello, F. M. Ramos, and M. Médard, "Random linear network coding on programmable switches," in *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 2019, pp. 1–6.

[18] G. Fettweis, M. Hassler, R. Wittig, E. Matus, S. Damjancevic, S. Haas, F. Pauls, S. Nam, and N. Grigoryan, "A low-power scalable signal processing chip platform for 5g and beyond-kachel," in *2019 53rd Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2019, pp. 896–900.

[19] K. Keutzer, S. Malik, and A. R. Newton, "From asic to asip: The next design discontinuity," in *Proceedings. IEEE International Conference on Computer Design: VLSI in Computers and Processors*. IEEE, 2002, pp. 84–90.

[20] K. Nii, M. Yabuuchi, Y. Tsukamoto, S. Ohbayashi, Y. Oda, K. Usui, T. Kawamura, N. Tsuboi, T. Iwasaki, K. Hashimoto *et al.*, "A 45-nm single-port and dual-port sram family with robust read/write stabilizing circuitry under dvfs environment," in *VLSI Circuits, 2008 IEEE Symposium on*. IEEE, 2008, pp. 212–213.

[21] J. P. Kulkarni, J. Keane, K.-H. Koo, S. Nalam, Z. Guo, E. Karl, and K. Zhang, "5.6 mb/mm 1r1w 8t sram arrays operating down to 560 mv utilizing small-signal sensing with charge shared bitline and asymmetric sense amplifier in 14 nm finfet cmos technology," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 229–239, 2017.

[22] F. Bai, B. Xiong, X. Xue, W. Song, W. Baofeng, N. Fu, B. Yu, H. Duan, X. Han, A. Minzoni *et al.*, "A two-port sram using a single-port cell array with a self-timed write-after-read control scheme to save 47% area & 63% standby power," in *ASIC (ASICON), 2017 IEEE 12th International Conference on*. IEEE, 2017, pp. 426–428.

[23] R. Wittig, M. Hasler, E. Matúš, and G. Fettweis, "Queue based memory management unit for heterogeneous mpsocs," in *Design Automation and Test in Europe (DATE)*, Florence, Italy, Mar 2019.

[24] G. Kalokerinos, V. Papaefstathiou, G. Nikiforos, S. Kavadias, M. Katevenis, D. Pnevmatikatos, and X. Yang, "Fpga implementation of a configurable cache/scratchpad memory with virtualized user-level rdma capability," in *2009 International Symposium on Systems, Architectures, Modeling, and Simulation*. IEEE, 2009, pp. 149–156.

[25] G. Kalokerinos, V. Papaefstathiou, G. Nikiforos, S. Kavadias, X. Yang, D. Pnevmatikatos, and M. Katevenis, *Prototyping a Configurable Cache/Scratchpad Memory with Virtualized User-Level RDMA Capability*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019, pp. 100–120. [Online]. Available: https://doi.org/10.1007/978-3-662-58834-5_6

[26] J. Ambrose, A. Molnos, A. Nelson, S. Cotofana, K. Goossens, and B. Juurlink, "Composable local memory organisation for streaming applications on embedded mpsocs," in *Proceedings of the 8th ACM International Conference on Computing Frontiers*, ser. CF '11. New York, NY, USA: Association for Computing Machinery, 2011. [Online]. Available: https://doi.org/10.1145/2016604.2016631

[27] A. Secco, I. Uddin, G. P. Pezzi, and M. Torquati, "Message passing on infiniband rdma for parallel run-time supports," in *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE, 2014, pp. 130–137.

[28] M. Hasler, R. Wittig, E. Matúš, and G. Fettweis, "Slicing fifos for on-chip memory bandwidth exhaustion," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 2, pp. 441–450, 2019.