OPEN
Compute Project

Polymorphic Architecture for Future AI Applications

Editors: Weifeng Zhang, Danny Moore and Bijan Nowroozi

Contributors:

Weifeng Zhang, Lightelligence
Allan Cantle, Nallasway
Kevin Cameron
Brian Hirano, Micron
Marian Verhelst
Danny Moore, Rambus
Greg Compton
Ritu Gupta, Sony
Igor Muskatblit
Huihuo Zheng, Argonne National Lab
Zhibin Xiao, Moffett AI
Winston Liu, KeySight
Robert Feng, Portwell
Keith McKay, ScaleFlux
Yang Ki, Samsung
Matt Bergeron, Meta
Jonathan Zhang, Meta
Manoj Wadekar, Meta
Michael Choi

Executive Summary

Rapid progress of artificial intelligence in the past decade has been driving tremendous computational demands in data centers. While AI algorithms and related data tonnage continue evolving quickly with larger and larger models, complexity in system resource scaling is growing and there is no prominent sign of AI algorithm convergence across different AI application domains. The nature of these two-dimensional evolutions, across-domain evolution (spatial divergence) and across-generation evolution (temporal divergence), brings unparalleled challenges to the design and development of underlying AI computing architectures.

This whitepaper proposes a **polymorphic architecture** to address these challenges based on hardware software co-design. With large-scope scalability, polymorphic computing empowers a **fractal computing** infrastructure with hierarchical transformability and composability. It is dynamic and virtualized, thus able to adapt to the characteristics of the running applications such that it can reconfigure and adapt itself *across multiple computing hierarchies* into a new logical accelerator, which executes the application just as effectively as a purpose-built ASIC accelerator.

Polymorphic architecture takes advantage of modern architecture trends, including coarse-grained domain-specific accelerators, chiplet technologies, composable memory, and the advancement of high bandwidth and increasingly fault tolerant interconnect technologies, to achieve the required composability and transformability. From the heterogeneous chiplets, compute nodes, all the way to racks and beyond, polymorphic architecture allows dynamic resource sharing, partitioning, and re-composition for high execution efficiency and resource utilization while overcoming domain specific accelerator's (DSA's) inflexibility of programming. Polymorphic architecture also leverages AI application's relatively deterministic parallelism during AI HW/SW co-design to enable rapid architectural re-composability.

Table of Contents

1 Introduction 4
2 Hardware Software Co-Design 6
2.1 Why Is Hardware Software Co-Design Important Now? 6
2.2 Why is Hardware Software Co-design Particularly Important for AI? 7
3 Compute Architecture Trends and Challenges 9
3.1 Computation Scale-up 9
3.2 Computation Scale-out 12
3.2.1 Intra-Chip Interconnect 12
3.2.2 Inter-Chip Interconnect 13
4 A System Engineering Perspective 14
4.1 System building blocks 15
5 Memory Hierarchies and Adaptability 15
5.1 Intra-Chip Memory Performance Challenges 16
5.2 Inter-Chip Memory Pooling and Sharing 16
6 Software Ecosystem 17
6.1 AI Ecosystem and Polymorphic Architecture 18
6.2 Ecosystem Goals for Polymorphic System Architecture 19
7 Polymorphic Architecture 20
7.1 Architecture for Adaptability 21
7.2 Composable Memory Architecture 22
7.3 Computation Partitioning and Co-Optimizations 24
8 Conclusion 25
9 Glossary 27
10 References 28
11 License 30
12 About Open Compute Foundation 31

1 Introduction

The proliferation of artificial intelligence and machine learning applications, from object detection, natural language processing, graph neural networks, to generative networks for content creation, Monte Carlo pruning (i.e. AlphaGo), and protein simulation inference, has been driving a massive surge in the computation demands of the hardware required to run these applications. At the same time, machine learning (ML), particularly deep learning (DL), continues rapid evolution with increasing model complexity utilizing techniques such as GAN, capsules, diffusion, and larger data models to further improve AI application accuracy and quality. The size of AI models has doubled every year and computation capacity for AI training has increased about 10 times each year due to data size explosion. Reviewing the MLPerf training results in recent years (see Figure 1), a similar trend is also observed. The computation demand required for AI training is accelerating much faster than Moore's Law. The hardware that is currently being developed is incapable of keeping pace with the increased demands of the algorithms themselves.

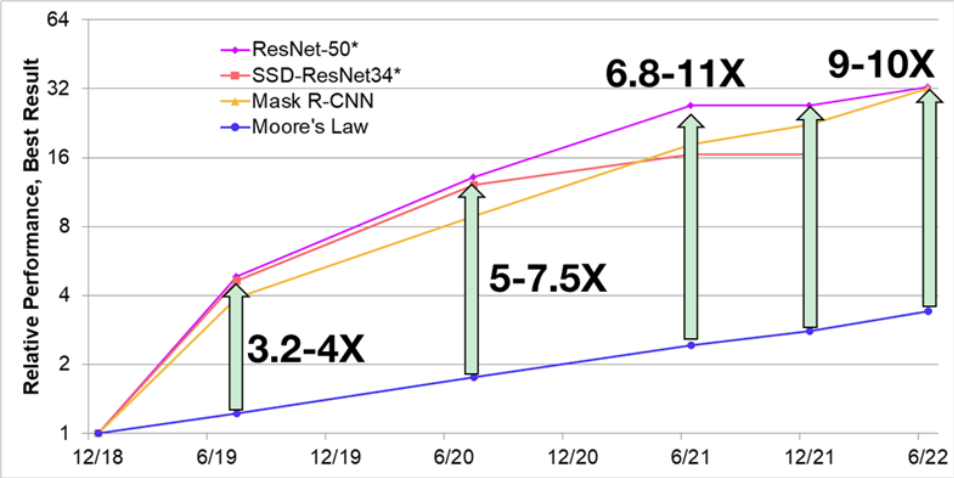

Figure 1: Performance of MLPerf Training, ahead of Moore's Law[1]
(* indicates benchmark increased accuracy target in MLPerf Training v0.6)

Contemporary generations of computing acceleration processors are struggling to meet the growing computation requirements. Both chip area and power consumption have increased dramatically over the years. As shown in Figure 2, peak performance correlates highly with power consumption in publicly announced AI accelerators and processors. Furthermore, common approaches such as resource scale-out to add additional accelerators in an attempt to gain more computation capacity, does not provide the full performance benefits that are needed. The total system performance only scales up sub-linearly despite the linear expansion of resources. Consequently, the larger number of compute nodes added, the lower total system efficiency, and therefore the more costly such scaled systems will be.

The problem is aggravated when migrating the architectures across different AI application sub-domains. For example, personalized recommendation and autonomous driving applications have very diversified constraints and priorities on computation throughput, latency, and memory usage. Specialized AI accelerators may work very efficiently in one AI application domain, but are less performant in other domains. Thus, we might observe big performance fluctuations with specialized AI processors when compared to general-purpose AI accelerators.

With no obvious sign of AI models converging across subdomains (spatial scaling across algorithm domains) and with the fast evolution of the AI algorithms themselves (temporal scaling with algorithm generations), it is critical that the acceleration architecture designed for future AI applications needs to address the following challenges:

1) Ensure the efficiency and scalability of the architecture for various model sizes, large or small.
2) Be resilient to the spatial evolution and temporal evolution in AI algorithms.

