RISKA: Towards an Open-source RISC-V based Domain-specific System-on-Chip for SKA Data Processing

Arunkumar M. V.
Harshal G. Hayatnagarkar*
arunkumar.mv@thoughtworks.com
harshalh@thoughtworks.com
Engineering for Research
ThoughtWorks Technologies India
Pune, Maharashtra, India - 411006

ABSTRACT

The Square Kilometre Array, the largest radio telescope once completed, is one of the poster examples of computing in and beyond the exascale era. The computing performance and energy efficiency are required to be at least a couple of orders of magnitude better than the current top supercomputers. To meet these requirements, GPUs and FPGAs accelerators have been explored time and again, but these are yet to meet their promised potential due to limitations rooted in the algorithmic and architectural characteristics of the solutions.

For the SKA data processing to be done within meaningful time and resources, we therefore believe that a clean-slate approach could be useful. In this position paper, we propose *RISKA*, an open-source domain-specific system-on-chip to be designed specifically for SKA's data processing needs. The scope of the current proposal is to address the requirements of SKA's Science Data Processor supercomputer (SKA-SDP), with an intention for future expansion to other systems. We discuss a preliminary design of RISKA which combines open-source RISC-V CPU cores with SKA-SDP specific custom-designed accelerators and on-chip memory in order to widen the performance and energy efficiency bottlenecks in the SKA-SDP. Finally, we propose development of RISKA should be based on an open-source collaborative model.

ACM Reference Format:
Arunkumar M. V. and Harshal G. Hayatnagarkar. 2021. RISKA: Towards an Open-source RISC-V based Domain-specific System-on-Chip for SKA Data Processing. In *Proceedings of Fifth Workshop on Computer Architecture Research with RISC-V (CARRV 2021).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Square Kilometre Array once completed will be the largest and most sensitive radio telescope in the world. It is being constructed at two different sites: SKA-Low Frequency Array and SKA-Survey Array at an Australian site, and SKA-Mid Frequency Array at a South African site. It will be commissioned in two phases, SKA Phase-1 (*SKA-1*) by early 2020s, and thereafter SKA Phase-2 (*SKA-2*). SKA-1 will be a fraction of the full SKA-2 size. SKA-Survey Array is not part of the SKA-1 design.

Since its proposal, SKA has stood out among the big science projects for its ambitious science objectives, enormous data volume and velocity, and the corresponding exascale data processing challenges, often mentioned as the big data challenge of the 2020s. But what sets it apart from many other exascale computing problems, is that SKA data processing is data-intensive in nature, with the compounded effect of data volumes and data velocity [13, 46]. The data volumes streaming from the radio antennae are expected to be an average of 16 Tb/s together from the SKA-1 Low and SKA-1 Mid arrays [41]. At this rate, the telescope would generate more than an exabyte of data per day, making real-time processing necessary to reduce this data before it can be stored in the data centers for later use by scientists for their respective research.

SKA data processing is performed by a collection of subsystems. Data flows from the antenna stations to the *Central Signal Processor* (*CSP*) and then to the *Science Data Processor* (*SDP*). The SDP subsystem transforms the observation data into the data products, to be later used by the scientists for their research [29]. The most challenging computing requirements come from a small set of critical algorithms related to radio interferometry imaging in SDP [46]. In addition, these SKA compute systems are required to consume much less energy than the most efficient supercomputers available today [41].

In light of these algorithmic and architectural challenges, this paper attempts a clean-slate approach via the proposal of RISKA and its corresponding hypothesis — *"Could a System-on-Chip (SoC) specifically designed for the most time-consuming algorithms of SKA data processing address its overall challenges?"*.

The development of an SoC is a long enough process to rule out its feasibility for SKA-1 deployment, and hence this proposal targets requirements only for the SKA-2 deployment. Due to the overall complexity of the problem, this paper focuses on a small set of SKA-SDP imaging algorithms mentioned above.

The remainder of the paper is organized as follows. First, we analyze the SDP compute profile in Section 2 to gain insights about known bottlenecks and remedies. This is followed by the RISKA proposal in Section 3. Thereafter in the next Section 4, we describe a preliminary design of the SoC. Further, in Section 5, we elaborate

*Corresponding author

CARRV 2021, June 17, 2021,

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Algorithm 1 Tile-based Gridding

```
for all tiles in global grid do
    for all visibilities in tile do
        for v dimension in convolution kernel (that overlaps with tile) do
            for u dimension in convolution kernel (that overlaps with tile) do
                convolve visibility value with convolution kernel and add back to grid
```

on other important issues related to development and evolution. Finally, we conclude the paper with details of the RISKA source repository, and a call for collaboration.

