MG-DmDSE: Multi-Granularity Domain Design Space Exploration Considering Function Similarity

Jinghan Zhang*, Aly Sultan*, Hamed Tabkhi† and Gunar Schirner*
*Department of Electrical and Computer Engineering, Northeastern University, Boston, USA
†Department of Electrical and Computer Engineering, University of North Carolina Charlotte, USA
Email: *zhangjinghan@ece.neu.edu, *sultan.a@northeastern.edu , †htabkhiv@uncc.edu, *schirner@ece.neu.edu

***Abstract*—Heterogeneous accelerator-rich (ACC-rich) platforms combining general-purpose cores and specialized HW accelerators (ACCs) promise high-performance and low-power streaming application deployments in a variety of domains such as video analytics and software-defined radio. In order to benefit a domain of applications, a domain platform exploration tool must take advantage of structural and functional similarities across applications by allocating a common set of ACCs. A previous approach [1] proposed a GenetIc Domain Exploration tool (GIDE) that applied a restrictive binding algorithm that mapped applications functions to monolithic accelerators. This approach suffered from lower average application throughput across and reduced platform generality.**

This paper introduces a Multi-Granularity based Domain Design Space Exploration tool (MG-DmDSE) to improve both average application throughput as well as platform generality. The key contributions of MG-DmDSE are: (1) Applying a multi-granular decomposition of coarse grain application functions into more granular compute kernels. (2) Examining compute similarity between functions in order to produce more generic functions. (3) Configuring monolithic ACCs by selectively bypassing compute elements within them during DSE to expose more functionality. To assess MG-DmDSE, both GIDE and MG-DmDSE were applied to applications in the OpenVX library. MG-DmDSE achieves an average 2.84x greater application throughput compared to GIDE. Additionally, 87.5% of applications benefited from running on the platform produced by MG-DmDSE vs 50% from GIDE, which indicated increase platform generality.

I. Introduction

Stream computing is essential in many domains, e.g. video analytics, software-defined radio, and radar. Strict area and power limitations challenge edge-deploying streaming applications. Using heterogeneous accelerator-rich (ACC-rich) platforms [2] that combine general-purpose processor(s) (SW) and custom ACCs (HW) is the primary approach for efficient, high-performance stream computing. Designing one platform for each application is also not economical, as the non-recurring engineering cost can not be recovered.

In the context of the domain (i.e. many applications), only binding and scheduling to an existing platform has been done, e.g. [3]. This paper focuses on allocating a common platform for many applications. In order to benefit a wider range of applications within a domain, a platform exploration tool must take advantage of inherent structural and functional similarities across applications by allocating a common set of ACCs.

In the Domain Design Space Exploration (DmDSE) flow proposed in [1], many simplifying assumptions were made. The most restrictive of them was the one to one binding between application functions and dedicated monolithic ACCs allocated in the platform. An illustrative example of this is given in Fig. 1 where the function $CannyEdgeDetector$ is bound to the monolithic $ACC_{CannyEdgeDetector}$. This simplifying assumption resulted in diminished ACC utilization by ignoring opportunities to accelerate other functions with similar processing. As a result, only 50% of domain applications considered in [1] achieved a throughput increase of 3x when compared to a pure SW platform.

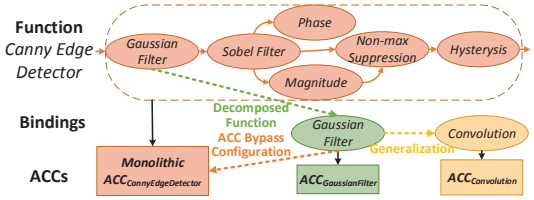


Fig. 1: Decomposition and Similarity in Functions

To overcome these limitations, this paper introduces a Multi-Granularity based Domain Design Space Exploration tool (MG-DmDSE) which attempts to increase ACC utilization. This improves platform generality and average application throughput. MG-DmDSE improves ACC utilization by:

1) Applying a multi-granular decomposition of coarse grain application functions into more granular ones, which leads to more granular ACCs being allocated that have wider applicability. Fig. 1 shows $CannyEdgeDetector$ decomposed by MG-DmDSE into more granular compute kernels such as $GaussianFilter$ and $SobelFilter$. This results in a more granular allocation of corresponding ACCs.

