Software driven SoC Architectural Exploration for AI and ML accelerators with RISC-V

Simon Davidmann, Lee Moore, and Larry Lapides
Imperas Software Ltd.
Oxford, United Kingdom
LarryL@imperas.com

***Abstract*— SoC developers and system designers are looking at hardware acceleration options for AI and Machine Learning applications moving from cloud-based algorithms to dedicated hardware. Since the algorithms are already configured with multicore support the tradeoffs become focused on the structure of processor arrays and the optimum performance requirements at each node. In addition to the flexibility offered by the open standard ISA of RISC-V to configure the core features to match the compute requirement, RISC-V offers the options to add custom extensions and instructions that allows a greater degree of system optimization. New extensions can be targeted at the application workload or as dedicated communication channels between the cores, nodes and/or interfaces to the NoC. This paper covers a methodology to evaluate the hardware options by enabling early system architectural exploration using software to uncover the optimum design configurations.**

***Keywords*— RISC-V, Multicore, Processor, SoC,**

I. Introduction

Artificial Intelligence (AI) and Machine Learning (ML) research and development based on software developed around cloud/datacenter resources has produced a wide range of algorithms that produce useful results with large databases and real-world datasets. To support the next level of requirements, optimized hardware can be developed for edge, near edge or datacenter with dedicated accelerators. Since the software platforms and algorithms already support distributed array processing as used in datacenters this helps the move to multicore configurations for dedicated hardware. However, a large number of options and configurations still need to be evaluated in order to select the most optimized design for the target application solution.

The goal of dedicated hardware accelerators is to optimize the performance around a target application or use-case. In this way the hardware design can first be developed around the general requirements and then further refined by tuning of the software to address the application features in detail. In this way the high-level hardware and software developments can proceed in parallel until the ideal balance or optimized performance is achieved.

The chart in Fig. 1, shows the history of multicore adoption, which has increased to compensate for the reduced improvements with single core performance and frequency [1].

This paper will explore the limitations on multiprocessor design as defined by Amdahl's Law and the implications for hardware accelerators. This will be illustrated with an example SoC processor array based on a heterogeneous design featuring both Arm and RISC-V CPU cores.

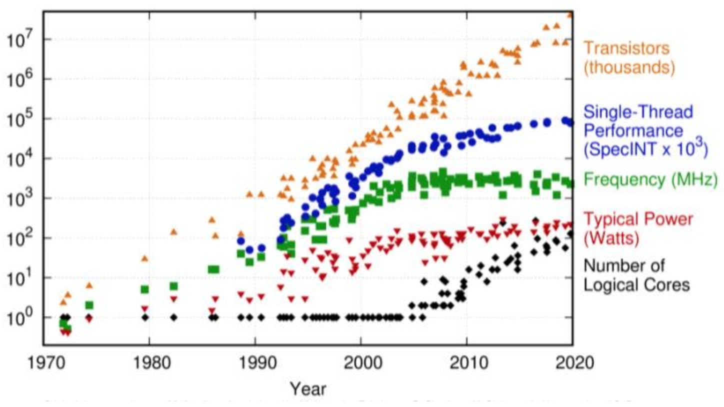


Fig. 1. 48 years of microprocessor trend data [1]

II. Amdahl's Law

One important consideration for any developer of multicore pressor designs is Amdahl's law [2]. Named after Gene Amdahl, a computer scientist who worked at IBM before founding Amdahl Computers, Amdahl's law gives a theoretical level of performance that can be achieved by use of parallel processing to speed up a given task. In any algorithm some operations can be performed in parallel and can be mapped with efficient scale to multiple processing elements (PE), however some other functions will need to be processed in a strict order and/or have other dependences. This produces a scale of reference given the percentage of any particular application that has compute requirement that cannot be processed in parallel. Using

Amdahl's laws as a guide it can be possible to predict the optimum number of hardware resources that would be the most beneficial before reaching a point of diminishing return, that is the point at which the additional hardware gives proportionally reduced overall improvements. The formula is shown in Fig. 2, and a graph illustrates the efficiency limits that could be expected for a number of examples in Fig. 3. For an SoC design to operate efficiently as a hardware accelerator for an AI algorithm, the detailed analysis and operational dependences can help to guide the insights into the potential design structure.

$$S_{latency}(s) = \frac{1}{(1 - p) + \frac{p}{s}}$$

- S_{latency} is the theoretical speedup of the execution of the whole task;
- s is the speedup of the part of the task that benefits from improved system resources;
- p is the portion of execution time that the part benefiting from improved resources originally occupied.

