FPGA Accelerated NoC-Simulation – A Case Study on the Intel Xeon Phi Ringbus Topology

Oliver Jakob Arndt, Christian Spindeldreier, Kevin Wohnrade,
Daniel Pfefferkorn, Martin Neuenhahn, and Holger Blume

Leibniz Universität Hannover, Institute of Microelectronic Systems, 30167 Hanover, Germany
{arndt, spindeldreier, wohnrade, pfefferk, neuenhahn, blume}@ims.uni-hannover.de

ABSTRACT

Complex signal processing algorithms targeted on architectures with increasingly high numbers of parallel processing units require high performance core-interconnections (i.e., low latencies, high throughput, no pinch-offs or bottlenecks). Therefore, assisting techniques, exploring characteristics of diverse topologies of common as well as innovative Network-on-Chips (NoCs), are necessary for the development of chips with massive parallel processing cores. In contrast to analytic NoC models, event driven NoC simulations can handle even complex task graphs, but however feature long simulation times. Enabling the simulation of even complex task graphs, in this work, we propose to use FPGA accelerated simulation. While we extend such a simulator in order to imitate cache coherence communication-behavior, we also present a translation of real measured profiles to task graphs for in-depth simulation of the communication behavior of an existing NoC-based manycore. Therefore, this approach is able to not only deal with synthetic scenarios, but analyse the communication behavior of real world applications. Additionally, a simulation of the *Histograms of Oriented Gradients* algorithm, running on the Intel Xeon Phi manycore, exhibiting a 70-stop ring-bus, exemplifies this approach.

1. INTRODUCTION

Signal processing chips continuously turn into landscapes of diverse even heterogeneous processing units. Programming these chips implies to distribute work over multiple units, synchronize tasks, and transfer data between cores' private caches, cache hierarchy-levels, and global memory. Depending on the software's complexity and individual demands on the underlying hardware, the communication often appears as the most critical bottleneck of a final application. Thus, low-latency and high bandwidth interconnections are required. Due to the trend of integrating increasingly high numbers of processors to a single die (i.e., manycore), these interconnection topologies become more complex and refer to network communication techniques and theories: *Network-on-Chip* (NoC). Classic approaches like the ring bus within Intel Sandy Bridge processors [1] or the ARM AHB bus used in the esa NGMP [2] lag of performance when scaling the number of cores. NoCs, consisting of nodes, network-interfaces, physical interconnect and network switches promise to provide a versatile usable interconnection network similar to Ethernet [3] addressing the interconnect bottleneck in manycore architectures. The basic architecture of a NoC is variable, while topologies like mesh, star or ring are widely used. As well several routing algorithms [4] are used to obtain the maximum performance of a NoC-based manycore depending on the application. For instance, the Intel Xeon Phi (Knights Corner) manycore card from the field of high-performance computing features 61 x86-cores and therefore also requires powerful interconnections. Not only used for rapid prototyping, this accelerator card's massive core-parallelism combined with 512 bit vector-registers and SIMD operations can dramatically speedup for instance image processing algorithms. However, as the cores of the Xeon Phi are arranged on a ring-bus, high data communication exchange between cores may limit the application's resulting performance.

Camera based driver-assistance algorithms, featuring high computational complexity as well as a huge amount of memory during the process, require powerful platforms. As these algorithms often provide high theoretical parallelism, parallel platforms like the Intel Xeon Phi can serve as viable implementation targets. Previous works presented a parallel implementation of the *Histograms of Oriented Gradients* (HOG) algorithm [5] for feature-based pedestrian detection running on the Intel Xeon Phi manycore architecture (stand-alone). While exhibiting both, functional and data parallelism, this implementation is supposed to introduce a high amount of communication demands, potentially pushing performance boundaries of NoC implementations.