2) Examining compute similarity between functions in order to produce more generic functions. In Fig. 1, MG-DmDSE generalizes $GaussianFilter$ into a generic convolution kernel with a corresponding generalized ACC.

3) Configuring monolithic ACCs by selectively bypassing compute elements within them during DSE to expose more functionality. MG-DmDSE in Fig. 1 can bind a decomposed function $GaussianFilter$ to a configured monolithic $ACC_{CannyEdgeDetector}$.

The benefits of MG-DmDSE are demonstrated using 40 vision applications built on the OpenVX library. With a 0.1 mm^2 ACC area budget, MG-DmDSE achieves an average 2.84x greater application throughput compared to the previous GenetIc Domain Exploration tool (GIDE) in [1]. Additionally, MG-DmDSE produced a more general platform that benefited 87.5% applications, while GIDE only benefited 50%.

This paper is organized as follows: Section II summarizes the related work. Section III introduces the flow and inputs of MG-DmDSE. Section IV describes the platform alloca-

978-3-9819263-5-4/DATE21/©2021 EDAA

tion. Section V provides a platform assessment method and introduces a fast binding algorithm that considers function decomposition and ACC configuration. Section VI describes experimental setup and Section VII evaluates and analyzes the benefits of MG-DmDSE. Section VIII concludes this paper.

II. Related Work

To efficiently execute applications with heterogeneous kernels, recent work focused much on designing application-specific platforms, e.g. [4]. Those, however, only optimize for individual kernels in one application and suffer low flexibility, being not efficient for other applications. Platform-based design, e.g. [5], aims to broaden the design scope to many applications by using statistical information (e.g. operation types, data transfer volume) to alter platform templates. It then applies app-specific binding, communication routing, and resource allocation. However, due to the limited specialization, the platforms don't reach the highest efficiencies.

[1] introduces a DmDSE method GIDE, focusing on many applications to generate an ACC-rich domain platform, however, as mentioned in the introduction, it makes too many simplifying assumptions which in turn limit the generality.

All of the above platform design approaches do not consider compute the similarity between application functions or multi-level decomposition of functions and ACCs. In addition, they do not consider ACC configurability to support multiple functions. Hence, these approaches result in platforms with reduced generality.

From an architecture perspective, reconfigurable platforms such as FPGAs, [6] support multiple applications but not concurrently by taking advantage of an FPGA's inherent flexibility. However, this flexibility comes at the cost of performance when compared to ASIC implementations such as the aforementioned ACC-rich approach. The performance loss can be as much as 5x [7] in terms of application throughput.

[8] focuses on high-level architecture solutions that are inspired by Coarse Grain Reconfigurable Architectures (CGRAs) to avoid the need for hardware ACCs. CGRA based approach shares similarities with our work, which aims to provide a common platform for many applications. These approaches, however, use more flexible CGRA structures that are programmable and consequently target a different flexibility/efficiency design point.

III. Domain Design Space Exploration Flow

Domain Design Space Exploration (DmDSE) is needed to determine accelerator allocation for a platform to efficiently execute domain applications. In general, different optimization goals are possible, e.g. minimize power consumption or maximize throughput with various constraints (e.g. cost, area). In this work, we focus on improving average performance for all applications under a certain area constraint for ACCs.

Fig. 2 describes the general flow of Multi-Granularity Domain DSE (MG-DmDSE), which is a double nested DSE loop. The outer Domain DSE explores different platform allocations discussed in Section IV. The inner Application DSE (App DSE) performs platform assessment, which is repeated for each application to evaluate its performance on the platform,

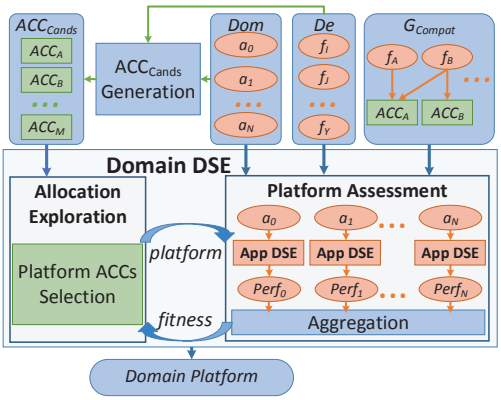


Fig. 2: Domain DSE Flow

the details of this process are discussed in Section V. The results of the execution of the inner processes are used to calculate each platform's fitness for the domain.

