SSR: Spatial Sequential Hybrid Architecture for Latency Throughput Tradeoff in Transformer Acceleration

Jinming Zhuang
University of Pittsburgh, USA
jinming.zhuang@pitt.edu

Zhuoping Yang
University of Pittsburgh, USA
zhuoping.yang@pitt.edu

Shixin Ji
University of Pittsburgh, USA
shixin.ji@pitt.edu

Heng Huang
University of Maryland, USA
heng@umd.edu

Alex K. Jones
University of Pittsburgh, USA
akjones@pitt.edu

Jingtong Hu
University of Pittsburgh, USA
jthu@pitt.edu

Yiyu Shi
University of Notre Dame, USA
yshi4@nd.edu

Peipei Zhou
University of Pittsburgh, USA
peipei.zhou@pitt.edu

ABSTRACT

With the increase in the computation intensity of the chip, the mismatch between computation layer shapes and the available computation resource significantly limits the utilization of the chip. Driven by this observation, prior works discuss spatial accelerators or dataflow architecture to maximize the throughput. However, using spatial accelerators could potentially increase the execution latency. In this work, we first systematically investigate two execution models: (1) sequentially (temporally) launch one monolithic accelerator, and (2) spatially launch multiple accelerators. From the observations, we find that there is a latency throughput tradeoff between these two execution models, and combining these two strategies together can give us a more efficient latency throughput Pareto front. To achieve this, we propose spatial sequential architecture (SSR) and SSR design automation framework to explore both strategies together when deploying deep learning inference. We use the 7nm AMD Versal ACAP VCK190 board to implement SSR accelerators for four end-to-end transformer-based deep learning models. SSR achieves average throughput gains of 2.53x, 35.71x, and 14.20x under different batch sizes compared to the 8nm Nvidia GPU A10G, 16nm AMD FPGAs ZCU102, and U250. The average energy efficiency gains are 8.51x, 6.75x, and 21.22x, respectively. Compared with the sequential-only solution and spatial-only solution on VCK190, our spatial-sequential-hybrid solutions achieve higher throughput under the same latency requirement and lower latency under the same throughput requirement. We also use SSR analytical models to demonstrate how to use SSR to optimize solutions on other computing platforms, e.g., 14nm Intel Stratix 10 NX.

CCS CONCEPTS

• **Computer systems organization** → **Heterogeneous (hybrid) systems**; • **Hardware** → **Hardware-software codesign**.

KEYWORDS

Heterogeneous Computing, Domain-Specific Accelerator, Versal ACAP, Transformers, Design Space Exploration, Latency Throughput Tradeoff, Deep Learning

FPGA '24, March 3–5, 2024, Monterey, CA, USA

ACM ISBN 979-8-4007-0418-5/24/03.
https://doi.org/10.1145/3626202.3637569

ACM Reference Format:
Jinming Zhuang, Zhuoping Yang, Shixin Ji, Heng Huang, Alex K. Jones, Jingtong Hu, Yiyu Shi, and Peipei Zhou. 2024. SSR: Spatial Sequential Hybrid Architecture for Latency Throughput Tradeoff in Transformer Acceleration . In *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '24), March 3–5, 2024, Monterey, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3626202.3637569

1 INTRODUCTION

Latency and throughput are two crucial performance metrics when deploying deep learning models on various computing platforms. Depending on the nature of the applications and different user expectations, different application scenarios have different latency requirements. For example, the latency requirement in autonomous driving is more stringent than that in video conferencing. The former requires milliseconds or submillisecond latency [1, 2, 3, 4] for a life-critical system whereas the latter has a looser latency requirement of hundreds of milliseconds. Furthermore, throughput is also needed to be considered. For example, in data center services, e.g., Microsoft [5, 6, 7, 8], Google [9], AWS [10], etc, higher throughput means less amount of data center servers and therefore less power consumption for the same workload. On the other hand, it can also support more volumes of users while ensuring real-time user content updates with the same amount of servers. For autonomous vehicles, to safely navigate the changing environments, higher throughput means processing higher amounts of sensor data to make real-time decisions [11].

The two factors are also intertwined and there is a design tradeoff between latency and throughput. In general cases, a system can not get high throughput and low latency simultaneously. If a design requires higher throughput which can be achieved by batching more data, the system would have to sacrifice latency. While users can only explore latency throughput tradeoff by changing the batch size when using the off-the-shelf deep learning framework on GPUs, FPGA accelerators [12, 13, 14, 15, 16] and other tiled accelerators [17, 18, 19, 20, 21, 22, 23, 24] provide more flexibility and users have a larger design space to explore the latency throughput tradeoff.

By using on-chip local scratchpad memory and configurable processing elements, users can design customized accelerators (accs) that fit certain computations, and this is called accelerator (acc) customization. There are different strategies when mapping multiple layers within a deep learning model onto FPGAs or tiled accelerators. One common method is to design one unified acc that can compute different layers within the model graph and the unified accelerator is launched **sequentially** to finish all the layers [26].

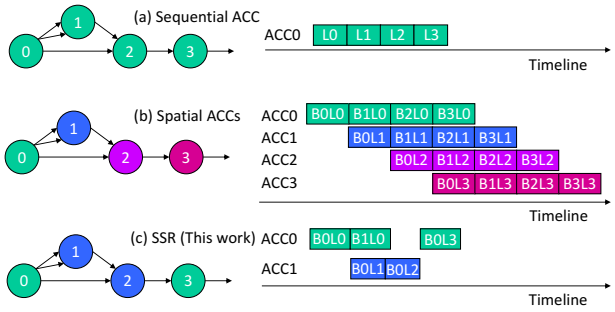


Figure 1: Execution models for sequential, spatial, and our proposed spatial-sequential-hybrid architecture (SSR).

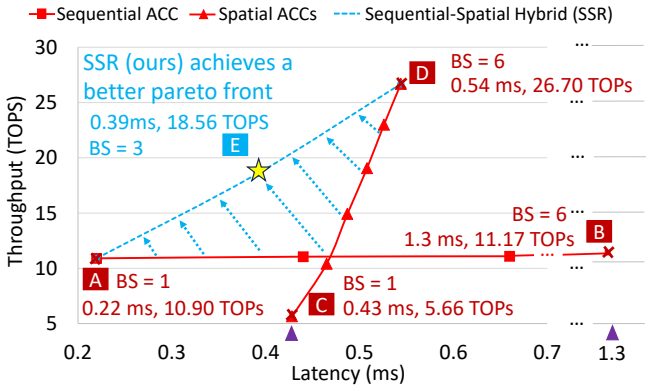


Figure 2: Latency and throughput tradeoff under different strategies for one representative vision transformer model, i.e., DeiT-T [25]**. SSR (ours) achieves a better Pareto front than sequential acc and fully spatial accs designs.**

The execution model timeline is shown in Figure 1(a). Here we use a graph with four layers `0-3` to illustrate. The arrows show the layer dependencies in the graph. Where there is only one acc, `ACC0`, four layers `L0`, `L1`, `L2`, `L3` are launched sequentially while honoring the dependencies in the graph. When users increase the batch size, in the timeline, `L0-L3` will become longer. We also apply the sequential acc design strategy and map one representative deep learning application, an INT8 quantized vision transformer model DeiT-T [25] for image classification task on AMD ACAP VCK190 [27]. We sweep the batch size from 1 to 6 and find the customized monolithic acc that gives the highest throughput under each batch size. We plot the latency and corresponding throughput for each batch size as a 2-D scatter plot and add the trendline as shown in Figure 2. From point `A` to point `B`, the latency increases from 0.22 ms to 1.3 ms. The effective throughput slightly increases from 10.90 TOPS to 11.17 TOPS, which means the sequential acc design strategy achieves 10.9% utilization of the peak INT8 computation performance (102 TOPS) for AMD VCK190. The underlying reasons for such a utilization are: (1) the computation and communication patterns for different layers in DeiT-T vary a lot; (2) there is a huge mismatch between the small matrix multiply layer shape and the huge computation resource. Therefore, the first question arises: *Can we achieve a higher throughput?*

A common solution is to apply an alternative design strategy, i.e., implementing **spatial accs** [8] and mapping each layer with a dedicated specialized acc, i.e., fully spatial acc design. The corresponding execution model timeline is shown in Figure 1(b), where there are four accs, `ACC0-ACC3`. Since there are dependencies between layers 0-3, four layers in the same batch data `B0L0`, `B0L1`, `B0L2`, `B0L3` have to be launched sequentially. As can be observed from Figure 1(b), if there is only one batch, `ACC0-ACC3` will be severely underutilized. However, when there are more batches, e.g., `B1-B3`, the executions for different layers from different batches can be pipelined. Therefore, the utilization of `ACC0-ACC3` is greatly improved. This also matches the trendline in Figure 2 from point `C` with throughput as 5.66 TOPS to point `D` with throughput improved to 26.70 TOPS.

