International Journal of Scientific Research in Computer Science, Engineering and Information Technology

ISSN : 2456-3307 OPEN ACCESS

Available Online at : www.ijsrcseit.com

doi : https://doi.org/10.32628/CSEIT2390655

Automatic Generation of Network Function Accelerators Using Component-Based Synthesis

Sai Teja Nagapuri, Rajasree Tella, Meghana Reddy Guntireddygari

New Jersey Institute of Technology, Newark, New Jersey, USA

ARTICLEINFO

Article History:

Accepted: 10 Dec 2023

Published: 26 Dec 2023

Publication Issue

Volume 9, Issue 6

November-December-2023

Page Number

297-302

ABSTRACT

The document discusses the development of a compiler called SyNAPSE, which aims to address the challenge of programming for multiple hardware targets in networked systems. It focuses on optimising the performance, efficiency, and resource consumption of networked systems by dividing packet processing across multiple hardware platforms. The key problem it is solving is the difficulty in developing high-performance network functions for multiple platforms, each with its own programming language and hardware features. SyNAPSE aims to provide a solution that allows for 'write once, run anywhere' code that is portable across different platforms and automatically provisioned on the hardware best-suited for the task. It explores a large search space of different mappings of functionality to hardware, allowing for optimization based on programmer-specified objectives such as minimising memory consumption or maximising network throughput.

Keywords: Synapse, Network Function, Component-Based Synthesis, Tofino-Based Switch.

I. INTRODUCTION

The networking community is trying to make networks better by using special hardware and software that can be customized and upgraded. They are debating which type of hardware is the best for this, such as special switches, network processing units, programmable chips, or regular computer software. Each type of hardware has its own strengths. Some are good at certain tasks, like matching information in packets, while others are better at analyzing the content of the packets. Some people think that using a combination of different types of hardware is the best solution because it can make the network faster, more efficient, and cheaper. For example, some systems divide the work between different types of hardware to make it more efficient, like using a special chip to search for specific information and a regular computer to analyze the rest of the data.

Hybrid designs are very difficult to create because each platform they use has its own programming language, hardware features, and debugging challenges. It's

already hard to create high- performance designs for just one platform, so asking developers to learn how to create them for multiple platforms is a big challenge. What developers really need is the ability to write code that can work on any platform and automatically adapt to the best hardware for the job. Some new tools have shown that it is possible to write code that can work on multiple platforms.

The problem of developing high-performance network functions for multiple hardware platforms is important due to the increasing adoption of programmable data planes in networking. As the networking community embraces programmable data planes to implement network functions, the debate over which programmable hardware is best suited for these capabilities remains active. The challenge lies in the diverse strengths of different platforms and the difficulty in developing high- performance network functions for each. This problem is crucial as it hinders the seamless deployment of network functions across heterogeneous hardware platforms, impacting performance, efficiency, and resource utilisation. Addressing this challenge is essential for achieving upgradeability, customizability, and innovation in networked systems.

II. BACKGROUND

2.1 Introduction to Synapse:

SyNAPSE Approach: The prototype compiler, SyNAPSE, employs a methodology known as component-based synthesis. It recognizes that expert developers invest significant effort in tuning common algorithms and data structures.SyNAPSE provides a library of abstract data types and algorithms, referred to as "components," which can be implemented in various ways across different hardware platforms. This allows developers to choose implementations that suit their goals without needing to delve into the internal details of each component.

2.1.1 Abstracting NF Development with SyNAPSE:

Simplified Development for NF Developers:

SyNAPSE simplifies the development process for Network Function (NF) developers. They only need to focus on the control flow and networking logic of their application, programming against the abstract SyNAPSE component APIs. The compiler takes care of selecting the appropriate implementations based on the specified hardware platform and performance objectives.This separation of concerns enables the use of advanced symbolic execution techniques during compilation, leading to optimized deployments based on hardware capabilities and specified goals.