A. Domain Definition

The foundation of DmDSE is the formalization of a domain. A set of applications belonging to a domain is defined as a set of applications that share a common set of functions and compute patterns. To allow reasoning about what constitutes a domain, we formally define it as a set of graphs.

$$\begin{aligned} &Dom = \{a_0, a_1, ..., a_N\}, \quad a_i = g(N(F), E) \\ &N(F) = \{n_0(f_A), n_1(f_B), ..., n_n(f_N)\} \\ &E = \{e_0, e_1, ..., e_m\}, \quad e_i((n_{src}, n_{dst}), d) \end{aligned} \tag{1}$$

In Eq. (1), domain Dom is a set of streaming applications (a_0 .. a_N), each captured as a dataflow graph [9]. Each application a_i contains a set N of processing nodes (n_0 .. n_n) and a set of E edges (e_0 .. e_m) representing the communication between nodes. Each node n_i is an instance of a function f. Multiple instances of the same function f may exist within and across applications in a domain. Each edge e_i is the directed communication between its source node n_{src} and destination node n_{dst} with a data transfer of size d.

B. Function Decomposition

To enable multi granularity exploration, MG-DmDSE needs the set De of large functions within the domain that can be decomposed into multiple small functions. Eq. (2) shows that each function f_I follows the same definition as an application. The definition can be recursive, e.g. function $HarrisCorners$ can be decomposed once to $Trace$ and Det, and those can then be further decomposed to the multiple functions $SumOfProduct$. The recursive nature of Eq. (2) allows MG-DmDSE to select the appropriate granularity of function to achieve the desired design criteria of the platform.

$$De = \{f_I.f_J, ..., f_Y\}, \quad f_I = g(N(F), E) \tag{2}$$

C. ACC Candidates Pool

Both large monolithic function ACCs and decomposed granular function ACCs may be good candidates for domain platform allocation. The trade-off between both types

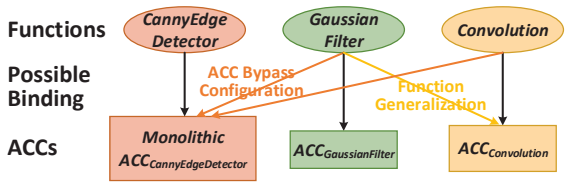


Fig. 3: Partial Binding Compatibility Graph Example

of ACC candidates is efficiency vs area and power saving. Monolithic ACCs have a larger area and increased power consumption relative to decomposed granular ACCs, however, they achieve higher performance during function execution. Small decomposed ACCs only accelerate finer grain functions. As a result, using small decomposed ACCs can result in better area utilization and reduced energy consumption.

MG-DmDSE considers ACC candidates (ACC_{Cands}) for all functions at different granularities. Regardless of granularity, each ACCs has an estimated area based on the number of processing operations within them and the size of their scratch-pad memory.

D. Binding Compatibility Graph

The binding compatibility between functions and ACCs has a dramatic effect on ACC utilization, application performance, and platform generality. Eq. (3) captures the binding compatibility as a bipartite graph G_{Compat}. It contains one set of nodes representing all domain functions (considering both original functions F and decomposed functions De), another set of nodes of ACC_{Cands}, and a set of edges (E) representing the binding between two sets. Each edge (e) means its source function f_J can be be bound on ACC_K. Each edge is annotated with a potential performance penalty b (e.g. when executing on a less specialized ACC).

$$\begin{aligned} &G_{Compat} = (F \cup De, ACC_{Cands}, E) \\ &E = \{e_0, e_1, ..., e_m\}, \quad e_i((f_J, ACC_K), b) \end{aligned} \tag{3}$$

Fig. 3 shows a partial G_{Compat} example. Previous work [1] only considered one-to-one function and ACC binding (black lines). MG-DmDSE expands G_{Compat} by adding more binding possibilities with either an area or a performance penalty. These possibilities are created by 1) monolithic ACC bypass configuration (orange lines) and 2) function generalization (yellow line).