Fig. 2. Amdahl's Formula [2]

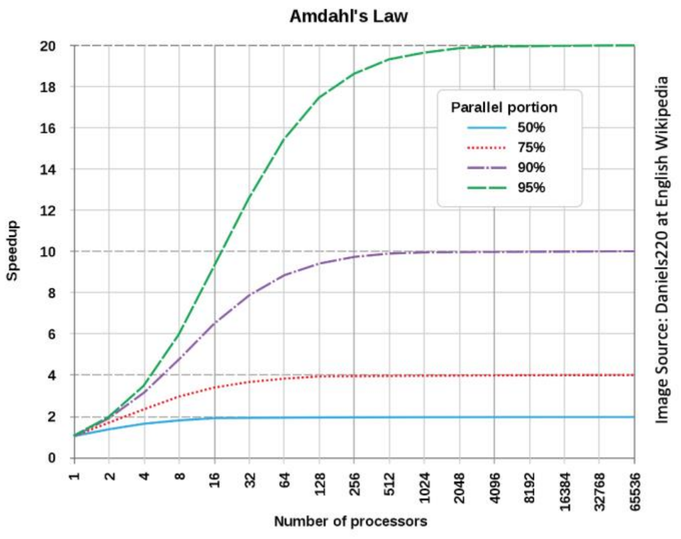


Fig. 3. Chart of processors/speedup [2]

While an interesting concept for first order approximations, much of the expected efficiency could be lost due to data transfer inefficiencies, or a theoretical design that exceeds the cost (silicon die area) or power budget anticipated for the final product. In addition to inefficiencies around data movement, another factor is the inter-core communications. Within a multiprocessor system a number of communication needs could arise from synchronizing activities, buffer limits, exceptional events, as well as routine processor errors or other interrupt conditions. The frequency of these events and the corresponding message bandwidth may have a significant impact on the prospects of achieving the theoretical speed ups predicted with just calculations-based estimates.

While a purely hardware view of the workload and algorithm partitioning may appear to over some interesting support for some configurations, the software useability needs to be considered also. When the hardware is finalized, the software side of the solution may need to be evolved further to address changing requirements and further optimization. The debug and code maintenance tasks cannot be understated. In some situations, the software will be developed by other users in the future and developers not familiar with the detailed hardware tradeoffs, so care must be taken to provide debug and performance processing features that assist the future developers to monitor and adapt the software.

III. MULTICORE PROCESSOR ARRAYS

A number of options for configuring processing elements are well documented. Typical designs contain arrays of processors and hierarchy. The block diagram in Fig. 4, shows this as a two-dimensional array. Depending on the workload of the algorithm, the Processing Element (PE) could be a single core, dual-core or multicore. Additional options could be to consider a heterogeneous multicore PE, such as mixing scalar and vector processors even with the same base instruction set architecture, or using dedicated hardware for either data processing or as a communication channel. All of these options have various merits and advantages that can be considered as the workload requirements for a targeted set of application needs are profiled.

As with any design process there are two items that need to be considered. First, the future roadmap of requirements and how well this PE will cover the anticipated changes ahead, or if the hardware evolves into a second generation will the software investment migrate efficiently. Second, as by definition the PE will be a programmable unit, how will adopters develop, debug and optimize the future software and support maintenance updates over the lifetime of the hardware. As much of the design performance and efficiency depends on the operation of the PE this design can be a major focus to evaluate the different configuration options, however in isolation as a single unit it may not be possible to fully evaluate all aspects of the design until it is implemented with the larger multicore array. So, while the PE can be consider as becoming the main unit of the multicore array, it may be necessary to iterate on the design features after reviewing the initial results with a multicore evaluation set-up.

