Wibheda+: Framework for Data Dependency-aware Multi-constrained Hardware-Software Partitioning in FPGA-based SoCs for IoT Applications

Deshya Wijesundera†, Alok Prakash⋆, Thilina Perera†, Kalindu Herath†, Thambipillai Srikanthan⋆
Nanyang Technological University, Singapore
†{deshyase001,pere0004,kalindub001}@e.ntu.edu.sg, ⋆{alok, astsrikan}@ntu.edu.sg

ABSTRACT

FPGA-based system-on-chip (SoC) devices for Internet of Things (IoT) applications require hardware-software (HW-SW) partitioning techniques to optimize for performance under stringent area and power constraints. To obtain an optimally partitioned design it is necessary to account for the data communication cost between hardware and software. However, the large design space during partitioning makes it challenging to account for this cost while optimizing for stringent constraints using an exhaustive approach. Hence, we propose Wibheda+, a heuristic based framework for fine-grained data dependency-aware multi-constrained HW-SW partitioning that can be employed to partition designs for FPGA-based SoCs used in IoT. Wibheda+, evaluated on 10 applications from the CHStone benchmark suite, has been shown to find solutions with 99% accuracy within several milliseconds compared to several minutes or hours taken in a state-of-the-art and an exhaustive approach, respectively.

1. INTRODUCTION

Modern FPGAs are not only suited for accelerating critical parts of an application, but also for realizing an entire System-on-Chip (SoC), constituting processors, programmable logic, memory subsystems, etc. Hence, while the critical section of the application can still be accelerated on the programmable logic, the rest of the application can be executed on the accompanying on-chip processors [18]. An efficiently partitioned design effectively combines the flexibility of software with the performance acceleration and low power/energy consumption of hardware [16]. Thus, efficient partitioning of the application between the hardware and software components is a crucial step in allowing designers to exploit the benefits offered by both worlds.

However, partitioning a design into its hardware and software components is one of the biggest challenges in architecting an embedded system, as badly partitioned designs can lead to inefficiencies due to data dependencies between components [14]. Partitioning decisions must typically be made early in the design of a product. However, the lack of proper analysis at this stage can result in higher non-recurring engineering (NRE) costs and time-to-market (TTM) delays, or even an inability to meet design requirements [5]. Hence, it is important for the partitioning methodology to analyze the applications at this early design stage and accurately model the system while considering the plethora of different implementation possibilities [20].

The granularity at which to partition an application poses yet another challenge for the designer during the partitioning step. Different approaches have been proposed to partition an application at a coarse (functions [10]/loops [20]) or fine (basic blocks [15]) granularity. Compared to a fine-grained approach, in a coarse-grained approach the design space to select the best application code segments for acceleration is significantly reduced [12]. On the other hand, a fine-grained approach may yield higher return on investment (ROI) of the hardware area, since one could select the most profitable code segments to be accelerated, in this technique [14]. The higher ROI is significantly beneficial in the domain of always-connected and low cost Internet of Things (IoT) devices, which necessitates designs with extremely tight constraints in terms of size (area), power consumption, costs, etc. This has resulted in FPGA vendors such as Lattice Semiconductor and Intel offering low power, small FPGAs specifically targeted for IoT [13] [2].

However, the larger design space in fine-grained approaches and the resulting increased data communication between the many fine-grained accelerators and the remaining software components, necessitates a partitioning technique that considers and accurately models the data communication costs during partitioning. Existing state-of-the-art (SoA) work [15] proposes a fine-grained (basic block level) hardware-software (HW-SW) partitioning methodology that selects the most beneficial basic blocks for hardware acceleration while taking into account data dependencies between the basic blocks. However, this work does not consider the area constraint directly during the selection process. Instead, it selects the most profitable basic blocks in an iterative fashion and obtains their area post-selection. Figure 1 shows the drawback of this approach for the *DFADD* application from the CHStone benchmark suite. The horizontal axis in this figure indicates the area constraints in terms of look-up-tables (LUT). The vertical axis shows the deviation of estimated execution cycles from an exhaustive selection technique for the SoA [15] and proposed techniques. In this example, execution cycles is the number of clock cycles needed for execution of the entire application that is partitioned between hardware and software. Here, normalized values (vertical axis) greater than 1 indicate the selection of sub optimal design points. Figure 1 shows that there are large deviations in the estimated execution cycles of the SoA at certain design points. At LUT constraints 1000, 1500, 2000 and 2500, the normalized values exceed 2, depicting estimation errors exceeding 100%. As shown in Fig. 1, the average error across the design space for *DFADD*

This work was presented in part at the international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2018) Toronto, Canada, June 20-22, 2018.