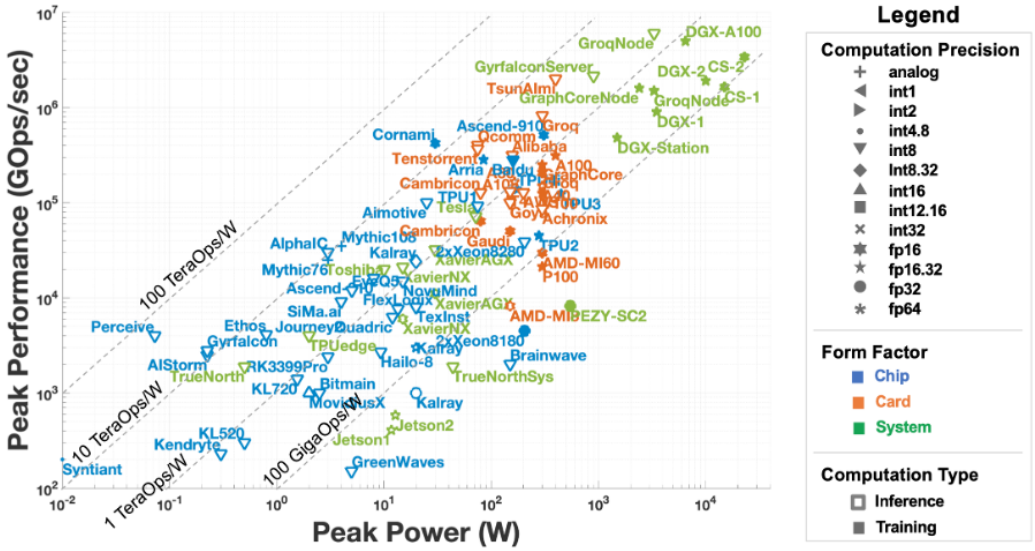

Figure 2: peak performance vs. power scatter plot of publicly announced AI accelerators and processors[2,3]

This paper focuses on hardware software co-design to propose a future-oriented polymorphic accelerator architecture for AI training and large-scale inference applications offering both scalability and transformability. A Polymorphic architecture adapts to the characteristics of the running applications such that it can reconfigure and transform the computing resources to compose a new compute unit (i.e., a logic accelerator) within the architecture, which executes the applications just as effectively as a customized ASIC accelerator. The architecture is highly scalable across various AI sub-domains (i.e., spatial scalability), such as vision, speech recognition, natural language processing, etc. It also scales across multiple generations of AI algorithms (i.e., temporal scalability), now and future. This paper identifies the critical components of this architecture, including compute, interconnect, and memory hierarchies, to achieve the required polymorphism.

2 Hardware Software Co-Design

Hardware software co-design is a broad topic. Generally speaking, AI hardware/software co-design refers to the concurrent design of the hardware and software components of an Artificial Intelligence system to maximize overall performance and efficiency. Key elements of AI HW/SW co-design include:

System Architecture: The overall system architecture defines how hardware accelerators integrate with the CPU and memory subsystem. Key considerations include accelerator flexibility, host-accelerator bandwidth, memory hierarchy, and power delivery. The architecture impacts performance and scalability.

Hardware Accelerators: Specialized hardware units like GPUs, TPUs, and ASICs that are optimized to accelerate critical AI computations like matrix multiplication and neural network inferences. The hardware design aims to maximize throughput and efficiency.

Software Frameworks: AI software frameworks like TensorFlow, PyTorch, Caffe, etc. that can target different hardware back-ends. The frameworks provide optimization opportunities through techniques like quantization, pruning, compiler optimizations etc.

Co-Design: Hardware and software designers collaborate closely to match the capabilities of the hardware accelerators to the requirements of AI models and frameworks. The goal is to fully exploit the available hardware resources.

Automated Co-Optimization: Using techniques like autoML and neural architecture search to automatically find optimal hardware-aware neural network designs, precision specifications, and partitioning between hardware and software.

Benchmarking: Rigorously benchmarking design choices on relevant AI tasks and datasets to guide design decisions for optimized overall performance.

The key benefits of AI hardware/software co-design are faster and more power efficient AI computations compared to CPU/GPU-only solutions. Effective co-design is crucial for the next generation of AI accelerator chips and platforms.

In this paper, we limit ourselves to discussions on computing chip/system (hardware) and AI algorithms (software). In addition, we focus on the resulting impact of co-design (such as architecture adaptability, software execution efficiency and so on), rather than the implementation process of co-design (such as the design flow, languages, and tools to carry out the co-design).

2.1 Why Is Hardware Software Co-Design Important Now?

Hardware software co-design can be dated back to the early 1990s[4,5]. For decades, hardware and software have been designed by independent teams each dealing with their own unique challenges. Hardware teams focus on constraint driven, application specific strategies, while software teams deal with managed complexity driven problem specific solutions. Such strategies can quickly become stale from the start as the rate of development between hardware and software components are significantly unaligned. AI model structures and algorithm parameter requirements (doubling every 3 months) change on a much faster cadence than much longer hardware development cycles (18 months to 24 months). This forces, for example, ASIC providers into rapid development of simplified solutions to broaden the application scope and product lifetime to reach as many customers as possible over a longer applicable lifetime. On the other hand, software teams often start development from a relatively unknown hardware platform and continue development during evolving hardware capabilities or under a highly conservative assumption of hardware architecture, often resulting in underutilization[6].

The co-design approach is garnering a renewed interest as it addresses these problems by leveraging the expertise of both hardware and software domains. Co-design should be driven via the top-down approach first, independent of how hardware and software are implemented (obviously, some general aspects of HW/SW need to be factored in). Under this approach, the goal of co-design is to reach a common understanding of the problem with a top-level solution, break it down into coarse-grained functional blocks with clear interfaces and verification methodologies, and then be followed by an architecture design. During this process, systems designers can weigh trade-offs to decide where complexity should reside, in the software or in the hardware. Thus, co-design enables assessment of the implications of algorithmic decisions on the hardware efficiency early on and without the hardware fully present, e.g., through analytical (transaction level) models of HW efficiency of certain computational kernels. And vice-versa, co-design must be able to assess the implications of hardware features to the algorithmic properties as quickly as possible without developing the actual hardware. With rapid design and update cycles of both algorithmic and hardware innovations, co-design results in jointly optimal solutions of both components. Ultimately this design pattern facilitates the creation of hardware aware software and vice versa.

2.2 Why is Hardware Software Co-design Particularly Important for AI?

Generally speaking, AI application software architecture is composed of:

User Interface Layer - This is how users interact with the AI application, such as through a website, mobile app, voice interface, etc. It handles taking in user input and displaying outputs from the AI.

Processing Layer - This is where the "brain" of the AI resides. It includes the machine learning models that power the AI capabilities. Some common components are:

- Natural Language Processing (NLP) - For processing and understanding text/speech
- Computer Vision - For analyzing visual data like images or video

- Speech Recognition - For transcribing spoken audio into text
- Predictive Models - Statistical models that make predictions from data

Data Layer - Stores and manages the data used to train, evaluate and serve the AI models. This includes databases, data warehouses, storage systems like S3.

Infrastructure Layer - Provides the computational resources needed to develop, train and serve AI models. This includes GPUs for model training, container orchestration systems like Kubernetes to deploy models, and cloud services like AWS, GCP, Azure.

The different layers integrate with each other to provide end-to-end AI capabilities. For example, user input flows to the NLP module, whose predictions are used by a predictive model to generate a response, which is then sent back to the user through the interface. The architecture's goal is to be modular and scalable.

From a system's point of view, general computing is more modest and balanced in its requirements for architecture, while HPC are typically code dense for general mathematical operations and high throughput and this results in systems architecture featuring distributed parallel processing requiring high compute density.

Compared with traditional general-purpose computing and high-performance computing (HPC), AI work loads have a high degree of complexity that require more intensive computing such as tuned for matrix math and ML ops, significantly higher memory capacity, huge network bandwidth, and low communication latencies all simultaneously. These unique properties lead to the development of domain specific accelerators (DSA) with holistic system level solutions to address execution efficiency and usability. Thus, the co-design of AI accelerators is often driven by compromises between general-purpose and DSA architectures.