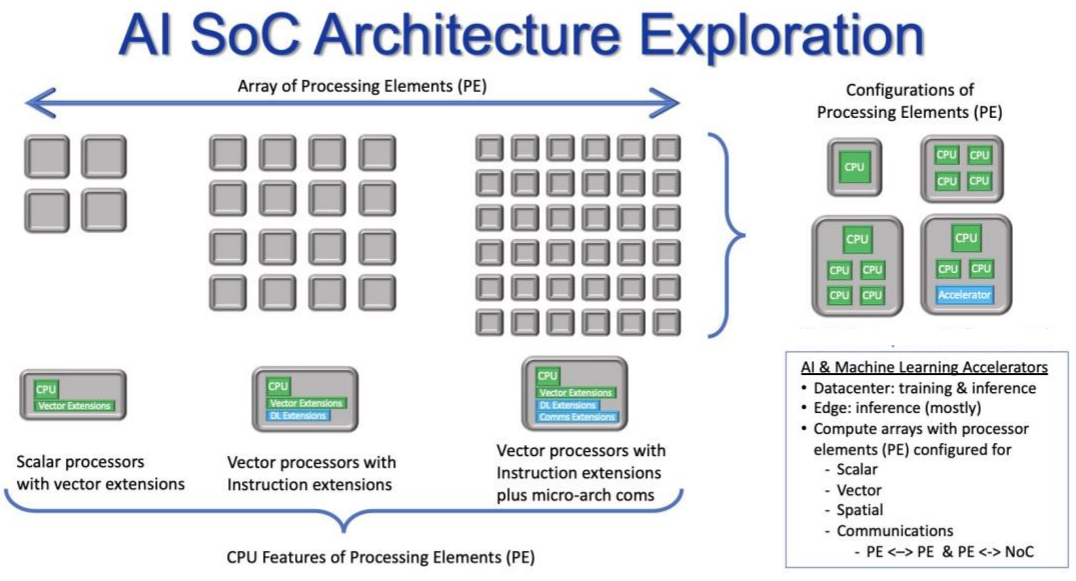

AI SoC Architecture Exploration
Array of Processing Elements (PE)
Configurations of Processing Elements (PE)
CPU
Vector Extensions
DL Extensions
Comms Extensions
Accelerator
Scalar processors with vector extensions
Vector processors with Instruction extensions
Vector processors with Instruction extensions plus micro-arch coms
CPU Features of Processing Elements (PE)
AI & Machine Learning Accelerators
• Datacenter: training & inference
• Edge: inference (mostly)
• Compute arrays with processor elements (PE) configured for
- Scalar
- Vector
- Spatial
- Communications
- PE <-> PE & PE <-> NoC

Fig. 4. Multicore arrays configurations for Processing Elements (PE) hierarchy

Once the configuration of the basic PE building block is chosen, the next step is to select the number and configurations of PE's with the overall multicore design. This could be an array with communications over a common shared NoC (Network-on-Chip) structure, or with groups of PE's with local communication and then a NoC across a large number of clusters. To manage the overall control and operations a number of Control Processors (CP) can be configured to manage the multicore array and external interfaces. Given the range of requirements this may share the same ISA as the PE cores, or be a perhaps a different ISA more tuned to the control requirements.

Another aspect of developing a new multicore SoC is the anticipation of future generations of hardware design. Giving consideration to the bandwidth of the CP and how this will scale with the future potential PE arrays may be a useful consideration in selecting the ISA and configurations of the CP. However, it is not required or perhaps even desirable to have all cores uniform in features and performance. Given the target workload and application area it may be useful to have a combination of resources that are tuned to the requirements of the application or algorithm.

One other configuration choice to be considered is having an additional applications processor external to the AI accelerator array as a front-end to the AI processing. This applications processor might often use a familiar ISA and operating system to aid user familiarity and ramp up. For example, if the user entry into the AI accelerator is an applications processor running Linux, and the AI routine is a Linux application that makes use – transparently to the user – of the AI accelerator, this could help adoption of the new SoC.

One universal requirement is the debug features and resources that future developers will need to support software development and maintenance. This will need to be considered all through the design project:

- Pre-Silicon: Project concept, development and profiling
- Post-Silicon: First silicon bring-up, firmware development, demos and evaluations
- Production: Adoption by end users, including complete lifecycle of software management

IV. Overview of AlexNet

AlexNet [3] is a well-known convolutional neural network (CNN) algorithm, which achieves high accuracy in image recognition, but which also has extensive compute requirements. The AlexNet structure is shown in Fig. 5, which illustrates the potential for parallel processing. The number of parameters is 58 million (Floating-Point 32bit) with a computational cost of 1 billion multiply-add operations. The AlexNet performance requirements are illustrated in Fig. 6.

