Investigating the Applicability of In-Network Computing to Industrial Scenarios

Ike Kunze*, René Glebke*, Jan Scheiper*, Matthias Bodenbenner†, Robert H. Schmitt†, Klaus Wehrle*
*Communication and Distributed Systems · {kunze, glebke, scheiper, wehrle}@comsys.rwth-aachen.de
†Laboratory for Machine Tools and Production Engineering (WZL) · {m.bodenbenner, r.schmitt}@wzl.rwth-aachen.de
All authors are affiliated with *RWTH Aachen University*, Aachen, Germany

***Abstract*—Moving computation functionality into the network and onto networking devices has recently shown great promise for performance improvements, e.g., by reducing path lengths and processing latencies, or by increasing bandwidth efficiency. Yet, In-Network Computing (INC) is challenged by the limitations of networking hardware which is generally not designed for complex calculations. Thus, while possible, developing INC approaches is a constant struggle between desired and available functionality.**

In this paper, we demonstrate the applicability of INC to industrial contexts by offloading a seemingly simple task from an industrial assembly scenario – coordinate transformations – to programmable switches and SmartNICs. We find that even such a task puts heavy demands on the devices, but that the right dose of approximation and diligent problem reformulation can enable the necessary operations on all platforms, thus setting the stage for improving latencies by significantly reducing communication paths. Yet, our results further indicate that these gains in latency come in hand with a trade-off regarding the achievable accuracy.

***Index Terms*—in-network computing, latency, approximation**

I. Introduction

The recent advent of programmable networking devices (PNDs) has reignited the interest in moving computations into the network. Ranging from key-value stores [1] and data aggregation [2] to network telemetry [3], many studies have already demonstrated the potential of this new generation of *In-Network Computing (INC)*, largely capitalizing on the privileged positions of PNDs within the network.

Conceptually building upon the generic pipeline structure described by Bosshart et al. [4], the approaches are implemented using a variety of platforms with widely differing capabilities and restrictions. On the one hand, ASIC-based switches map the pipeline structure to specialized hardware, enabling them to process several terabits per second, but at the same time restricting them to such functionality that can be executed predictably and fast [5]. On the other hand, CPU-based software implementations allow great computational flexibility but cannot provide the strict performance guarantees and extremely high data rates of dedicated hardware. Hybrid platforms, which express pipelines as programs on specialized but still computationally restricted network processing units (NPUs), aim to provide a middle ground between high achievable speed and computational flexibility. The performance of INC approaches thus heavily depends on their suitability to the available computation platforms: approaches naturally mapping to traditional networking hardware can fully use the high throughput of ASICs, while more demanding applications may require higher flexibility at the cost of slower performance.

Differences in performance especially impact applications with strict requirements, such as latency-critical industrial scenarios. For these, INC has already been shown to be beneficial through a shortening of communication paths and thus a reduction of (horizontal) communication latency [6]. Yet, little work investigates the tangible impact of (a) the inherent trade-off regarding the hardware platforms and (b) the design choices needed to implement the INC functionality.

In this work, we thus analyze inherent performance trade-offs between different INC platforms and explore different ways to implement a critical task from a real-world industrial assembly scenario. Specifically, we contribute the following:

- We show the efficacy of INC for latency-critical industrial environments using the example of an *in-network coordinate transformation* task.
- We explore methods harnessing the capabilities of different platforms (ASIC, SmartNIC, Linux PC) and languages ($P4_{16}$, eBPF, C) to enable required, yet absent mathematical functionality (multiplications, trigonometric functions).
- We evaluate the implications of our design choices in a testbed study by analyzing the accuracy, speed, and throughput of our conceived variants whose code is available at [7].

Structure. Sec. II introduces the peculiarities of our INC platforms before Sec. III presents the specific requirements of industrial INC and our real-world use case. Sec. IV highlights challenges and trade-offs when implementing a coordinate transformation on our platforms, and we evaluate the performance of our approaches in Sec. V. We discuss related work in Sec. VI before concluding in Sec. VII.

II. Computing Resources in the Network

INC functionality can be placed on various platforms at different positions in or close to the network, entailing inherent performance implications. The two most prominent variants are PNDs, commonly programmed using P4, and offloading features offered by some modern operating system kernels. In the following, we briefly present the available methods.

PISA. The protocol-independent switch architecture (PISA) and the corresponding P4 programming language [4] enable a flexible programmability of network devices. PISA maps to the requirements of forwarding-related packet processing and

 DOI: 10.1109/ICPS49255.2021.9468247.

consists of the following main components: (a) programmable parsers/deparsers responsible for (de-)serializing packets at the start/end of a pipeline; (b) a fixed number of match-action unit (MAU) stages in between, which allow for table lookups and packet modifications; and (c) a programmable bus carrying information through the pipeline. The actual functional scope, e.g., regarding possible operations in the MAUs, and the respective performance depend on the concrete PISA platform.