Defining a NoC for a new manycore platform is a challenging task. The optimal network in terms of performance and chip area depends on the target system architecture as well as the target application class, so a realistic simulation of appropriate NoC topologies is demanded. There are several approaches to simulate the behavior of NoCs. Pure software based simulators either suffer from high computation times when simulating a high level of detail or from low accuracy when neglecting some levels of detail. Hardware

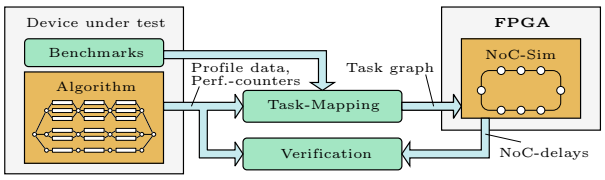


Figure 1: From HOG profile to NoC simulation.

This work was presented in part at the international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2017) Bochum, DE, June 7-9, 2017.

accelerated NoC simulators take advantage of the hardware performance, which allows the simulation of more complex and thus more realistic NoC topologies by mapping them to FPGAs. Processor cores are simulated by so-called virtual cores (VC) in order to spare FPGA-resources for the NoC to be analysed. This approach allows the simulation of communications of real-world applications via a NoC at a very high level of detail on short timescales.

In this work, we propose to use FPGA accelerated NoC simulation in order to enable the simulation of complex and thus realistic NoC communications. Therefore, we extend an existing FPGA accelerated NoC simulation framework to be able to imitate cache coherence communications, usually found in today's processors. Demonstrating the simulation of real-world applications, we use real measured profilings (timings and performance counter information) to be translated to a simulation task-graph, which is then mapped to the FPGA NoC simulator. As a case study, we reconstruct the ring-bus topology of the Intel Xeon Phi (Knights Corner) manycore processor on an FPGA and simulate the communication behavior of a HOG pedestrian detection algorithm. Following the idea of rating influences of various NoC-topologies, we furthermore evaluate the HOG communication on different NoC topologies. More precisely, this work makes the following contributions:

- **Conceptual design** of an FPGA accelerated NoC simulator for coherent cache communications
- **Reconstruction** of the Intel Xeon Phi ring-bus NoC and parameterization using benchmark results
- **Simulation** of the *Histograms of Oriented Gradients* (HOG) algorithm using real profiles of the Xeon Phi
- **Evaluation** of the simulation accuracy in order to identify application specific bottlenecks
- **Rating** of HOG runtime performances on different bus topologies via presented simulation tools

The remainder of the article is structured as follows: In Section 2, we briefly recap basics on NoCs on parallel processors and NoC simulations. Section 3 introduces the Intel Xeon Phi platform specifications and the implementation specifics of the HOG algorithm, representing the exemplary case study. While we first introduce the conceptual design of cache communications, also found on the Xeon Phi, in Section 4, we iteratively present all aspects considered in our model (e.g., using benchmarks), afterwards. After evaluating the results in Section 5, Section 6 concludes this paper.

2. RELATED WORK

Connecting multiple computational devices to the same memory combined with coherent caches enables parallelization by using shared-memory approaches [6], which can avoid processor idle times due to costly offload data-transfer delays. Depending on the parallelization scheme, shared and mutually updated cache lines or synchronizations between threads can involve high communication rates on the NoC. However, as sharing data as well as passing data between private caches (i.e., cache misses) cause long waiting times, for processes exhibiting higher data dependencies, communication may arise as the performance limiting bottleneck. In order to evaluate NoC-influences on the runtime performance of parallel processes, this work focuses on communication events enabling a more detailed NoC-analysis.

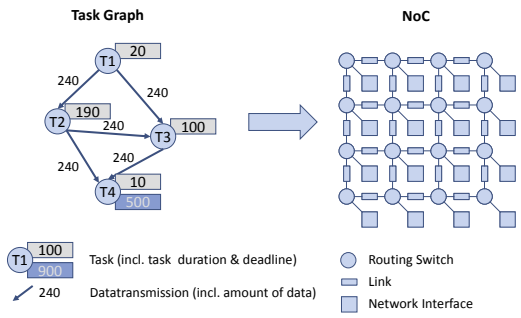


Figure 2: Task graph to NoC mapping.