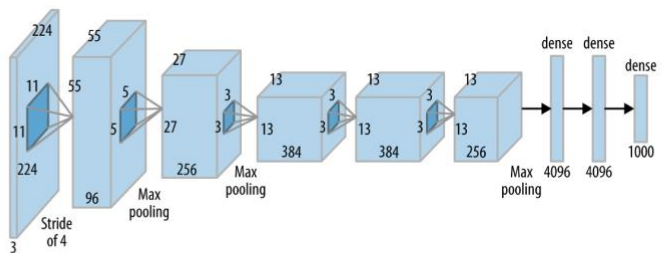


Fig. 5. Convolution layers have lots of calculations, parallelized these layers map AlexNet over the 16 CPU cores

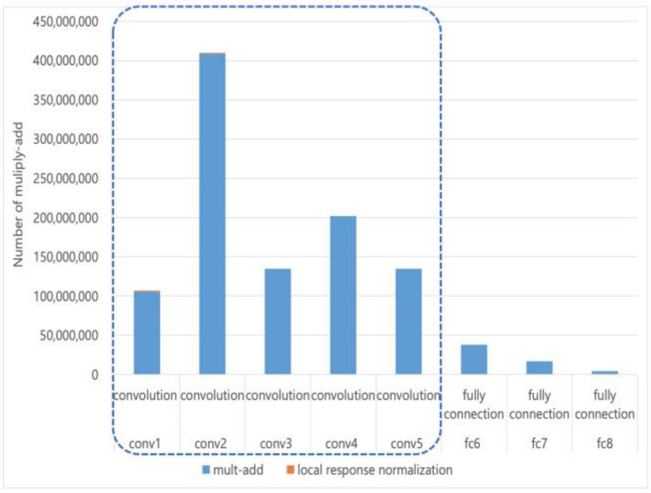


Fig. 6. Workload chart of Multiply-Adds

V. Software Driven Profiling

Having explored the details of the AlexNet algorithm it appeared that with a PE based on a single core of medium performance would have sufficient compute resources to undertake the key central processing steps. In reviewing the guidance offered by Amdahl's law, a common approach is to consider the symmetry of the operations across multiple of 2 or units of 8, 16, or 32, as a way to estimate the number of PE's that could prove to be the most efficient. But with the additional impact on the area (SoC die size) budget the advantages of 32 were much less attractive than an optimized configuration of 16 PE. To oversee the operations of the multicore array and act at the main interface point a single high-performance core was selected. Again, due to the single thread workload requirements and overall lack of parallelism in many of these oversight functions, a multicore solution gives only marginal improvements, so a single core was selected as the main CP. The design was configured with 17 (16 + 1) cores. The block diagram is shown in Fig. 7.

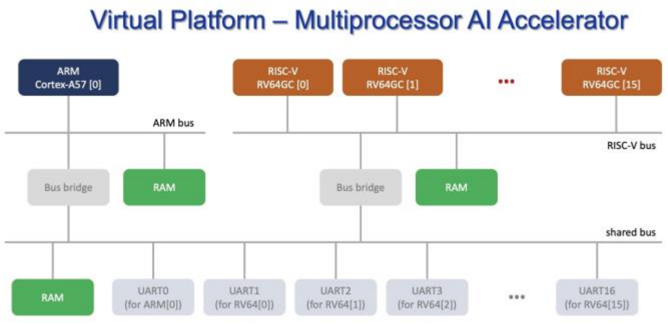
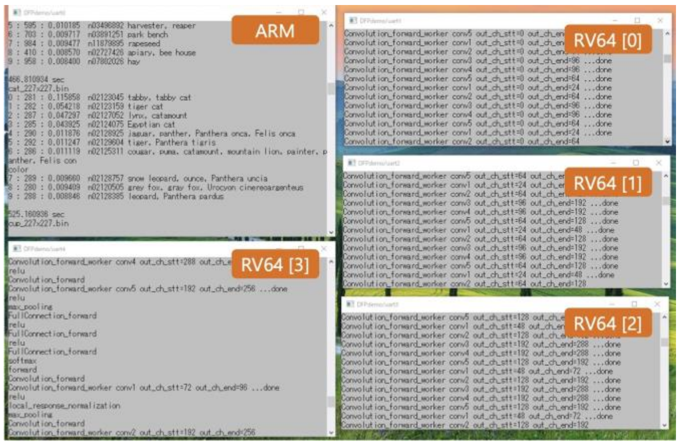