While the general-purpose accelerators are highly flexible and easily deployable in a wide variety of use-cases, they suffer from low execution performance and power efficiency. Conversely, a DSA architecture improves performance and power efficiency by customizing the design towards the characteristics of the application. The DSA architecture trades the versatility of a general purpose solution to achieve a power and performance improvement by more effectively and efficiently utilizing the native computing power of the hardware. With deep customizations, DSAs often include special computing units with unique architecture-supported parallel mechanisms, data types, and domain-specific languages (DSL) etc.

Moreover, computation in AI algorithms exhibit additional sets of characteristics, such as regular parallelism, reduced control flow, and relative determinism. Because the traditional architecture is well equipped with logic to handle control flow, branch prediction, and so on, the properties of regular parallelism and reduced control flows will shift the focus of the AI architectures, bringing new optimization opportunities to AI co-design.

DSA architectures may be based on an optimized Instruction Set Architecture (ISA), which may include domain specific data types or precision requirements. Therefore, they often suffer from

inflexibility and poor portability, leading to performance loss or even rendering the hardware completely unusable when deployed in different AI domains or systems architectures

Therefore, some of the primary challenges of AI co-design are how to accelerate AI for both high performance and power-efficiency while keeping any optimized components/DSAs flexible and scalable across various AI domains.

3 Compute Architecture Trends and Challenges

As AI applications continue to push the boundaries of current computing systems, one constraint is that the subsystem components (such as compute, memory, interconnect, and so on) have their own design challenges. One example is the trend of Moore's Law and its impact on processors. Following this trend is economically necessary for producers and technically necessary for developers, yet increasingly more and more difficult on both fronts. While the efficiency of computation continues to increase, performance gains in a single hardware accelerator or improvements in AI algorithms alone are not sufficiently offsetting the explosion of computation demands.

In this section, we identify the limitations and missing features in contemporary compute architectures when enabling large scale computation with rapid evolution. For a solution to be adaptive and composable, the architectural resources need to have dynamic runtime reconfigurability, allowing for decomposition into smaller compute functions if the application's requirements are less demanding, or re-composition into larger compute functions otherwise.

3.1 Computation Scale-up

In the past few decades, the number of transistors on an integrated circuit has been doubling every 18 months, as predicted by Moore's Law. At the same time, the CMOS-based digital manufacturing process continues to scale down from 28nm, 16nm, 7nm, to 3nm, and so on. This abundance of transistors has brought great opportunities for continuous architectural innovations and improvements in computational power. Additionally, some of these improvements, including higher memory bandwidths, deeper microarchitecture pipelining, multithreaded execution, and various data movement mechanisms (e.g., cache hierarchies and data prefetches) are able to reduce the adversarial effects of Memory Wall[7] during increased computation scale-up.

However, digital electronics such as transistors, start to approach their physical limitations at scales that are not far from present and thus Moore's Law is showing the signs of slowing down. The scale-up of computation has moved toward chip multi-processing (CMP), multicore, and even kilo-core architectures to boost chip level parallelism. But, with the end of Dennard Scaling around 2006, this densification of CMOS has led to an increased power density, causing

devices to heat up due to current leakage. Even though packing more cores into a chip increases its computing power, the limited power budget of chips imposes serious scalability problems because simultaneously utilizing all the cores on chip is no longer feasible. Some cores in a chip must remain passive (turned off) or clocked (slowed down) to stay within a particular power budget. These less active or inactive cores are termed as Dark Silicon[8,9]. Dark silicon leads to a low utilization of chip resources, causing economical chip-level performance to plateau.

Recently, DSAs have gained a lot of traction. Compared with traditional hardwired ASIC accelerators, DSA often offers coarse grained SW programmability based on a unique domain specific abstraction model with an optimal ISA. An optimal ISA language usually includes coarse-grained instructions (operations) so that it can leverage as much the native hardware capability as possible to speed up domain specific applications.

As a rule of thumb, the industry typically seeks 10x to 50x gain for DSA. Here we examine some of the most promising approaches to achieving this gain.

A promising technology that is gaining popularity to boost computing power is through chiplet technology. By packaging diversified domain-specific functions into a single device, chiplet technology enables the flexibility of heterogeneous computing at the device level, thus efficiently adapting to various compute workloads even within a single device.

Further, these architectural trends create great opportunities for hardware software co-design.

1. Recomposition for logical ISA extension: The ISA or the domain specific language (DSL) acts like a contract between hardware and software. The granularity of ISA, such as complexity of ops and data types, is often a trade-off between the accelerator's performance, flexibility, and ease of use (e.g., software optimizations). Once this contract is agreed between all parties, the most important interface (or the top-level functional breakdowns) between hardware and software is settled. After this, sub-functionalities (such as data movement, synchronization, interrupt and message handling) can be refined and negotiated between two sides. Thus, ISA co-design is critical for a DSA architecture to succeed.
   With the fast evolution of AI algorithms, new AI functions/operations may not be supported (or be supported with low efficiency) by the existing DSA instructions. Thus, recomposing the underlying computing/interconnect resources to form a new logical function essentially expands the DSA capability with high efficiency.

2. Dynamic resource sharing and reconfiguration across hierarchies: As advanced packaging technology continues to proceed, heterogeneous chiplets are expected to become dominating architectures in the future. However, current heterogeneous architectures have difficulty orchestrating the workloads among all chiplets. So, this architecture requires industry standard protocols and interface definitions for interoperability between chiplets and interconnects. At the same time, the resources inside chiplets are not readily shareable. Furthermore, the failure of Dennard Scaling leads to Dark Silicon, thus causing computing resource underutilization among chiplets. So it also makes sense to focus on intra-chiplet and inter-chiplet resource composability and resource sharing during co-design.

3. Exploration of large-scale computing granularity and sparsity: As AI models continue to grow profusely, computation requirements imposed by the AI model also dramatically increase. However AI models often contain redundant parameters, leading to wasted computation using precious computing resources. This may be particularly true when looking at large-scale foundation models[10]. Foundation models are often trained on a broad set of unlabeled data such that they can be customized to target a different use case during inference, with minimal fine-tuning, thus leading to parameter redundancy and the wasting of compute resources, not to mention the overhead imposed on the memory subsystem.
   One method to reduce the redundant computation is by pruning the model which adds to sparsity. Additionally, computation in the pruned model can be further reduced with low precision computation, by utilizing data types of int8, FP8, FP4, etc. The enablement of efficient sparse computing requires significant hardware software co-design, focussing on the granularity of sparsity, dynamic vs static sparsity, and balanced vs. irregular sparsity. Hardware software co-design needs to be well aligned with sparsity aware task mapping and resource clustering to avoid underutilized compute resources, and non-portability of applications.

Here, to address the challenges above and explore new opportunities at the same time, we propose a polymorphic architecture via **transformable heterogeneous chiplet architecture**. As a comparison, coarse-grained reconfigurable architecture (CGRA)[11,12] may offer a certain level of flexibility across different applications domains, but it doesn't have sufficient heterogeneous chiplet level hardware efficiency to address AI application parallel patterns[13,14] and doesn’t have dynamic composability of computing resources across different heterogeneous cores. Without such critical composability we won’t be able to realize the spatial and temporal scalabilities as needed.