PISA in Hardware. ASIC-powered switches, such as the Intel Tofino, map PISA's pipeline directly onto hardware, offering the significant advantage of line-rate processing capabilities. They are programmable using P4, but physical resources are limited, and operations are conceptually restricted to those that can be performed fast and with predictable delay [5]. For example, multiplications can only be performed when one factor is statically defined, and more complex functionality not needed for general networking tasks, e.g., the evaluation of trigonometric functions, is not natively supported.

PISA in Software. More flexible PNDs build upon the P4 behavioral model, a software implementation of PISA targeting processor-based systems. It alleviates some of the restrictions—for example, multiplications of two variables are usually possible—but comes at the cost of lower and less predictable performance. Additionally, for some PNDs, such as the Netronome SmartNIC, P4 is internally compiled into a restricted C dialect which can be directly used for programming. Some devices also allow calling C functions from within P4 via externs. This enables constructs such as loops, and lifts the constraints of the pipeline-based approach of P4 even further.

In-Kernel Processing. On Linux machines, small programs can be placed into the kernel using the extended Berkeley Packet Filter (eBPF), a sandboxed, lightweight VM first introduced for BSD. Using a RISC-like instruction set, eBPF supports computations on packets without passing them to user space applications, but can only access a subset of kernel functionality, rendering, e.g., mathematical libraries unavailable.

While the functional scope of INC methods is limited and approaches are thus challenging to implement, their privileged positioning in the network provides opportunities for industrial settings, which we next demonstrate with a concrete use case.

III. Industrial In-Network Computing

The industrial sector is becoming increasingly interconnected, commonly captured by the term Industrial Internet of Things (IIoT). Ideas such as the Internet of Production (IoP) [8] provide holistic concepts going beyond interconnecting the different components, as they also focus on how gathered data can be leveraged and how the systems can be operated. The IoP is further characterized by diverse application fields, ranging from processes requiring extremely low latencies [6] to approaches needing high bandwidths to transmit all sensor data [8]. In this context, INC provides a novel solution space: the volume of sensor data can already be lowered *in the network*, and latencies can be reduced by giving earlier responses *from within the network*. In this paper, we focus on the latter setting, i.e., speeding up processes by shortening communication paths. We showcase the benefits of INC functionality for the IIoT by mapping the requirements of an example from industrial assembly onto PNDs.

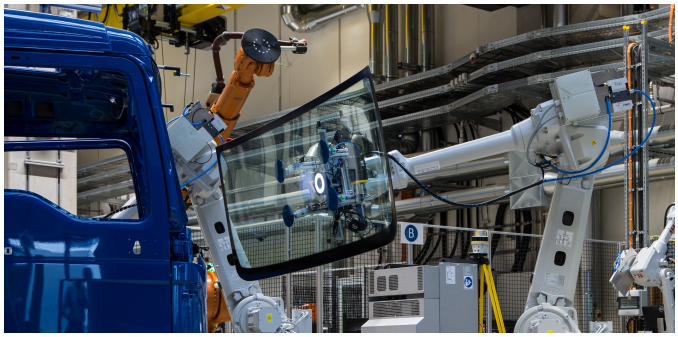

Fig. 1. Line-less Mobile Assembly Systems (LMASs) require sharing precise position measurements of machinery and objects in real-time [9]. We aim to harmonize readings from multiple sources through In-Network Computing.

A. Setting and Requirements: Industrial Assembly

Setting. Industrial assembly is an essential part of production and describes processes that permanently merge bodies and parts. Novel assembly paradigms such as *Line-less Mobile Assembly Systems (LMASs)* [10] replace conventional assembly lines by fully mobilized products and resources and greatly benefit from a backbone of metrological coordinate measurements for accurate positioning and collaboration of the mobile entities. For example, when mounting a windscreen to a moving truck cabin using a robot, as illustrated in Fig. 1, the positions of the cabin and the robot's end-effector are tracked to reduce misalignments during the mounting process.

One approach to meet the requirements of all subtasks in an LMAS is to combine multiple large-scale metrology systems with varying capabilities [11]. For this, the coordinate measurement systems have to be joined to a reference frame, i.e., local coordinate measurements are first transformed into a global coordinate system and then into the respective local coordinate systems of the controlled robots [9].

Requirements. In general, all measurement-related operations must be conducted in real-time to meet usual control loop cycle times of industrial robots in the *single-digit millisecond* range ($10^{-3}\,\mathrm{s}$) [12]. For the illustrated mounting of the windscreen this includes the measurement itself, the coordinate transformations, and the intermediate communication. Additionally, some manufacturing processes, such as drilling, require maximum uncertainties in the order of *tens of micrometers* ($10^{-5}\,\mathrm{m}$) [13], so that the coordinate transformations also have to be performed with *high accuracy*.

Such demanding requirements are characteristic of the industrial domain. Ideally, a supporting coordinate transformation framework should be able to satisfy both requirements, i.e., speed and accuracy. As the main benefit of INC is speeding up communication, we especially investigate its applicability to this task subject to the accuracy requirements. Before we describe our system in detail in Sec. IV, we first define the concrete operations that it has to perform.