When choosing from these two strategies, sequential vs. spatial, the optimal design varies under different design constraints. For example, in Figure 2, if the latency requirement is 0.43 ms, sequential acc is more favorable than spatial acc as point `A` achieves a higher throughput and a smaller latency than point `C`. This is intuitive to understand. When the batch size is 1, as each spatial acc has a smaller resource than the one monolithic acc, each layer takes longer execution time on separate spatial accs than on one monolithic acc. However, if the latency requirement is 1.3 ms, spatial acc is more favorable than sequential acc as point `D` achieves a higher throughput and smaller latency than point `B`. This is also intuitive to understand. When the batch size is large, spatial accs tend to have better customization and more batches fill the pipeline gaps and improve the utilization. Based on this observation, one follow-up question arises: *Can we combine sequential acc and spatial acc strategies together and gain the best of both worlds?*

Our answer is "Yes". The key idea is to enable more scheduling flexibility to map any layers to any accs where the number of accs can be one to the maximum number of layers. We illustrate such a sequential-spatial hybrid architecture (SSR) in Figure 1(c). In this approach, there are two accs, `ACC0` and `ACC1`. Layer 0 and 3 map to acc0. Layer 1 and 2 map to acc1. By using such hybrid architecture, users can find an even better throughput than sequential acc and spatial acc strategies. For example, in Figure 2, if the latency requirement is 0.43 ms, the SSR hybrid strategy (point `E`) achieves throughput 18.56 TOPS, which is 1.70x throughput improvement than the sequential acc strategy (point `A`) and 3.28x than the spatial acc strategy (point `C`). **The new design points enabled by the SSR strategy constitute a better Pareto front in latency throughput tradeoff.** That is, our SSR sequential spatial hybrid solutions achieve higher throughput under the same latency requirement or lower latency under the same throughput requirement compared with the sequential-only solution and spatial-only solution. In summary, our main contributions are:

- **Design Challenges Analysis:** To understand the performance, we first perform an in-depth kernel profiling of DeiT-T on Nvidia GPU A10G in Section 2. Then we discuss the challenges of exploring latency throughput tradeoff for deep learning applications and propose our design principles.
- **SSR Accelerator and Framework:** We propose SSR accelerator, a novel sequential and spatial hybrid accelerator template, and SSR framework, a programming mapping solution, in Section 4 to leverage the ACAP's heterogeneous components within the same system-on-chip, including FPGA and AIE vector cores.

- **SSR Implementations:** We deploy the SSR framev plore latency throughput tradeoff of four models on ' Section 5. Our on-board experiments demonstrate that ious latency constraints, SSR achieves average throug as 2.53x, 35.71x, and 14.20x under different batch sizes to the 8nm Nvidia GPU A10G, 16nm AMD FPGAs ZC U250. The average energy efficiency gains are 8.51x, 21.22x, respectively.
- **Open-source Tools and Discussions on Mapping** We open-source our tools with detailed guides to rep of the results presented in this paper: https://githul research-lab/SSR. We also discuss mapping insights ir

2 DESIGN CHALLENGES AND PROPOS SOLUTION

Exploring latency throughput tradeoff requires a deep understanding of the performance. To understand the performance of different layers within a deep learning application, we first perform an in-depth kernel profiling by using TensorRT [28] to deploy an INT8 quantized DeiT-T inference on Nvidia GPU A10G. Built with 8nm fabrication, the Nvidia A10G GPU has 72 stream multiprocessor (SM)s with 4 tensor cores per SM, reaching the peak INT8 performance as 140 TOPS and peak FP32 performance as 35 TFLOPS, as specified in Table 1. We profile DeiT-T and set the batch size as 6. The measured end-to-end latency is 1.43 ms. We show the kernel time breakdown in Figure 3. We have the following observations: ① **The matrix-multiply or convolution-type kernel utilization is low.** This includes matrix-multiply (MM), batch matrix-multiply (BMM), and patch embedding, i.e., convolution. We calculate the effective throughput in these layers as 18 TOPS, which is only 13% of the peak INT8 throughput on A10G (140TOPS). ② **The non-linear layers including Softmax, GELU, and LayerNorm take significant GPU cycles.** These layers consume less than 1% of the total computation operations, however, take around 28% of the total time. These layers are mapped to CUDA cores on the GPU. ③ **The data layout change kernel consumes non-negligible GPU cycles, around 8% of the total latency.** The data layout change kernel, i.e., Transpose, is introduced either implicitly as certain data layouts are favorable for GPU Tensor Cores computation, e.g., the least dimension of the tensor is aligned with 32, or explicitly as specified in the model. ④ **The data type conversion kernel Reformat to convert between INT8 and FP32 also consumes non-negligible GPU cycles, around 5% of the total latency.** This happens, e.g., when the FP32 output from Softmax needs to be used as the input of the next matrix-multiply layer.

Table 1: Comparisons between Nvidia GPU A10G and AMD Versal ACAP VCK190 on peak FP32 and INT8 performance, and peak off-chip bandwidth (BW).

Hardware Specification	FP32	INT8	Off-chip BW
Nvidia GPU A10G [29]	35 T	140 T	600 GB/s
AMD ACAP VCK190 [27]	6.4 T	102.4 T	25.6 GB/s

We deploy the same INT8 quantized model, DeiT-T, on the AMD ACAP architecture [30] VCK190 [27] board using CHARM [19]. CHARM [19] is the state-of-the-art deep learning inference accelerator and mapping framework on ACAP architecture, which features

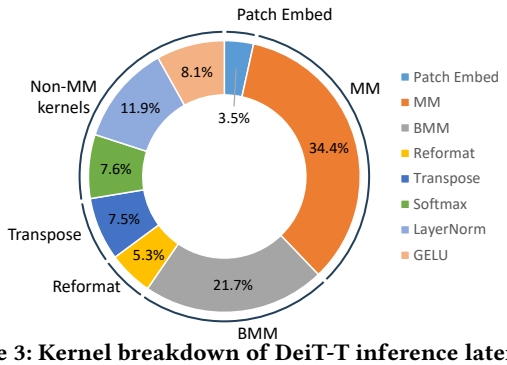


Figure 3: Kernel breakdown of DeiT-T inference latency on GPU A10G, batch size = 6.

FPGA, AIE vector processors, and CPU on the system-on-chip. The end-to-end latency when using CHARM [19] is 12ms, 8.4x larger than that of GPU A10G under batch size 6. The main reason is that CHARM maps heterogeneous accelerators on ACAP and the data transfer among accelerators has to go to/from off-chip DDR. As specified in Table 1, the VCK190 board has 25.6 GB/s off-chip bandwidth, which is much smaller than that of A10G. ⑤ **Programming on ACAP creates new unsolved challenges.** Without careful de-

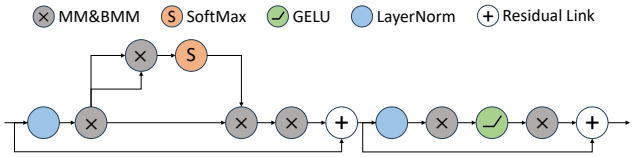


Figure 4: Layers & their dependencies in a transformer block.

We further plot the layers within a transformer block in DeiT-T and show the dependencies between different layers in Figure 4. When considering the sequential spatial hybrid strategies, we can consider mapping different layers on one physical accelerator. For example, we can map all MM and batch MM layers using one MM accelerator and map all the other non-MM layers to separate accelerators. Using only one MM accelerator can potentially give us the lowest achievable latency for MM layers as discussed in Section 1. However, we should also consider the data communication between this one MM accelerator and all the other non-MM accelerators. For example, the dataflow design and input & output data layout design of this MM accelerator should be carefully chosen. Otherwise, it could be the case that the data layout of this MM accelerator matches with one neighboring non-MM accelerator but it does not match another one. Therefore, it needs data layout change, which means extra communication overhead in addition to the computation of each layer. Therefore, ⑥ **when considering sequential spatial hybrid strategies, the data dependencies in the graph will make the communication patterns between accelerators more complex, and the inter-acc communication should be co-optimized during the accelerator design time.**

To tackle these challenges, we propose SSR to optimize performance, which brings the latency of mapping DeiT-T on VCK190 from 12 ms to 0.54 ms when batch size is 6, achieving a 22.22x speedup. Our SSR solution beats the latency of GPU A10G by

2.53x. SSR also enables efficient latency throughput tradeoff design space exploration as described in Section 1. How does SSR achieve this? **First**, SSR explores sequential spatial hybrid strategies when mapping MM and BMM layers to enable the latency and throughput tradeoff. SSR map these layers onto the AIE part of ACAP. **Second**, SSR considers on-chip forwarding when the model size fits on-chip. This greatly reduces the communication. But it also means the design complexity of the on-chip buffers increases. We discuss how to apply SSR in general cases when the model size does not fit on-chip in Section 6. **Third**, SSR designs efficient accelerators for nonlinear layers (Softmax, GELU, and LayerNorm), data layout change (Transpose), and data type conversions (Reformat) on the FPGA part. The flexibility provided by FPGA enables customization for various types of non-MM layers, which GPU CUDA cores lack. **Fourth**, SSR enables a fine-grained pipeline between MM layers on the AIE and non-MM layers on the FPGA to hide the non-MM latency, which further reduces the latency. **Fifth**, SSR considers the inter-acc communication during the layer-to-accelerator mapping stage and also the accelerator design stage. This further reduces the inter-acc communication overhead.