3.1.1 Beyond CSP and Amdahl’s Law

Communicating Sequential Processes (CSP) language constructs, views software as a collection of threads passing messages to each other. Contemporary host processors are generally architected more for parallel processing and less for sequential processing, e.g. FPGAs and GPUs. Similarly, Amdahl’s Law says that in a symmetric architecture the maximum throughput is determined by the parallel task with the longest latency. However, Amdahl does not consider heterogeneous computing where task swapping between processors can level up the task time, nor does it consider the cost of task to task communication. Heterogeneity in processors and software-defined-networking (SDN) adds new levels of complexity to computing. CSP can provide a foundation for AI HW-SW Co-Design, but implemented in hardware as Communicating asynchronous State-Machines (CaSMs). Programming Protocol-independent Packet Processors (P4) is a domain specific language that would be well suited for implementation of these CaSMs.

To obtain maximum performance, software can be written in a way with the maximum number of separate computing elements (CSPs/CaSMs), with maximum channels of communication. It is then the job of compilers and the runtime system to collapse that into the available hardware - making things sequential and multiplexing communication. Like hardware, parallel/distributed

software has critical paths which need to be allocated resources appropriately for best performance.

The CaSM paradigm maps to regular Object Oriented (OO) programming by considering messages as being equivalent to method calls on the CaSM object. CaSM objects may be anything in scale from a logic gate, chiplet, accelerator, to a remote computer. A challenge is the lack of ability to reason about the individual channels of communication between the computing components, that would allow, for example, reallocation of resources for the best latency/throughput/power trade-off.

Tools for handling hardware software co-design with the CaSM paradigm would be usable for polymorphic architecture in AI. In particular there would be compilers for different blocks (for different accelerators) within the AI and Network on Chip/Software Defined Networking (NoC/SDN) compilers for communication, along with simulators or monitoring to evaluate results. From a resource management perspective, the blobs of compute in a large (CaSM) application can be mapped to containers in an SDN networking space, which makes control through existing tools like Kubernetes possible.

3.2 Computation Scale-out

Future AI applications, with scale up to tens or hundreds of trillions of parameters and peta-scale data size, should likely drive computation to scale out to many thousands of cores at the chip level and thousands of chips at the system level. This trend has manifested itself as the compute architecture moves from chip multiprocessor (CMP) architectures and multi-core towards many-core and kilo-cores. Considering these many cores as networked processing units, intra-chip interconnect via Network on Chip (NoC) and inter-chip interconnect between the processing engines (PE) become critical in terms of network scaling efficiency, performance, and power consumption.

Many studies have focused on achieving low latency and power efficiency in interconnect itself. In the era of composable computing, we believe that the focus will go much further. The ultimate form should make compute resources across a very large number of PEs sharable and adaptive such that these PEs can be dynamically transformed into many smaller DSAs for concurrent execution of many small applications or into a single combined DSA which a large application can run on.

3.2.1 Intra-Chip Interconnect

NoC components can play major roles in latency and power consumption, often responsible for parallelism degradation in the multi-core architecture due to cross-core data dependency or a limited power budget shared between NoC components and execution cores. In addition, with variable communication demands on the underlying NoC, different traffic loads may lead to unbalanced consumption of the NoC resources (such as bandwidth and buffers). These create a perfect opportunity for hardware software co-design to target a flexible NoC architecture.

For example, previous work[9] demonstrated a NoC architecture that connected many clusters of cores via a 2D mesh structure. This approach not only helps reduce the number of routers for interconnection, but also explores dark-silicon-aware mapping to allocate active cores along with dark cores across the clusters. With shared communication resources within a cluster, it can achieve a limited bandwidth reconfigurability while meeting the power constraints (through power gating different components of cores), ultimately mitigating the performance degradation due to dark silicon.

NoC also expands computation to effectively a much larger silicon area, forming a wafer scale compute engine. However, traditional electronic NoC structures suffer significant signal loss when extending beyond certain physical distances. Silicon-photonics or low loss graphene nanotube based optical NoC address these challenges head on and may become the dominating physical layer connectivity in the near future.

With the polymorphic AI architecture, we'll explore the intra-chip interconnect topology, communication patterns, and more fine-grained resource sharing to enable flexible composition. At the same time, we'll exploit the emerging interconnect architecture, such as optical NoC[15], for higher bandwidth and lower latency to make the polymorphic architecture more viable.

3.2.2 Inter-Chip Interconnect

As previously described, AI has a complex set of requirements. Scale-up of compute within a single chip is insufficient to meet the large-scale computation demands without experiencing exponential growth of power consumption. Thus, a pool of compute nodes is often needed to scale out computation beyond a single server. Each of the nodes in the pool can be a DSA, which itself typically is scaled out with many parallel processing cores.

Compute pooling has gained traction due to its ability to ease the growing compute pressure by AI applications. This paradigm has excellent parallels with the hierarchy of AI computational tasks: scale-up suits closely coupled high density compute needed for gradient descent, inference and update cycles, whereas scale-out fits neatly with the demands of loosely coupled design space exploration requirements of hyperparameter search.

At the same time, data centers are also changing. Instead of the traditional pools of multi-purpose servers,the disaggregation has advanced the idea of specialized resources such as a "Compute Rack" or an "AI Rack" and is being deployed in data centers to support load matched resource pooling. This type of disaggregated design breaks the inflexibility of resource sharing in traditional servers, significantly improving the overall resource utilization.

Computation scale-out via resource pooling is essentially the same pattern as many-core scale-out at the chip level. Both share the same challenge of how to make resource interconnects efficient. Large-scale resource pooling is even more challenging because the physical limitations such as distances between servers cause additional latency and power concerns.

Current interconnect technology mainly focuses on high bandwidth and low latency, without much attention on interconnect reconfigurability and composability. With a hope of improved latencies and bandwidths for large-scale interconnect in the future, both chip-level and system-level scale-out can be co-designed to achieve efficient polymorphism across multiple hierarchies.

4 A System Engineering Perspective

Historically, computing architecture has evolved from stand-alone computers that are “scaled-up” to multiple processors, typically running single applications in a coherent memory space, to “scaled-out” Beowulf Clusters, either running different applications and/or sharding a single application across multiple nodes.

Today we see a mixture of scale-up and scale-out that includes many types of heterogeneous accelerators. These lead to a vision of ideal software composable rack scale systems, allowing for scale-up and scale-out domain-specific accelerators (DSAs) to be easily built for any given application.

Yet, with all this software-driven, flexible, and composable innovation, we appear to have entirely neglected the simple laws of physics - the fact that “data movement consumes significant energy.” This is the fundamental issue that all solutions should squarely focus on.

Ergo, a DSA means exactly that a computing machine has the sole purpose of being the most efficient computer for that single application. Any form of generalizing the architectural solution with software will impact the application's performance/watt/cost ratio.

DSAs at the system level can provide several orders of magnitude of increased performance and/or savings of energy and cost, especially when considering the large-scale AI machines that are rapidly emerging.

When architecting a DSA, one should adhere to the following principles:

1. When data is idle, don’t burn any power.
2. Don’t move the data unless you absolutely have to, i.e., keep data and compute as close as possible.
3. If you must move the data, move it as efficiently as possible.

To be fair, when we look at the Cambrian explosion of heterogeneous accelerators, we can see the above guiding principles being adhered to. This is now being nicely extended with heterogeneous chiplets within a package. But this now needs to also be extended to the system architecture level.

Another way to look at this is that we see DSA characteristics repeating themselves recursively, like fractals - i.e., we see DSA at the heterogeneous silicon level, and again at the heterogeneous chiplet level. We now need to create the DSA at the system level. This pattern

holds at the system level and can also be segregated into its own set of fractal layers from a node level, to a rack scale level, all the way to a data center level, or greater.