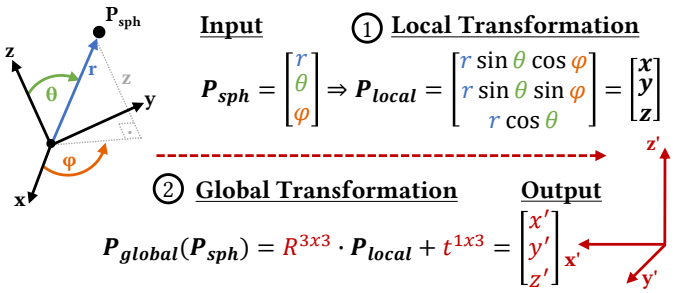

Fig. 2. Network devices in our scenario first transform spherical coordinates from sensors $(r, \theta, \phi$; left) into a local Cartesian representation (x, y, z). An affine transformation then yields a global representation $(x', y', z'$; bottom).

B. Problem Outline: Coordinate Transformation

In our LMAS, the global reference frame uses a standard 3-dimensional Cartesian system, but parts of our equipment produce readings in *spherical* coordinates. Our aim, as visualized in Fig. 2, is to use INC to first transform the spherical readings (r, θ, ϕ) to a local Cartesian representation (x, y, z) (Step ①). This *local transformation* entails five multiplications and five evaluations of the trigonometric functions sine and cosine. We next transform the local representation into a global Cartesian system and then to the local system of the receiving robot (Step ②). These two affine transformations each require one vector-matrix multiplication and one subsequent translation. For simplicity, we combine these transformations into a single *"global" transformation* entailing nine distinct multiplications and additions. Overall, we perform one spherical to Cartesian transformation followed by one affine Cartesian transformation. Their combined complexity already puts heavy demands on some PNDs, as we elaborate in the following.

IV. In-Network Coordinate Transformation

The tailoring of PNDs and eBPF towards high-throughput packet processing introduces several restrictions that we have to overcome to implement our coordinate transformation. Most notably, our PNDs only offer integer arithmetic. To support calculations on all possible scales of readings in our use case, we employ a *restricted fixed-point (FP)* arithmetic on top of the available integer representations. Since mathematical libraries are unavailable on our platforms, we additionally have to emulate the sine and cosine functions using our FP arithmetic. For this, we devise two variants: (i) pre-computed values in lookup tables, which is a natural fit to the architecture of PNDs, and (ii) an approximation variant based on Chebyshev polynomials [14]. We next give the ideas behind our approaches before presenting platform-specific details.

A. Restricted Fixed-Point Arithmetic

All our platforms support integer calculations of at least 32 bits. We consequently store each FP number in variables of this common minimum width so that all platforms can support at least parts of the necessary operations *natively*. In our FP representation, $FP_d \in \{\pm[0 \dots 2^d].[0 \dots 2^{31-d}]\}$, the most significant bit stores the sign, the following d bits the decimal part, and the rest the fractional part, with d fixed.

Choice of fixed-point position. The fixed format of our FP representation necessarily induces inaccuracies in calculations, e.g., due to rounding [15]. We hence need to choose the fixed point such that the resulting decimal range is *sufficiently* large, while the number of bits for the fractional part is *maximized*. To support medium-sized shop floors with linear dimensions of up to 50 m, we choose $d = \lceil \log_2(50) \rceil = 6$ bits for the decimal part. The remaining $31-6 = 25$ bits provide sufficient space to support the needed minimal fractional accuracy of 10^{-5} m. Note that the required dimensions and accuracies are highly use-case-specific and we can adapt d to accommodate for changing requirements. For example, when less accuracy is needed we can support higher dimensions. We represent angles using the same format with $d = 6$ to keep a common data type, although a tighter representation is generally possible.

Required operations. For our coordinate transformation problem, we need to support addition and multiplication. Since d is fixed, the position of the fixed point stays at the same position during addition, so that we can simply apply the native addition routines. Likewise, for multiplication, eventual native routines suffice, but we need to right-shift the product by d bits to obtain a number in our FP format again. Bit shifting is a basic operation supported by all platforms.

B. Trigonometric Functions

To calculate the spherical transformation, we need to provide the trigonometric functions using our FP representation. Accounting for typical strengths of our PNDs, we first explore the possibility to pre-compute the required values and store them in match-action tables. Alternatively, we use the extended computational abilities of our CPU-based platforms and devise an approximation method that allows a fast, ad-hoc computation of the needed values.

Lookup tables. For our lookup approach, we generate tables with possible radian values of θ and φ, the angles in the spherical representation (see Fig. 2), as keys. The associated actions write the respective sine and cosine values into metadata fields to be used in the multiplications. A naïve table setup containing all 2^{32} possible key-value pairs would become prohibitively large. We thus use several techniques to reduce the table size, resulting in three concrete table layouts: (i) $Layout_L$, (ii) $Layout_M$, and (iii) $Layout_S$.