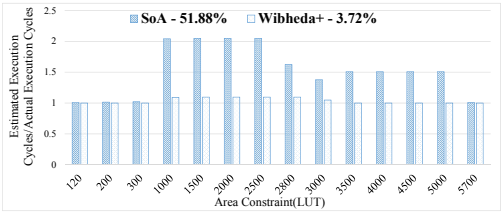


Figure 1: Comparison of Estimated Performance - *DFADD*

using the SoA is 51.88% in comparison to 3.72% in the proposed techniques. Thus, we identify the need to not only develop rapid and reliable techniques for the fine-grained single constrained HW-SW partitioning problem but also propose techniques to accommodate multiple design constraints such as LUT, digital signal processing (DSP) block, flip-flop (FF) and power.

Hence, in this paper, we propose Wibheda+, a framework for rapid data dependency-, area- and power constraint-aware fine-grained (basic block level) HW-SW partitioning that can be applied to applications of varying size and complexity. Major contributions of this work are (a) methodology to analyze data communication cost between basic blocks and memory components; (b) rapid technique to model the HW-SW partitioning problem to an Oregon Trail inspired knapsack problem using the proposed system level model; (c) scalable integer linear programming (ILP) based heuristic formulation to select the most profitable partition considering (i) data communication cost of basic blocks and memory components, (ii) area constraints in terms of LUT, DSP, FF and (iii) power constraints. Wibheda+, compared against the SoA, not only achieves better accuracy but also advances the SoA by incorporating multiple design constraints without compromising the runtime.

2. RELATED WORK

The problem of finding the optimal partition under a given constraint, e.g. area, is NP-complete [3]. Thus, any exact formulation will lead to an exponential increase in runtime with increase in design space. Therefore, many heuristics have been proposed in order to find near optimal solutions while reducing runtime [20] [15]. Also, approaches to combine multiple heuristics have been reported [10]. However, the solution quality of these approaches is highly dependent on the initial solution seed [20].

The accuracy of a partitioning decision is significantly dependent on the data communication model. As a result of partitioning, data manipulated by hardware has to be updated in the appropriate memory location so that software uses the updated data value and vice versa. This results in an overhead for the partitioning step. Thus, it is important that a partitioning algorithm facilitates accurate modeling of data communication. However, the current state of practice offers only limited accuracy at modeling time [20].

Also, the granularity of the partitioning algorithm is a vital factor in obtaining an optimal partition [17]. Thus, automatic partitioning strategies using hardware accelerators at different granularities and runtime systems to incorporate different fine/coarse-grained accelerators into an extensible processor according to application requirements have been proposed [12] [15].

Fine-granularity HW-SW partitioning is significantly beneficial in the domain of IoT devices due to the stringent area/power requirements. As a result, applications of FPGA based HW-SW partitioned systems in IoT is becoming increasingly popular [9]. Chen et al. identify that the long and iterative design cycles which is a bottleneck to meet stringent TTM requirements is one of the key challenges in development of IoT SoC devices due to the lack of suitable automation tools [8]. Further, the authors continue to highlight the importance of developing fast and accurate estimation models that can incorporate area, performance and power goals for HW-SW partitioning in IoT.

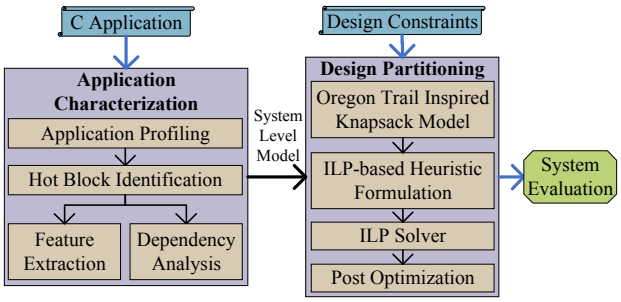


Figure 2: Wibheda+

To this end, Zuo et al. [20] present an automated power/area-aware HW-SW partitioning framework. However, due to the coarse granularity of the partitioning step the usefulness of the system for low power/area requirements is limited. Moreover, the methodology is heavily reliant on manual intervention and also requires a significant amount of FPGA platform specific data to be provided by the user. On the contrary, Prakash et al. [15] propose a rapid methodology for memory-aware HW-SW partitioning at basic block level. However, authors only consider the number of basic blocks as the design constraint that results in large errors for an area-constrained design. Thus, we identify the need for rapid and reliable methodologies for data dependency-aware multi-constrained HW-SW partitioning which can be applied to designs of varying size and complexity.