Binding $GaussianFilter$ on $ACC_{CannyEdgeDetector}$ results in a small area penalty due to introducing configurability hardware in the form of bypass busses to the monolithic ACC. Binding $GaussianFilter$ on the generic $ACC_{Convolution}$ incurs a performance penalty when compared to binding it directly to $ACC_{GaussianFilter}$ because the more general $ACC_{Convolution}$ performs more processing operations than $ACC_{GaussianFilter}$. Binding $Convolution$ on $ACC_{GaussianFilter}$ is not possible because $ACC_{GaussianFilter}$ takes advantage of optimizations within $GaussianFilter$ like automated weight generation. This paper assumes that the performance of the generalized $Convolution$ function on $ACC_{Convolution}$ is the same as the performance of $GaussianFilter$ on $ACC_{Convolution}$ which is implicit given that the performance of both is defined by the number of operations within an ACC.

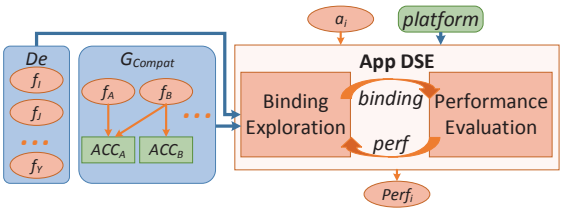


Fig. 4: Application DSE Flow

IV. Platform Allocation

The outer process of DmDSE is platform allocation as illustrated in Fig. 2. MG-DmDSE expands the GenetIc Domain Exploration (GIDE) algorithm in [1] to explore different platform allocations considering both monolithic and decomposed ACCs under an ACCs area budget.

MG-DmDSE captures different ACC allocations as a chromosome where each chromosome represents a different possible platform. After randomly generating an initial generation of chromosomes, the fitness value of each chromosome is produced based on MG-DmDSE's platform assessment methodology discussed in Section V, which includes multi-granularity function decomposition and binding. The Elitist strategy [10] is employed during traversal in order to locate the best-evaluated platforms. Genetic operators Selection, CrossOver, and Mutation are used to generate more platform chromosomes. This process repeats until there are no improvements in new chromosome generations for 10 iterations.

V. Platform Assessment

Platform assessment evaluates platform fitness to guide allocation exploration. The assessment is composed of two parts. The first part App DSE, in Section V-A, explores single application binding on the allocated platform to maximize its performance, which is evaluated in terms of throughput. The second part is the aggregation of all application performance evaluation results to provide a platform fitness value. This part is described in Section V-B.

A. Application DSE

Fig. 4 describes the flow of App DSE in MG-DmDSE. The objective of App DSE is to explore possible bindings between application a_i and $platform$ in order to find the optimal binding that maximizes application performance $Perf_i$.

The recursively decomposable function set De and the binding compatibility graph G_{Compat} are passed to the DSE to explore possible bindings at multiple decomposition granularities. The App DSE is a loop between two processes, binding exploration in Section V-A1, and performance evaluation in Section V-A2.

1) Binding Exploration: The goal of MG-DmDSE is to both improve platform generality as well as maximize application throughput. One way to achieve this is to consider multiple decomposition granularities of functions in the set De during binding exploration. Another way is to rely on function generalization and ACC bypassing configuration which are

both captured in the binding compatibility graph G_{Compat}. Both of these approaches increase application binding flexibility to improve the probability of achieving high throughput. However, they increase binding exploration complexity and render an exhaustive search impractical.

In this subsection, a fast binding algorithm is proposed that efficiently takes advantage of decomposition as well as generality and configurability captured in G_{Compat}. The algorithm in Algorithm 1 first decomposes the application according to the allocated ACCs in the platform. Then the algorithm calculates the number of possible bindings between the decomposed application functions and the allocated platform ACCs using G_{Compat}. If the number of possible bindings is small (less than 10000), then it uses an exhaustive search to explore all possible bindings. If the number is large, the algorithm uses the greedy heuristic binding method Local Search, which is introduced in Algorithm 2.