2 SDP COMPUTE ANALYSIS

The past work in understanding the challenges to realize SKA-SDP offers a rich trove of insights into the characteristics of algorithms on different architectures. Analyzing this compute profile reveals that three algorithms concerned with image synthesis namely *Gridding*, *Degridding*, and *Two-dimensional FFT* (*2D-FFT*) make up a significant proportion of the overall compute load [29, 46].

Gridding is essentially a two-dimensional convolution operation on the incoming visibilities (samples of the Fourier-transformed sky) to populate the global grid representing the sky, whereas its inverse operation degridding estimates visibilities based on an input grid [37, 51]. A tile-based gridding approach [8] is depicted in Algorithm 1. It has low arithmetic intensity, and offers an opportunity for parallelism with tile-based processing (the outermost loop). *W-projection* and *AW-projection* are the most commonly used gridding algorithms although a recent approach namely *Image Domain Gridding* (IDG)[44] has also shown promising results. The SKA community has made substantial efforts to implement W-projection [27, 37, 51] and Image Domain gridding/degridding [44, 45] on CPU, GPU and FPGA architectures. Gridding and degridding are inverse operations with similar properties and bottlenecks [46].

The W-projection gridding mainly suffers from poor memory locality of incoming visibilities making it memory intensive. Thus, memory bandwidth becomes a bottleneck [10, 46, 51]. Another bottleneck is the large number of atomic additions to the global grid which makes exploiting parallelism difficult [37, 46, 51].

The third important algorithm *2D-FFT* has been well-studied, and is widely used beyond radio astronomy. Thus, many vendor-optimized solutions of 2D-FFT are available for various architectures. However, these implementations also suffer from various bottlenecks including low utilization on CPU and low interconnect efficiency and inadequate memory size on discrete accelerators [38, 46].

2.1 Bottlenecks and Remedies

Based on the past literature, we have compiled a non-exhaustive list of bottlenecks in the SKA-SDP algorithms along with possible remedies to widen these bottlenecks.

(1) **Nature of Algorithms.** Computational complexity and overall behavior of algorithms contributes to emergence and severity of bottlenecks on specific architectures. For example, W-projection gridding and IDG perform differently on CPUs, GPUs, and FPGAs [45].

Remedy — Choosing the right data-intensive algorithms is essential [20]. Introducing modularity and composability in algorithms could also help to accommodate and constrain architecture-specific optimizations.

(2) **Low Arithmetic Intensity.** Fewer computations performed per byte of data means longer time will be spent in data movement, leading to higher time and energy expenditure and idle compute, something commonly seen in *Data-intensive computing*.

Remedy — To minimize time and energy cost of data movement, bringing compute closer to the data via near-memory and in-memory computing strategies could be useful [12, 46].

(3) **Irregular Memory Accesses.** Such irregularities stem from the choice of data structure, behavior of algorithms, and nature of the data itself. They cause poor data locality, and worsen the memory bandwidth bottleneck [13, 37, 44, 51].

Remedy — Improving data locality of SKA-SDP algorithms across architectures is a hard problem, especially without making any assumptions about architectural characteristics [10, 12].

(4) **Off-chip Memory Bandwidth.** Device memory accesses are expensive and can quickly become a bottleneck in algorithms where frequent memory accesses and updates are necessary, like in the case of gridding and 2D FFT algorithms [10, 37, 38].

Remedy — Strategies like caching, prefetching, scheduling multiple load-stores improve memory bandwidth utilization. Another remedy is to convert algorithms from memory-bound to compute-bound [10, 44]. Use of more expensive on-chip High Bandwidth Memory (*HBM*) also shows improved results [12, 27].

(5) **CPU Performance per Watt.** Performance per watt of CPUs is far lower than accelerators primarily because of the inherent general-purpose nature and design of CPUs [18, 24, 29, 46].

Remedy — Vector units in CPUs can improve performance per watt for vectorizable algorithms [29]. As we observed in the literature, accelerators (in particular custom accelerators) are the best bet to overcome this limitation [18, 24, 29, 46].

(6) **Data Transfer to-and-fro Discrete Accelerators.** Discrete accelerators require data transfers to-and-fro host via relatively slower interconnects like PCIe, which means more time could be spent for data transfer than computation [38].

Remedy — Integrated accelerators could forgo the need for host-accelerator data transfer, thereby avoiding the PCIe interconnect bottleneck altogether.

(7) **Memory Size in Discrete Accelerators.** Accelerators with smaller memories can store only small chunks of data, making the algorithm responsible for partitioning and supplying chunks of data to accelerators [45].

Remedy — For problems too large to fit into the memory of a discrete accelerator, integrated accelerators are a possible alternative [12, 47].