4.1 System building blocks

When considering system-level DSAs, we have three fundamental building blocks to focus on:

1. Compute - including Processors, Heterogeneous Accelerators
2. Memory - including Cache, HBM, DDRx, SSDs, HDDs, Tape
3. Interconnect - Including copper/optical links and switches

Any system-level DSA will precisely configure these building blocks to ensure the most performant and energy efficient solution at the lowest possible cost. With the expanding set of options under each of these categories, the architectural options can be virtually infinite.

For example, classic HPC workloads like QCD, weather modeling, CFD etc have yielded the classic 3D Torus DSA centered entirely around the compute requirements with relatively little memory or I/O beyond nearest neighbor interconnect. AI problems, on the other hand, have a streaming dataflow topology that can be narrow or wide depending on the different layers of the neural network. They increasingly require larger memory capacities and bandwidths.

Architectures in the data center have continued to rely on top-of-rack networking strategies that can support a wide variety of computing and data workloads, but at a penalty of costly system overhead - the antithesis of DSA.

With the upcoming CXL[16] revolution promising software composability down to these three primary building blocks, allowing us to logically build a DSA, we run the danger of getting none of the benefits! The promise of full software composability may end up yielding far higher power, higher latency, poorer performing solutions which will have forgotten the entire intent of DSA in the first place.

The only real solution is to go back to basics and build an extremely flexible, modular architecture consisting of the three fundamental building blocks that can be easily configured with one another in myriad combinations. These building blocks can then be scaled up into large-scale DSA systems. To achieve the lowest cost and power, this modular architecture will need tight packaging, while still being easily configurable into any DSA implementation. These are the core ideas behind the Polymorphic Architecture in this whitepaper.
The activities in the OCP HPC subproject have taken these system-level DSA requirements and conceptualized a potential solution called the HPC Module (HPCM)[25] to address this need.

5 Memory Hierarchies and Adaptability

As AI models and datasets grow larger, the amount of data to be processed during AI training or inferencing may exceed terabyte or even petabyte scales. Loading data from memory/storage

and moving data across compute elements is costly in terms of both performance (interconnect bandwidth requirement) and energy. Minimizing data movement requires a new memory and storage architecture where compute elements can efficiently share often-duplicated data in AI training.

5.1 Intra-Chip Memory Performance Challenges

The memory wall has long caused performance issues in traditional architectures due to the unbalanced development of computing and memory speeds. CPU and DSA efficiencies are declining due to reduced memory capacity and bandwidth per core. Various techniques like cache hierarchies, data prefetching, and recently high bandwidth memory (HBM) have aimed to improve memory bandwidth and latency.

With computation scaling across many cores, memory blocks in a chip may be distributed exclusively for individual cores, shared among all cores, or a mix of both. Complicated coherency issues arise when these memory blocks couple with multiple cache levels.

In the polymorphic AI architecture, compute elements comprise modules and chiplets within a module. Therefore, composability of compute elements requires memory and storage resources to also be composable to meet capacity and performance requirements of the assembled system.

Currently, interconnect technologies between memory blocks in a chip package are maturing (e.g., UCIe[17] and BoW[18]) and beyond with CXL. Polymorphic architecture can build on these standards for composability and efficient sharing of memory resources.

5.2 Inter-Chip Memory Pooling and Sharing

Data storage capacity in a chip is limited to the state of the art in density techniques, which is still far short of system level capacity needed for heavy workloads such as AI/ML. Building an extended memory capacity with high bandwidth at the system level is essential to reduce the memory wall bottleneck for very large AI data sets. Efficient peer-to-peer memory sharing across domains reduces isolation challenges such as stranded memory and fragmentation. Hosts can have a coherent copy of shared regions or portions in the host cache. Sharing (pooling) memory among hosts increases data flow efficiency and improves utilization. A futuristic DC/HPC environment requires a composable, disaggregated memory resource pool with sharing and pooling capabilities.

CXL technology enables such capabilities, for example, defining cache coherency mechanisms.

A quick look at some of CXL (3.0) capabilities for memory sharing and pooling include:

1. CXL fabric scaling up and out
2. Features like port-based routing, peer-to-peer communication, and back-invalidate to support pooling/sharing
3. Tiered memory pools:
    a. Local scope: DRAM on motherboard (<100 ns latency)
    b. System scope: local CXL memory in chassis (<200 ns latency)
    c. Rack scope: disaggregated global memory in rack (<600 ns latency)
4. Supporting various endpoint device types:
    a. MHD: multiple headed device
    b. MLD: multiple logical device
    c. GFAM: global fabric attached memory

Built on CXL capabilities, the proposed polymorphic architecture will leverage hardware-software co-design in the control plane to dynamically optimize system resources through composability and in the data plane to minimize data movement and optimize efficiency. Other aspects of this architecture will leverage methods like architecture-aware compute partitioning, which will be explained later, and enable storage pools to preprocess and filter data before moving across the fabric to greatly improve network efficiency and reduce preparation at compute nodes. Further aspects, such as optimization to storage pool management, enables own data reduction and virtualizes storage for compute node schedulers.

6 Software Ecosystem

In the new paradigm of composable computing and application co-design, the software ecosystem linking an application to hardware must evolve to meet emerging challenges and opportunities. Currently, scale-out HPC server computing faces significant challenges integrating accelerators, containerization, and coping with legacy software like schedulers, monolithic kernels, file systems, and performance monitoring/telemetry originally developed for legacy systems. The HPC and cloud communities have, through experience, learned and implemented many lessons, but these learnings have not necessarily been fully integrated back into the system stack.

For inter-device and system synchronization, the lack of a heterogeneous programming model has significantly hampered current scale-out HPC implementations [20]. As we increase device diversity and interconnect quality, we must support adoption with improved programmability models and standards designed from the start, not as an afterthought.

Significant strides have been made scheduling and porting workloads [21,22], including integrating software-defined networking and topology, accelerator-aware scheduling, and power consumption considerations. However, we must go beyond simple virtualization for much greater integration between hardware and workload management that is needed to support truly

dynamic polymorphic composition for creating the efficient exascale AI platforms to power us forward.

Today, one major paradigm holding back progress is how we think about operating systems. There has been much evolution but no revolution in the last 2 HPC system generations. OS kernel multi-threading is still under development for many popular HPC OSes, let alone thinking about multiple threads running on multiple systems. Increasingly, we have added more layers to the HPC userland system stack to provide basic system-wide services that should ideally be built into the system-wide OS.

While many challenges remain, the industry has been working on solutions built on lessons learned. The demands of AI and the shift to a composable co-design paradigm offer the opportunity to implement these best-of-breed ideas from the ground up in this new generation of systems.

6.1 AI Ecosystem and Polymorphic Architecture

Currently, much of the AI industry and research focuses on coarse grained optimizations such as through training Foundation Models [10] or MLPerf Training and Inference benchmarks to drive hardware improvements to increase workload performance. These constraints tend to lead to large-scale systems with uniform hardware optimized for training specific models using specific frameworks. While other models may see improvements running on systems optimized for Foundation Models or MLPerf, the hardware and software could often be better organized and optimized for data scientists exploring new models for their problem domains. Some users may want concurrent AI training jobs running the same or different models to explore approaches to increase accuracy.

The AI ecosystem includes algorithms, frameworks, and infrastructure like compute, interconnects, and memory.

- Framework software and interfaces specify composable operators and formats for AI models, compile and target execution based on underlying hardware capabilities and overall system topology. Some system components traditionally considered the "operating system" or "language runtime environment" are responsible for moving work to AI hardware based on the AI compiler and framework output.
- AI infrastructure includes specialized compute devices to accelerate common AI operations in training or inference. These may use traditional digital designs or novel analog compute implemented in different process technologies or media. Additionally, AI infrastructure provides compute-to-compute (potentially coherent), compute-to-memory, peer-to-peer memory-to-storage, and switch connectivity to increase scalability.