Algorithm 1 Application Decomposition

1: **function** APPDECOMPOSE($a, plat, De, G_{Compat}$)
2: $aDe = a$
3: $nSet \leftarrow aDe.N(F)$
4: **while** $nSet$.isNotEmpty() **do**
5: $n(f) = nSet$.pop()
6: **if** $f \in De$ **then**
7: isACCed = false
8: **for each** $ACC \in plat$ **do**
9: **if** $(f, ACC) \in G_{Compat}$ **then**
10: isACCed = true
11: **if** isACCed == false **then**
12: $aDe.n \leftarrow De.f.N(F)$ ▷ decompose f into a graph
13: $nSet = nSet \cup aDe.n.N(F)$
14: **return** aDe

The goal of Algorithm 1 is to take the original application graph a and decompose it into a graph of finer granularity aDe depending on the allocated ACCs in the platform $plat$ and the binding compatibility graph G_{Compat}. The algorithm first copies graph a into aDe (line 2) and stores its nodes into $nSet$ (line 3). It then pops the last node $n(f)$ from $nSet$ and determines if its function f is decomposable by checking if it's in the set De (line 5-6). If it is then Algorithm 1 checks to see if any of the ACC allocated in $plat$ are compatible with f (line 9). If there is no allocated ACC candidate that's compatible with f, it is decomposed and the decomposition graph of f ($De.f$) is added to the graph aDe at node n (line 12). The new added nodes in aDe are stored into $nSet$ (line 13). Once $nSet$ is empty, all nodes in aDe are checked and every function in aDe either has a compatible allocated ACC or cannot be decomposed. Upon completion Algorithm 1 returns a platform-specific decomposed application graph.

The local search binding in Algorithm 2 starts by assuming a first $curBinding$ equal to pure SW (line 2-4) where all functions are bound to the general-purpose processor in the platform. Then the while loop (line 7) evaluates all binding pairs between functions in aDe and ACC in $plat$. Each pair is added to the current binding and the newly modified binding is evaluated by EVAL described in Section V-A2 (line 10-18). The binding pair that improves the current binding the most by exceeding the last $bestPerf$ (line 16) is added to the current binding. A modification to the $curBinding$ triggers another iteration of the while loop (line 7). Algorithm 2 greedily

Algorithm 2 Binding Exploration LocalSearch

1: **function** LOCALSEARCH($aDe, plat, G_{Compat}$)
2: $binding(plat.SW, plat.ACCs, aDe.N(F))$
3: $binding.SW \leftarrow aDe.N(F)$
4: $binding.ACCs \leftarrow \emptyset$
5: bestPerf = APPEVAL($binding$)
6: isBindingUpdated = true
7: **while** isBindingUpdated **do**
8: $curBinding = binding$
9: isBindingUpdated = false
10: **for each** $n(f) \in aDe$ **do**
11: **for each** $ACC \in plat$ **do**
12: **if** $(f, ACC) \in G_{Compat}$ **then**
13: $tempBinding = curBinding$
14: $tempBinding.ACC \leftarrow f$
15: tempPerf = EVAL($tempBinding$)
16: **if** tempPerf > bestPerf **then**
17: bestPerf = tempPerf
18: $binding = tempBinding$
19: isBindingUpdated = true
20: **return** $binding$

constructs a new current binding by adding a new optimal binding pair at every while loop iteration. The algorithm stops when there is are no performance improvement gained by adding any new bindings (line 7,19) and returns the obtained binding (line 20).

2) Performance Evaluation: Given an application binding to a platform, performance evaluation uses the analytic model (based on [1]) to estimate throughput. It computes application throughput based on the inter-kernel pipelined execution. The estimated throughput is equal to output volume per iteration over the pipe duration which is determined by the slowest processing or communication component.

B. Aggregation and Fitness

After obtaining each application's performance, the entire allocated platform needs to be assessed. This is done by aggregating these performance results to produces a single fitness value per platform. The Average of all evaluated application's Logarithmic Throughput (ALT) is used for aggregation because it focuses both on overall application performance as well as platform generality [11]. The fitness calculation is shown in Eq. (4).

$$fitness = \sum_{i=0}^{N} (\ln throughput_{(a_i, plat)})/N \tag{4}$$

VI. Experimental Setup

Fig. 5 illustrates the applied ACC-rich platform template in experiment. It includes 1) An ARM CortexA57 processor with 4 ARMv8-A cores at 1.4GHz (3.3 W power and approx 2000 MIPS). 2) A multi-layer interconnect fabric (32 bit-width, 200MHz) with eight concurrent channels (4R and 4W),