3 RELATED WORK

In this section, we first introduce existing approaches of sequential, spatial, and hybrid accelerators in Sections 3.1, 3.2, 3.3, and discuss their key features. We then summarize the comparisons between SSR and the prior works in Table 2.

3.1 Sequential Accelerators

GPUs are typically used as sequential accelerators in frameworks such as Tensorflow [39], Pytorch [40], etc. With a lot of computing resources, GPUs achieve high throughput by batch processing. TensorRT [28] provides general solutions for mapping deep learning models on GPUs. However, it does not provide customization on certain model workloads. Gemmini [41] is an automatic accelerator generator. It can generate both systolic-array-based and parallel vector engines like hardware accelerators. Gemmini has been widely applied to deep learning acceleration. For example, Sehoon et. al. [18] use Gemmini in Transformer inference. The authors identify the characteristics of Transformer-based models and propose various optimization methods. ViTCoD [34] designs a dedicated accelerator for sparse and dense workloads to boost hardware utilization for vision transformers. Auto-ViT-Acc [36] designs an FPGA accelerator for multi-head attention and an FPGA-aware quantization algorithm to make better use of FPGA resources. HeatViT [35] accelerates vision transformer on embedded FPGAs using image-adaptive token pruning and 8-bit quantization. However, these sequential accelerators use a generic accelerator for all layers with different shapes, which possibly leads to shape mismatch and results in larger latency.

3.2 Spatial Accelerators

Different from deep learning training, real-time AI inference applications usually do not have large batching inputs to fully explore parallelism, and therefore, many throughput-optimized systems for batch processing can only use a small portion of resources for a single inference request. Microsoft BrainWave [5, 6, 7, 8] targets real-time AI inference in the data center scale production system. It explores parallelism within a single task and achieves much lower latency on FPGAs compared with GPUs without sacrificing system-level throughput. Andrew et. al. [37] identify the gap between hardware's peak performance and achievable performance in real applications on Intel Stratix 10 NX FPGA. To minimize this gap in small batch AI inference, they re-implement BrainWave [5, 6, 7, 8] and propose enhanced neural processing unit (NPU) architecture on Intel Stratix 10 NX FPGA. By leveraging the flexibility of FPGA, they achieve significantly higher hardware utilization over GPUs with a comparable peak performance.

3.3 Hybrid Accelerators

DNNExplorer [38] proposes a hybrid design methodology. Specifically, applying spatial accelerators for the first several layers and using a generic accelerator for the rest layers to enable deep networks while achieving acceptable performance. DNNExplorer only supports a fine-grained pipeline between linear kernels, which can reduce latency to a certain extent, while in our work, we extend the pipeline to nonlinear kernels to further reduce end-to-end latency. SET [17] is a framework that automatically schedules deep neural network (DNN) nodes onto tiled accelerators. SET proposes a universal notation and formally defines the mapping space for analyzing tradeoffs among different schedule choices. However, it assumes a very flexible Network-on-Chip (NoC) to connect the accelerators which consumes non-negligible resources and may cause large overhead because of the data congestion in the NoC. CHARM [19] composes heterogeneous accelerators for deep learning applications on ACAP. However, CHARM does not support on-chip data forwarding which results in longer inference latency. DiviML [31] formalizes the DNN partition problem on the heterogeneous computing systems in which different accelerators such as GPUs are connected through PCIe links. DiviML proposes a linear programming model to search for both model and data parallelism and a heuristic schedule algorithm to optimize both latency and throughput. However, in DiviML, data transfer only happens after one layer finishes its computation, and overlap between computation and communication is not supported. Herald [33] and MAGMA [32] optimize DNN on heterogeneous computing systems, but different accelerators can only communicate with each other via off-chip memory, resulting in high latency.

We summarize the comparisons of SSR with prior works in Table 2. SSR adopts sequential spatial hybrid strategies, enables more scheduling flexibility to map layers to accelerators, designs fine-grained pipelines across different types of accelerators, and co-optimizes inter-acc communication with accelerator designs. All together, SSR achieves a better latency throughput Pareto front.

4 SSR ACCELERATOR ARCHITECTURE AND SSR DESIGN FRAMEWORK

In this section, we first introduce SSR framework and heterogeneous architecture overview in 4.1 and 4.2. We then discuss hardware design methodologies and how to do efficient design space exploration in Sections 4.3 and 4.4. Section 4.5 discusses code generation and compilation flow.

4.1 SSR Framework Overview

Figure 5 illustrates the proposed SSR framework. The automatic framework takes the transformer model and hardware resource

Table 2: Comparisons between SSR (ours) and prior works.

Prior Works	Computing Platform	Architecture Features					
	Type	Spatial Accelerator	Hardware Specialization	On-chip Forwarding	Fine-grained Pipeline	Hybrid	Inter-acc Comm. &Acc Co-Design
TensorRT [28]	GPU	×	×	×	×	×	×
DiviML [31]	CPU+GPUs	✓	✓	×	×	✓	×
MAGMA [32], Herald [33]	ASIC	✓	✓	×	×	✓	×
ViTCoD [34]	ASIC	×	✓	×	×	×	×
SET [17]	ASIC	✓	✓	✓	×	✓	×
HeatViT [35], Auto-ViT-Acc [36]	FPGA	×	✓	×	×	×	×
BrainWave [5, 6, 7, 8, 37], Intel NPU [37]	FPGA	✓	✓	✓	×	×	×
DNNExplorer [38]	FPGA	✓	✓	✓	×	✓	×
CHARM [19]	ACAP	✓	✓	×	✓	✓	×
SSR (Ours)	ACAP and FPGA	✓	✓	✓	✓	✓	✓

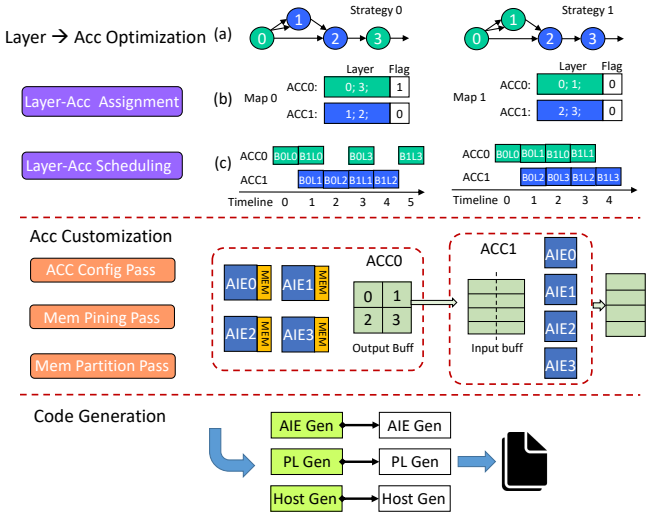


Figure 5: SSR framework overview.

constraints as input and generates the spatial sequential hybrid execution scheduling as well as the corresponding hardware implementation on the Versal ACAP heterogeneous system. Our SSR framework systematically optimizes the system throughput under certain latency constraints through two levels of optimization including Layer→Acc level and Acc-Customization level.