3. METHODOLOGY

In this section, we discuss the system level model and details of the Wibheda+ framework shown in Fig. 2.

3.1 System Level Model

Figure 3 shows the proposed system level model for application characterization, comprising of c computation units and m memory blocks denoted by $\{\Delta = C_1, C_2, C_3, \cdots, C_c\}$ and $\{\Gamma = M_1, M_2, M_3, \cdots, M_m\}$ respectively, interconnected via a data communication network. Here, C_i represents computation unit i and M_j represents memory block j. The set of all computation units is denoted by Δ, and the set of all memory blocks by Γ.

In this work, an application is represented using the proposed model where basic blocks are mapped to computation units and statically assigned variables to memories. Further, each computation unit is characterized by a set of features, defined by (i) Software Time (ST): Given that the full application is executed in the processor space (SW), the execution time of a computation unit; (ii) Hardware Time (HT): Similar to ST, provided that the full application is executed in the FPGA space (HW), the execution time of a computation unit (iii) Area (L,D,F): resource consumption of a computation unit in terms of LUT, DSP, FF in HW ; (iv) Power (P): dynamic power consumption of a computation unit in HW; and (v) Frequency (α): number of times a computation unit is executed during application execution. Thus, each computation unit (C_i) is represented by a tuple as in Eq. (1). Further, each computation unit C_i accesses the

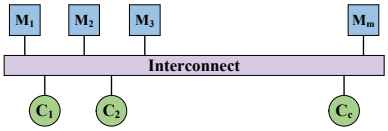


Figure 3: System Level Model

memory blocks M_j for data communication. Thus, M_{ij} represents the number of accesses to memory block M_j from computation unit C_i during application execution.

$$C_i = \{ST_i, HT_i, L_i, D_i, F_i, P_i, \alpha_i\} \quad (1)$$

3.2 HW-SW Partitioning Framework

Wibheda+ consists of two phases, application characterization and design partitioning. During the application characterization phase, the input C application is mapped to the system level model presented in subsection 3.1. At the same time, all features of computation units (C_i) and memory accesses (M_{ij}) are obtained using modified LLVM [1] and LegUp [7] tool-chains. In the design partitioning phase, the system level model and design constraints are provided to the HW-SW partitioning engine that models the problem as an *Oregon Trail inspired knapsack* problem. Next, the problem is formulated into an *ILP-based* heuristic and solved using an ILP solver. Finally, we perform a post optimization process of the solution to further enhance accuracy.

3.2.1 Application Characterization

The C application is first profiled using LLVM profiler to obtain the most frequently executed basic blocks, henceforth referred to as 'hot blocks'. These hot blocks are considered as computation units while each assignment in the LLVM profile is considered as a memory block. Even though, the user has the flexibility to select the number of computation units for analysis, we select 25 hot blocks since the application spends most of its time in executing these basic blocks. However, the methods proposed in this work are independent of the number of selected computation units.

3.2.1.1 Feature Extraction.

The LLVM profile is used to extract features of computation units. We use an estimation based methodology [18] to extract the ST, while HT, area and power extraction is done using the LegUp tool. However, it should be noted that currently, LegUp only provides data at function level granularity. As a result, we modify LegUp to obtain data at a basic block level granularity.

3.2.1.2 Dependency Analysis.

Data flow information between computation units and memory blocks are extracted using a control data flow graph. The process of obtaining the data dependency cost between a single computation unit and a memory block is as follows. Let d_{uk} be data produced by instruction i_k in C_u. Also, assume that d_{uk} is consumed by β instructions of C_v, where $u \neq v$. Thus, we define the data dependency cost of C_v, when accessing d_{uk} as β. Since, the execution frequency of C_v is known to be α_v (refer Section 3.1), the total data dependency cost is $(\beta * \alpha_v)$. This procedure is repeated for all nodes of the control flow graph using an LLVM analysis pass. Based on the data obtained from this pass, we develop a data flow graph (DFG) with nodes and edges representing computation units and dependency costs respectively. However, there can be multiple edges between two nodes of the DFG as two computation units may share multiple data between each other. Thus, each edge in the DFG is mapped to the corresponding M_{ij} values explained in Section 3.1.

