Retrospective: Understanding Sources of Inefficiency in General-purpose Chips

Rehan Hameed Wajahat Qadeer Omid Azizi Alex Solomatnikov Benjamin C. Lee Christos Kozyrakis Mark Horowitz

I. Introduction

This paper, published at ISCA 2010, explored why ASIC energy and area efficiency was so much higher than a general purpose CPU and what could be done to close that gap. It was motivated by the fact that compute had started moving from desktops and servers to power constrained devices such as smartphones, tablets, intelligent cameras and other embedded devices. At the same time we had hit the end of Denard scaling so computer architects could no longer rely on just technology node scaling to get the required energy reduction. Instead they had to find architectural ways to create fundamentally more efficient designs with less waste.

The paper showed that data movement overhead is the biggest culprit behind the inefficiency of programmable compute systems. Application-specific HW blocks don't do "better compute" – rather they are much less wasteful in data movement compared to programmable processors. Their datapaths incorporate dedicated storage elements close to compute blocks eliminating expensive read/writes to memories and register files. To bring similar efficiency to programmable processors one needs to add specialized datapaths, which the paper called "magic instructions", which combined compute and storage in a manner matching the data flow of the specific algorithms being accelerated. The work was an early example showcasing the advantage of adding complex dataflow optimized compute+memory units to scalar / vector processors.

The paper used H.264 encode as the target application for this study. This application was chosen as it has been shown to benefit tremendously from specialized HW - a dedicated H.264 ASIC outperforms a high-end CPU at a fraction of silicon area while consuming 500x times lower energy! The paper carried out a detailed analysis of the energy consumption in various components of a computing systems while executing this application, and analyzed designs with various degrees of specialization including a generic CMP based on RISC cores, cores with standard extension such as SIMD and VLIW, cores with custom instructions of various complexity as well as a fully custom ASIC design. By comparing the energy consumption across these designs, it determines where energy is wasted in an instruction set based design and how various specializations help eliminate or reduce that waste.

In a generic CPU, arithmetic operations account for less than 1% of total energy. Most of the energy is 'wasted' in energy-expensive data accesses from memories and register files, with the control overhead of instruction sequencing being the second major component of energy consumption. Adding vector datapath extensions similar to GPUs and Intel's SSE instructions helps amortize the instruction execution overhead over multiple operations. This helps improve CMP performance by 14× and energy by 10×. However compute still accounts for less than 10% of total energy with data accesses, and instruction execution overhead accounting for the remaining 90%. Thus the vector CMP remains 50x worse compared to the ASIC.

To bridge the remaining gap we must satisfy two requirements:

- Each compute instruction executes hundreds of operations to effectively amortize the high cost of instruction fetch and sequencing
- Accesses to register files and memories are significantly reduced by retaining data inside the compute units

To meet these goals requires using custom storage structures with algorithm-specific communication links to directly feed large amounts of data to functional units without explicit register accesses. The connectivity of these storage elements to functional units is matched to the dataflow of the target algorithms. These instructions with merged compute and storage elements enable a very high degree of parallelism, increase data-reuse in the datapath and reduce communication bandwidth and power at register file as well as memory level. Adding these complex instruction to an existing programmable engine yields a programmable solution which can achieve high efficiency. The final CMP design leveraging such instructions was within 3x of the energy consumption of the dedicated ASIC.

II. Key Insights and Relevance in 2023

Over the last decade the key insights presented in the paper, the need to create "magic" instructions with optimized storage/compute and the advantages of embedding these optimized dataflows into a processor, have been validated by several commercial and academic designs. Today, these insights are specially significant because of the growing importance of AI workloads. AI compute requirements far exceed traditional algorithms requiring tens to hundreds of TeraOps, making the traditional processor design approaches infeasible. At the same time the AI algorithms are rapidly evolving with new algorithms coming up every month thus requiring a high degree of programmability. The most successful designs in this space have followed our insights of adding specialized hardware into a powerful programmable engine.

We continued developing the ideas presented in the paper further, creating the Convolution Engine (CE) [5], specialized for the convolution-like data-flow that is common in image and video processing, computer vision and convolutional neural networks. The CE instruction set achieves high energy efficiency in these domains by capturing data reuse patterns, eliminating data transfer overheads, and enabling a large number of operations per memory access. This approach was later implemented commercially in specialized AI processor chips developed at Kinara [1], which have been deployed in large volumes to enable high performance, real-time AI inference in power-and-cost-constrained edge devices [7].