At the Layer→Acc level, given an application graph, the Layer→Acc scheduler will first generate the layer-accelerator assignment map by partitioning the graph into multiple sub-graphs and allocating each one to a specific accelerator. For example, as shown in Figure 5(a), the graph consists of four layers. In strategy 0 (left), layers {0, 3} are assigned to Acc0, and layers {1, 2} are assigned to Acc1. Based on the different layer-accelerator assignment maps, the scheduler can determine the execution order of the nodes with the dependency in the application graph being resolved. Assume there are two batches of input, denoted by B0 and B1, in strategy 0, it requires 6 units of time to finish two batches. In contrast, strategy 1 (right), requires 5 unit time. When considering the actual time in each unit, the Acc-Customization plays an important role, thus it leads to a coupled Layer→Acc/Acc-Customization problem. After the Layer→Acc assignment and scheduling, our framework will allocate the initial resource allocation constraints on each accelerator. Then the Acc-Customizer will optimize the configuration of each accelerator including the AIE array design, memory pinning strategy (❶), and non-linear kernel fine-grained pipeline design (❷). Most importantly, to reduce the data transfer overhead between different accelerators, we apply an inter-acc communication and accelerator co-design and introduce a customized memory partitioning strategy (❸). Guided by the configuration provided by the SSR scheduler, the automatic code generator will generate the source code for the host CPU, PL, and AIE respectively.

4.2 SSR Heterogeneous Architecture Overview

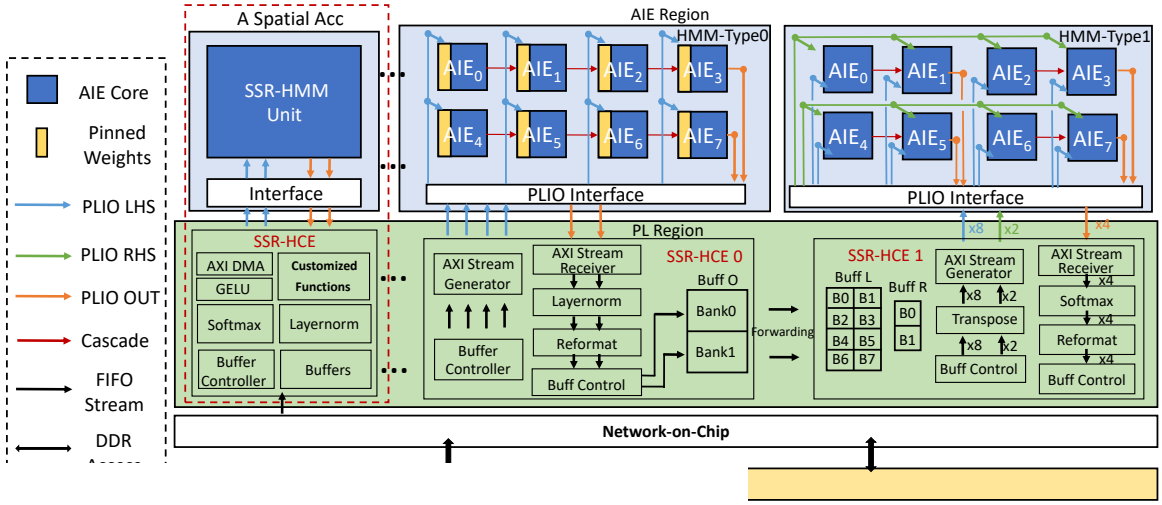
The hardware architecture overview in our SSR framework is shown in Figure 6. It consists of N (∈ 1,...,n) spatial accelerators implemented on the AIE and PL. Within each spatial accelerator, there are two basic blocks, the heterogeneous matrix multiply (HMM) unit, and the heterogeneous customized engine (HCE).

The AXI DMA in the spatial accelerator is responsible for sending the AXI request to the NoC that loads the image data/stores the final results from/to the off-chip DDR4 memory. The HMM units handle the computation-intensive MM and BMM kernels using the high throughput AIE arrays. The HCE units contain senders and receivers to transfer the data between AIE and PL. The sender and receiver modules are not only responsible for generating the AXI stream protocols needed by the AIE array but also for computing the nonlinear and element-wise kernels. SSR supports extension for future applications as any customized function units can be included in our HCE units for data pre/post-processing. The intermediate data can move between different spatial accelerators through on-chip forwarding directly.

4.3 SSR Hardware Design Methodology

After introducing the overall SSR architecture, we elaborate on the detailed hardware design methodology.

❶ **HMM configuration and memory pinning strategy.** In order to sustain the computation of 400 AIEs under the limited PLIO constraint [42], we design two types of HMMs demonstrated in Figure 6. For HMM-type0, by pinning the weights to the local memory of AIEs it only takes one operand (activations) to reduce the utilized PLIOs. However, the multi-head attention layers in transformer models involve two activation operands, which cannot be implemented by HMM-type0. Thus HMM-type1 is designed to deal



rchitecture overview.

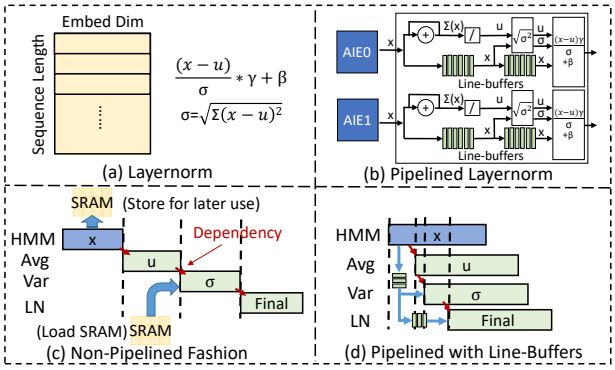


Figure 7: Element-wise and nonlinear kernel pipeline.

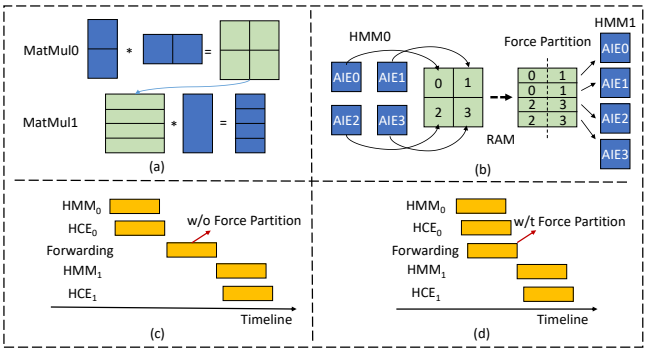


Figure 8: On-chip data forwarding between spatial accelerators with force RAM bank partition.

with such general matrix multiply operations. To apply the weights pinning and PLIO reduction strategy to the entire application graph, we mark each Layer→Acc assignment with an optimizable flag. This is achieved by checking if attention layers are included in the assignment. For example, in Figure 5(a), nodes 1 and 2 represent the multi-head attention layers with two activation inputs. When applying strategy 0, only non-attention layers are assigned to accelerator 0, thus we enable the optimization for searching the configuration to pin all the weights in the local memory of AIEs. By using this strategy, SSR enables high utilization of AIEs without routing congestion, for example, 394 AIEs out of a total of 400 AIEs are successfully implemented in the SSR-Spatial design.

❷ Fine-grained pipeline for element-wise and nonlinear kernels. In order to reduce the latency of the non-computation-intensive kernels, we explore the fine-grained pipeline between the HMM and HCE units. The operations whose data reuse distance are one, such as Transpose, VectorAdd and Reformat (data type conversion), can be easily fused with the HMM kernels. However, nonlinear operations such as Softmax, LayerNorm, and GeLU perform the reduction in an array resulting in the reuse distance larger than 1. Take the LayerNorm operation as an example as shown in Figure 7(a), before calculating the final results, it computes the average(μ) and standard derivation(σ) along the embedding dimension. Moreover, the dependency also exists between average and standard derivation. If without any pipeline design, these operations can take even longer time compared to the computation-intensive HMM Units in Figure 7(c). To reduce the latency and improve hardware utilization, we apply the bypass line-buffer structure in the customized Layernorm kernel on the PL side to overlap the latency in different stages as depicted in Figure 7(b). As illustrated in Figure 7(d), it receives data from HMM units and temporally pushes it into the line buffer. Right after the average μ of the first row is ready, it will read the data from the line buffer and calculate the standard derivation σ, so that the dependency can be resolved with a small waiting time. In general, this methodology can also be applied to other nonlinear kernels which reduces its latency to nearly half.