NoC analysis is usually performed to evaluate the performance and cost (in terms of e.g. silicon size and energy consumption) of a NoC part of a many core processor for a given algorithm or class of algorithms. Figure 2 visualizes a schematic mapping of a simple task graph onto a mesh topology NoC. This analysis is always a trade-off between effort, flexibility and accuracy. The most accurate analysis is the cycle accurate simulation at register transfer level (RTL-simulation) [7]. Although, there are several approaches to speed up this simulation [8, 9], it is impossible to simulate data traffic of real world applications. On the other hand, there are more abstract analysis techniques like colored petri nets [7], zero-load models [10], or approaches using the queuing theory [11]. These approaches are quite flexible but lag the needed accuracy. A compromise allowing the cycle accurate analysis of NoCs with realistic amounts of data traffic is the FPGA-based NoC-emulation or FPGA accelerated simulation of NoCs [7, 12].

In order to test the performance of a NoC, usually randomly generated traffic or synthetic task graphs are used [13], as there are no standard benchmarks for NoC [14]. There are only few examples which try to use task-graphs generated from real applications representing algorithms and applications [15]. As these applications have not been implemented using a manycore processor with complex communication architecture, no comparison of simulated with measured results has been performed so far.

3. CASE STUDY

In this section, we introduce the Intel Xeon Phi manycore processor as well as the HOG implementation targeted on this massive parallel platform, representing the case study application. Previous works denoted a performance limitation of the HOG algorithm for increased numbers of parallel threads used on the Xeon Phi [16]. Therefore, in this work, a detailed analysis of of this application's communication behavior and potential bottlenecks will be presented.

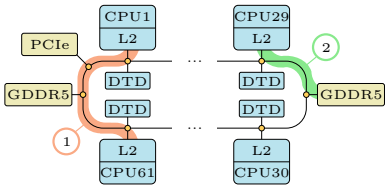
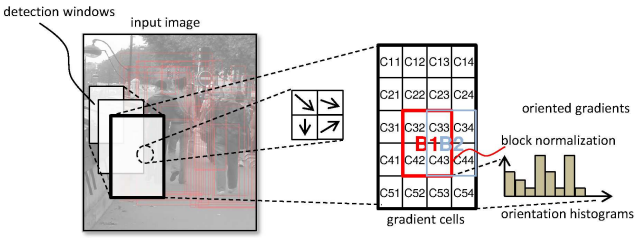
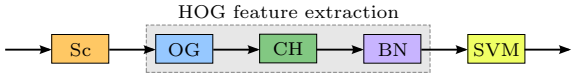


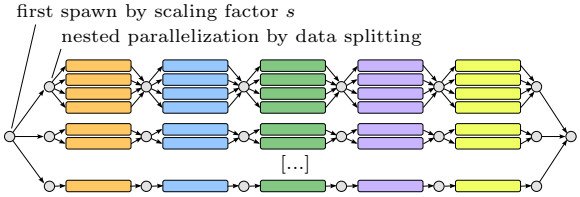
Figure 3: Intel XeonPhi NoC ring-bus topology.



(a) Feature extraction in the HOG algorithm.



(b) Algorithmic pipeline of processing stages.



(c) Nested parallelization in two levels.

Figure 4: HOG: algorithm basics and parallelization.

The Intel Xeon Phi (Knights Corner) manycore (Figure 3) features 61 cores with an in-order instruction pipeline at 1.2 GHz and each with four hyperthreads, 32 kB L1 and 512 kB private L2 cache. Interestingly, the cores are arranged alongside a ring bus with 70 bus stops: 61 cores (each with a private L2 cache), eight GDDR5 controllers, and a PCIe interface. While the 512 bit wide data bus is the largest, both the address bus and acknowledge bus exist twice in order to minimize performance limitations introduced by controlling commands. As all buses exist in two ring-directions, the Xeon Phi entirely exhibits ten buses.