3.2.2 Design Partitioning

After obtaining the system level model, the partitioning algorithm strives to find the optimal set of computation units for hardware acceleration to minimize the overall application execution time within the design constraints (area and/or power). Formally, the problem is defined as *"given a finite set of computation units and the data transfer costs between computation units and memory blocks, find the optimal set of units and memories to be implemented as hardware accelerators such that the application execution time is minimized within area and/or power constraints"*.

3.2.2.1 Architecture Model.

In this work, we perform our analysis based on the architecture model described in [14] and summarized below:

(i)Placement of memory blocks: When a computation unit is implemented in hardware, all memory blocks accessed by the computation unit are also placed alongside in the FPGA BRAMs as un-cached local memories. This is shown in Eq. (2), where $C_i = HW$ means that computation unit i is implemented in hardware. This assumption is required to exploit potential benefits of hardware acceleration [4].

(ii)Data communication penalty (Ω): A data communication overhead is incurred by computation units implemented in software while accessing data stored in un-cached local memories (δ). In contrast, when computation units and memory blocks are both in software the access time will be the cache access time (γ). Thus, there will be a data communication penalty as shown in Eq. (3). Further, the data communication penalty effectively encapsulates the speeds of communication interfaces. On the other hand, based on assumption (i), the penalty incurred by computation units in hardware to access the un-cached local memories is encapsulated in the HT, obtained during feature extraction.

(iii)Static data allocation: The model assumes that all data will be initialized prior to the execution of the application. Thus, applications with dynamic data input are not considered in Wibheda+.

(iv)Atomicity of computation units: Since basic blocks are mapped as computation units, it is guaranteed that the control flow of the application will not deviate from a computation unit to another while the former computation unit is executed. However, data flow requires a computation unit to access the memories.

(v)Resource sharing: It is assumed that none of the computation units share hardware resources in terms of LUT, DSP and FF. As shown in [19], this assumption is based on the fact that resource sharing negates the objective (minimizing execution time) of HW-SW partitioning.

$$C_i = HW \Rightarrow M_j = HW \iff M_{ij} > 0, \quad \forall C_i \in \Delta,\ \forall M_j \in \Gamma \quad (2)$$

$$\Omega = \begin{cases} \delta, & \text{if } M_{ij} > 0;\ \forall C_i = SW,\ \forall M_j = HW \\ \gamma, & \text{if } M_{ij} > 0;\ \forall C_i = SW,\ \forall M_j = SW \end{cases} \quad (3)$$

Oregon Trail inspired Knapsack Model. Based on the architecture model and assumptions, it is evident that even though certain computation units might be profitable candidates for hardware acceleration, due to data dependencies, they might have a negative impact on the overall execution time of the application. This effect is analogous to the **0-1 Oregon Trail knapsack problem** presented in [6]. This problem discusses filling a knapsack with a set of items where the value of an item of type x depends on the presence of another item of type y in the knapsack. Inspired by

this similarity, the proposed HW-SW partitioning problem is mapped such that, in the partitioned design space, the execution time of a computation unit placed in SW ($T_{C_{i,SW}}$), depends on itself (C_i) and the penalty incurred due to presence of other computation units placed in HW ($\Delta \backslash C_i$) that have overlapping data dependencies with C_i. The proposed model is shown in Eq. (4), which shows that the execution time of a computation unit in SW is a function (f_i) of itself (C_i) and other computation units in HW ($\Delta \backslash C_i$). However, based on assumption (i), the execution time of a computation unit placed in HW ($T_{C_{i,HW}}$) is not affected by this model and is still equal to HT_i. This is shown in Eq. (5).

$$T_{C_{i,SW}} = f_i(C_i, \Delta \backslash C_i) \tag{4}$$

$$T_{C_{i,HW}} = HT_i \tag{5}$$

3.2.2.2 ILP-based Heuristic Formulation.

The next step in the framework is to propose a runtime efficient algorithm to solve the problem while providing near optimal results. It has been shown that ILP-based formulations can optimally solve the HW-SW partitioning problem [3]. However, the time complexity of ILP-based exact solutions is exponential, which in-turn reduces their usefulness for larger design spaces. To alleviate this problem, we propose an ILP-based heuristic formulation as explained in detail in the subsequent sections. The input to the ILP consists of the set of computation units (Δ), set of memory blocks (Γ) and their dependencies (M_{ij}). The design constraints are provided in terms of the LUT (ζ), DSP (ξ), FF (λ) and power (ρ) budgets, where $\zeta, \xi, \lambda, \rho \in Z^+$. Next, we define a binary decision variable x_i, as shown in Eq. (6), for each C_i which indicates whether the computation unit is placed in hardware ($x_i = 1$) or software ($x_i = 0$).