Most successful AI accelerators have followed similar approaches of embedding specialized magic instruction engines into a base programmable engine. One of the clearest examples is NVidia GPUs which added tensor cores [3] to its vector SMs enabling the accceleration of various AI algorithms including the widely used cuDNN library. Similarly TPU [6] uses a small set of massively parallel instructions which leverage a large systolic array with storage elements directly embedded in it. The TPU works along side a traditional CPU where the CPU uses its instruction set to accelerate a range of matrix math / linear algebra operations.

Coarse-grained re-configurable array (CGRA) architectures adopt an alternate approach to enable flexibility. CGRAs consist of an array of programming elements (PEs) where each PE is connected to neighboring PEs. The connectivity between PEs is configurable allowing multiple algorithmic and data flows to be mapped [4]. However, the overhead of the routing between the PEs can still limit the area and energy efficiencies if each PE is very simple; therefore, CGRAs also employ complex PEs tailored to specific applications to prevent data routing from lowering efficiency. [2]

III. What was missed in the paper

This paper focused on executing a specific function, and as expected, creating a more flexible computing solution, the CMP with magic instructions, results in worse efficiency than the ASIC. In our case the penalty was around 3x confirming our intuition that flexibility in the HW inevitably results in lower efficiency. While that is true at the level of an individual compute kernel, processor flexibility can lead to greater system level efficiency. Practical implementations of this approach in Kinara AI chips have shown that the added flexibility at processor level gives the system level compiler more freedom in managing data flow and data reuse across memory hierarchies for each application and minimize data movement which would otherwise dominate the energy consumption. Thus, counter-intuitively the flexible instruction set approach leads to greater overall system efficiency.

The extent of communication between accelerated functions and rest of the application is another factor not discussed in the paper. Workloads such as video encode work on large chunks of data so a standalone accelerator doesn't require frequent communication with other parts of the system. However, applications like cryptography and some AI inference workloads require close interaction with a complicated software stack (OS, network, storage and other applications etc). For such workloads flexibility offered by magic instructions in general purpose processors is even more attractive.

Of course the greater flexibility that this approach provides comes at a cost. The paper poses the design problem as a trade-off between flexibility and efficiency. However it does not directly address the third axis of this tradeoff which is the ease of developing code. The instruction set based approach presented in the paper makes it possible for a programmer to merge specialized compute with regular code unlike using standalone fixed function units which typically require code restructuring and partitioning of the algorithm to effectively use them.

However, using these instructions, optimizing loops and variables is nevertheless substantially more complex for a compiler than using standard vector / SIMD instructions. Compilers are good at lowering code, however using these complex instruction usually requires lifting code, which is a harder compilation task. Today these specialized instructions are often manually used by programmers in their code which limits productivity. NVidia and Google both have extensive compiler efforts, and have a large number of programmers developing optimized libraries using these special instructions which are then used by the end customers. Enabling the compilers and code generation frameworks to automatically make use of these instructions is the current research frontier.

References

[1] Kinara.ai. [Online]. Available: https://kinara.ai/products/

[2] J. Melchert, K. Feng, C. Donovick, R. Daly, R. Sharma, C. Barrett, M. A. Horowitz, P. Hanrahan, and P. Raina, "Apex: A framework for automated processing element design space exploration using frequent subgraph analysis," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 33–45. [Online]. Available: https://doi.org/10.1145/3582016.3582070

[3] NVIDIA. Nvidia tesla v100 gpu architecture. [Online]. Available: http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

[4] R. Prabhakar, Y. Zhang, D. Koeplinger, M. Feldman, T. Zhao, S. Hadjis, A. Pedram, C. Kozyrakis, and K. Olukotun, "Plasticine: A reconfigurable architecture for parallel patterns," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 389–402.

[5] W. Qadeer, R. Hameed, O. Shacham, P. Venkatesan, C. Kozyrakis, and M. A. Horowitz, "Convolution engine: balancing efficiency & flexibility in specialized computing," in *ISCA*, 2013, pp. 24–35.

[6] K. Sato and C. Young. An in-depth look at google's first tensor processing unit (tpu). [Online]. Available: https://cloud.google.com/blog/products/ai-machine-learning/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu

[7] S. Ward-Foxton. Ai startup deep vision raises funds, preps next chip. [Online]. Available: https://www.eetimes.com/ai-startup-deep-vision-raises-funds-preps-next-chip