Fig. 7. Block diagram of virtual platform with 17 CPU cores

Developed using the open-source models available from OVPworld [4], a virtual platform was built using the model library for an Arm Cortex-A57 [5] as the control processor, and the 16 PE's based on RISC-V cores configured as RV64GC [6] for the AI accelerator array. The virtual platform was completed with interconnects between the Arm and RISC-V subsystems including standard peripherals and interfaces.

With the virtual platform set-up, the next step was to map the original software implementation on to the array. With a virtual platform this can be achieved with a simple direct memory load to each PE, which is useful for a quick start to the profiling and investigation phase. However, the actual start-up process can be developed after the initial tests confirm this as a useful configuration.

Having pre-loaded all the programs the multiprocessor design can now be simulated and with the use real world datasets start to profile the operation of the complete design. The debug and analysis tools provided by Imperas [7] offers a single user interface to inspect any of the 17 cores, and cross trigger debug events between any of the cores, peripherals, or interfaces. The simulation view can be seen in Fig. 8, heterogenous debug in Fig. 9, and some sample results for one of the test images are shown in Fig. 10.



```
5 : 595 : 0.010185  n03496892 harvester, reaper
6 : 703 : 0.009717  n03891251 park bench
7 : 984 : 0.009477  n11879895 rapeseed
8 : 410 : 0.008570  n02727426 apiary, bee house
9 : 958 : 0.008400  n07802026 hay

466.810934 sec
cat_227x227.bin
0 : 281 : 0.115858  n02123045 tabby, tabby cat
1 : 282 : 0.054218  n02123159 tiger cat
2 : 287 : 0.047297  n02127052 lynx, catamount
3 : 285 : 0.043925  n02124075 Egyptian cat
4 : 290 : 0.011876  n02128925 jaguar, panther, Panthera onca, Felis onca
5 : 292 : 0.011247  n02129604 tiger, Panthera tigris
6 : 286 : 0.011119  n02125311 cougar, puma, catamount, mountain lion, painter, p
anther, Felis con
color
7 : 289 : 0.009660  n02128757 snow leopard, ounce, Panthera uncia
8 : 280 : 0.009409  n02120505 grey fox, gray fox, Urocyon cinereoargenteus
9 : 288 : 0.008846  n02128385 leopard, Panthera pardus

525.160936 sec
cup_227x227.bin
```

```
Convolution_forward_worker conv4 out_ch_stt=288 out_ch_e
relu
Convolution_forward
Convolution_forward_worker conv5 out_ch_stt=192 out_ch_end=256 ...done
relu
max_pooling
FullConnection_forward
relu
FullConnection_forward
relu
FullConnection_forward
softmax
forward
Convolution_forward
Convolution_forward_worker conv1 out_ch_stt=72 out_ch_end=96 ...done
relu
local_response_normalization
max_pooling
Convolution_forward
Convolution_forward_worker conv2 out_ch_stt=192 out_ch_end=256
```

```
Convolution_forward_worker conv5 out_ch_stt=0 out_ch_en
Convolution_forward_worker conv1 out_ch_stt=0 out_ch_en
Convolution_forward_worker conv2 out_ch_stt=0 out_ch_en
Convolution_forward_worker conv3 out_ch_stt=0 out_ch_end=96 ...done
Convolution_forward_worker conv4 out_ch_stt=0 out_ch_end=96 ...done
Convolution_forward_worker conv5 out_ch_stt=0 out_ch_end=64 ...done
Convolution_forward_worker conv1 out_ch_stt=0 out_ch_end=24 ...done
Convolution_forward_worker conv2 out_ch_stt=0 out_ch_end=64 ...done
Convolution_forward_worker conv3 out_ch_stt=0 out_ch_end=96 ...done
Convolution_forward_worker conv4 out_ch_stt=0 out_ch_end=96 ...done
Convolution_forward_worker conv5 out_ch_stt=0 out_ch_end=64 ...done
Convolution_forward_worker conv1 out_ch_stt=0 out_ch_end=24 ...done
Convolution_forward_worker conv2 out_ch_stt=0 out_ch_end=64
```

```
Convolution_forward_worker conv5 out_ch_stt=64 out_ch_en
Convolution_forward_worker conv1 out_ch_stt=24 out_ch_en
Convolution_forward_worker conv2 out_ch_stt=64 out_ch_en
Convolution_forward_worker conv3 out_ch_stt=96 out_ch_end=192 ...done
Convolution_forward_worker conv4 out_ch_stt=96 out_ch_end=192 ...done
Convolution_forward_worker conv5 out_ch_stt=64 out_ch_end=128 ...done
Convolution_forward_worker conv1 out_ch_stt=24 out_ch_end=48 ...done
Convolution_forward_worker conv2 out_ch_stt=64 out_ch_end=128 ...done
Convolution_forward_worker conv3 out_ch_stt=96 out_ch_end=192 ...done
Convolution_forward_worker conv4 out_ch_stt=96 out_ch_end=192 ...done
Convolution_forward_worker conv5 out_ch_stt=64 out_ch_end=128 ...done
Convolution_forward_worker conv1 out_ch_stt=24 out_ch_end=48 ...done
Convolution_forward_worker conv2 out_ch_stt=64 out_ch_end=128
```

```
Convolution_forward_worker conv5 out_ch_stt=128 out_ch_e
Convolution_forward_worker conv1 out_ch_stt=48 out_ch_en
Convolution_forward_worker conv2 out_ch_stt=128 out_ch_e
Convolution_forward_worker conv3 out_ch_stt=192 out_ch_end=288 ...done
Convolution_forward_worker conv4 out_ch_stt=192 out_ch_end=288 ...done
Convolution_forward_worker conv5 out_ch_stt=128 out_ch_end=192 ...done
Convolution_forward_worker conv1 out_ch_stt=48 out_ch_end=72 ...done
Convolution_forward_worker conv2 out_ch_stt=128 out_ch_end=192 ...done
Convolution_forward_worker conv3 out_ch_stt=192 out_ch_end=288 ...done
Convolution_forward_worker conv4 out_ch_stt=192 out_ch_end=288 ...done
Convolution_forward_worker conv5 out_ch_stt=128 out_ch_end=192 ...done
Convolution_forward_worker conv1 out_ch_stt=48 out_ch_end=72 ...done
Convolution_forward_worker conv2 out_ch_stt=128 out_ch_end=192
```