$Layout_L$. Leveraging the periodicity and symmetry of sine and cosine, we restrict the inputs to the range $[0, 2\pi]$, reducing the table size to 2^{28}. We then employ the *sum of angle identity*, i.e., $sin(x) = \sin a + b = \sin a \cdot \cos b + \cos a \cdot \sin b$, to *split* an *input x* into a *higher part a* and a *lower part b* as exemplified in Fig. 3. By placing the values for the first bits (blue) in one table (left) and the remaining bits (green) in a second table (right), we can essentially trade table size against two additional multiplications and an addition. By splitting the bit representation of the values exactly in half, we thus further reduce the needed space to two tables with 2^{14} entries each.

$Layout_M$/$Layout_S$. We additionally experiment with smaller table configurations. Making full use of the periodicity and symmetry of sine and cosine, we can restrict the inputs to

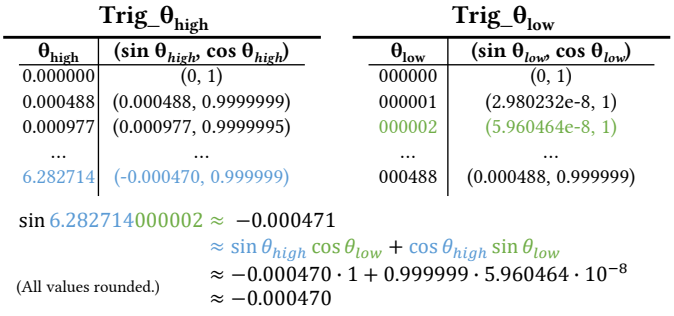

Fig. 3. The sum of angle identity allows us to save table space by performing split lookups on the high (blue, left) and low (green, right) bits of trigonometric functions.

the range $[0, \pi/2]$, yielding 2^{26} overall values and 2^{13} entries for each of the split tables (*Layout*$_M$). For *Layout*$_S$, we additionally sacrifice some accuracy and cut off the least significant 4 bits, resulting in 2^{22} overall values and 2^{11} entries in the split tables.

Approximation via polynomials. We survey several techniques for approximating the trigonometric functions on our CPU-based platforms (see, e.g., [16] for an overview of available options). Due to the absence of mathematical libraries, we focus on such techniques that minimally rely on functions and operations not natively available on our platforms. We choose a technique based on Chebyshev polynomials described by Wallace [14], which we adapt to our setting. For brevity reasons, we omit the derivation of our approximation formula; in essence, it yields a polynomial of degree 6, which requires a total of 9 multiplications and 7 additions to approximate the sine using our FP arithmetic. The domain of the function is $[-\pi, \pi]$, which suffices due to the periodicity of the sine; we use the *modulo* operation to restrict our inputs accordingly.

C. Platform-specific Details

The different architectures of our platforms do not allow us to use a shared codebase. We leverage this fact to tailor our solutions to the main peculiarities of the platforms.

Userspace. As a baseline variant, we implement a single-threaded Linux C program (***Userspace***) that busy-waits on a socket and performs a transformation once a packet arrives. To harness the higher calculation accuracy available, it takes input in our FP format but casts it into doubles and uses the trigonometric functions from the standard library.

eBPF. We implement four flavors for eBPF: a lookup variant for each of the presented table layouts (*Large*, *Medium*, and *Small*) and one variant for our *Chebyshev* approximation. After compiling to eBPF using the bcc compiler collection[1], we attach the resulting programs at two different points in the Linux stack: as a traffic control action[2] (***TC***) and in the *XDP* datapath in the device driver[3] (***XDP***).

Netronome SmartNIC. We implement the same four flavors for the SmartNICs (**N**). The lookup variants are purely *P4*-based, while the Chebyshev approximation version uses P4 for parsing only and performs calculations in *C*.

Since the multiplications may yield intermediate results larger than 32 bits, we split the factors using temporary variables and reassemble the resulting products at the ends of our calculations. (We employ a similar method in our Tofino variant below.) Note that this is only necessary for the *P4* versions; the *C* externs allow us to use 64-bit integers so that our two 32-bit inputs cannot cause overflows.

Intel Tofino. On Tofino, multiplications are only possible with at least one static factor. Our variant (***Tofino***) thus emulates the multiplication of two payload values, a and b, using a scheme that mimics the classic "long multiplication" method: We first initialize a product accumulator value $c = 0$. In each step i ($0 \leq i \leq 31$), we then calculate $a \ll i$, i.e., we left-shift a by i bits, right-filling with zeros. Then, if bit i of b equals 1, we add the result to the accumulator, i.e., $c = c + (a \ll i)$. After the last iteration, we have $c = a \cdot b$.

The parallelization capabilities of Tofino allow us to perform one such multiplication per pipeline pass. Reusing the result of $r \cdot \sin\theta$ (see Fig. 2) and accounting for our table splitting technique, the spherical transformation requires twelve multiplications. The *Chebyshev* approximation variant incurs even more multiplications but yields less precise values. We thus only implement the lookup-based variant. Since table sizes do not affect performance on Tofino, we only apply the most accurate variant, i.e., Layout$_L$.