In order to detect pedestrians – e.g., in scene interpretation for autonomous driving – HOG image-features to be classified are extracted inside of detection windows (DW) in different scales and positions on the input image I (Figure 4(a)). Enabling a fix size of the feature vector to be classified by an SVM, instead of scaling the DW by factor s, we inversely scale the width and height of an image by $\frac{1}{s}$ to I_s (stage *Sc* in Figure 4(b)). For each scaled image I_s, the entire HOG feature extraction in calculated, while DW positions are only differentiated in the very last stage (SVM). HOG features are calculated by firstly calculating oriented gradients (OG), which are used to generate orientation histograms inside cells of size 8×8 pixels (CH). After these histograms are normalized to an entire vector length of 1 inside blocks of 2×2 cells, which overlap each other by one cell, the concatenation of all blocks inside a DW position represents the final SVM feature vector. As visualized in Figure 4(c), a first parallelization (*functional decomposition*) is applied by image scales, as each scaled input image I_s can be processed independently in its own algorithmic pipeline. Producing a higher level of concurrencies, a further (nested) parallelization (*domain decomposition*) is applied inside of each algorithmic stage by splitting the data.

The HOG implementation used in this work is parallelized using the MPAL abstraction layer [17], which enables an improved portability between platforms by offering an independent generic API, which then refers to any external parallelization framework. Furthermore, MPAL can be used to automatically extract detailed profiling information, such as timings of spawns, synchronizations and tasks, or task associated performance counters – as far as they are available on the particular target platform. Using these profiling methods, a detailed call graph of parallel tasks, their dependencies, and requested memory resources (using performance-counters) can be extracted and thus mapped onto the FPGA-based NoC-simulation.

4. SIMULATION

In this section, we introduce the construction of the NoC simulation, which is adapted to the architectural specifications of the Intel Xeon Phi according to our case study application. We first describe the extensions to the given FPGA accelerated NoC simulation framework to match the used application's architecture. Secondly, we describe the translation of profile information using timing and performance counter measures of the parallel HOG implementation to a simulation conform task graph to be mapped onto the NoC simulation. In order to parameterize our simulation, we make use of small micro-benchmarks, which will be presented in the following as well.

4.1 NoC Model

Distributed tag directories (DTD), one assigned to each L2 cache, manage cache coherence on the Xeon Phi, while it is not necessarily the local L2-cache's content managed in a DTD. In order to evenly distribute memory commands over the ring, avoiding certain bus areas to emerge as a bottleneck, the DTDs' memory associativity is distributed over the ring using a hashing protocol. Therefore, once a cache line is not mirrored in the local L2, a request to the address-associated DTD on the bus is emitted to ask, if this cache line is mirrored and the request can hence be answered by another L2 cache on the ring (remote cache).

Due to unknown hashing methods for DTD associations, we decided to focus on modelling the data bus and therefore did not respect control packets in this first approach. Thus, the created model can reflect the upper performance bound (best case) of a ring bus communication behavior. In order to determine data dependencies of the HOG when executed on the Xeon Phi, we use profiles automatically measured by MPAL and extract the call graph, timing information, and the tasks' corresponding core IDs (i.e., tracing). Performance counter information identify the amount of data, which are communicated between cores, while we verify these numbers with algorithmic knowledge about the algorithm. As the associativity of memory addresses to memory controllers is also hashed to avoid bottlenecks on a single memory controller interface, in our simulation we randomly spread read/write main memory accesses over the eight GDDR5 controllers. The existing NoC model can either execute a task on virtual cores or send a message on the bus. As we model coherent caches, which may rather cause a constant rate of bus communications than sending a bundle of messages, we split up tasks in multiple chunks, intermediately sending smaller data-bursts to mime coherence (*task-splitting*). An increased number of splits improves the model toward the actual coherence behavior, while also complicating the task graph to be mapped to the FPGA.

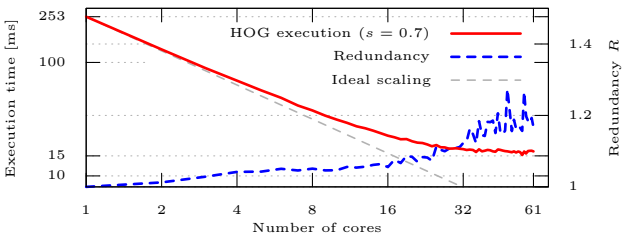


Figure 5: Parallel scaling behavior of the HOG.