Fig. 8. Multicore simulation of virtual platform

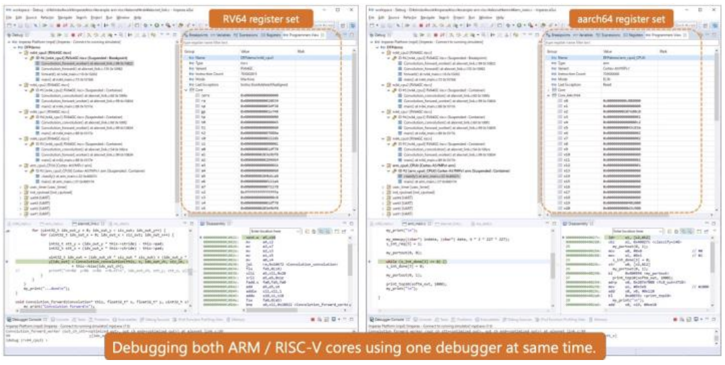
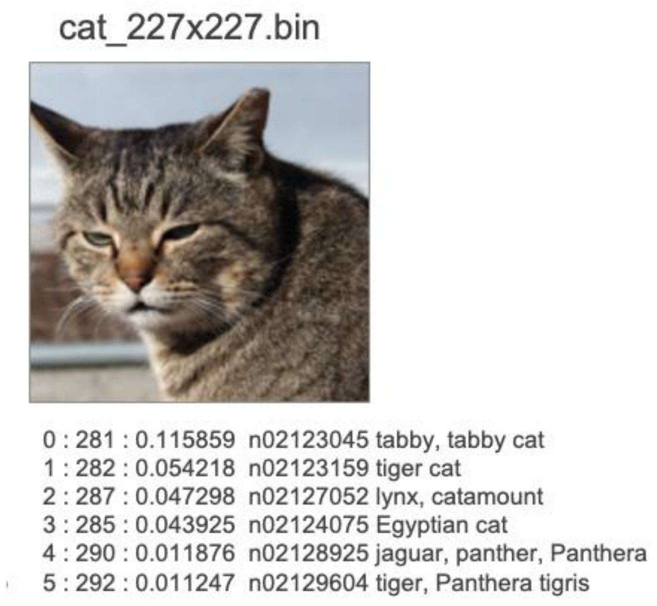