As AI models scale up and/or scale out (example: Mixture of Experts) and new hardware implementations and technologies are applied to AI workloads, each of these areas will tend to evolve independently of most other factors for incremental improvement. For example:

algorithms will continue developing with existing or slightly modified models on ever-increasing datasets. Meanwhile, frameworks will improve known model performance and identify opportunities to more efficiently use existing hardware in response to model changes or new operators. Similarly, AI hardware may introduce new instructions, data types, or methods to utilize hardware differently.

A polymorphic architecture provides an environment where hardware is aware of software and software is aware of hardware and all hardware and software components recognize system level capabilities. It offers an implicit level of abstraction that allows future component versions to remain compatible while taking advantage of new beneficial features. All parts share information about capabilities and connections. With multiple models potentially running concurrently, feedback from each component about utilization or data will inform scheduling and placement optimization of other executing models. The system might select different scheduling policies to optimize for external constraints (e.g. overall power usage) and aggregate memory bandwidth available for AI workloads.

6.2 Ecosystem Goals for Polymorphic System Architecture

Co-design enables the ecosystem to explore new software algorithms more closely fitting problem domains and leveraging the latest AI hardware capabilities and accelerator devices. The AI software framework can inquire about compute and AI hardware architecture capabilities to determine allowable instruction and capability combinations for expressing model operators and when to fuse operators. The overall system topology can also be dynamically mapped to compose hardware resources according to specified constraints or policies described by applications (e.g. power, bandwidth, etc.) There are also new levels of integration unleashed such as heterogeneous ISA within a system and so on.

A novel software framework can incorporate polymorphic system capabilities and discern where new hardware instructions will improve training or inference efficiency. It will ascertain application characteristics at runtime (e.g. sparsity, reduced precision capability, etc.) and perform optimizations to reduce data footprint and required bandwidth to move data. With memory utilization telemetry, it may augment memory resources based on relative distances between memory and compute as well as peer-to-peer data movement between memories.

The polymorphic architecture's success relies on building a collaborative ecosystem focused on customer requirements spanning different models and problem domains. Both software and hardware should share information to leverage specialized hardware components and interconnects as needed, dynamically adapting to global system behavior to ensure predictable performance within infrastructure resource constraints.

Discussion in the OCP AI HW-SW Co-Design Workstream will decide responsibilities, division of labor, and specifications to enable the Polymorphic System Ecosystem to work towards the common goal.

7 Polymorphic Architecture

Future AI applications will have diverse computational demands, requiring flexible scalability across domains as well as across multiple generations of algorithms. As domain-specific architectures continue to evolve with improved efficiency, we need a new computing paradigm with agility and adaptability to meet the needs of AI applications. In this section, we leverage a conceptual top-down, application-driven co-design approach to enable a general framework of polymorphic architecture. The essence of this polymorphism includes the following important aspects:

- **Disaggregation and pooling:** The base compute units in this architecture are composed of interconnected heterogeneous components, such as processors including ASICs and/or chiplets, a common interconnect fabric, and a shared local memory pool. These compute units are similarly arranged into a compute node with a local interconnect fabric and shared memory layers. Finally, multiple compute nodes are aggregated into a pod with a configurable network fabric connecting the top tier of memory and storage. Importantly, the function units inside a processor/chiplet should also be organized similarly (e.g. via CGRA). This facilitates composability at multiple levels of computation, from chiplet to chip, system, and clusters of servers.

- **Composability:** Using pools of disaggregated compute units, memories, and interconnect fabrics, a large array of logical processors can be dynamically created and reconfigured on demand to meet the computation needs of the assigned workload. For example, dynamically combining different computing resources from heterogeneous processors and chiplet DSA pools can broaden programmability to target a much wider range of workloads (even if a particular DSA itself lacks general programmability). By scaling composability across multiple hierarchies, this architecture can enable a form of fractal computing with great agility to enable adaptability and transformability.

- **Adaptability and transformability:** The dynamically composed logical processor can be redefined and adapted specifically for the application (large or small). The runtime software needs to understand the application behavior and insights to guide how the available architectural resources should be transformed. The goal is to enable the logical processor to execute the application as efficiently as a purpose-built ASIC processor tailored for that exact application.

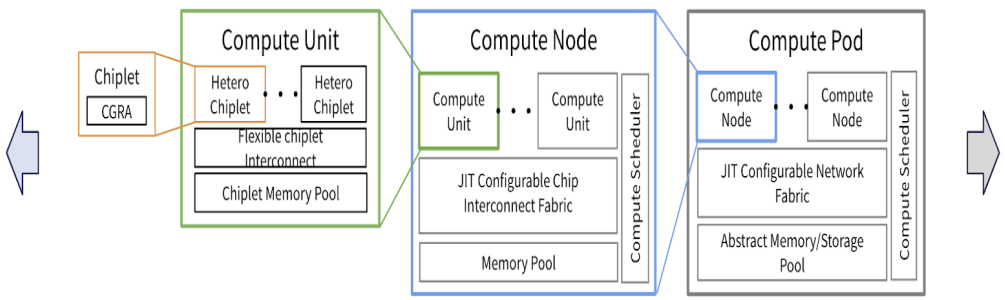

Figure 3: Polymorphic architecture with heterogeneous composability across multiple system hierarchies. The combination of heterogeneity, composability, and transformability are essential enablers for polymorphic architecture.

Figure 3 shows the proposed architectural diagram to enable polymorphism for AI applications. From processor/chiplet to compute unit, system compute node, and all levels beyond, architectural resources (compute, memory, interconnect) have the ability to be partitioned, shared, and re-composed to form new computing capabilities. With sufficient flexibility and efficiency, polymorphic architecture can enable a form of fractal computing, with great adaptability and transformability.

In this paper, we focus discussions on compute organizations at the chip and system levels. The transformability can be easily extended into higher hierarchies.

7.1 Architecture for Adaptability

Computation for AI algorithms can generally be described as an execution graph, where nodes represent computation and edges represent data flowing between nodes. Computation at each node may be further divided into a more fine-grained execution graph. Thus, one of the most important aspects in hardware-software co-design is balancing the granularity of computation, maximizing software performance while maintaining hardware flexibility and power efficiency. The coarser the computational granularity, the more efficient the hardware architecture, however this comes at the cost of flexibility. And just as flexibility can be improved by increasing the fineness of computational granularity, this will come at the cost of computational efficiency.

There are a few approaches to designing the polymorphic microarchitecture at the chip level:

- The first is general computing with massive thread-level parallelism (TLP), augmented with domain-specific acceleration. A typical example in this category is Nvidia's Hopper GPU which includes powerful domain-specific tensor cores and structural sparsity support. Compared to conventional TLP architectures, polymorphic architectures have the ability to dynamically compose (fuse) and decompose functional units at the hardware level.

- The second approach is dataflow-based accelerators, such as the systolic array in Google's TPU. The essence of interconnected and disaggregated computing in polymorphic architectures makes it natural to work with computational flow graphs. Furthermore, since the computation graphs in AI training and inference are relatively deterministic, the configuration of computational resources (compute units, memory access patterns, and interconnect topology) can be explicitly redefined (or "nearly hardwired") per computation graph or subgraph.

The properties of AI computation graphs also enable dynamically composing a logical processor tailored to a particular subgraph or coarse-grained operator. With recent advancements in interconnect technologies (e.g. optical network-on-chip), bandwidths and latencies for inter-chiplet data transfer are approaching those of on-chip data movement. Thus, it is feasible to compose a new logical processor to execute a previously unsupported operator through dynamic resource sharing and reconfiguration among chiplets. More coarse-grained logical processors can be further composed into even larger processors, and so on (i.e., fractal computing).