2.1.2. Working of SyNAPSE

Automatic Generation of Network Function Accelerators Using Component-Based Synthesis :

SyNAPSE is a prototype compiler designed to automatically generate network function accelerators using a component-based synthesis methodology. It explores a large search space of different mappings of functionality to hardware, allowing programmers to specify performance objectives such as minimising memory consumption or maximising network throughput. The SyNAPSE prototype supports deployments across x86 and Tofino platforms, uncovering thousands of deployment options. By using components as the key abstraction for space exploration, SyNAPSE constrains the search space while considering trade-offs and performance predictions for different tasks on various hardware platforms. This approach enables the automatic provisioning of code paths on the hardware best-suited for specific tasks, advancing the state of the art in network function accelerator generation.

2.2 : KEY WORDS:

In-network compute, Network function virtualization, Programming abstraction

III. Previous State of Art

The previous state of the art in network function deployment involved the use of programmable data planes to implement network functions (NFs), such as deep packet inspection, filtering, and WAN optimization. The debate centred around which programmable hardware platform was best suited for these capabilities, with options including Protocol Independent Switch Architecture (PISA) switches, Network Processing Units, FPGAs, and x86 software. Hybrid dataplane designs, which use an ensemble of platforms, were found to offer higher throughput, lower latency, better energy efficiency, and lower cost than single-platform approaches. However, developing high- performance NFs for multiple platforms was challenging, and there was a need for a "write once, run anywhere" approach to code deployment.

IV. METHODOLOGY

To improve the methodology, the following steps can be considered:

1. Refinement of Heuristics: The document mentions that the work on developing heuristics is still at an early stage. Further research and development can focus on refining and expanding the set of heuristics used by SyNAPSE. This could involve exploring more elaborate heuristics to outperform the simple ones that have been tried so far.
2. Performance Prediction Model: Incorporating a more advanced performance prediction model into the heuristics can enhance the decision-making process during the search for optimal deployment options. This model should account for predicted performance of different tasks on different hardware platforms and predict the transition cost of transferring packets from one platform to another.
3. Support for Additional Platforms: The document mentions the support for deployments across x86 and Tofino platforms. Future work can focus on expanding the platforms supported by SyNAPSE, such as FPGAs, Network Processing Units (NPUs), or Infrastructure/Data Processing Units (IPUs/DPUs).
4. Automated Search Process: Developing an automated search process that can efficiently explore the space of possible deployments and select the best solution based on specific performance objectives. This could involve the use of machine learning algorithms or optimization techniques to guide the exploration of the deployment options.
5. Integration of Real-world Workloads: Incorporating real-world workloads and traffic patterns into the evaluation process can provide more realistic insights into the performance of different deployment options. By focusing on these improvements, the methodology of SyNAPSE can be enhanced to better address the challenges of programming for multiple hardware targets and optimise network function performance and resource consumption.
6. Advancements in Network Function Accelerators

Using Component-Based Synthesis:
The paper significantly advances the state of the art by introducing SyNAPSE, a prototype compiler that explores a wide range of mappings of functionality to hardware platforms to optimise network function (NF) deployment based on performance objectives. This approach addresses the challenge of programming for multiple hardware targets by enabling a "write once, run anywhere" capability, allowing code to be portable across different platforms and automatically provisioned on the hardware best-suited for specific tasks. SyNAPSE's component-based synthesis methodology supports deployments across x86 and Tofino platforms, uncovering thousands of deployment options and demonstrating substantial improvements, such as reducing controller traffic by an order of magnitude and halving memory usage. These advancements represent a significant leap forward in the development of network function accelerators and their deployment optimization.