3 RISKA PROPOSAL

RISKA (acronym for *RISC Implementation for Square Kilometre Array*) is our proposal of an open-source System-on-Chip (SoC) with SKA-SDP specific accelerators to achieve high performance at low energy consumption. This SoC would sit in place of the server CPUs to meet the extreme challenges in the SKA-SDP. For the non-extreme compute needs, the commercially available CPUs, GPUs and FPGAs could as well suffice.

The RISKA has a three-fold motive. First, the SKA-SDP would require an exascale supercomputer with ambitious goals for performance and energy efficiency [41]. These goals cannot be fulfilled by CPUs and discrete accelerators [10, 29]. Second, SKA-SDP is a case of domain-specific supercomputers as echoed in [30]. Third, SKA-SDP would require high-performance computing (HPC) and data-intensive computing (DIC) in a single environment [13, 40].

At present, the *Fugaku* supercomputer is the closest example for us to relate to these requirements. It has achieved the top spot in the Top500 and Graph500 lists [34], and 10th spot in the Green500 list. As we understand, its success can be attributed to the following decisions by its designers: 1. Targeting a small number of application domains, 2. Co-designing the hardware and software, and 3. Designing a custom SoC, namely *A64FX* [31, 39]. Observing parallels between Fugaku and SDP requirements, the Fugaku and A64FX serve as inspirations for RISKA proposal and development.

We anchor the RISKA proposal and its preliminary design in the insights from the analysis presented in Section 2. To deliver the exascale performance at the desired energy efficiency, we would need to combine two sets of ideas namely *Domain-specific Architectures* (*DSAs*, also called as *Accelerators*) and *Host-accelerator Integration*.

For the DSAs, the guiding principle comes from the Hennessy-Patterson vision to mitigate the diminishing returns due to fading of Moore's Law and Dennard Scaling [24]. The vision proposes a shift towards domain-specific custom-designed hardware, especially when intrinsic parallelism is available in algorithms. SKA-SDP specific accelerators can be designed for taking advantage of the intrinsic parallelism in the gridding and degridding algorithms. This approach has far lower execution overheads compared to general-purpose architectures [18, 26, 29].

For the integration, the host (CPU) and accelerators into a single chip or package. Such integration via *Uniform Memory Access* (*UMA*) can forgo the need for data transfers between host and accelerators, thereby avoiding the off-chip interconnect bottleneck, an issue frequently faced by discrete accelerators especially when frequent and large data transfers are involved [10, 14, 27, 38]. With the availability of larger and faster memory banks, such integration could help improve data locality, utilization of memory bandwidth, and energy consumption.

The host-accelerator integration for exascale computing was also explored in AMD's *Exascale Node Architecture* (*ENA*) via combining CPU and GPU with on-chip high-bandwidth memory [47]. Our proposed architecture bears a close resemblance to the ENA with the main difference being the addition of SKA-specific accelerators. In addition, Apple's M1 SoC has a few design elements relevant for RISKA such as integrated memory, graphics accelerator, and neural network accelerator [21].

While the preliminary design proposed in this paper targets the SKA-SDP, we intend to expand its scope to other SKA compute systems, especially to those deployed in *SKA Regional Science Centers* (*SKA-RSCs*). In addition, we believe that some of the RISKA ideas could be cross-pollinated to other domains beyond radio astronomy, such as meteorology and drug discovery.

Rest of the paper discusses the RISKA design elements and development process.

4 DESIGN ELEMENTS

The elements in the preliminary design for RISKA SoC are depicted in Figure 1. We would adapt this design to widen the known bottlenecks as well as the emergence of new ones during RISKA development. We acknowledge that an effective design space exploration is necessary to discover the feasible and optimal performance, energy and area specifications for the design elements. During this optimization process, we may not discover a single RISKA specification but many. Thus, it could lead to a possibility of multiple SoC variants, each covering a subset of the overall use-cases. RISKA variants are discussed in Section 5.2.

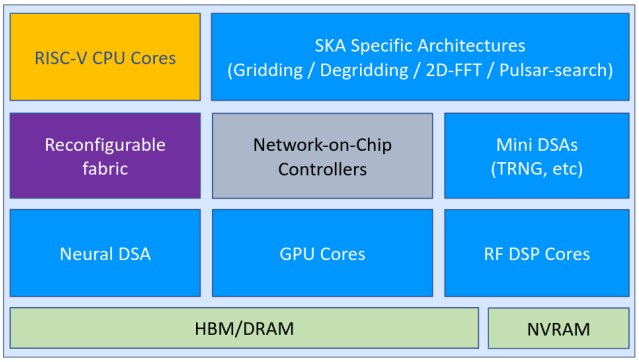

Figure 1: Basic design elements

4.1 RISC-V CPU Cores

For scientific computing, we believe that choosing an open ISA and architecture is in the best interest. The free and open RISC-V ISA fits this description [48], and it has so far generated enough interest in academia and industry alike. The ISA has many implementations, ranging from smaller cores for embedded/IoT applications to high-performance out-of-order processors [3, 11, 17, 22, 50].