$$x_i = \begin{cases} 1, & \text{if } C_i = HW, \\ 0, & \text{otherwise.} \end{cases} \quad \forall i = 1, 2, 3, \cdots, c \tag{6}$$

The objective of the partitioning problem is to minimize application execution time. While the execution time of a computation unit placed in HW is directly obtained through feature extraction (refer Section 3.2.1.1), its execution time during SW execution is modeled in Eq. (4). Since, this SW execution time depends on whether other computation units are implemented in HW or SW, an exact ILP formulation models all possible design points in the full design space (2^n combinations for n computation units), which in turn leads to the exponential increase in runtime of the partitioning algorithm. To alleviate this combinatorial explosion, in the proposed heuristic, the total data communication penalty of a computation unit in SW, is assumed to be the summation of individual communication penalties of computation units in HW on the SW computation unit in consideration. This limits the search space to all possible combinations of only two computation units (for n computation units $n * (n-1)$ combinations). Therefore, we effectively reduce the time complexity of the problem to $\mathcal{O}(n^2)$.

We explain this assumption using an example of 5 computation units. Consider finding the execution time of the application when computation units 1 and 3 are implemented in hardware. Equation (7) shows the total execution time (T_{ex}) of the application and Eq. (8), the individual execution time of the computation units. Here, the execution time of computation units in hardware and software are modelled as in Eq. (5) and Eq. (4) respectively. Next, in Eq. (9) we apply the proposed algorithm, where the total data communication penalty of computation units 1 and 3, on computation unit 2 (similarly for 4 & 5) is assumed to be equal to the sum of individual data communication penalties.

$$T_{ex} = \sum_{1}^{5} T_{C_i} \tag{7}$$

$$\begin{gathered} T_{C_1} = HT_1, T_{C_3} = HT_3, T_{C_2} = f_2(C_2, \Delta \backslash C_2), \\ T_{C_4} = f_4(C_4, \Delta \backslash C_4), T_{C_5} = f_5(C_5, \Delta \backslash C_5) \end{gathered} \tag{8}$$

$$f_2(C_2, \Delta \backslash C_2) = ST_2 + f_2(C_2, C_1) + f_2(C_2, C_3) \tag{9}$$

Thus, based on the proposed algorithm, we generalize the execution time model for C_n in Eq. (10). Here, the three terms refer to Hardware Time, Software Time and data communication penalty of C_n due to computation units which are placed in hardware. In Eq. (10), the "$\wedge$" in the last term of the data communication penalty refers to the logical AND operator and implies, if $M_{nj} > 0$ and $M_{ij} > 0$, then M_{nj} is considered in the summation. The impact of the terms of Eq. (10) for the two cases of C_n in hardware ($x_n = 1$) and software ($x_n = 0$) are shown in Eq. (11) and Eq. (12) respectively.

$$f_n(C_n, \Delta \backslash C_n) = \underbrace{(HT_n * x_n)}_{\text{Hardware Time}} + \underbrace{(ST_n * (1 - x_n))}_{\text{Software Time}} + \underbrace{(\Omega * (1 - x_n) * (\sum_{C_i \in \Delta \backslash C_i} x_i * \sum_{M_j \in \Gamma} (M_{nj} \wedge M_{ij}) * M_{nj}))}_{\text{Data Communication Penalty}} \tag{10}$$

$$f_n(C_n, \Delta \backslash C_n) = HT_n + 0 + 0 \tag{11}$$

$$f_n(C_n, \Delta \backslash C_n) = 0 + ST_n + \Omega * (\sum_{C_i \in \Delta \backslash C_i} x_i * \sum_{M_j \in \Gamma} (M_{nj} \wedge M_{ij}) * M_{nj}) \tag{12}$$

Thus, the objective can be expressed as Eq. (13). Substituting Eq. (10), we derive the final objective function in Eq. (14). Equation (15) - (18) represent the design constraints.