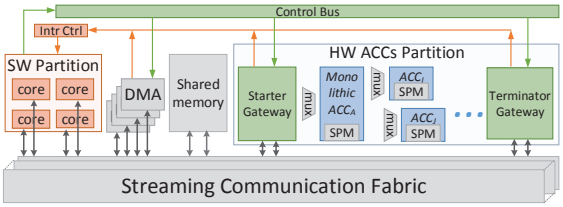
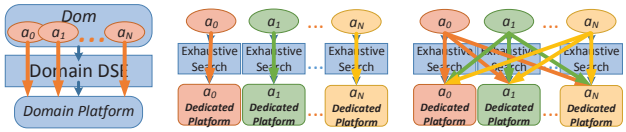


Fig. 5: Target Platform

and 0.5 pJ per byte data transfer on each channel. 3) Four DMAs between shared memory and ACCs. 4) ACCs internal communication fabric that allows directly inter-ACC communication. ACC characteristics are an input to the exploration. For this paper, each ACC's area, processing speed and power consumption has been estimated based on kernel complexity and parallelization opportunities.

We use 40 OpenVX applications in [12], [13] captured in annotated dataflow graphs. Applications graphs range from 3 to 14 functions or nodes, with 2 to 17 edges. The applications are composed of 44 unique processing functions, including 9 new functions generated from decomposition. In platform allocation, 35 monolithic and 9 decomposed ACC candidates are explored. Each ACC has an estimated area based on the number of parallel processing operations, the size of scratch-pad memory, and the overhead of bypass configuration.



(a) Domain Plat (b) OOP (c) FOP

Fig. 6: Experiments Settings

Both MG-DmDSE and previous GenetIc Domain Exploration (GIDE) [1] are applied to generated domain platforms for many applications. This paper re-implements GIDE including its published assumptions and limitations, marked as DmDSE-1to1. The DmDSE-1to1 uses a one to one fixed binding between functions and ACCs, without function decomposition, generalization, and ACC configuration.

In order to compare the efficiency of a domain platform, e.g. obtained to Fig. 6a, having a performance upper and lower bound are beneficial. To give an upper performance bound for DmDSE, this paper uses Own Optimal Platform (OOP) [1]. In Fig. 6b, OOP considers each application in isolation where ACCs are allocated through exhaustive search to achieve maximum throughput for each application. OOP performance aggregates application performance on their own optimal platforms given an area budget.

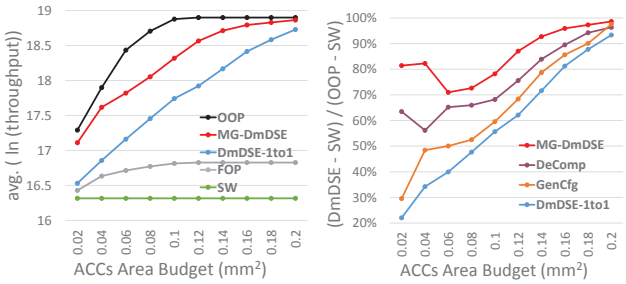
To represent the limitations of application-dedicated DSE, this paper uses Foreign Optimal Platform (FOP) [1]. In Fig. 6c, FOP maps all applications in a domain on to all dedicated platforms for each application. The unified FOP performance metric at each area constraint aggregates across all application and platform combinations.

MG-DmDSE allocated platforms should exceed FOP performance and attempt to match OOP.

VII. Experimental Results

A. Overall Benefit

Fig. 7a shows the performance of different platform allocations over increasing ACCs area budgets. We use the Average Logarithmic Throughput (ALT) for performance comparison. OOP yields the upper bound for a given area constraint. OOP plateaus after 0.12 mm^2, after allocating all beneficial ACCs for each app. FOP plateaus at the very same threshold as it uses the same platforms as the OOP, but executes other applications



(a) Overall Benefit (b) Benefit of Separate Step

Fig. 7: Platform Performance

on it. However, the FOP has a much lower performance than OOP, due to foreign binding. This signifies the limitations of an application-dedicated DSE. Its platform only accelerates the application it was designed for but creates much penalties for all others. SW is the lowest performance due to fairly slow and limited parallel execution on the multi-core processor.