For being free and open ISA, RISC-V offers an opportunity for open-source hardware development. Since approved ISA and its extensions are frozen, it helps in avoiding vendor lock-in. The only caveat of choosing RISC-V is that it is too young an ISA among its peers to have its capabilities ratified.

The RISC-V ISA features a modular and extensible design with a minimal base integer instruction set. Additional functionality can be introduced via standard or custom ISA extensions, which opens up new avenues for DSA integration. The modularity enables

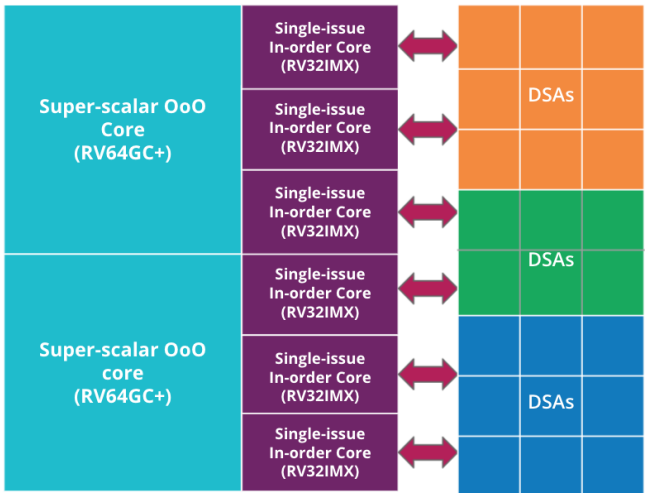

Figure 2: Many-core architecture for DSA integration

complex architectures such as heterogeneous many-core architectures (see Figure 2) where larger high-performance cores perform the general-purpose processing and smaller energy-efficient cores control the DSAs. Similar architectures have been shown to greatly improve performance and energy efficiency compared to conventional techniques [15, 42, 49].

4.2 SKA SDP-specific Accelerators

Top candidates for SKA-specific accelerators include the architectures for previously discussed imaging algorithms gridding, degridding and 2D-FFT [18, 26, 29, 46]. Although specialization is favorable, it may not be realistic in some cases. Conventional architectures like GPUs and DSPs have well defined programming models and lesser design overhead and can be leveraged wherever sufficient. As certain SDP pipelines like pulsar search and single-pulse/fast-transient pipeline involve heuristics based on machine learning capabilities [5], architectures like a *Neural DSA* could be useful.

Effective integration of the DSA will be as important as its micro-architecture itself. Strategies like *UMA* for DSA to avoid host-accelerator data transfers will be necessary. Other techniques include the use of dedicated cache for the DSA to enable fast communication [32] and intelligent last level cache bypassing for large writes [36].

4.3 Memory

Gridding, degridding, and 2D-FFT implementations are memory-bound because of data locality related issues. To quote [46], *"Solving the data-locality problem will be key: there should always be a suitable compute element near the data, on every level"*. Memory-centric approaches like *In-Memory Computing* (IMC) and *Near-Memory Computing* (NMC) are necessary for addressing memory bandwidth issues [12], and hence we would like to place sufficiently large memories on RISKA SoC.

3D-stacked *High-Bandwidth Memory* (HBM) can provide higher bandwidth but at higher energy-consumption [12]. Such integrated memories have been used in Fugaku's A64FX [39]. AMD's *Exascale Node Architecture* (ENA) also involves stacking HBM-DRAM on GPU chiplets and embraces the NMC approach. It tries to maximize bandwidth by minimizing the data movement overheads [47]. To improve energy efficiency, low-power low-bandwidth on-chip DDR memories have also been used in commercial devices like Apple's M1 SoC [21]. In RISKA, we intend to bring data closer to the compute so that it could compensate for the memory bandwidth saturation, caused by irregular memory accesses in gridding and other algorithms.

4.4 Reconfigurable Element

Due to the trade-offs in evolving a design and taping it out, RISKA may have only a small number of critical accelerators, thus leaving others to software-only implementations. For such scenarios, we propose a reconfigurable element in the RISKA design. This element could host any accelerator, which would be suitable for continuously evolving algorithms [23].

4.5 Network-on-Chip

In many-core many-accelerator SoCs using traditional bus interconnect topologies, on-chip communication may become a bottleneck. NoCs introduce alternate architectures for on-chip communication which tries to address these bottlenecks [43]. As on-chip communication efficiency will be crucial for RISKA due to the data-intensive nature of processing, such alternate architectures may become essential.