7. Key insights in the Design:

The key insights in the design of SyNAPSE that enabled it to advance the state of the art include the use of component-based synthesis, which allows for the exploration of a large search space of different mappings of functionality to hardware. This approach enables the compiler to consider a wide range of implementation options to tune network functions (NFs) to meet specific performance objectives, such as minimising memory consumption or maximising network throughput. Additionally, SyNAPSE's ability to explore different deployment options and consider trade- offs sets it apart from rule-based translation approaches, allowing for more flexible and optimised solutions. The incorporation of heuristics and performance prediction models further enhances its capabilities, making it a significant advancement in the field of network function accelerators..

V. CONCLUSION

The design is evaluated through experiments to understand the impact of different performance targets on the systems generated by SyNAPSE. The prototype of SyNAPSE is implemented and run using a running example of a NAT, and the performance of different solutions is evaluated by examining the fraction of packets sent to the controller, CPU load on the controller, and switch resource utilisation. The evaluation involves exploring the search space to find distinct solutions targeting different performance objectives, such as CPU load, resource utilisation, and throughput. The key results include the identification of multiple valid solutions with different trade-offs, demonstrating the flexibility and optimization capabilities of SyNAPSE in generating network function accelerators. Additionally, the evaluation highlights the potential of SyNAPSE to reduce controller traffic and memory usage, showcasing its effectiveness in optimising NF deployment for various performance objectives.

VI. REFERENCES

[1]. Theophilus Benson, Aditya Akella, and David A Maltz. 2010. Network traffic characteristics of data centers in the wild. In Proceedings of the 10th ACM SIGCOMM conference on Internet measurement. 267–280.

[2]. Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming ProtocolIndependent Packet Processors. SIGCOMM Comput. Commun. Rev. (jul 2014).

[3]. Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN. In Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM (SIGCOMM '13).

[4]. Cristian Cadar, Daniel Dunbar, and Dawson R. Engler. 2008. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. In 8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, December 8-10, 2008, San Diego, California, USA, Proceedings, Richard Draves and Robbert van Renesse (Eds.). USENIX Association, 209–224.
http://www.usenix.org/events/osdi08/tech/full_p ape rs/cadar/cadar.pdf

[5]. Jian Chen, Xiaoyu Zhang, Tao Wang, Ying Zhang, Tao Chen, Jiajun Chen, Mingxu Xie, and Qiang Liu. 2022. Fidas: Fortifying the Cloud via Comprehensive FPGA-Based Offloading for Intrusion Detection: Industrial Product. In Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA '22).
[6] Xiang Chen, Hongyan Liu, Dong Zhang, Zili Meng, Qun Huang, Haifeng Zhou, Chunming Wu, Xuan Liu, and Qiang Yang. 2022. Automatic performance-optimal offloading of network

functions on programmable switches. IEEE Transactions on Cloud Computing (2022).

[6]. Sharad Chole, Andy Fingerhut, Sha Ma, Anirudh Sivaraman, Shay Vargaftik, Alon Berger, Gal Mendelson, Mohammad Alizadeh, Shang-Tse Chuang, Isaac Keslassy, Ariel Orda, and Tom Edsall. 2017. DRMT: Disaggregated Programmable Switching. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17).

[7]. Intel Corporation. 2022. Intel Tofino 3. https://www.intel.com/content/www/us/en/products/network- io/programmable-ethernetswitch.html

[8]. ETSI. 2012. Network Functions Virtualisation - White Paper. http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[9]. Jiaqi Gao, Ennan Zhai, Hongqiang Harry Liu, Rui Miao, Yu Zhou, Bingchuan Tian, Chen Sun, Dennis Cai, Ming Zhang, and Minlan Yu. 2020. Lyra: A cross-platform language and compiler for data plane programming on heterogeneous asics. In Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication. 435–450.

[10]. Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. 2018. Sonata: Query-driven streaming network telemetry. In Proceedings of the 2018 conference of the ACM special interest group on data communication. 357–371.