Therefore, even though the set of AI model operators (like Open Neural Network Exchange (ONNX) operators) has not yet converged, it is still beneficial to define a common set of operators based on hardware-software co-design best practices. We can then leverage the composability of the architecture to expand programmability.

In addition, emerging computing paradigms (such as processing in memory, neuromorphic computing, photonic computing, etc.) can also be adopted in microarchitecture co-design. For example, PIM accelerators integrate processing elements with memory technology. Among these PIM accelerators, analog computing leverages in-place analog multiply-add capabilities with augmented flash memory circuits. Considering computation as processes that traverse data performing operations, rather than processors that have data brought to them, PIM architectures gain performance when dispatching work is cheaper than moving data.

Another aspect in polymorphic architecture co-design is taking advantage of computation sparsity in AI models. FPGAs are incapable of exploiting model sparsity because they are too fine-grained and reconfiguration latency is too high.

Finally, power efficiency is crucial for future AI applications. Through adaptability and transformability, unused resources (dark silicon) can be powered down after architectural reconfiguration. Dark silicon can be dispersed among active computing resources to mitigate thermal issues, thus improving the power budget[9].

7.2 Composable Memory Architecture

AI training and inference applications require significant memory capacity to store input data, intermediate results/activations, and model data itself. During execution, AI workloads are dispatched to the compute units within each node, and appropriate data is transferred between the node's local memory and underlying memory hierarchies. The amount of data transferred depends on the compute node's throughput requirements.

Since each compute node is composable, the memory structure should also be composable to meet the needs of a dynamically constructed node. Thus, memory pooling is proposed to offer flexible allocation of memory capacity and bandwidth.

To make memory pools efficient, memory architecture co-design must account for key application characteristics in AI:

1. Huge data consumption (large memory footprint)
2. Massive data transfers (high bandwidth needs)
3. High-speed data movement (low latency)

Here are some points worth exploring during co-design:

- Memory processing engine (MPE): Memory access latency critically impacts computation performance, so offloading memory access to a dedicated on-chip acceleration engine is essential (similar to the DPU concept in data centers). In addition to accelerating data transfers, MPEs can support compression, encryption, and domain-specific operations like reshaping and concatenating data. That is, MPEs can integrate basic **near-data computing**.

MPEs are agnostic to heterogeneous computing architectures and can support tiered memory as needed. This aligns with emerging memory accelerator interface standards like Smart Data Accelerator Interface (SDXI)[23].

- Memory coherency: Coherent memory access may be required across memory hierarchies. Protocols like CXL support coherent access between accelerators and memory pools. However, maintaining efficient coherent access across composable memory pools is challenging.

- Dynamic bandwidth reconfiguration: When parts of a memory pool are inactive, their bandwidth can be released and shared with other pool sections or pools. Similar to dark silicon optimizations in computing, unused memory resources can be dispersed to reduce conflicts among active resources (e.g. memory banks).

Additionally, the following issues should be addressed when implementing memory pools:

- Define pooling APIs for new memory topologies and data movement paradigms (e.g. RDMA, CXL)

- Develop abstractions to achieve composability and QoS allocation goals:

    - Define memory resource properties: capabilities, latency, topological connectedness
- Support tiered memory at different grades

- Enable topology-aware dynamic memory allocation and composition
    - Make memory an active rather than passive device

- Define APIs for moving computation closer to memory/storage
    - Computational storage
    - Virtual vs. physical composability

Some of these issues are already being addressed in open standards[24] and industry activities[19]. They should be incorporated during AI co-design.

7.3 Computation Partitioning and Co-Optimizations

A key part of polymorphic architecture is enabling deep and tight interaction (cohesion) between application software (algorithms) and hardware architecture. Figure 4 shows a proposed execution flow to run AI applications on a polymorphic architecture:

1. **Application characterization:** An AI application has an attribute description that specifies its requirements for throughput, latency, power budgets, QoS, etc. The description can be generated by an offline AI compiler and runtime framework, or from online learning.

2. **Resource description:** This describes the currently available resources to execute the AI application. It may include compute capability, resource configurability, topology, interconnect bandwidths and latencies, memory capacity and hierarchy, power constraints, etc. The runtime system (e.g. task scheduler) should maintain the resource description.

3. **Resource provision predictor (RPP):** The RPP consumes the application and resource descriptions to generate a reconfiguration profile. This guides the **Composition Controller (CC)** to dynamically compose the underlying computing resources into a new logic processor. RPP also generates an application execution plan, including compute partitioning (task breakdowns) and mapping scheme. The execution plan dispatches computation tasks (e.g. AI computation graph) to the newly composed processor. The mapping scheme efficiently utilizes hardware resources, potentially shared with concurrent tasks from other AI applications. RPP may be a dedicated ASIC accelerator, using a pre-trained AI model for prediction or online learning for more accurate predictions.

4. **Composition controller (CC):** Composes hardware resources as specified by the RPP predictor, satisfying resource provision constraints or policies.

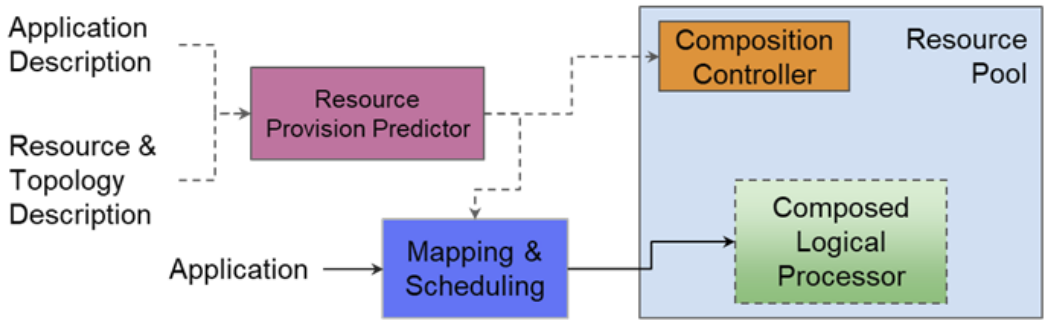

Figure 4: AI application execution flow on the polymorphic architecture

When characterizing AI applications, there are opportunities for more co-design ideas to achieve better performance or efficiency. For example, sparsity and reduced precision optimizations can reduce the application's data footprint and bandwidth needs while meeting accuracy goals. Similarly, an architecture-aware AI compiler can utilize allowable operator combinations to better express model computation and make fusion decisions.

Goals may be similar but require different optimizations such as hardware provision scenarios during software co-optimization, with different strategies:

1. **Limited resource provision:** Optimize the application and compose limited hardware to maximize performance under power, QoS, etc constraints.

2. **Unlimited resource provision**: Identify the application's maximal parallel execution potential (ultimately limited by Amdahl's Law) to maximize performance/watt/dollar. Don't provide extra resources if performance stops scaling due to things like interconnect latency limits.

8 Conclusion

Artificial intelligence has become increasingly democratized in our daily lives. This is evident from the recent popular application ChatGPT, which reached 100 million daily active users (DAUs) in just two months. As AI algorithms (models) grow more complex, some, for example, exceeding 10 trillion parameters or expansive numbers of parameters spanning multiple models, the computation demands on AI infrastructure are unprecedented. As explained earlier, the 2-dimensional evolution of AI applications, caused by across-domain divergence (spatial) and across-generation divergence (temporal), brings unparalleled challenges to AI computing architectures.

In this paper, we proposed a polymorphic architecture based on hardware-software co-design to address these challenges. With scalability and transformability, polymorphic architecture adapts to running application characteristics, dynamically reconfiguring itself into a new logical accelerator whose throughput is effectively similar to a purpose-built ASIC.