❸ Inter-acc communication and accelerator co-design. When exploring the spatial-sequential architecture, the data communication patterns between accelerators become more complex and are prone to cause communication overhead because of the mismatch in accelerator configurations or memory conflicts. For example, the latency overhead appears in the consecutive matrix multiply scenarios as shown in Figure 8(a) where MatMul0 and MatMul1 are mapped to HMM0 and HMM1 respectively. The output matrix of Matmul0 serves as the input activation of Matmul1. When designing the HMM kernels for MM with size M×K×N, there are three corresponding parallel choices at the AIE array level including

here noted as A, B, and C. In other words, the A×B×C AIEs work concurrently with the A×C AIEs generating the output at the same time. In Figure 8(b), while HMM0 parallels on A and C forming a 2×2 AIE array, the HMM1 parallels on A and B forming a 4×1 AIE array. Since A×C tiles of output will be transferred through PLIO and received by the downstream PL BRAM/URAM simultaneously, to prevent the HMM0 from stalling, A×C bank partitioning is required shown in the RAM of HMM0. However, the data stored in the 2×2 banks needs to be forwarded to the subsequent HMM1 as input activation in the format of 4×1, resulting in bank conflicts. One straightforward solution to resolve bank conflicts is to introduce a non-overlapping operation that moves the data from RAM0 to RAM1 sequentially thus introducing a huge latency overhead in the pipeline as illustrated in Figure 8(c). In this work, we propose a force-partition strategy to resolve the bank conflicts, while maintaining low latency. More specifically, during the runtime of optimization, we parse the data transaction among accelerators. For the pairs that do have communication, we configure the parallelism of the them to be divisible by each other. For example, the parallel parameters A, C of HMM0 should be fully divisible by the A, B of HMM1 or vice versa. Then we force the RAM bank partition of the subsequent HMM1 to be compatible with the previous HMM0. As illustrated in Figure 8(b), originally four banks of RAM are sufficient to guarantee the execution of the 4×1 HMM1 unit. However only by partitioning the RAM to 4×2, can the on-chip forwarding latency be overlapped by HMM0 shown in Figure 8(d).

4.4 SSR Design Space Exploration.

Layer→Acc evolutionary algorithm (EA). The main challenge to optimize the spatial-sequential solution is the extremely large design space. For example, the complexity for only Layer→Acc scheduling is already over $O(9.9^n)$ [17] where n is the number of layers in the graph. To solve this problem, we propose several heuristics at Layer→Acc and Acc-Customization levels that explore the design space efficiently. At the Layer→Acc level, we apply an evolutionary algorithm [43] based solution to optimize the throughput of the system while achieving the latency constraints demonstrated in Algorithm 1. In our framework, the algorithm takes the execution graph, hardware resources, and latency constraints as input. By using the Layer→Acc and Acc-Customization passes, it can generate the specialized configuration for each accelerator and the Layer→Acc scheduling that will be used by our automatic generation to implement the design. The algorithm is inspired by the processes of biological evolution. It first randomly generates some Layer→Acc strategies shown in Figure 5(a-b) as the population and evaluates the design points in the current population through proposed SSR optimization passes ("SSR_DSE" Lines 3-5). Then it selects the best assignment strategy to do crossover which generates the children generation (Lines 8-12). By introducing the mutation to the children generation it is possible to obtain a better assignment strategy (Lines 13-18). After evaluating all the design points, it will record the throughput optimal point under latency constraints and update the new population by selecting the top solutions (Lines 19-24).

During the SSR Layer→Acc and Acc-Customization processes (line 5 & 18 defined in lines 27-37), by using a greedy algorithm, it first generates the Layer→Acc scheduling pipeline and the data

Algorithm 1 SSR Evolutionary Algorithm

Input: Execution Graph (**G**), Hardware Constraints (**HW_Cons**), Latency Constraints (**Lat_Cons**)
Output: SSR Spatial Acc Configuration (**Conf**), Layer-Acc scheduling (**schedule**)
Hyperparmeters: **nAcc**, **nBat**, **nPop**, **nChild**, **nIter**
▷ **nAcc** and **nBat** refers to the number of accelerators and batch of graphs, **nPop**, **nChild** and **nIter** are the parameters for EA search

```
1  assign_pop = zeros(nPop) #initialize layer-acc assignment
2  layer_acc_flag = 1 #enable inter-acc aware Acc-Customization
3  #Initialize first generation
4  assign_pop[:]=layer_acc_assign(nAcc)
5  latency, cost_thput_par[i], Conf, schedule=SSR_DSE(assign_pop[:],G)
6
7  for iter in range(nIter): #Run EA by nIter generations
8      # Choose the best parent assignment and do single point crossover
9      for k in range(nChild//2):
10         p1,p2 = assign_pop [select(cost_thput_par[:])]
11         ch1,ch2 = sp_crossover(p1,p2)
12         assign_chi.append(ch1,ch2)
13     # Randomly exchange two layer-acc assignment to do mutation
14     for k in range(nChild):
15         assign_chi[k]=mutate(assign_chi[k])
16         #Launch SSR optimization passes
17         latency, cost_thput_chi[k], Conf, schedule =
18         SSR_DSE(assign_chi[k], G)
19         if latency < Lat_Cons and cost_thput_chi[k]>best_thput:
20             best_thput = cost_thput_chi[k]
21             final_Conf, final_schedule = record(Conf, schedule)
22     # Select top design points as new population
23     assign_pop = population_update (assign_pop, assign_chi)
24     latency = cost_update (cost_thput_par, cost_thput_chi)
25 return final_Conf, final_schedule
26
27 def SSR_DSE (assign, Graph, layer_acc_flag):
28     #Gready Algorithm based Layer->Acc scheduling
29     acc_trans, schedule = layer_acc_schedule (assign, Graph)
30     # First-round memory allocation based on data transfer among Accs
31     mem_alloc = mem_allocation (acc_trans)
32     # Determine AIE, PLIO, RAM, and DSP for each Acc
33     hw_part = hw_partition (mem_alloc, schedule)
34     # Launch SSR Acc-Customization DSE to get latency, throughput
35     latency, thput, Conf = SSR_Acc_DSE (hw_part, schedule, ...
36                            assign, acc_trans, layer_acc_flag)
37     return latency, thput, Conf, schedule
```

transaction between accelerators with the dependencies resolved according to the layer-accelerator mapping (Lines 28-29). More specifically, for a layer in the graph, we assign it to the pipeline as soon as its corresponding accelerator is available and its dependencies are already resolved as illustrated in Figure 5(c). Then by analyzing the data transaction among accelerators, it determines a minimum memory allocation strategy that can buffer both the activations and weights on-chip while keeping the accelerator running without memory stall (Lines 30-31). Before doing the Acc-Customization (Lines 35-36, Algorithm 2), the framework preallocates the resources to each accelerator including AIE, PLIO, RAM, and DSP. While the number of AIE together with PLIO is proportional to the total number of operations assigned to the accelerator, the memory budget is assigned according to the memory allocation strategy (Lines 32-33).

Inter-acc communication aware optimization at the Acc-Customization level. In the Acc-Customization stage, SSR searches the configurations of each accelerator represented as a config_vector (h1, w1, w2, A, B, C, Part_A, Part_B, Part_C). In the configuration, (h1, w1, w2) define the workload allocation per AIE, (A, B, C) determine the AIE array parallelism, and (Part_A, Part_B, Part_C) determine the extra bank partitions for inter-acc communication

Algorithm 2 SSR inter-acc comm. aware customization

▷ **hw_part**, **schedule** and **acc_trans** are described in Algorithm 1. **hw_part** contains the resource constraints for nAcc accelerators

```
1  def SSR_Acc_DSE (hw_part,schedule,assign,acc_trans,inter_acc_flag)
2      # Return the order for searching Accs
3      index = trace_assignment(schedule)
4      for i in index:
5          final_thput = 0 #Initialize final throughput
6          #exhaustive search the configuration in the design space
7          for conf_vector[i] in Design_Space:
8              util <-- Eq1 (conf_vector[i])
9              #Check if resource utilization is under the constraints
10             if util > hw_part[i]:
11                 continue
12             #If inter-acc-aware is enabled,
13             if inter_acc_flag==1:
14                 #Check if the current configuration aligns with others
15                 if force_partition(conf_vector[i],assign)==false:
16                     continue
17                 else: #Force memory partitioning to avoid overhead
18                     update(conf_vector[i])
19             cycle, thput <-- Eq2 (conf_vector[i],assign)
20             if thput > final_thput:
21                 final_thput = thput
22                 final_cycle = cycle
23                 final_conf_vector[i] = conf_vector[i]
24     final_cycle, final_thput= comm_overhead(final_cycle, schedule)
25 return final_cycle, final_thput
```

aware optimization. In our design space, we find all integer solutions that make sure a single AIE workload can be fit in 32Kb AIE local memory and AIE utilization doesn't exceed the number of AIE. SSR sequentially launches the DSE for each accelerator according to the order of the accelerator appearing in the Layer→Acc scheduling (Lines 2-4). This ensures that the other accelerators can get the information from the accelerators they depend on as much as possible. For example, for the first Layer→Acc scheduling shown in Figure 5, Acc0 will be searched before Acc1. Then SSR exhaustively searches the configuration of each accelerator within the design space defined before and makes sure the configurable meets the utilization constraints (Lines 6-11). The utilization can be calculated by Equation 1 where the RAM_util represents the number of RAMs needed in each partition and the DSP_Util is the DSP utilization for each nonlinear processor. Then in order to avoid the communication overhead among accelerators due to the memory conflict problem discussed in Section 4.3, SSR takes two steps. First, it checks the AIE array configuration (A, B, C) of the current accelerator to align with the other accelerator that has data transactions. Then force memory bank partition is able to be launched (Line 12-18). The performance of each accelerator for its layers can be calculated by Equation 2, since the nonlinear layers can be fully overlapped by MM kernels we omit it in the equation. After recording the configurable of each accelerator with the best performance (Line 20-23), it fine-tunes the communication overhead based on the knowledge of all the accelerators(Line 24).

$$\begin{aligned} AIE &= A * B * C \\ PLIO &= (A + C) * B \\ RAM &= Part_A * Part_B * Part_C * RAM\_util \\ DSP &= A * C * DSP\_util \end{aligned} \tag{1}$$

$$\begin{aligned} Cycle &= \frac{M * N * K}{A * B * C * MAC/Eff} \\ Throughput &= \frac{\#OPs}{Cycle/Freq} \end{aligned} \tag{2}$$

Table 3: Different vision transformer models configurations.