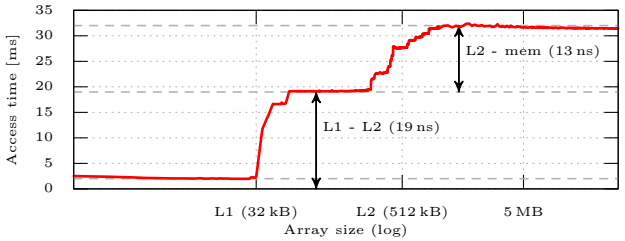


Figure 7: Memory level delays.

In previous versions of the NoC simulation framework FNOCEE, only circuit switching communication techniques were supported and evaluated. The bidirectional data bus of the Xeon Phi has a latency of one clock cycle per bus stop, and can therefore be seen as two independent shift registers per direction, while each register stage is connected to a DTD/L2-cache or a memory controller. In order to simulate this bus, packet switching was added to FNOCEE in this work. This additionally allows the evaluation of different switching technologies using the same NoC-Topology and task graphs. Two basic routing algorithms, XY-Routing and YX-Routing, are used for packet switching and will be extended by complex routing algorithms in the future.

To emulate the two Xeon Phi data buses, a NoC featuring 70 VCs was designed. Three different types of VCs are used to emulate different Xeon Phi bus stops: compute core, memory-, and PCIe-controller. As public available information about the bus topology of the Xeon Phi is limited, we analyzed several die shots of the Xeon Phi [18] to obtain the arrangement of the cores. The memory controller VCs are evenly distributed over the ring. Between two memory VCs there are 8 compute VCs, but only once the PCIe VC replaces a compute VC (No. 4) between two memory VCs.

4.2 Task Graph

As the original HOG exhibits plenty of tasks, resulting in a huge and complex task graph not fitting into the FPGA's local memory, we firstly profiled only the most communication intensive scaling cascade (i.e., $s = 0.7$) on the Xeon Phi for our study. As can be seen in Figure 5, this parallel version can only achieve a maximum speedup of 17 at a number of 61 cores. Behavioral parameter study according to MPAL's automated parameter extraction denotes an influence of scheduling overhead and work imbalance, which we not focus in this work. Furthermore, an increase of the tasks' actual execution time (written as the percentage increase $R(n)$ compared to the sequential execution), up to 20 % (50 ms) can be registered. Since no mutex locks are used in the HOG algorithm, the increase can only be caused by inter-core and core-to-memory communication delays, directly influenced by the NoC topology. In measured profile information, communication delays are hidden in task execution delays, including communication and computational portions. Therefore, we used these task timing information combined with the according cache miss rates (extracted with performance counters), to separate both portions. Finally, the pure computational portion is used to generate the task graph to be mapped to the FPGA and communication delays are used to judge simulation results. As visualized in Figure 6, we used micro-benchmarks to determine the delays involved by different types of cache misses, described in the next paragraph. Timing information, read from profile data can not be used on the FPGA directly, but have to be translated as follows. This work basically models cache coherence communication instead of message passing applications, thus we translate tasks, sharing cache lines, to discrete memory events by splitting them up (*task splitting*). While L1 cache misses do not stress the bus if the cache line can be provided by the L2 cache, missing cache lines in L2 cache can either be loaded from a remote cache (*remote-fill*) or RAM (*mem-fill*). Furthermore, prefetched but not used data as well as modified cache lines evicted and thus written back to main memory cause additional traffic on the bus. Since performance counters for *mem-* and *remote-fills* are namely available, but return no meaningful values on the Xeon Phi, we additionally used algorithmic knowledge and call-graph information to determine whether a L2 cache miss results in a *remote-* or *mem-fill*.