RV64 register set
aarch64 register set
Debugging both ARM / RISC-V cores using one debugger at same time.

Fig. 9. Heterogenous multicore debug and analysis



```
0 : 281 : 0.115859  n02123045 tabby, tabby cat
1 : 282 : 0.054218  n02123159 tiger cat
2 : 287 : 0.047298  n02127052 lynx, catamount
3 : 285 : 0.043925  n02124075 Egyptian cat
4 : 290 : 0.011876  n02128925 jaguar, panther, Panthera
5 : 292 : 0.011247  n02129604 tiger, Panthera tigris
```

Fig. 10. Example results for cat image

VI. RISC-V OPTIMIZATIONS

Within the AlexNet example above, a RISC-V core was selected as the main processor array. Since RISC-V is an open standard ISA it allows SoC developers a number of options in selecting processor cores, such as develop a new design based on the specification, using one of the many open source cores, or selecting from the growing number of commercial core providers. In addition, with any of these starting options, an SoC developer can modify and extend a core with custom instructions. This allows for a core to be optimized for the application or targeted workload. Also in multicore processor design one major consideration is the overhead of core to core, PE to PE, and PE to NoC communications. Using the heavyweight traditional processor bus protocols may affect the processing performance and overall efficiency of the system. With a custom RISC-V design a lightweight inter-core communication link could be added with custom instructions and state registers.

In addition to the flexibility that RISC-V offers SoC developers, the verification challenges for a custom or extended processor need to be considered also, more detail on this area can be found in the Imperas paper presented at Embedded World 2020 on RISC-V Verification [8].

VII. CONCLUSIONS

This paper illustrates the design configurations and options that SoC developers can consider in developing a multicore design for an AI application. In building a custom accelerator a number of tradeoffs need to be evaluated to develop an optimized solution. In addition to the profiling the design options, considerations for future software design, debug, analysis and maintenance needs to be included.

In looking ahead to further optimizations, the PE could be profiled further to perhaps add custom instructions. Also, the new RISC-V vector extensions offer more options in mapping AI algorithms to dedicated hardware accelerators.

ACKNOWLEDGMENT

The authors would that to thank the engineers at eSol Trinity for the implementation and results presented in this paper. The authors would like to thank RISC-V International for the specifications referenced in this paper. The authors would like to thank Arm for the Arm Cortex-A57 core features and documentation referenced in this paper.

REFERENCES

The template will number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use “Ref. [3]” or “reference [3]” except at the beginning of a sentence: “Reference [3] was the first ...”

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors’ names; do not use “et al.”. Papers that have not been published, even if they have been submitted for publication, should be cited as “unpublished” [4]. Papers that have been accepted for publication should be cited as “in press” [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

[1] Background and repository with 48 years of Microprocessor trend data https://github.com/karlrupp/microprocessor-trend-data

[2] Amdahl’s law https://en.wikipedia.org/wiki/Amdahl's_law

[3] Krizhevsky, Alex; Sutskever, Ilya; Hinton, Geoffrey E. (2017-05-24). "ImageNet classification with deep convolutional neural networks"

[4] OVP planforms and models available at https://www.ovpworld.org

[5] Arm Cortex-A57 https://www.arm.com

[6] RISC-V Specifications https://riscv.org/technical/specifications

[7] Imperas Software commercial simulation solutions, debug and analysis tools https://www.imperas.com

[8] Imperas Software paper at Embedded World 2020 on RISC-V verification “Impact of RISC-V Adaptability on SoC Verification Methods”