Such techniques have shown positive results. The Kalray MPPA 2 processor combines CPU cores with shared memory *Network-on-Chip* (NoC) [6]. In a similar approach, the A64FX SoC combines ARM cores with HBM and NoC [31]. Unlike RISKA, neither design supports accelerator cores.

5 DEVELOPMENT AND EVOLUTION

Designing an SoC and its software ecosystem is a mammoth task considering the complexity, competency, costs, and time [16].

On the complexity front, hardware-software co-design has been recommended in the SKA-SDP community [13]. We also see similar recommendations in general for exascale computing [33]. For a recent example, the co-design approach has certainly contributed to the success of the Fugaku supercomputer to deliver exascale performance [39].

On the cost front, an open-source model of innovation could be the solution, mainly because the burdens become more distributed. Such a model would also lead to more modular and composable design, rather than monolithic design, with a reliable ecosystem evolving around it over time [28].

Open-source development will come with its own set of organizational and technological challenges. For example, on the technology side, integrating the multiple IP blocks both commercial and open into a single chip will certainly be challenging. The area, energy, and performance of individual blocks have to meet the design budget [26]. An existing ecosystem which follows the above model is *Chipyard* [1], an open-source framework for developing full-system hardware using agile practices. If such an ecosystem could thrive

with innovations and investments, then an open-source model for RISKA development may be viable.

On the competency front, we believe that a collaborative open-source model for RISKA development would work best. The collaboration could bring in diversity of ideas and skills, as well as help in cross-pollinating RISKA ideas elsewhere. We share the details of the repository later in Section 7.

5.1 Guidelines for Designing DSAs/Accelerators

To ease general and contextual challenges in design and development of accelerators, a set of guidelines are shared in the book [26]. These guidelines provide an effective way to simplify design, reduce costs of development and increase overall performance of the accelerator. Here we discuss these guidelines in the context of RISKA.

(1) ***Use dedicated memories to minimize the distance over which data is moved.*** Memory architectures in GPAs involving multiple cache levels are expensive in terms of area and energy. Software-controlled scratchpad memories consume far less area and energy compared to multi-level caches [4]. For example, a two-way set associative cache consumes approximately 2.5 times the energy of the software-controlled equivalent [26]. Such memories tailored for a domain will suffice instead of expensive hardware-controlled caches [19], provided a co-design approach is followed for explicitly managing these memories to minimize data movement (see section 5 on co-design). Often, understanding of how compilers optimize and generate code for an architecture helps in minimizing data movements. In the case of RISKA, we could use such a memory architecture for each data-intensive accelerator, for example the gridding/de-gridding accelerator.

(2) ***Invest the resources saved from dropping advanced microarchitectural optimizations into more arithmetic units or bigger memories.*** In making CPUs more performant, resource-intensive strategies like out-of-order execution, multi-threading, and multi-processing are used. The rationale behind this guideline is that such strategies can be avoided when designing architectures for narrow application domains due to the assumed domain expertise [19], consequently freeing up resources which can be used for more arithmetic units and bigger memories. Thus, RISKA could have bigger, faster, and better on-chip memories to accommodate the needs of CPU cores and accelerators.

(3) ***Use the easiest form of parallelism that matches the domain.*** Leveraging the easiest inherent parallelism in the application domain to design the architecture aids towards a simpler programming model. For example, SIMD is easier than MIMD to specify and implement by programmers and compiler writers respectively. In the case of RISKA, a gridding accelerator can be designed around inherent parallelism across frequency channels, and perhaps 2D planar iterations via some collision detection algorithms [13, 46].

(4) ***Reduce data size and type to the simplest needed for the domain.*** Using the narrowest possible data type suited for the domain will improve effective memory bandwidth and make the applications less memory bound. It will also aid in increasing the number of arithmetic units in a given area. For example, certain use-cases in SKA-SDP may permit lower precision floating point calculations [2, 7, 35]. The RISKA design team can take advantage of it for those cases via RISKA variants (see the section 5.2).

(5) ***Use a domain-specific programming language to port code to the DSA.*** A domain-specific language (*DSL*) is designed around a domain's terminology and rules so that the software specification reads naturally from that domain. A DSL can be written for a DSA/accelerator in order to minimize efforts of code generation/translation for that accelerator [25]. In the case of RISKA, we do not have clarity currently about which of the accelerators would require an accompanying DSL. However, based on current understanding we could hazard a guess for gridding and pulsar searching accelerators.

Many of these guidelines provide a clear design choice amongst competing alternatives. But sometimes, a choice may be optimal for one usage scenario and be sub-optimal for another. Thus, one-size-fits-all becomes a fallacy, which we discuss in the following section.