Model	#Head	Embed. Dim	Depth	Model (M)	MACs (G)
DeiT-T	3	192	12	5.6	1.3
DeiT-160	4	160	12	4	0.9
DeiT-256	4	256	12	7.4	2.1
LV-ViT-T	4	240	12	6.75	1.6

Table 4: Experimental hardware platforms.

GPU	Board	NVIDIA A10G
	Fabrication	8nm
	Frequency	1.71GHz
	TDP	300W
	Library	TensorRT-8.6.1.6
FPGA	Board	AMD U250
	Fabrication	16nm
	Frequency	250MHz
	TDP	225W
FPGA	Board	AMD ZCU102
	Fabrication	16nm
	Frequency	250MHz
	TDP	90W
ACAP	Board	AMD VCK190
	Fabrication	7nm
	Frequency	PL:230MHz, AIE:1GHz
	TDP	180W

4.5 Automatic Code Generation & Compilation

Our SSR framework includes a Python interface to take model description as input and the output is the design source code files including ARM CPU host code, FPGA high-level synthesis code, and AIE intrinsic C/C++ code. Based on our analytical model-guided design space exploration, the code generation toolflow can instantiate the code template to generate the design source code files. SSR framework calls corresponding backend tools in AMD Vitis [44] 2021.1 to generate both the hardware bitstream and host binaries, which can be readily deployed on the board.

5 EXPERIMENTS

5.1 Experimental Setup

We evaluate SSR on AMD ACAP VCK190 [27] board with PL and AIE running on 230MHz and 1GHz respectively. We compare SSR with other state-of-the-art implementations of FPGA and GPU on four transformer-based applications shown in Table 3. The experiments setup for GPU, FPGA, and ACAP is summarized in Table 4. On GPU, we use ONNX 1.14.0 and TensorRT 6.1[28] to convert deep learning models from Pytorch and deploy inference with TensorRT. Then we measure the performance on Nvidia A10G GPU [29] and use nvidia-smi [45] to measure the power consumption. On FPGA, we apply HeatViT [35] on AMD Zynq ZCU102 [46] and AMD Alveo U250 [47] as our baseline. AMD Board Evaluation and Management [48] is used to measure the power of ACAP boards.

5.2 Performance & Energy Efficiency Comparisons

5.2.1 Comparison of performance and energy efficiency among GPU, FPGA, and ACAP. We apply the proposed SSR framework to four applications under three different batches. We verify the SSR designs on the AMD Versal VCK190 board and compare the latency, throughput, and energy efficiency with TensorRT [28] solution on

Table 5: Performance and energy efficiency comparisons across different solutions.

		TensorRT [28] on A10G GPU			HeatViT [35] on ZCU102			HeatViT [35] on U250			SSR (ours) on VCK190		
model	Metrics	Batch=1	Batch=3	Batch=6	Batch=1	Batch=3	Batch=6	Batch=1	Batch=3	Batch=6	Batch=1	Batch=3	Batch=6
DeiT-T	Latency (ms)	0.76	1.03	1.43	5.50	15.14	29.79	2.23	5.60	10.66	0.22	0.39	0.54
	Throughtput (TOPS)	3.19	7.05	10.16	0.44	0.48	0.49	1.09	1.30	1.36	10.90	18.62	26.70
	Energy Eff (GOPS/W)	26.54	40.76	48.37	46.82	48.96	49.25	14.02	16.66	17.04	246.15	368.75	453.32
DeiT-T-160	Latency (ms)	0.73	1.05	1.45	4.22	11.81	23.18	2.21	5.67	10.88	0.21	0.37	0.50
	Throughtput (TOPS)	2.39	4.98	7.21	0.41	0.44	0.45	0.79	0.92	0.96	8.19	14.92	20.90
	Energy Eff (GOPS/W)	20.05	28.59	34.98	44.86	46.58	46.94	10.44	12.13	12.57	196.03	296.11	360.90
DeiT-T-256	Latency (ms)	0.81	1.17	1.69	9.10	25.56	50.51	3.52	9.07	17.24	0.40	0.66	0.98
	Throughtput (TOPS)	5.09	10.56	14.63	0.45	0.48	0.49	1.17	1.36	1.43	10.30	18.73	25.22
	Energy Eff (GOPS/W)	38.53	51.78	66.78	543.55	46.48	46.16	15.05	17.43	18.27	229.37	363.59	423.89
LV-ViT-T	Latency (ms)	0.92	1.37	1.91	7.24	20.27	39.95	3.11	7.91	15.11	0.38	0.62	0.85
	Throughtput (TOPS)	3.39	6.84	9.81	0.43	0.46	0.47	1.01	1.18	1.24	8.21	15.10	22.03
	Energy Eff (GOPS/W)	21.34	35.79	45.19	43.97	46.20	45.52	12.53	14.69	15.32	181.74	296.74	360.04

Table 6: Comparisons on the optimal throughput (TOPS) under four different latency constraints (ms) for four solutions including TensorRT on GPU A10G, and SSR designs (ours) on VCK190 for DeiT-T. SSR-hybrid includes designs from SSR-sequential and SSR-spatial.

Latency Constraints	GPU (TensorRT)	SSR-sequential (ours)	SSR-spatial (ours)	SSR-hybrid (ours)
2 ms	11.32	11.17	26.70	26.70
1 ms	5.28	11.12	26.70	26.70
0.5 ms	×	11.05	19.37	19.37
0.4 ms	×	10.90	×	18.56

Note: × means can not find a valid solution under the latency constraint.

Table 7: Latency comparison for DeiT-T between SSR analytical modeling and on-board measurements.

# of Accs	Estimation(ms)	On-board(ms)	Error Rate
1	1.29	1.30	1%
2	1.14	1.08	-6%
3	0.88	0.85	-4%
4	0.81	0.83	3%
5	0.77	0.79	2%
6	0.54	0.54	-1%

Nvidia A10G GPU, HeatViT [35] solution on AMD ZCU102 [46] and U250 FPGAs [47].

As shown in Table 5, SSR outperforms all three other solutions under 3 different batch sizes in terms of latency, throughput, and energy efficiency. For SSR, the reported latency is measured when the number of accelerator(s) is set as the batch number. For all four applications with 3 different batch sizes of each, the average throughput gains SSR achieves are 2.53x, 35.71x, and 14.20x when compared to Nvidia A10G GPU, AMD ZCU102, and U250 FPGA. The average energy efficiency gains are 8.51x, 6.75x, and 21.22x, respectively. Specifically, when batch size = 1, the throughput gains are 2.84x, 21.67x and 9.38x, and the energy efficiency gains are 8.38x, 4.76x and 16.52x; when batch size = 3, the throughput gains are 2.37x, 35.54x and 14.05x, and the energy efficiency gains are 8.64x, 7.01x and 21.80x; when the batch size comes to 6, the throughput gains are 2.38x, 49.92x, and 19.18x, and the energy efficiency gains are 8.51x, 8.50x, and 25.35x, when compared to Nvidia A10G GPU, AMD ZCU102, and U250 FPGA respectively.

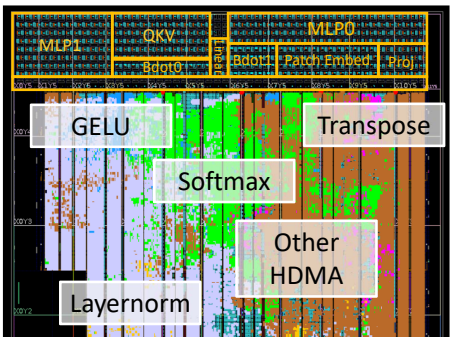
Table 8: SSR hardware utilization for DeiT-T on INT8 mode.