Adding the Cartesian transformation, entailing nine multiplications, overstretches the pipeline resources. In our solution, we split the program onto two distinct pipelines which causes one additional pipeline pass for each multiplication to finish the preceding multiplication and prepare the subsequent one. Overall, our approach for transforming the original spherical format to the final Cartesian format requires 42 pipeline passes. We realize these using the internal recirculation engines, thus reducing the available throughput to 1/42.

We next investigate how our highly device-specific design decisions and the device limitations themselves actually impact the achievable performance.

V. Performance Evaluation

We experimentally evaluate our implementations on a local testbed consisting of two 32-port Intel Tofino switches and two 25G Netronome SmartNICs, as shown in Fig. 4, with userspace programs running on an Intel Core i5-4590 CPU.

In our general setup, the host of SmartNIC 1 creates 107 Byte UDP packets containing random values for the (r, θ, φ) tuple introduced in Fig. 2 and sends them to Switch 1 (Step ①). Switch 1 then forwards the packets to the test targets for transformation into the target Cartesian (x, y, z) coordinates, inserting a timestamp *T1* upon transmission (Step ②). On the targets, we take timestamps directly before (*T2*) and after (*T3*) the actual transformation (Step ③). We then return the packets to Switch 1, which inserts timestamp *T4* upon arrival and forwards them to the host of SmartNIC 1 for evaluation (Step ④). We evaluate the conformance of our

[1] https://github.com/iovisor/bcc
[2] https://www.kernel.org/doc/html/latest/networking/filter.html
[3] https://www.kernel.org/doc/html/latest/networking/af_xdp.html

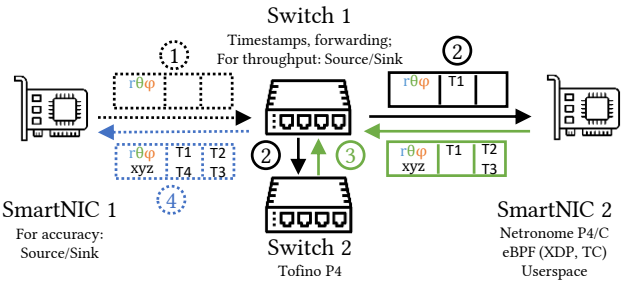

Fig. 4. Our testbed comprises two Intel Tofino switches and two Netronome SmartNICs. We timestamp on Switch 1 (T1, T4) and on our targets (T2, T3).

implementations to the requirements stated in Sec. III-A using the following metrics: (i) the *total transformation* time as seen by Switch 1 ($T4 - T1$); (ii) the raw *calculation* time on the platforms ($T3 - T2$); (iii) the *accuracy* of the transformations (based on (r, θ, φ) and (x, y, z)); and (iv) the *reliability* based on the sustainable throughput and the packet loss.

For investigating the *reliability*, we leave out Step ① and instead use the packet generation capabilities of Switch 1 to generate high numbers of slightly larger UDP test packets (111 Byte). We forward them to the targets for transformation and afterward return them to Switch 1 where we count the number of departing and arriving packets for certain intervals. This allows us to determine the throughput in packets per second as well as the number of lost packets.

A. Processing Times

To analyze the processing speeds of our implementations, we perform 15 000 coordinate transformations for each approach. Fig. 5 shows our results. The light bars denote the mean *total* time for a single transformation, including the respective mean *calculation* times represented by the darker bars, both shown with 99% confidence intervals.

High-level findings. The userspace and eBPF variants yield the fastest *calculation* times as they are executed on a CPU, but most of the *total* time is consumed by the communication overhead of handling the packets in the device driver and/or traversing the stack. Consequently, N_{Cheby} and *Tofino* have the fastest *total* times as their placement deep in the network incurs little additional processing overhead. This effect is also visible for the eBPF variants, which have slightly faster *total* times than the userspace. The lookup-based Netronome variants perform worst, with N_{Large} and N_{Medium} not being able to satisfy the required speed.

Caching and Table Sizes. The main reason for the bad performance of N_{Large} and N_{Medium} is that the layout of our problem forbids a sensible use of caching. Typically, first memory lookups are slow, but when entries are often queried, as is the case for general networking-related lookups, the entries are cached and can then be retrieved much faster. As the coordinate values change frequently, the table access is random and the Netronomes cannot benefit from these caching effects. However, we can observe that smaller tables, as evidenced by N_{Small}, can improve the lookup time of non-cached values. This effect is not visible for our eBPF variants.

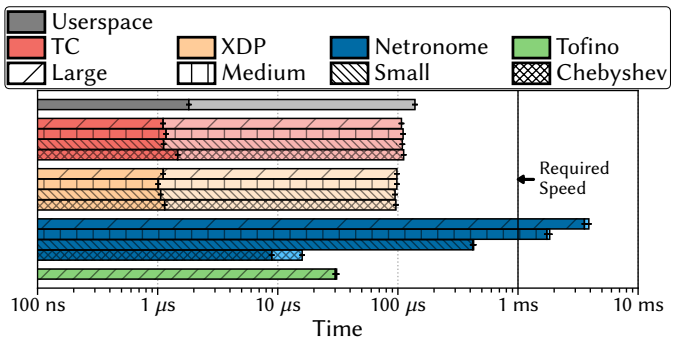

Fig. 5. Mean *calculation* (dark) and *total* times (light) of a single coordinate transformation on the different platforms given in seconds. Note the logarithmic scale of the x-axis.