5.2 RISKA Variants

Designing SoCs have been compared with city planning in the book [26], for that different components or IP blocks (*Intellectual Property* blocks) compete against each other for resources performance, area, and energy. There are trade-offs to be made in designing, guided by the use-cases. For example, the SoC for a watch design could be very different from that of a phone, or a laptop. With modern SoCs containing a multitude of IP blocks, resource allocation becomes a major issue. Thus, the scalability of IP blocks in area, energy, and performance to meet the design budgets is essential, making the design space exploration an optimization problem under constraints. Solving for optimization could lead to multiple solutions. In the case of RISKA, it could mean multiple design variants.

Given the diversity of SKA science objectives, we could explore the idea of a small number of RISKA variants guided by the SDP reference hardware architecture [9] to cover as many use-cases as possible. The current proposed hardware architecture categorizes the SDP compute servers into the 4 different *personalities* based on their functionalities namely 1. Receive, 2. Processing, 3. Service, 4. Storage. The processing servers contain throughput-optimized cores, and will deliver the bulk of required computational resources for processing [9]. The different variants of RISKA can make up these throughput-optimized cores.

The data processing afterwards in the *SKA-Regional Science Centers* would be more guided by the research questions of the scientists. This processing could involve exploratory ad-hoc steps. Suitability of RISKA in these scenarios is of interest to us for future work.

So, in a nutshell, we are interested in exploring if RISKA could provide maximum coverage for data processing with a minimum number of variants.

6 CONCLUSION

In this paper, we discussed the challenges of exascale computing and beyond for SKA-SDP, its compute profile, known bottlenecks, and potential remedies. Based on these details, we proposed an open-source SKA-SDP specific system-on-chip *RISKA* to cater to performance and energy efficiency requirements. We later presented a preliminary design of RISKA, followed by a prospective development and evolution process based on an open, collaborative model. In realizing RISKA, the openness of RISC-V ISA and its flourishing ecosystem play a pivotal role. Developing RISKA would be quite challenging if not impossible otherwise. Finally, we intend to keep the RISKA design extensible enough for other SKA compute needs, and hopefully the ideas can be applied to other domains beyond radio astronomy.

7 OPEN SOURCE REPOSITORY

We would like to conclude the paper by sharing details of the RISKA source code repository and a call for wider collaboration. The repository URL is: https://github.com/iamharshal/riska.

8 ACKNOWLEDGMENTS

We thank **Dr. T. Prabu** for his valuable review comments on this paper.

REFERENCES

[1] Amid, A., Biancolin, D., Gonzalez, A., Grubb, D., Karandikar, S., Liew, H., Magyar, A., Mao, H., Ou, A., Pemberton, N., Rigge, P., Schmidt, C., Wright, J., Zhao, J., Shao, Y. S., Asanović, K., and Nikolić, B. Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs. *IEEE Micro 40*, 4 (2020), 10–21.

[2] Arunkumar M V, Sai Ganesh Bhairathi, H. PERC: Posit Enhanced Rocket Chip. In *Fourth Workshop on Computer Architecture Research with RISC-V* (May 2020).

[3] Asanović, K., Avizienis, R., Bachrach, J., Beamer, S., Biancolin, D., Celio, C., Cook, H., Dabbelt, D., Hauser, J., Izraelevitz, A., Karandikar, S., Keller, B., Kim, D., Koenig, J., Lee, Y., Love, E., Maas, M., Magyar, A., Mao, H., Moreto, M., Ou, A., Patterson, D. A., Richards, B., Schmidt, C., Twigg, S., Vo, H., and Waterman, A. The rocket chip generator. Tech. Rep. UCB/EECS-2016-17, EECS Department, University of California, Berkeley, Apr 2016.

[4] Banakar, R., Steinke, S., Bo-Sik Lee, Balakrishnan, M., and Marwedel, P. Scratchpad memory: a design alternative for cache on-chip memory in embedded systems. In *Proceedings of the Tenth International Symposium on Hardware/Software Codesign. CODES 2002 (IEEE Cat. No.02TH8627)* (2002), pp. 73–78.

[5] Bolton, R., Broekema, P. C., Cornwell, T. J., van Diepen, G., Holli, C., Johnston-Holli, M., Preston, L. L., Mika, A., Nijboer, R., Nikolic, B., Salvini, S., Rampadarath, H., Scaife, A., Stappers, B., and Wortmann, P. Parametric models of SDP compute requirements. Tech. Rep. SKA-TEL-SDP-0000013, SKA Science Data Processor Consortium, 2019.

[6] Boyer, M., de Dinechin, B. D., Graillat, A., and Havet, L. Computing routes and delay bounds for the network-on-chip of the kalray mppa2 processor. In *ERTS 2018-9th European Congress on Embedded Real Time Software and Systems* (2018).

[7] Braam, P. Posits and computing at 200 pb/sec for the ska telescope. In *Conference for Next Generation Arithmetic (CoNGA'19)* (2019).