Modules	REG	LUT	BRAM	URAM	DSP	PLIO	AIE
Total	849527	619956	624	104	1797	199	394
AXI DMA	10316	5482	12	0	12	–	–
Layernorm	308736	256678	0	0	1024	–	–
Softmax	179544	78549	192	0	336	–	–
GeLU	3888	2400	0	0	0	–	–
Transpose	13541	5720	0	0	0	–	–
Other HCE	333502	271127	420	104	425	–	–
HMM	0	0	0	0	0	199	394

5.2.2 Latency throughput tradeoff. In Table 6, we demonstrate the latency throughput tradeoff by comparing the throughput of A10G GPU, SSR-sequential design, SSR-spatial design, and SSR-hybrid design under certain latency requirements. In general, all the platforms achieve higher throughput when the latency constraints become looser. As described in Section 1, the GPU designs can only explore the latency throughput tradeoff by changing the batch size. Thus for the real-time scenarios with stringent latency constraints, e.g., <2ms as illustrated in Table 6, the small workload can't sustain the computation of GPU, and this results in relatively lower throughput. Moreover, GPU is unable to meet more critical latency requirements, e.g., <0.5ms.

Since the SSR-spatial design is specialized for each layer in the application, it can achieve high computation utilization when the pipeline is filled with a sufficient number of batches. However, due to the resource partitioning, it has to sacrifice latency. Therefore it cannot meet the most critical time budget (<0.4ms). While the SSR-sequential design is capable of meeting all the latency constraints, due to the lack of specialization, it leads to shape mismatches between layers and the accelerator. Therefore it can't achieve high throughput. Among the design points, by adopting all the hardware optimization techniques and covering large design space, our proposed SSR-hybrid design is able to meet all the latency requirements and achieves the highest throughput under each latency constraint.

5.2.3 Analytical modeling VS. On-board implementations. We compare the latency of the DeiT-T model between the reported results by the SSR analytical model and the real on-board measurements in Table 7. The design points are verified under the number of batches=6 with different numbers of accelerators. The error rate in percentage refers to the difference between the estimated latency by the SSR analytical model and the real on-board implementation. On average, the SSR modeling achieves less than 5% error rate indicating that it can predict the hardware behavior accurately.



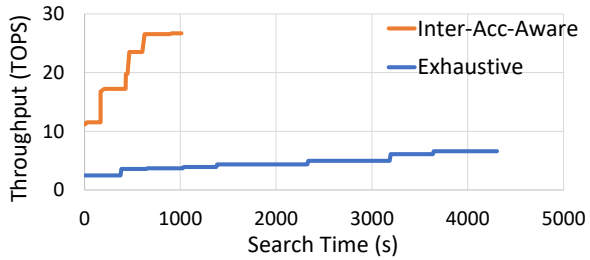


Figure 10: Search time comparison between inter-acc aware search and exhaustive search for DeiT-T.

5.2.4 Implementation layout & resource utilization breakdown. The implementation layout of the proposed SSR-spatial design is shown in Figure 9. In this case, we design specialized MM accelerators on the AIE array for every node within one block of DeiT-T, e.g. QKV layer, attention layers, and MLP layers. The nonlinear kernels including layernorm, and softmax are implemented on the PL side. The corresponding hardware utilization breakdown is shown in Table 8 where specialized HMM units utilize 394 (98.5%) AIEs and perfectly match the shape of layers in the DeiT-T model providing high AIE utilization. For the HCE units that support fine-grained pipeline, 799.8k (44.4%) REG, 588.8k (65.4%) LUT, 624 (64.5%) BRAM, 104 (22.5%) URAM and 1785 (90.7%) DSPs are utilized.

5.2.5 Search Efficiency. We apply the SSR design space exploration to optimize the throughput of the end-to-end inference under the latency constraints of less than 2ms. We compare the search efficiency of two proposed communication-aware strategies in Figure 10. The inter-acc aware strategy optimizes the communication overhead among accelerators by considering the configuration and bank partition of the other accelerators and thus is capable of pruning large inefficient design space. The baseline strategy exhaustively searches the design space and finally post-verifies the configuration of each accelerator and adds the communication overhead. We conduct the search on an Intel Xeon Gold 6346 CPU utilizing 16 cores that run at 3.10GHz. For DeiT, compared to the baseline exhaustive search, the SSR inter-acc aware strategy finds the optimal solution of 26.70 TOPs within 1000s whereas the exhaustive search takes more than 4000s and still can not find high throughput designs.

5.2.6 SSR Step-by-step optimization analysis. SSR enables several design optimizations, including (1) on-chip data forwarding, (2) spatial accelerators, and (3) fine-grained pipeline. We measure the baseline design on VCK190 which none of the three optimizations is enabled. The latency of the baseline design is 12 ms for the DeiT-T model under batch=6, which is 22.2x slower than SSR 0.54 ms (ours). Compared to the baseline, when feature (1) is enabled, SSR achieves a 3.4x latency reduction on DeiT-T. When feature (2) is enabled, it gives 2.4x more latency reduction. When feature (3) is further applied, SSR achieves another 2.7x latency reduction.

6 DISCUSSION OF MAPPING INSIGHTS

Q1: Can we leverage SSR in other architectures?
A1: Yes. SSR can be applied to other architectures.
SSR can be used as a general solution and we can apply SSR mapping method to other platforms, for example, Intel Stratix 10 NX FPGA [49], which has AI-optimized tensor blocks with up to 143 INT8 TOPS, 16MB on-chip memory, and 512GB/s high bandwidth memory. We use SSR analytical models to estimate the latency after we change the hardware resource configurations to be fed into the modeling. In our modeling, we use data from [37] and [8] to get a reasonable INT8 computation efficiency for MM kernels and other non-MM kernels on Intel Stratix 10 NX. The modeled latency when adopting SSR to map DeiT-T on Intel Stratix 10 NX FPGA is 0.49ms, which is comparable to 0.54ms on VCK190 (0.41m ms if VCK190 has 102GB/s off-chip bandwidth). This indicates one of the key contributions of SSR , i.e., SSR provides a general mapping solution that can improve performance across platforms.
Q2: Can we leverage SSR when model sizes do not fit on-chip?
A2: Yes. If a model can not fit on a single board, we can leverage SSR to explore how the model is most effectively partitioned onto multiple devices.
Extensive works have discussed partitioning a large model onto multiple devices spatially whereas part of the model could fit onto the chip. Microsoft Catapult/Brainwave projects deploy large applications (machine learning, search engine, etc.) onto multiple directly connected FPGAs [5] within a server rack or onto a larger number of FPGAs connected with secondary rack-scale networks for inter-FPGA communication [6, 7, 8]. Specifically, we can use a similar assumption as in [8], where the system stores deep learning models' weights in distributed on-chip SRAM memories. For example, the DeiT-Base model is 16x larger than DeiT-T in parameter size. According to the inter-FPGA latency reported in [8, 7], we can scale out our design onto 12 VCK190 boards connected via 100Gb/s QSFP28 with 0.1 ms inter-FPGA board communication overhead across each board.

7 CONCLUSION AND ACKNOWLEDGEMENT

In this work, we propose SSR accelerator & SSR framework to design the sequential spatial hybrid architecture to explore latency throughput tradeoff for deep learning applications and achieve a better Pareto front than sequential-only and spatial-only designs.

We acknowledge the support from NSF awards 2213701, 2217003, 2324864, 2328972, and the University of Pittsburgh New Faculty Start-up Grant. We thank all the reviewers for their valuable feedback and AMD/Xilinx for hardware and software donations.

REFERENCES

[1] Manouchehr Rafie. Autonomous vehicles drive ai advances for edge computing. https://www.3dincites.com/2021/07/autonomous-vehicles-drive-ai-advances-for-edge-computing/.

[2] CERN. Colliding particles not cars: CERN's machine learning could help self-driving cars, 2023. Last accessed JANUARY 25, 2023.

[3] Minjia Zhang, Samyam Rajbandari, Wenhan Wang, Elton Zheng, Olatunji Ruwase, Jeff Rasley, Jason Li, Junhua Wang, and Yuxiong He. Accelerating large scale deep learning inference through {DeepCPU} at microsoft. In *2019 USENIX Conference on Operational Machine Learning (OpML 19)*, pages 5–7, 2019.

[4] AMD/Xilinx. Versal: The First Adaptive Compute Acceleration Platform (ACAP)(WP505).