Memory delay benchmark: The first benchmark periodically reads a data-array of varying size and measures the access-time. Figure 7 shows the average access-time per cache-line (64 Bytes). The read operation was repeated multiple times to get an average value and to lessen the influence of control overhead. The characteristic shape is defined by the particular cache sizes and their access delays. Every time the array-size surpasses a cache level the average access-time rises and indicates the delay of the next cache level. At 280 kByte the L2 cache starts to perform loads from

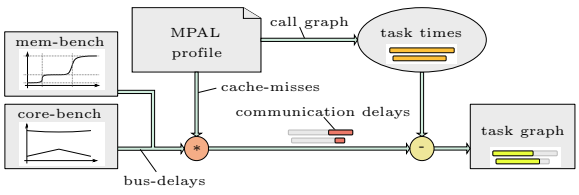


Figure 6: Visualization of task map creation.

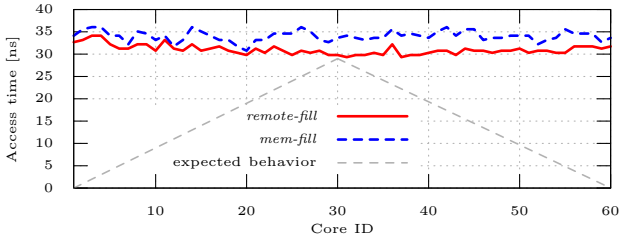
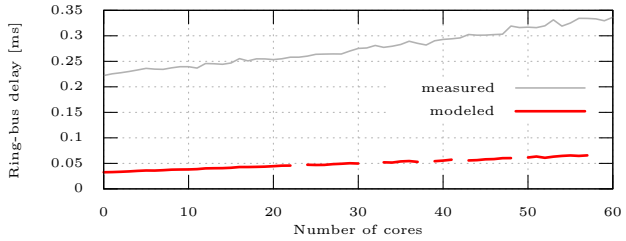
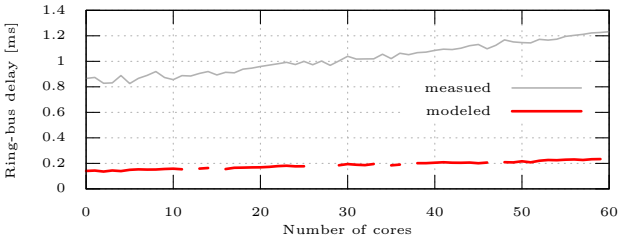


Figure 8: Core-to-core delays.



(a) Algorithmic stage *Scale* (Sc).



(b) Algorithmic stage *Histogram calculation* (CH).

Figure 9: Extracted and simulated communication delays of exemplary stages.

the main memory (visible in the performance counter) and therefore the access-time rises. After the array-size exceeded 1 MB the access time saturates at 32 ns. The difference between the plateau for the L2 cache and the saturation for sequential data access from the main memory is 13 ns. This additional delay is caused by the ring, DTDs, and memory accesses and characterizes the platform's communication delay according to ② in Figure 3.

Core communication benchmark: In the next benchmark, one core pre-loads a specific data-content which is then read again from a different core triggering remote-cache communication corresponding to ① in Figure 3. The array is sized to fit inside the core's private L2 cache to ensure that a remote-load is executed. During the Benchmark's execution, all cores subsequently read the data from the first core's private L2 cache, resulting in an increasing distance (number of hops) individually using the shortest path on the bidirectional ring bus. These measurements where repeated and the average access-time per cache-line and a corresponding regression curve are depicted in Figure 8. Normally, a ring-bus would produce an increasing delay with increasing distance and therefore form a triangle shape also shown in Figure 8. One aspect of this benchmark reveals that the employed hashing of the address space of the Xeon Phi *hides* the distances along the ring. In order to compare the results, each core reads the same data-array from memory (no preload). On the one hand, Figure 8 shows, that the core-distance is not influencing the access-times. On the other hand, a remote-load is only marginally faster (2 % - 12 %) than a mem-load. In both cases, the access-times correspond to the first benchmark in saturation which verifies the communication delay for a cache fill to be 13 ns.