[8] Broekema, P. *Commodity compute- and data-transport system design in modern large-scale distributed radio telescopes.* PhD thesis, Vrije Universiteit Amsterdam, 2020.

[9] Broekema, C. SDP CDR Closeout Documentation: SDP Hardware Decomposition View. Tech. Rep. SKA-TEL-SDP-0000013, SKA Science Data Processor Consortium, 2019.

[10] Brown, A., Armour, W., Dulwich, F., and Chauveau, S. SDP Memo 72: Vertical prototyping of the gridding algorithm on GPU. *SDP Memo 072* (2018).

[11] Chen, C., Xiang, X., Liu, C., Shang, Y., Guo, R., Liu, D., Lu, Y., Hao, Z., Luo, J., Chen, Z., Li, C., Pu, Y., Meng, J., Yan, X., Xie, Y., and Qi, X. Xuantie-910: A commercial multi-core 12-stage pipeline out-of-order 64-bit high performance risc-v processor with vector extension : Industrial product. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)* (2020), pp. 52–64.

[12] Corda, S., Veenboer, B., Awan, A., Kumar, A., Jordans, R., and Corporaal, H. Near Memory Acceleration on High Resolution Radio Astronomy Imaging. *2020 9th Mediterranean Conference on Embedded Computing (MECO)* (2020), 1–6.

[13] Cornwell, T., and Humphreys, C. B. SKA Exascale Software Challenges. In *SKA Memo 128* (2010), SKA Consortium.

[14] Daga, M., Aji, A. M., and chun Feng, W. On the Efficacy of a Fused CPU+GPU Processor (or APU) for Parallel Computing. *2011 Symposium on Application Accelerators in High-Performance Computing* (2011), 141–149.

[15] Davidson, S., Xie, S., Torng, C., Al-Hawai, K., Rovinski, A., Ajayi, T., Vega, L., Zhao, C., Zhao, R., Dai, S., Amarnath, A., Veluri, B., Gao, P., Rao, A., Liu, G., Gupta, R. K., Zhang, Z., Dreslinski, R., Batten, C., and Taylor, M. B. The celerity open-source 511-core risc-v tiered accelerator fabric: Fast architectures and design methodologies for fast chips. *IEEE Micro 38*, 2 (2018), 30–41.

[16] Deneroff, M., and Solutions, E. Building an soc: How to do it? what will it cost? In *Workshop on System-on-Chip Design for HPC* (2015).

[17] Dörflinger, A., Albers, M., Kleinbeck, B., Guan, Y., Michalik, H., Klink, R., Blochwitz, C., Nechi, A., and Berekovic, M. A comparative survey of open-source application-class RISC-V processor implementations. In *Proceedings of the 18th ACM International Conference on Computing Frontiers* (Virtual Event Italy, May 2021), ACM, pp. 12–20.

[18] Fang, Z., Javadi, F., Cong, J., and Reinman, G. Understanding Performance Gains of Accelerator-Rich Architectures. In *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)* (2019), vol. 2160-052X, pp. 239–246.

[19] Fang, Z., Javadi, F., Cong, J., and Reinman, G. Understanding Performance Gains of Accelerator-Rich Architectures. In *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)* (2019), vol. 2160-052X, pp. 239–246.

[20] Fox, G. C., Jha, S., Qiu, J., and Luckow, A. Towards an understanding of facets and exemplars of big data applications. In *Proceedings of the 20 Years of Beowulf Workshop on Honor of Thomas Sterling's 65th Birthday* (2014), pp. 7–16.

[21] Frumusanu, Andrei. Apple Announces The Apple Silicon M1: Ditching x86 - What to Expect, Based on A14, November 2020.

[22] Gala, N., Menon, A., Bodduna, R., Madhusudan, G., and Kamakoti, V. Shakti processors: An open-source hardware initiative. In *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)* (2016), IEEE, pp. 7–8.

[23] Hayatnagarkar, H., MV, A., and Padalkar, B. Reconfigurable Domain-specific Architectures in the post-Moore's Law World: Implications for Software Engineering. *Preprint* – (2020).

[24] Hennessy, J., and Patterson, D. A New Golden Age for Computer Architecture. *Communications of the ACM 62* (2019), 48 – 60.

[25] Hennessy, J., and Patterson, D. A new golden age for computer architecture. *Communications of the ACM 62* (2019), 48 – 60.

[26] Hennessy, J. L., and Patterson, D. A. *Computer Architecture, Sixth Edition: A Quantitative Approach*, 6th ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2017.

[27] Hou, J., Zhu, Y., Du, S., Song, S., and Song, Y. FPGA-Based Scale-Out Prototyping of Degridding Algorithm for Accelerating Square Kilometre Array Telescope Data Processing. *IEEE Access 8* (2020), 15586–15597.