Polymorphic architecture leverages architecture trends like multiple ISAs found across processor types, domain-specific accelerators, chiplets, and fast interconnects to achieve the required composability and transformability. Through dynamic resource partitioning and sharing, it obtains re-composable resources for high efficiency while overcoming DSA's inflexibility. It also utilizes new opportunities from AI applications' characteristics (high parallelism, determinism) during co-design to enable fast recomposition.

The AI hardware-software co-design principle can be applied at various scales. For example, AI can create fast models of low-level hardware for use in higher-level simulations. In OCP ODSA,

this could unleash the capability to rapidly generate fast Chiplet models from contained IP blocks.

To realize the proposed Polymorphic Architecture co-design, we need more minds working together to envision the next level of hurdles and unleash the required innovation, so this white paper also serves as an open call for more participation in the OCP AI Co-Design workgroup. Please come and join our effort to define this new computing architecture. Together, we can solve the problems mentioned above and make a real difference for humanity.

9 Glossary

AI	Artificial Intelligence
CXL	Compute Express Link
UCIe	Universal Compute Interconnect Express
SDXI	Smart Data Accelerator Interface
BOW	Bunch of Wire
OCP	Open Compute Platform Foundation
CGRA	Coarse Grained Reconfigurable Architecture
DSA	Domain Specific Architecture
NOC	Network-on-Chip
Fractal Computing	A compute architecture with dynamic resource sharing and recomposition to enable general efficient computing at every level of computing granularities

10 References

1. David Kanter, *MLPerf 2Q22 Briefing,* MLCommons 2022
2. A. Reuther, P. Michaeleas, et al, “AI Accelerator Survey and Trends”, IEEE High Performance Extreme Computing Conference (HPEC), 2022
3. S. Zhu, T. Yu, et al, “Intelligence computing: the latest advances, challenges, and future”, Intelligent Computing Vol 2, Jan 3, 2023, DOI: 10.34133/icomputing.0006
4. M. Horowitz and K. Keutzer, "Hardware-software co-design", *Proc. SASIMI*, pp. 5-14, 1993.
5. D. Thomas, J. Adams and H. Schmitt, "A model and methodology for hardware-software co-design", *IEEE Design and Test*, vol. 10, no. 3, pp. 6-15, Sept. 1993
6. Y. Zhu, “Maximizing GPU utilization in large-scale machine learning infrastructure”, GTC 2022
7. W. Wulf, S. McKee, “*Hitting the Memory Wall: Implications of the Obvious”*, ACM SIGARCH Computer Architecture News, Vol 23, Issue 1, 1995
8. H. Esmaeilzadeh, E. Blem, et al, “Dark silicon and the end of multicore scaling”, International Symposium on Computer Architecture (ISCA), 2011, San Jose
9. M. Hoveida, F. Aghaaliakbari, et al, “Efficient mapping of applications for future chip-multiprocessors in dark silicon era”, ACM Transaction on Design Automation of Electronic Systems, Vol 22, No 4, 2017
10. Bommasani R, Hudson DA, et al. “On the Opportunities and Risks of Foundation Models”, arXiv:2108.07258 https://arxiv.org/abs/2108.07258. Jul 2022 (v3)
11. W. Hartenstein, A. G. Hirschbiel, M. Riedmuller, and K. Schmidt, “A novel ASIC design approach based on a new machine paradigm”, *IEEE J. Solid-State Circ.* 26, 7 (1991), 975–989.
12. L. Liu, J. Zhu, et al, “A Survey of Coarse-Grained Reconfigurable Architecture and Design: Taxonomy, Challenges, and Applications”, ACM Comput. Surv., Vol. 52, No. 6, October 2019, DOI: https://doi.org/10.1145/3357375
13. S. Rashidi, S. Sridharan, et al, “Enabling SW/HW co-design exploration for distributed DL training platforms”, IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2020
14. N. Jouppi, G. Kurian, et al, “TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings”, International Symposium on Computer Architecture (ISCA), 2023, Florida
15. H. Meng, “Optical Network-on-Chip for large scale chiplet architectures”, the 2nd International High Performance Chiplet and Interconnect Architectures (HiPChips) in HPCA-2023, Montreal, 2023
16. Compute Express Link, https://www.computeexpresslink.org/
17. Universal Chiplet Interconnect Express, https://www.uciexpress.org/specification

18. S. Ardalan, H. Cirit, et al, “Bunch of wires: an open die-to-die interface”, 2020 IEEE Symposium on High Performance Interconnects (HOTI), DOI: 10.1109/HOTI51249.2020.00017
19. OCP FTI, Data Centric Computing, fka Computational Storage, https://www.opencompute.org/projects/data-centric-computing
20. S. Ashby, P. Beckman, et al, “The opportunities and challenges of exascale computing”, Summary Report of ASCAC Subcommittee, DoE Office of Science, 2010
21. D. Narayanan, K. Santhanam, et al, “Heterogeneity-aware cluster scheduling policies for deep learning workloads”, 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2020
22. T. Hoefler, T. Bonato, et al, “HammingMesh: a network topology for large-scale deep learning”, SC '22: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, November 2022
23. Smart Data Accelerator Interface (SDXI), https://www.snia.org/sdxi
24. OCP Future Technology Initiative (FTI), Software Defined Memory workstream, https://www.opencompute.org/wiki/OCP_Future_Technologies_Initiative/Software_Defined_Memory
25. A. Cantle, “CXL Enablement with High Performance Compute Module (HPCM)”, CMS Tech Summit 2023, OCP Subproject https://www.opencompute.org/wiki/HPC

11 License

OCP encourages participants to share their proposals, specifications and designs with the community. This is to promote openness and encourage continuous and open feedback. It is important to remember that by providing feedback for any such documents, whether in written or verbal form, that the contributor or the contributor's organization grants OCP and its members irrevocable right to use this feedback for any purpose without any further obligation.

It is acknowledged that any such documentation and any ancillary materials that are provided to OCP in connection with this document, including without limitation any white papers, articles, photographs, studies, diagrams, contact information (together, “Materials”) are made available under the Creative Commons Attribution-ShareAlike 4.0 International License found here: https://creativecommons.org/licenses/by-sa/4.0/, or any later version, and without limiting the foregoing, OCP may make the Materials available under such terms.

As a contributor to this document, all members represent that they have the authority to grant the rights and licenses herein. They further represent and warrant that the Materials do not and will not violate the copyrights or misappropriate the trade secret rights of any third party, including without limitation rights in intellectual property. The contributor(s) also represent that, to the extent the Materials include materials protected by copyright or trade secret rights that are owned or created by any third-party, they have obtained permission for its use consistent with the foregoing. They will provide OCP evidence of such permission upon OCP’s request. This document and any "Materials" are published on the respective project's wiki page and are open to the public in accordance with OCP's Bylaws and IP Policy. This can be found at http://www.opencompute.org/participate/legal-documents/. If you have any questions please contact OCP.

Footer:

12 About Open Compute Foundation

At the core of the Open Compute Project (OCP) is its Community of hyperscale data center operators, joined by telecom and colocation providers and enterprise IT users, working with vendors to develop open innovations that, when embedded in product are deployed from the cloud to the edge. The OCP Foundation is responsible for fostering and serving the OCP Community to meet the market and shape the future, taking hyperscale led innovations to everyone. Meeting the market is accomplished through open designs and best practices, and with data center facility and IT equipment embedding OCP Community-developed innovations for efficiency, at-scale operations and sustainability. Shaping the future includes investing in strategic initiatives that prepare the IT ecosystem for major changes, such as AI & ML, optics, advanced cooling techniques, and composable silicon. Learn more at www.opencompute.org.