$$minimize \sum_{C_i \in \Delta} T_{C_i} \tag{13}$$

Objective function:

$$minimize \left(\sum_{C_n \in \Delta} (HT_n * x_n) + \sum_{C_n \in \Delta} (ST_n * (1 - x_n)) + \sum_{C_n \in \Delta} \Omega * (1 - x_n) * (\sum_{C_i \in \Delta \backslash C_i} x_i * \sum_{M_j \in \Gamma} (M_{nj} \wedge M_{ij}) * M_{nj}) \right)$$

$$where\ i, n = 1, 2, 3, \cdots, c;\ j = 1, 2, 3, \cdots, m \tag{14}$$

Subject to:

$$LUT\ Constraints: \sum_{C_i \in \Delta} x_i * L_i \leq \zeta \quad \forall C_i \in \Delta \tag{15}$$

$$DSP\ Constraints: \sum_{C_i \in \Delta} x_i * D_i \leq \xi \quad \forall C_i \in \Delta \tag{16}$$

$$FF\ Constraints: \sum_{C_i \in \Delta} x_i * F_i \leq \lambda \quad \forall C_i \in \Delta \tag{17}$$

$$Power\ Constraints: \sum_{C_i \in \Delta} x_i * P_i \leq \rho \quad \forall C_i \in \Delta \tag{18}$$

As a result of the heuristic formulation, the execution time obtained (T_{ex}) from solving Eq. (14) contains an additional penalty component due to the summation of individual data communication costs. Thus, we perform a post optimization process on the obtained solution. The post optimization is carried out using Algorithm 1 to calculate the post optimization penalty (T_{opt}). Initially, in Algorithm 1, for

Algorithm 1 Post Optimization Algorithm

Input: computation units (Δ), memory blocks(Γ), placement vector (x_i)
Output: post optimization penalty T_{opt}

```
1: for i ∈ Δ do
2:     for n = i + 1 ∈ Δ do
3:         if x_i & x_n then
4:             for j ∈ Γ do
5:                 if M_ij ∧ M_nj then
6:                     for v ∈ Δ do
7:                         if (1 − x_v) ∧ M_vj then
8:                             penalty = penalty + M_vj
9:     done
10: done
11: T_opt = Ω ∗ penalty;
```

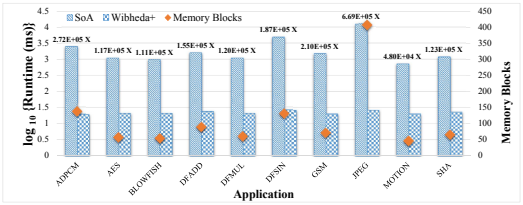


Figure 4: Criteria 1: Runtime Comparison

all combinations of two computation units in hardware we identify the overlapping memory accesses and populate the overlapping memory access vector (lines 1 - 5). Next, for each computation unit in software, we compare the overlapping memory access vector with the memory access pattern of the computation unit and if an overlap is identified the corresponding memory access value (M_{vj}) is added to the penalty. This process is repeated for all the computation units in software (lines 6 - 8). Finally, T_{opt} is calculated as in line 11 and the final execution time (T_f) using Eq. (19).

$$T_f = T_{ex} - T_{opt} \tag{19}$$

4. RESULTS AND DISCUSSION

Altera Cyclone V FPGA is used as the target device for feature extraction using LegUp. All experiments were carried out on a virtual machine running Ubuntu on an Intel Xeon E5-1650V2 CPU host at 3.5 GHz with 8 GB RAM. To test the scalability and accuracy of Wibheda+, we have used 10 benchmarks from the CHStone benchmark suite [11]. As mentioned in Section 3.2.1, we considered the 25 most frequently executed computation units of each benchmark. Evaluation of Wibheda+ is done under three criteria.

Criteria 1: The runtime of Wibheda+ and the SoA [15] is shown in Fig. 4, where the horizontal axis represents the benchmarks while the vertical axes on left and right represent the log_{10} value of runtime and the number of memory blocks respectively. Here, we use logarithmic values for clarity. As observed, the runtime of Wibheda+ is in the order of **milliseconds** while that of the SoA is in the order of minutes. Also, the speedup obtained by Wibheda+ with respect to an exhaustive approach for each benchmark is indicated as numbers above the bar in Fig. 4. Here, we see the significant speedup with increasing number of memory blocks due to the reduction in time complexity, from exponential in the exhaustive approach to quadratic in Wibheda+.

Criteria 2: Criteria 2 compares the performance of Wibheda+ and the SoA [15] in terms of the accuracy of results under stringent area constraints (LUT). Figure 5 shows the results for 9 applications while Fig. 1 in Section 1 shows the same for the *DFADD* application. Here, the horizontal axis indicates the area constraint in terms of LUT, distributed across the design space in each application while the vertical axis shows the deviation of estimated execution cycles from an exhaustive selection technique. Also, the legend indicates the average error across the design space. As can be seen in Fig. 5, Wibheda+ shows insignificant errors in most cases with a maximum error of only 13% at an area constraint of 1000 in Fig. 5i. However, the SoA work shows significant errors at multiple constraints in most applications with the maximum error exceeding 100% as can be seen in Fig. 1.