The domain platforms from MG-DmDSE and DmDSE-1to1 have much higher performance than application isolated design FOP. MG-DmDSE and DmDSE-1ot1 has 1.52 and 1.01 higher average ALT over the whole area than FOP. MG-DmDSE has 0.51 more ALT than DmDSE-1to1 due to function decomposition, function generalization and ACC configuration. DmDSE-1to1 only allocates monolithic function ACCs and applies a fixed one to one binding between functions and ACCs. When the area budget becomes sufficiently large (0.14 mm^2), MG-DmDSE platform achieves 92.7% of the OOP improvement from SW execution, highlighting the tremendous benefit of the domain focus and added generality. Conversely, DmDSE-1to1 only achieves 71.6% improvement.

B. Methodology Analysis

To gain more insight into how MG-DmDSE achieves the great results, this section considers partial implementations. Table I shows GenCfg only adds function generalization and ACC bypassing configuration (which is G_{Compat} information) on top of DmDSE-1to1. Conversely, DeComp only considers function decomposition in binding and decomposed ACC candidates in allocation.

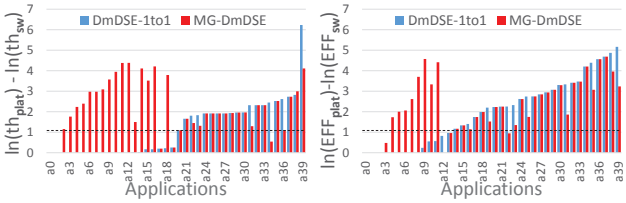
TABLE I: Analysis of MG-DmDSE Steps

	Function Decomposition	Function Generalization	ACC Configuration	Execution Time ($0.1mm^2$)
MG-DmDSE	Y	Y	Y	5.2 hours
DeComp	Y	-	-	4.9 hours
GenCfg	-	Y	Y	21.6 mins
DmDSE-1to1	-	-	-	6.9 mins

Fig. 7b demonstrates the four different DmDSE methods performance achievement of the OOP improvement (100%) from SW (0%). The red line MG-DmDSE and the blue line DmDSE-1to1 represent the same methods as in Fig. 7a. MG-DmDSE has 30.73% more improvement than DmDSE-1to1 on average over the whole area. After applying function generalization and ACC configuration, GenCfg enables flexible binding, increases ACC utilization, and improves by

7.2% compared with DmDSE-1to1. While DeComp with only function decomposition has 19.4% more improvement than DmDSE-1to1. DeComp has more improvement than GenCfg, because DeComp could partially accelerate heavy computing functions through decomposition, and these large functions are always a performance bottleneck. While GenCfg can only speedup some simple functions via its generalization and ACC bypassing, which leads to less improvement.

C. Platform Generality



(a) Throughput (b) Energy Efficiency

Fig. 8: Applications Performance on Domain Platform

Up to now, this paper discussed the aggregated performance of all applications in the domain. To gain more insight into the benefits, this subsection looks into the effect on application individually.

Fig. 8 graphs the logarithmic performance improvement (compared with pure SW execution) of all applications with an 0.1 mm^2 area budget. To quantify platform generality, we consider applications that achieve a greater than 3x performance improvement compared to pure SW (black dash lines, $ln(3) \approx 1.1$).

In the view of throughput, Fig. 8a shows that MG-DmDSE benefits 87.5% applications, while DmDSE-1to1 only benefits 50%. Similarly in Fig. 8b, MG-DmDSE benefits 15% more application than DmDSE-1to1 for energy efficiency (throughput per watt). MG-DmDSE's platform is more balanced in terms of performance across applications compared to DmDSE-1to1.

Some applications in MG-DmDSE have reduced performance than in DmDSE-1to1 due to different ACCs allocations that support more applications. However, unlike DmDSE-1to1, MG-DmDSE does not suffer from some applications dominating the platform to the detriment of all other applications. After removing the throughput outliers (top and bottom 10%), MG-DmDSE average throughput is 2.84x than DmDSE-1to1.

D. Scalability

In addition to the performance of the generated platform, the time for design space exploration is important to judge the scalability of the design process. For this, Table I's last column lists the DmDSE execution time under 0.1 mm^2 ACCs budget, running on the Intel(R) Xeon(R) CPU E5-2680 at 2.8GHz.