[11]. Rishabh Iyer, Luis Pedrosa, Arseniy Zaostrovnykh, Solal Pirelli, Katerina Argyraki, and George Candea. 2019. Performance Contracts for Software Network Functions. In 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19). USENIX Association, Boston, MA,517530.https://www.usenix.org/conference/nsd i19/presentation/iyer

[12]. Susmit Jha, Sumit Gulwani, Sanjit A. Seshia, and Ashish Tiwari. 2010. Oracle-Guided Component-Based Program Synthesis. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1 (Cape Town, South Africa) (ICSE '10). Association for Computing Machinery, New York, NY, USA, 215–224. https://doi.org/10.1145/1806799.1806833

[13]. Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. 2000. The Click modular router. ACM Transactions on Computer Systems (TOCS) 18, 3 (2000), 263–297.

[14]. Hongqiang Harry Liu, Xin Wu, Ming Zhang, Lihua Yuan, Roger Wattenhofer, and David Maltz. 2013. ZUpdate: Updating Data Center Networks with Zero Loss. SIGCOMM Comput. Commun. Rev. 43, 4 (aug 2013), 411–422. https://doi.org/10.1145/2534169.2486005

[15]. Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. 2016. One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon. In Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16).

[16]. Jedidiah McClurg, Hossein Hojjat, Pavol Černý, and Nate Foster. 2015. Efficient Synthesis of Network Updates. SIGPLAN Not. 50, 6 (jun 2015), 196–207. https://doi.org/10.1145/2813885.2737980

[17]. Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. 2017. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication. 15–28.

[18]. Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. 2017. Language-directed hardware design for network performance monitoring. In Proceedings of the Conference of

the ACM Special Interest Group on Data Communication. 85–98.

[19]. Netronome. 2022. Agilio LX SmartNICs. https://www.netronome. com/products/agilio-lx/

[20]. R. Pagh and F. F. Rodler. 2004. Cuckoo hashing. Journal of Algorithms (feb 2004). Issue 51. [22] Shoumik Palkar, Chang Lan, Sangjin Han, Keon Jang, Aurojit Panda, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. 2015. E2: A Framework for NFV Applications. In Proceedings of the 25th Symposium on Operating Systems Principles (SOSP '15).

[21]. Yiming Qiu, Jiarong Xing, Kuo-Feng Hsu, Qiao Kang, Ming Liu, Srinivas Narayana, and Ang Chen. 2021. Automated SmartNIC Offloading Insights for Network Functions. In Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles. 772–787.

[22]. Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K. Reiter, and Guangyu Shi. 2012. Design and Implementation of a Consolidated Middlebox Architecture. In 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12).

[23]. Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. 2012. Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service. In Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '12).

[24]. Armando Solar-Lezama, Rodric Rabbah, Rastislav Bodík, and Kemal Ebcioğlu. 2005. Programming by Sketching for Bit-Streaming Programs. In Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (Chicago, IL, USA) (PLDI '05). Association for Computing Machinery, New York, NY, USA, 281–294. https://doi.org/10.1145/1065010.1065045

[25]. Nik Sultana, John Sonchack, Hans Giesen, Isaac Pedisich, Zhaoyang Han, Nishanth Shyamkumar, Shivani Burad, André DeHon, and Boon Thau Loo. 2021. Flightplan: Dataplane disaggregation and placement for p4 programs. In 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21). 571–592. [28] Balajee Vamanan, Jahangir Hasan, and T.N. Vijaykumar. 2012. Deadline-Aware Datacenter Tcp (D2TCP). In Proceedings of the ACM SIGCOMM

[26]. 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (Helsinki, Finland) (SIGCOMM '12). Association for Computing Machinery, New York, NY

Cite this article as :

Sai Teja Nagapuri, Rajasree Tella, Meghana Reddy Guntireddygari, "Automatic Generation of Network Function Accelerators Using Component-Based Synthesis", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 9, Issue 6, pp.297-302, November-December-2023. Available at doi :
https://doi.org/10.32628/CSEIT2390655
Journal URL : https://ijsrcseit.com/CSEIT2390655