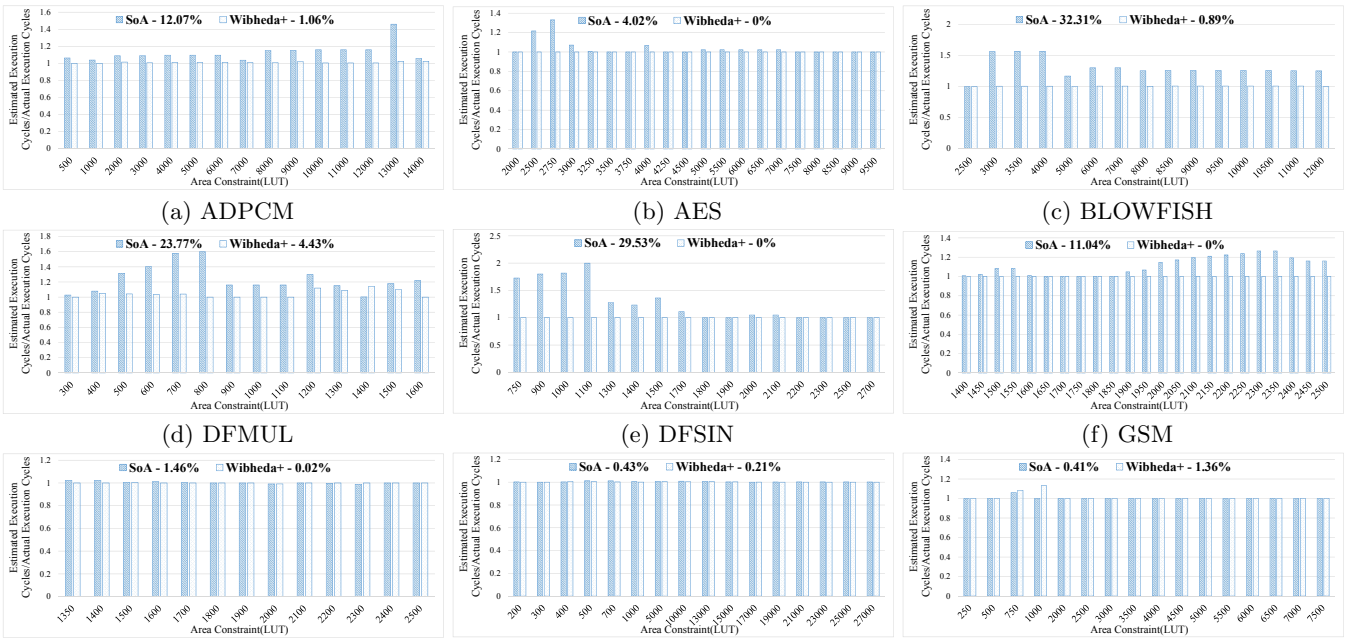
The SoA considers number of basic blocks as its design constraint. Typically, area constraints are provided as LUT, FF, etc., which are the actual resources available on an FPGA. Even though basic blocks can be expressed in FPGA resources, the resource consumption of different basic blocks typically vary significantly across an application as the sizes of basic blocks are not equal. Thus, representation of FPGA resources using a metric such as basic blocks can result in notable loop holes in obtaining optimal design points under resource constraints. This results in the selection of sub-optimal design points, which can have large deviations from the optimal design point. Averaging across all applications Wibheda+ shows an average estimation error of only **1.17%** in comparison to the SoA work which has an error of 16.76%. Thus, the accuracy of estimation under such stringent LUT constraints reflect the suitability of Wibheda+ across the full design space of an application.

Criteria 3: Wibheda+, not only improves the accuracy of a LUT constrained design but also advances the SoA by incorporating multiple design constraints. Thus, criteria 3 performs a holistic system evaluation of a SoC design under different LUT, DSP, FF and power constraints using constraints representing latest FPGA devices, Lattice iCE40 Ultra, ECP2/M and Intel Cyclone 10 LP series targeted for IoT applications. The design constraints used in the experiments are shown in Table 1. However, we evaluate results only against an exhaustive approach, due to the inability of the SoA to consider multiple design constraints.