[5] Andrew Putnam, Adrian M Caulfield, Eric S Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, et al. A reconfigurable fabric for accelerating large-scale datacenter services. *ACM SIGARCH Computer Architecture News*, 42(3):13–24, 2014.

[6] Adrian M Caulfield, Eric S Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, et al. A cloud-scale acceleration architecture. In *2016 49th Annual IEEE/ACM international symposium on microarchitecture (MICRO)*, pages 1–13. IEEE, 2016.

[7] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. Azure accelerated networking:{SmartNICs} in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 51–66, 2018.

[8] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, et al. A configurable cloud-scale DNN processor for real-time AI. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–14. IEEE, 2018.

[9] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[10] Amazon. Aws inferentia: High performance at the lowest cost in amazon ec2 for deep learning inference.

[11] Sidi Lu and Weisong Shi. Vehicle computing: Vision and challenges. *Journal of Information and Intelligence*, 1(1):23–35, 2023.

[12] Peipei Zhou, Jinming Zhuang, Stephen Cahoon, Yue Tang, Zhuoping Yang, Xingzhen Chen, Yiyu Shi, Jingtong Hu, and Alex K Jones. REFRESH FPGAs: Sustainable FPGA Chiplet Architectures. In *2023 14th International Green and Sustainable Computing Conference (IGSC)*, 2023.

[13] Xiaofan Zhang, Junsong Wang, Chao Zhu, Yonghua Lin, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. Dnnbuilder: An automated tool for building high-performance dnn hardware accelerators for fpgas. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2018.

[14] Yue Tang, Xinyi Zhang, Peipei Zhou, and Jingtong Hu. Ef-train: Enable efficient on-device cnn training on fpga through data reshaping for online adaptation or personalization. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 27(5):1–36, 2022.

[15] Xinyi Zhang, Yawen Wu, Peipei Zhou, Xulong Tang, and Jingtong Hu. Algorithm-Hardware Co-Design of Attention Mechanism on FPGA Devices. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(5s), sep 2021.

[16] Chen Wu, Jinming Zhuang, Kun Wang, and Lei He. Mp-opu: A mixed precision fpga-based overlay processor for convolutional neural networks. In *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, pages 33–37, 2021.

[17] Jingwei Cai, Yuchen Wei, Zuotong Wu, Sen Peng, and Kaisheng Ma. Inter-layer scheduling space definition and exploration for tiled accelerators. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, New York, NY, USA, 2023. Association for Computing Machinery.

[18] Sehoon Kim, Coleman Hooper, Thanakul Wattanawong, Minwoo Kang, Ruohan Yan, Hasan Genc, Grace Dinh, Qijing Huang, Kurt Keutzer, Michael W Mahoney, et al. Full stack optimization of transformer inference. In *Architecture and System Support for Transformer Models (ASSYST@ ISCA 2023)*, 2023.

[19] Jinming Zhuang, Jason Lau, Hanchen Ye, Zhuoping Yang, Yubo Du, Jack Lo, Kristof Denolf, Stephen Neuendorffer, Alex Jones, Jingtong Hu, Deming Chen, Jason Cong, and Peipei Zhou. CHARM: Composing Heterogeneous AcceleRators for Matrix Multiply on Versal ACAP Architecture. In *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '23, page 153–164, New York, NY, USA, 2023. Association for Computing Machinery.

[20] Zhuoping Yang, Jinming Zhuang, Jiaqi Yin, Cunxi Yu, Alex K Jones, and Peipei Zhou. AIM: Accelerating Arbitrary-precision Integer Multiplication on Heterogeneous Reconfigurable Computing Platform Versal ACAP. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2023.

[21] Zhuoping Yang, Shixin Ji, Xingzhen Chen, Jinming Zhuang, Weifeng Zhang, Dharmesh Jani, and Peipei Zhou. Challenges and Opportunities to Enable Large-Scale Computing via Heterogeneous Chiplets. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2024.

[22] Zheyu Yan, Xiaobo Sharon Hu, and Yiyu Shi. Computing-in-memory neural network accelerators for safety-critical systems: Can small device variations be disastrous? In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2022.

[23] Zheyu Yan, Xiaobo Sharon Hu, and Yiyu Shi. Swim: Selective write-verify for computing-in-memory neural accelerators. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 277–282, 2022.

[24] Zheyu Yan, Yifan Qin, Wujie Wen, Xiaobo Sharon Hu, and Yiyu Shi. Improving realistic worst-case performance of nvcim dnn accelerators through training with right-censored gaussian noise. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2023.

[25] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR, 2021.

[26] Chen Zhang, Guangyu Sun, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong. Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(11):2072–2085, 2018.

[27] AMD. Versal AI Core Series.

[28] Han Vanholder. Efficient inference with tensorrt. In *GPU Technology Conference*, volume 1, 2016.

[29] Nvidia. Nvidia aws a10g gpu data sheet.

[30] AMD/Xilinx. Versal Adaptive Compute Acceleration Platform.

[31] Yassine Ghannane and Mohamed S Abdelfattah. Diviml: A module-based heuristic for mapping neural networks onto heterogeneous platforms. *arXiv preprint arXiv:2308.00127*, 2023.

[32] Sheng-Chun Kao and Tushar Krishna. Magma: An optimization framework for mapping multiple dnns on multiple accelerator cores. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 814–830. IEEE, 2022.

[33] Hyoukjun Kwon, Liangzhen Lai, Michael Pellauer, Tushar Krishna, Yu-Hsin Chen, and Vikas Chandra. Heterogeneous dataflow accelerators for multi-dnn workloads. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 71–83. IEEE, 2021.

[34] Haoran You, Zhanyi Sun, Huihong Shi, Zhongzhi Yu, Yang Zhao, Yongan Zhang, Chaojian Li, Baopu Li, and Yingyan Lin. Vitcod: Vision transformer acceleration via dedicated algorithm and accelerator co-design. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 273–286. IEEE, 2023.

[35] Peiyan Dong, Mengshu Sun, Alec Lu, Yanyue Xie, Kenneth Liu, Zhenglun Kong, Xin Meng, Zhengang Li, Xue Lin, Zhenman Fang, et al. Heatvit: Hardware-efficient adaptive token pruning for vision transformers. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 442–455. IEEE, 2023.

[36] Zhengang Lit, Mengshu Sun, Alec Lu, Haoyu Ma, Geng Yuan, Yanyue Xie, Hao Tang, Yanyu Li, Miriam Leeser, Zhangyang Wang, et al. Auto-vit-acc: An fpga-aware automatic acceleration framework for vision transformer with mixed-scheme quantization. In *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*, pages 109–116. IEEE, 2022.

[37] Andrew Boutros, Eriko Nurvitadhi, Rui Ma, Sergey Gribok, Zhipeng Zhao, James C Hoe, Vaughn Betz, and Martin Langhammer. Beyond peak performance: Comparing the real performance of ai-optimized fpgas and gpus. In *2020 International Conference on Field-Programmable Technology (ICFPT)*, pages 10–19. IEEE, 2020.

[38] Xiaofan Zhang, Hanchen Ye, Junsong Wang, Yonghua Lin, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. Dnnexplorer: a framework for modeling and exploring a novel paradigm of fpga-based dnn accelerator. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[39] Martín Abadi. TensorFlow: Learning Functions at Scale. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, ICFP 2016, page 1, New York, NY, USA, 2016. Association for Computing Machinery.

[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[41] Hasan Genc, Seah Kim, Alon Amid, Ameer Haj-Ali, Vighnesh Iyer, Pranav Prakash, Jerry Zhao, Daniel Grubb, Harrison Liew, Howard Mao, Albert Ou, Colin Schmidt, Samuel Steffl, John Wright, Ion Stoica, Jonathan Ragan-Kelley, Krste Asanovic, Borivoje Nikolic, and Yakun Sophia Shao. Gemmini: Enabling

systematic deep-learning architecture evaluation via full-stack integration. In *Proceedings of the 58th Annual Design Automation Conference (DAC)*, 2021.
[42] Jinming Zhuang, Zhuoping Yang, and Peipei Zhou. High Performance, Low Power Matrix Multiply Design on ACAP: from Architecture, Design Challenges and DSE Perspectives. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2023.
[43] John H Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.
[44] Xilinx. Vitis unified software platform, 2022. Last accessed April 21, 2022.
[45] Nvidia. System Management Interface SMI | NVIDIA Developer.
[46] AMD. Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit .
[47] AMD. Alveo U250 Data Center Accelerator Card .
[48] AMD/Xilinx. Board evaluation and management Tool.
[49] Intel. Stratix10 NX FPGA.