Effect of Chebyshev Approximation. We also investigate the potential of entirely replacing the table lookups by direct computation strategies. Our N_{Cheby} variant yields a significant speedup and improves both the *calculation* and *total* time by one to two orders of magnitude. This demonstrates the usefulness of approximation techniques for improving processing speeds or, in our specific case, the benefits of replacing slow non-cached memory lookups by explicit calculations. In contrast, using the approximation even seems to slightly increase the compute time for TC_{Cheby}.

Impact of PISA Platform. Despite using numerous recirculations, *Tofino* achieves a performance in the same order of magnitude as N_{Cheby}. The extended capabilities of SmartNICs thus come with a significant performance penalty.

B. Calculation Accuracy

The accuracy of calculations is critical in the industrial domain, yet highly use-case-specific: some domains, e.g., aviation, require strict bounds on the maximum error while others tolerate fluctuations as long as the mean is stable. Analyzing the 15 000 (x, y, z) coordinates calculated in Sec. V-A, we capture these notions using two metrics: (i) the mean Euclidean distance between the points computed by our FP approaches and a reference point calculated using a floating-point implementation, and (ii) the percentage of transformations violating the 10 μm threshold of the drilling use case (see Sec. III-A) in any dimension.

As the accuracy values are similar for the same approaches on the different platforms, we combine the results for our analysis and present average values across all variants in Table I. Note that *Tofino* uses Layout$_L$ and is thus grouped together with TC_{Large}, XDP_{Large}, and N_{Large}.

Baseline and Lookup Variants. Using doubles and native libraries for calculation, *Userspace* achieves the highest accuracy. Our lookup-based PND variants using Layout$_L$ achieve a slightly worse accuracy due to the calculations in our FP arithmetic. Yet, they easily satisfy the required uncertainties with no observed violations. The same is true for Layout$_M$, which supports our motivation of leveraging periodicity and symmetry of sine and cosine to reduce the table sizes as a worthwhile option. However, decreasing the table size further, we observe that Layout$_S$ induces higher yet still tolerable

mean errors but can no longer provide absolute guarantees. Table size reductions show speed gains only for the SmartNICs but not for *eBPF* (see Fig. 5) so that they can become crucial trade-off parameters depending on the type of device used.

Effect of Chebyshev Approximation. Violating the 10 μm threshold on average, our *Chebyshev* variants have the worst accuracy; roughly 50 % of the transformations show at least one violation in any dimension. This means that we can neither give hard nor average guarantees and shows that, in the case of our SmartNICs, speeding up the computations by avoiding costly lookups yields a drastic penalty on accuracy.

C. Reliability

A coordinate transformation system must be able to reliably transform high rates of coordinate measurements. We thus judge the reliability of our variants based on the sustainable throughput rates as well as the corresponding packet loss. We generate our test packets at nine different rates from 1 Mbps to 25 Gbps and add some variant-specific rates to achieve a higher resolution around the region of first loss. Fig. 4 visualizes our results with the input rate in packets per second (PPS) on the x-axis and corresponding mean throughput values observed over 30 seconds on the y-axis. Values on the diagonal are loss-less; values below the diagonal are subject to loss.

High-level findings. Most of our variants achieve perfect reliability until they get close to their maximum throughput levels. The throughput then stabilizes at this maximum level and higher input rates only increase the loss. *Userspace*, e.g., can support up to 300 000 PPS without loss with a maximum throughput of around 450 000 PPS.

Netronome. Most of our Netronome variants can achieve slightly higher maximum throughput values than what they stabilize on for higher input rates. This is, e.g., visible for N_{Large} which slightly drops in throughput once first packet loss sets in. Operating the Netronomes at these sweet spots can thus ensure higher throughput rates at no loss while higher input rates will increase the loss and decrease the throughput.

Tofino. A similar observation can be made for *Tofino*. As can be seen, *Tofino* supports very high throughput rates at no loss despite using a high number of recirculations. However, there is a drastic increase in loss and decrease in throughput once the sweet spot is passed. This is due to the use of recirculations. In contrast to the other approaches, an increasing load on *Tofino* will not only cause incoming packets to be dropped, but also packets that are currently recirculating in the pipeline. Consequently, fewer transformations will finish successfully, resulting in a lower throughput rate. Note that we only investigate performance on a single input port and that higher rates are possible when using multiple ports.

D. Discussion

Based on our findings, we can draw several conclusions.