The performance achieved by accelerators recommended by Wibheda+ is nearly identical to ones from the exhaustive approach for most of the benchmarks in all experiments. The DFMUL application exhibits slightly lower performance in experiments 1 and 2. The accelerators selected by Wibheda+ in this case are moderately different than the ones selected by the exhaustive approach due to the overestimation of the communication penalty as discussed in Eq. (7) - (9), where the ILP is formulated assuming the overall penalty to be equal to the sum of individual penalties. Thus, when multiple computation units access the same memory block Wibheda+ erroneously incurs an additional penalty resulting in an error. However, the accelerators recommended by Wibheda+ for this application still achieves over 97% of the performance achieved by the exhaustive approach, but in a fraction of time. Overall, the average difference in performance across the 3 experiments is only **0.16%**. This affirms the suitability of the fine-grained acceleration achieved by Wibheda+ in system level design targeted for IoT devices with multiple, severely strict design constraints.

5. CONCLUSION

In this paper, we proposed Wibheda+, a framework for rapid fine-grained data dependency-, area- and power- constraint aware HW-SW partitioning. The proposed techniques achieve average estimation errors of only 1% across



(a) ADPCM (b) AES (c) BLOWFISH

(d) DFMUL (e) DFSIN (f) GSM

(g) JPEG (h) MOTION (i) SHA

Figure 5: Criteria 2: Comparison of Estimated Performance

Table 1: Criteria 3: Constraints for System Evaluation

Experiment No.	Constraint			
	LUT (ζ)	DSP (ξ)	FF (λ)	Power (ρ) (mW)
1	1000	4	1000	50
2	5000	8	5000	500
3	10000	40	10000	3000

various applications from the CHStone benchmark suite. It has also been shown to be extremely fast with runtime in the order of milliseconds as opposed to minutes/hours taken by an SoA/exhaustive approach for the same applications. In future, we plan to consider coarse-grained acceleration to further improve the application runtime, while still considering the various constraints used in this work.

Acknowledgment

This research project is partially funded by the National Research Foundation Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme with the Technical University of Munich at TUM-CREATE.

6. REFERENCES

[1] LLVM, Compiler Infrastructure.

[2] S. Anton. Intel Announces Cyclone 10 FPGAs for IoT Devices, 2017.

[3] P. Arato et al. Hardware-software partitioning in embedded system design. In *ISISP6*, 2003.

[4] P. Biswas et al. Introduction of local memory elements in instruction set extensions. In *DAC*, 2004.

[5] T. Brooks. Hardware-Software Partitioning in Embedded Systems—Barr Group, 2015.

[6] J. J. Burg et al. Experiments with the Oregon Trail Knapsack Problem. *Information Processing Letters*, 1999.

[7] A. Canis et al. Legup. *ACM TECS*, 2013.

[8] D. Chen et al. Platform choices and design demands for IoT platforms: cost, power, and performance tradeoffs. *IET CPS: Theory Appl*, 2016.

[9] H. Demel. FPGAs solve challenges at the core of IoT implementation—EDN, 2016.

[10] K. Garg et al. KnapSim - Run-time efficient hardware-software partitioning technique for FPGAs. In *SOCC*, 2015.

[11] Y. Hara et al. Proposal and Quantitative Analysis of the CHStone Benchmark Program Suite for Practical C-based High-level Synthesis. *Journal of Information Processing*, 2009.

[12] J. Henkel et al. An approach to automated hardware/software partitioning using a flexible granularity that is driven by high-level estimation techniques. *TVLSI*, 2001.

[13] Lattice Semiconductor. iCE40 Ultra/UltraLite/ UltraPlus-Lattice Semiconductor, 2017.

[14] A. Prakash et al. Modelling communication overhead for accessing local memories in hardware accelerators. In *ASAP*, 2013.

[15] A. Prakash et al. Rapid Memory-Aware Selection of Hardware Accelerators in Programmable SoC Design. *TVLSI*, 2017.

[16] M. U. Sharif et al. Hardware-software Codesign of RSA for Optimal Performance vs. Flexibility Trade-off. In *FPL*, 2016.

[17] F. Sun et al. A Synthesis Methodology for Hybrid Custom Instruction and Coprocessor Generation for Extensible Processors. *TCAD*, 2007.

[18] D. Wijesundera et al. Rapid Design Space Exploration for Soft Core Processor Customization and Selection. In *FPT*, 2016.

[19] P. Wilson. Chapter 2 - An FPGA Primer. In *Design Recipes for FPGAs*. Newnes, Second edition, 2016.

[20] W. Zuo et al. Accurate High-level Modeling and Automated Hardware/Software Co-design for Effective SoC Design Space Exploration. In *DAC*, 2017.