[28] Johnson, T. A., and Eigenmann, R. Context-sensitive domain-independent algorithm composition and selection. *ACM SIGPLAN Notices 41*, 6 (2006), 181–192.

[29] Jongerius, R. *Exascale Computer System Design: The Square Kilometre Array.* PhD thesis, Technische Universiteit Eindhoven, 2016.

[30] Jouppi, N. P., Yoon, D. H., Kurian, G., Li, S., Patil, N., Laudon, J., Young, C., and Patterson, D. A domain-specific supercomputer for training deep neural networks. *Communications of the ACM 63*, 7 (2020), 67–78.

[31] Matsuoka, S. The first "exascale" supercomputer Fugaku & beyond.

[32] McCalpin, John. Notes on Cached Access to Memory-Mapped IO Regions, May 2013.

[33] McMorrow, D. Technical Challenges of Exascale Computing. Tech. Rep. JSR-12-310, The MITRE Corporation, Apr. 2013.

[34] Meuer, Martin. The 56th edition of the TOP500 Supercomputers Rankings, November 2020.

[35] Miomandre, H., Nezan, J.-F., Ménard, D., Campbell, A., Griffin, A., Hall, S., and Ensor, A. Approximate buffers for reducing memory requirements in the ska. In *Preprint hal-02612369v1* (2020).

[36] Park, J. J. K., Park, Y., and Mahlke, S. A bypass first policy for energy-efficient last level caches. In *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)* (2016), pp. 63–70.

[37] Romein, J. An efficient work-distribution strategy for gridding radio-telescope data on GPUs. In *ICS '12* (2012).

[38] Salvini, S. SDP Memo: Fast Fourier Transforms. *SDP Memo 003, SKA-TEL-SDP-0000058* (2016).

[39] Sato, M., Ishikawa, Y., Tomita, H., Kodama, Y., Odajima, T., Tsuji, M., Yashiro,

H., Aoki, M., Shida, N., Miyoshi, I., Hirai, K., Furuya, A., Asato, A., Morita, K., and Shimizu, T. Co-Design for A64FX Manycore Processor and "Fugaku". In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2020), SC '20, IEEE Press.

[40] Simmonds, R., and Gounden, S. Sdp memo: Can sdp use existing big-data systems? Memo SKA-TEL-SDP-0000072, SKAO, 2016.

[41] Square Kilometer Array Organization. SKA Phase 1 Construction Proposal. Tech. rep., Feb. 2021.

[42] Swift, A. Esperanto accelerates machine learning with 1000+ low power risc-v cores on a single chip. RISC-V International Summit, 2020.

[43] Tsai, W.-C., Lan, Y.-C., Hu, Y.-H., and Chen, S.-J. Networks-on-Chip: Architectures, Design Methodologies, and Case Studies. *Journal of Electrical and Computer Engineering 2012* (2012), 634930:1.

[44] Veenboer, B., Petschow, M., and Romein, J. Image-Domain Gridding on Graphics Processors. *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2017), 545–554.

[45] Veenboer, B., and Romein, J. Radio-Astronomical Imaging: FPGAs vs GPUs. In *Euro-Par* (2019).

[46] Vermij, E., Fiorin, L., Jongerius, R., Hagleitner, C., and Bertels, K. Challenges in exascale radio astronomy: Can the SKA ride the technology wave? *The International Journal of High Performance Computing Applications 29* (2015), 37 – 50.

[47] Vijayaraghavan, T., Karunanithi, A., Kayiran, O., Meswani, M., Paul, I., Poremba, M., Raasch, S., Reinhardt, S. K., Sadowski, G., Sridharan, V., Eckert, Y., Loh, G. H., Schulte, M. J., Ignatowski, M., Beckmann, B. M., Brantley, W. C., Greathouse, J. L., and Huang, W. Design and Analysis of an APU for Exascale Computing. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (2017), IEEE, pp. 85–96.

[48] Waterman, A., Lee, Y., Patterson, D. A., and Asanovic, K. The RISC-V instruction set manual, Volume I: Base user-level ISA version 2.0. *EECS Department, UC Berkeley, Tech. Rep. UCB/EECS-2014-54* (2014).

[49] Zaruba, F., Schuiki, F., and Benini, L. Manticore: A 4096-core risc-v chiplet architecture for ultra-efficient floating-point computing, 2020.

[50] Zhao, J., Korpan, B., Gonzalez, A., and Asanovic, K. Sonicboom: The 3rd generation berkeley out-of-order machine. *Fourth Workshop on Computer Architecture Research with RISC-V* (May 2020).

[51] Zhu, Y., Zhao, X., Gao, X., You, H., and Li, Q. SDP Memo 082: Summarising Initial Scale-Out Prototyping Efforts. *SDP Memo 082* (2018).