Lookup-based PND. Our lookup-based variants generally achieve a sufficient level of accuracy to support the scenarios described in Sec. III-A. However, the Netronome implementations suffer from inefficient table lookup support so that N_{Large} and N_{Medium} cannot provide the required computational speed. Sacrificing some accuracy on average and lifting the hard guarantee requirement, as illustrated by N_{Small}, can help to improve the performance. In contrast, *Tofino* and *eBPF* are able to satisfy both the accuracy and speed requirements, despite *Tofino* requiring numerous recirculations.

	Userspace	Chebyshev	Layout Large	Layout Medium	Layout Small
Eucl. Dist. [μm]	0.2	19.7	0.5	0.4	2.9
Violations [%]	0	48.5	0	0	0.7

TABLE I
MEAN EUCLIDEAN DISTANCE AND PERCENTAGE OF 10^{-5} m THRESHOLD VIOLATIONS FOR THE DIFFERENT CALCULATION VARIANTS.

Chebyshev Approximation. The Chebyshev approximation significantly speeds up calculations when the lookups are the bottleneck (N_{Cheby}). This goes in hand with a severe accuracy penalty, with mean errors increasing by two orders of magnitude. Consequently, the Netronome variants currently cannot outright support systems that require both high speeds and high accuracy, as described in Sec. III-A, yet offer an intriguing potential for trading off accuracy against speed. For *eBPF*, we notice no positive effect of the approximation.

Userspace vs. PND. *Userspace*, the *eBPF* variants, and *Tofino* satisfy both the speed *and* accuracy requirements. They also represent the two extremes of INC: leaving functionality on end-hosts versus pushing it into the middle of the network. Our results suggest that INC is generally well-suited to speed up processes where forwarding latency significantly contributes to the *total* time. However, implementing tasks such as the coordinate transformation is challenging as the required calculations do not fit the original purpose of the devices. In contrast, userspace implementations are more straightforward but are tied to end-hosts which might add additional latency if no such devices can be placed deep in the network. Similarly, offloading in the form of *eBPF* is also tied to end-hosts. Overall, we conclude that some well-chosen industrial applications can profit from INC, but that careful design is paramount to satisfy common requirements.

VI. RELATED WORK

A prominent part of the work on INC aims to bring "classical" network functionality, such as congestion control and load balancing [5], network monitoring [3], [17], or heavy-hitter detection [18] into the data plane. Other work applies INC to support *application-level* functionality, such as in data centers [1], [2], [19]. Also falling into the latter category, we focus on cyber-physical system control, for which support via INC has already been studied in, e.g., [6], [20], [21]. However, the considered systems have no explicit or more lenient requirements than our LMAS setting, and we also examine the trade-off between latency and accuracy in INC.

Table lookups and approximations. Sharma et al. [5] use lookup tables to express multiplications as additions by leveraging the logarithmic realm. Ding et al. [17] develop solutions for approximating the logarithm in pure P4. Both works

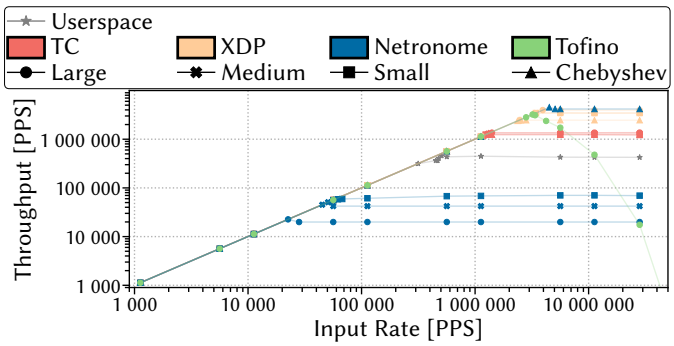

Fig. 6. Mean throughput of the different platforms in packets per second (PPS) subject to different input rates. Both axes have a logarithmic scale and all values below the diagonal represent loss.

also evaluate the impact of variable sizes on the achievable accuracy. We investigate the idea of splitting lookup tables to minimize space requirements while maintaining accuracy when facing very large input domains. Our results indicate that the *native* variable sizes and key lengths of PNDs can suffice for scenarios such as LMAS. Our experiments with Chebyshev polynomials further show that, while accuracy may be a concern, "nearly least maximum approximations" [16] are possible on PNDs with very satisfactory run-times.

Guidelines. Ports and Nelson [22] propose principles for INC and postulate that only stateless, reusable, common-case application *primitives* should be offloaded to the network, not whole applications. Our work suggests that on-path sensor value processing is a viable candidate for such primitives.

Further techniques. We include eBPF and userspace implementations as baselines for processing on end-hosts, but leave out further technologies such as netmap [23] and DPDK[4] as we focus on PNDs. We similarly do not consider FPGA-based variants based on, e.g., P4FPGA [24]. Investigating these technologies would help to complete the picture.

VII. Conclusion

The advent of programmable networking devices (PNDs) has been the spark to reignite the idea of In-Network Computing (INC). While many approaches already demonstrate that its time might actually have come [2], the development of new approaches is generally challenged by limitations and restrictions of the PNDs. Consequently, many new techniques build upon carefully designed trade-offs to achieve their respective goals. In this study, we showcase a common trade-off: processing speed versus achievable accuracy.