5. EVALUATION

In this section, we evaluate simulated communication delays with extracted communication times using benchmarks and performance counters. The NoC model described above was mapped on a BEEcube BEE4 prototyping platform equipped with a Xilinx Virtex-6 XC6VLX550T FPGA. The FNOCEE design required 89 % of the available slices and 77 % of the available RAM blocks, as shown in Table 1. The complete NoC comprising 70 VCs archived a clock frequency of 66 MHz, while the actual NoC simulation can be executed within only few ms. While this configuration uses the maximum amount of memory as task-graph storage, a further memory enlargements caused mapping errors.

Table 1: NoC Simulator FPGA ressources.

Slices	LUT	Register	RAMB36	DSP48
76,882	237,860	148,297	486	3

In this case study, all algorithmic stages are processed subsequently. Thus, in order to reduce the size of task graphs stored on the FPGA, we simulated bus communications stage by stage, but certainly kept all data- and cache-dependencies from the original call graph. As depicted in Figure 9, communication times, extracted from profile data, appear much higher than simulated communication delays, while both curves however follow the same trend. On the one hand, in this simulation, we did not consider communications on the coherence- and acknowledge-bus, while the responsible DTDs' address associativity is hashed, which hides regular communication characteristics of ring-topologies. As our simulation of the data bus can not reproduce the measured communication delays, the present Xeon Phi architecture apparently suffers from complex coherence communication. On the other hand, while we imitated all relevant communications occurring and thus effectively stressing the data bus, we only used one image scale HOG-pipeline, exhibiting relatively long running tasks compared to the others. While this limitation reduces the number of parallel tasks, shared and requested data, this simulation can not reproduce a bottleneck situation.

In fact, the simulated HOG execution does not suffer from high communication costs compared to the total execution time. However, hashed and thus complex coherence communications to DTDs introduce long waiting times for cache misses. As not all scaled input images I_s of the original HOG implementation exhibit sufficient parallelism, a nested parallelization is essential, resulting in more complex task graphs and therefore higher communication rates on the bus. Due to limited memory capacities on the used FPGA, these complex task graphs can however not be simulated yet.

Furthermore, we compared the modeled HOG behavior on the ring topology against a mesh topology each with circuit switching and packet switching as network communication strategies. Due to the low impact of the communication delays, the topology was not impacting the overall runtime. Only the network using circuit switching involved longer, but still negligible, waiting times for cache fills caused by collisions of communication routes on the network resources. Due to the relatively low communication rates of the used HOG model (only one scaled input image) without full parallelism, the mesh topology could only achieve a maximum speedup of 3 % in comparison to the ring topology.

6. CONCLUSION

Enabling the simulation of complex and thus most realistic NoCs, in this work, we proposed to use FPGA accelerated NoC simulators. We presented extensions on the FNOCEE framework in order to simulate cache coherence communications of shared memory multicore architectures. Thereby, translating real measured profiles to task graphs for simulation, enables a fast simulation and thus in-depth analysis of real-world applications and identification of potential bottlenecks. As an exemplary case study, we presented the simulation of the Intel Xeon Phi ring bus topology with a parallel HOG pedestrian detection.

As the Xeon Phi uses DTDs with hashed associativities along a separate coherence-control bus, the expected characteristic of communication delays along a ring bus is hidden. Firstly, programmers can therefore not consciously make use of data locality. And secondly, measured and simulated communication delays show divergent magnitudes. However, as this simulation takes all bus stressing components into account, but the simulation denotes no bottleneck on the data bus, the simulated case study obviously not suffers from a communication bottleneck. In contrast, the Xeon Phi itself suffers from complex and – from user perspective – closed coherence protocol resulting in long communication delays.

The current FPGA implementation of the simulator does not offer enough memory to pass the entire task graph of the HOG algorithm onto the FPGA. However, even if the presented use case doesn't reflect all aspects of the parallel HOG execution behavior on the Intel Xeon Phi, the proposed method and exemplary simulation can illuminate previously hidden aspects of communication behavior. Further improvements of this framework will make use of external memory to simulate task graphs of any complexity. Moreover, this approach not only helps inspecting software implementations, but can also explore different NoC topologies.

7. REFERENCES

[1] D. Molka, D. Hackenberg, R. Schne, and W. E. Nagel. Cache Coherence Protocol and Memory Performance of the Intel Haswell-EP Architecture. In *Intl. Conf. Parallel Processing*, pages 739–748. IEEE, 2015.