DmDSE-1to1 with a fixed binding is much faster in terms of execution time due to its narrow design space. GenCfg has a much larger design space considering the binding options in G_{Compat}. However, despite the increase in complexity, its execution time does not increase too much due to its use of the fast binding flow proposed in Section V-A1. DeComp and MG-DmDSE experience a significant execution time increase due to the overhead of function decomposition. Each function decomposition in both DeComp and MG-DmDSE generates a new application graph. This will be optimized in future work. Although the overhead of decomposition in MG-DmDSE is high, its performance improvement is significant, and the total exploration time (5.2 hours) is acceptable for allocating one unified platform for many applications.

VIII. Conclusion

This paper introduces Multi-Granularity Domain Design Space Exploration (MG-DmDSE) to increase the generality of domain platforms. MG-DmDSE employs function generalization and monolithic ACC configuration to increases platform ACC utilization. Additionally, it introduces function decomposition during application binding to generate more binding opportunities between allocated accelerators and application functions. This paper also presents a fast binding algorithm that deals with decomposition and complex binding compatibility. Under a 0.1 mm^2 ACC area budget, MG-DmDSE when applied to OpenVX applications demonstrates 2.84x greater application throughput when compared to previous domain platform exploration approaches like GIDE [1]. Additionally, the MG-DmDSE produced a more general platform that benefited 87.5% applications, while only 50% from GIDE.

Acknowledgment

This material is based upon work partially supported by the National Science Foundation under Grant No. 1319501.

References

[1] J. Zhang, H. Tabkhi, and G. Schirner, "Allocating one common acc-rich platform for many streaming applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2020.

[2] J. Cong, M. Ghodrat, M. Gill, B. Grigorian, K. Gururaj, and G. Reinman, "Accelerator-rich architectures: Opportunities and progresses," in *Design Automation Conference (DAC)*, 2014, pp. 180:1–180:6.

[3] J. Tang, Y. W. Hau, and M. Marsono, "Hardware/software partitioning of embedded System-on-Chip applications," in *Very Large Scale Integration (VLSI-SoC)*, 2015, pp. 331–336.

[4] A. Wicaksana *et al.*, "Case study: first-time success asic design methodology applied to a multi-processor system-on-chip," in *Application Specific Integrated Circuits-Technologies, Digital Systems and Design Methodologies*. IntechOpen, 2018.

[5] A. Enrici, L. Apvrille, and R. Pacalet, "A model-driven engineering methodology to design parallel and distributed embedded systems," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 2, pp. 1–25, 2017.

[6] S. Wildermann *et al.*, "Operational mode exploration for reconfigurable systems with multiple applications," in *International Conference on Field-Programmable Technology (FPT)*, 2011, pp. 1–8.

[7] E. Nurvitadhi *et al.*, "Accelerating binarized neural networks: Comparison of fpga, cpu, gpu, and asic," in *International Conference on Field-Programmable Technology (FPT)*, 2016, pp. 77–84.

[8] T. Nowatzki, V. Gangadhar, K. Sankaralingam, and G. Wright, "Domain specialization is generally unnecessary for accelerators," *IEEE Micro*, pp. 40–50, 2017.

[9] S. Stuijk, M. Geilen, and T. Basten, "Sdf^ 3: Sdf for free," in *Application of Concurrency to System Design (ACSD)*, 2006, pp. 276–278.

[10] W. Quan and A. Pimentel, "Towards exploring vast MPSoC mapping design spaces using a bias-elitist evolutionary approach," in *Digital System Design (DSD)*, 2014, pp. 655–658.

[11] J. Zhang, H. Tabkhi, and G. Schirner, "Mitigating Application Diversity for Allocating a Unified ACC-Rich Platform," in *International Conference on Computer Design (ICCD)*, Abu Dhabi, UAE, 2019, pp. 622–625.

[12] Intel, "Beta for Intel Computer Vision SDK (Intel CV SDK) Support," https://software.intel.com/en-us/computer-vision-sdk-support/code-samples 2017, accessed: 2017-09-12.

[13] AMD, "AMD OpenVX (AMDOVX)," http://gpuopen.com/compute-product/amd-openvx/ 2018, accessed: 2018-02-14.