Implementing our scenario from industrial assembly, we find that it is generally feasible to support industrial applications on PNDs. This is especially beneficial if placing the functionality into the network can shortcut communications paths as this has both latency and throughput benefits. However, simultaneously achieving the required speed and accuracy is not always possible, and we find that we can trade some accuracy for significant gains in speed, e.g., by adjusting lookup table sizes or using approximation. We thus argue that the use of INC in industrial settings should focus on settings with simple computations as the investigated coordinate transformation task already puts a heavy burden on our PNDs.

[4]https://www.dpdk.org

Acknowledgment

We thank Vladimir Gurevich for his insightful feedback and valuable support. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2023 Internet of Production – 390621612 and the REFLEXES project – 315171171.

References

[1] X. Jin et al., "NetCache: Balancing Key-Value Stores with Fast In-Network Caching," in *ACM SOSP*, 2017, pp. 121–136.
[2] A. Sapio et al., "In-network computation is a dumb idea whose time has come," in *ACM HotNets*, 2017, pp. 150–156.
[3] Q. Huang et al., "OmniMon: Re-Architecting Network Telemetry with Resource Efficiency and Full Accuracy," in *ACM SIGCOMM*, 2020, pp. 404–421.
[4] P. Bosshart et al., "P4: Programming Protocol-Independent Packet Processors," *SIGCOMM CCR*, vol. 44, no. 3, pp. 87–95, 2014.
[5] N. K. Sharma et al., "Evaluating the Power of Flexible Packet Processing for Network Resource Allocation," in *USENIX NSDI*, 2017, pp. 67–82.
[6] J. Rüth et al., "Towards In-Network Industrial Feedback Control," in *ACM NetCompute*, 2018, pp. 14–19.
[7] I. Kunze et al., "Coordinate Transformation on GitHub." [Online]. Available: https://github.com/COMSYS/coordinate-transformation
[8] J. Pennekamp et al., "Towards an Infrastructure Enabling the Internet of Production," in *IEEE ICPS*, 2019, pp. 31–37.
[9] R. Schmitt et al., "Investigation of the Applicability of an iGPS Metrology System to Automate the Assembly in Motion of a Truck Cabin," *Applied Mechanics and Materials*, vol. 840, pp. 58–65, 06 2016.
[10] G. Hüttemann et al., "Modelling and Assessing Line-less Mobile Assembly Systems," *Procedia CIRP*, vol. 81, pp. 724 – 729, 2019.
[11] F. Franceschini et al., "Large-Scale Dimensional Metrology (LSDM): from Tapes and Theodolites to Multi-Sensor Systems," *IJPEM*, vol. 15, pp. 1739–1758, 08 2014.
[12] J. Schrimpf et al., "Experiments towards Automated Sewing with a Multi-Robot System," in *IEEE ICRA*, 2012, pp. 5258–5263.
[13] P. Maropoulos et al., "A New Paradigm in Large Scale Assembly - Research Priorities in Measurement Assisted Assembly," *Int. J. Adv. Manuf. Technol.*, vol. 70, pp. 621–633, 01 2014.
[14] C. Wallace, "Approximating sin(x) to 5 ULP with Chebyshev polynomials," 2017. [Online]. Available: http://mooooo.ooo/chebyshev-sine-approximation/
[15] W. Alexander et al., "Chapter 6 - Finite Word Length Effects," in *Digital Signal Processing*. Boston: Academic Press, 2017, pp. 351–388.
[16] J. F. Hart et al., *Computer Approximations*. Malabar, Florida, USA: Krieger Publishing Co., Inc., 1978.
[17] D. Ding et al., "Estimating Logarithmic and Exponential Functions to Track Network Traffic Entropy in P4," in *IEEE/IFIP NOMS*, 2020, pp. 1–9.
[18] R. Ben-Basat et al., "Efficient Measurement on Programmable Switches Using Probabilistic Recirculation," in *IEEE ICNP*, 2018, pp. 313–323.
[19] C. Mustard et al., "Jumpgate: In-Network Processing as a Service for Data Analytics," in *USENIX HotCloud*, 2019.
[20] R. Glebke et al., "Towards Executing Computer Vision Functionality on Programmable Network Devices," in *ACM ENCP*, 2019, pp. 15–20.
[21] F. Cesen et al., "Towards Low Latency Industrial Robot Control in Programmable Data Planes," in *IEEE NetSoft*, 2020, pp. 165–169.
[22] D. R. K. Ports and J. Nelson, "When Should The Network Be The Computer?" in *ACM HotOS*, 2019, pp. 209–215.
[23] L. Rizzo, "netmap: A Novel Framework for Fast Packet I/O," in *USENIX ATC*, 2012, pp. 101–112.
[24] H. Wang et al., "P4FPGA: A Rapid Prototyping Framework for P4," in *ACM SOSR*, 2017, pp. 122–135.