[2] GR740: The ESA Next Generation Microprocessor (NGMP). `http://microelectronics.esa.int/ngmp`, 2017.

[3] W. J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Design Automation Conf.*, pages 684–689, 2001.

[4] A. Abbas, M. Ali, A. Fayyaz, A. Ghosh, A. Kalra, S. U. Khan, M. Usman S. Khan, T. De Menezes, S. Pattanayak, A. Sanyal, and S. Usman. A survey on energy-efficient methodologies and architectures of network-on-chip. *Computers and Electrical Engineering*, pages 333 – 347, 2014.

[5] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In *Intl. Conf. Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 886–893. IEEE, 2005.

[6] R. Membarth, F. Hannig, J. Teich, M. Krner, and W. Eckert. Comparison of Parallelization Frameworks for Shared Memory Multi-Core Architectures. In *Proc. Embedded World Conference, Nuremberg, Germany*. IEEE, 2010.

[7] M. C. Neuenhahn, J. Schleifer, H. Blume, and T. G. Noll. Quantitative comparison of performance analysis techniques for modular and generic network-on-chip. *Adv. Radio Science*, 7(C. 4):107–112, 2009.

[8] N. Genko, D. Atienza, G. De Micheli, J. M. Mendias, R. Hermida, and F. Catthoor. A complete network-on-chip emulation framework. In *Design, Automation and Test in Europe*, pages 246–251 Vol. 1, March 2005.

[9] M. Eggenberger and M. Radetzki. Scalable parallel simulation of networks on chip. In *2013 Seventh IEEE/ACM International Symposium on Networks-on-Chip (NoCS)*, pages 1–8, April 2013.

[10] A. Y. Weldezion, M. Grange, A. Jantsch, H. Tenhunen, and D. Pamunuwa. Zero-load predictive model for performance analysis in deflection routing NoCs. *Microprocessors and Microsystems*, 39(8):634–647, 2015.

[11] E. Fischer, A. Fehske, and G. P. Fettweis. A Flexible Analytic Model for the Design Space Exploration of Many-Core Network-on-Chips Based on Queueing Theory. In *Intl. Conf. Advances in System Simulation, ser. SIMUL*, 2012.

[12] D. Pfefferkorn, A. Schmider, G. Payá-Vayá, M. Neuenhahn, and H. Blume. FNOCEE: A Framework for NoC Evaluation by FPGA-based Emulation. In *Intl. Conf. Embedded Computer Systems (SAMOS)*, pages 86–95, 2015.

[13] S. Chai, Y. Li, J. Wang, and C. Wu. A List Simulated Annealing Algorithm for Task Scheduling on Network-on-Chip. *JCP*, 9(1):176–182, 2014.

[14] E. Salminen, T. Kangas, J. Riihimaki, and T. D. Hamalainen. Requirements for Network-on-Nhip Benchmarking. In *NORCHIP*, pages 82–85, 2005.

[15] J. Xu, W. Wolf, J. Henkel, and S. Chakradhar. A Methodology for Design, Modeling, and Analysis of Networks-on-Chip. In *Intl. Symp. Circuits and Systems*, pages 1778–1781 Vol. 2. IEEE, 2005.

[16] O. J. Arndt, D. Becker, F. Giesemann, G. Pay-Vay, C. Bartels, and H. Blume. Performance Evaluation of the Intel Xeon Phi Manycore Architecture Using Parallel Video-Based Driver Assistance Algorithms. In *Intl. Conf. Embedded Computer Systems (SAMOS XIV)*, pages 125–132. IEEE, 2014.

[17] O. J. Arndt, T. Lefherz, and H. Blume. Abstracting Parallel Programming and Its Analysis Towards Framework Independent Development. In *Intl. Symp. Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pages 96–103. IEEE, 2015.

[18] Intel Press Kit - Intel Xeon Phi Coprocessor 5110P/3000 Series. https://newsroom.intel.com/press-kits/intel-xeon-phi-coprocessor-5110p3000-series, 2012.