A Comparison of Different Optimization Algorithms for HW/SW Partitioning Using a High-Performance Cluster

Samah Rahamneh
Computer Engineering Department
The University of Jordan
Amman, Jordan
s.rahamneh@ju.edu.jo

Alvis Fong
Computer Science Department
Western Michigan University
Kalamazoo, MI, USA
alvis.fong@wmich.edu

Lina Sawalha
Computer and Electrical Engineering Department
Western Michigan University
Kalamazoo, MI, USA
lina.sawalha@wmich.edu

***Abstract*—Hardware/Software (HW/SW) co-design exploits the synergy between software and hardware to fulfill system design constraints. System designers have utilized diverse optimization algorithms to set boundaries between software and hardware. Discrete Particle Swarm Optimization (DPSO) and Genetic Algorithm (GA) are efficient meta-heuristic algorithms for HW/SW partitioning. However, these algorithms might suffer from premature convergence. Moreover, the accuracy and the speed of convergence of PSO depend on control parameters, which might vary among different applications. In this work, we extended DPSO and GA with distributed greedy local search mechanisms that improve the performance of DPSO and GA. We also tuned the acceleration parameters of DPSO using a neural network. We partitioned real-world applications implemented using OpenCL nd Intel's Hardware Research Acceleration Program (HARP) infrastructure. The results show that DPSO with tuned parameters improves the accuracy of DPSO by up to 62.8%, and its execution time by up to 29%. On the other hand, local search-based DPSO improves the accuracy of DPSO by up to 55.4%, and the local search-based GA improves the accuracy of GA by up to 82.6%. However, the local search-based technique increases the execution time of the algorithm.**

***Index Terms*—HW/SW partitioning, particle swarm optimization, genetic algorithm, machine learning, CPU-FPGA platforms.**

I. Introduction

Hardware/Software (HW/SW) co-design exploits the synergy between software and hardware, Field Programmable Gate Array (FPGA), to fulfill system design requirements and constraints. These requirements include execution time, FPGA utilization, and energy consumption. HW/SW partitioning is a crucial stage in HW/SW co-design. HW/SW partitioning is dividing an application into SW components and HW components. The HW components are implemented on the FPGA while the SW components are executed on the FPGA.

HW/SW partitioning is an optimization problem that aims at achieving a predefined objective function subject to a set of constraints. It could be formulated as a Single Objective Optimization (SOO) or Multi-Objective Optimization (MOO) problem. Many research studies carried out the partitioning process manually. However, the majority of research studies have used algorithms to perform the partitioning. These algorithms are mainly divided into three categories: exact, meta-heuristic, and hybrid algorithms. Exact algorithms fully explore the design space and find the optimal solutions. However, they are cumbersome and slow when used in a large/complex design space [1]. On the other hand, meta-heuristic algorithms balance the exploration and exploitation of design space to find sub-optimal solution(s). Hence, meta-heuristic algorithms are faster than exact algorithms [1]. In hybrid algorithms, a combination of two or more algorithms are used [2].

Due to the deployment of FPGAs in cloud data centers (e.g Figure 1), HW/SW co-design is used to meet the requirements not only for Embedded Systems (ESs) but also for cloud and High Performance Computing (HPC) applications [3]. Examples of cloud and data center systems that integrate FPGAs are Amazon EC2 [4], Intel's Hardware Research Acceleration Program [5] and Microsoft Catapult [6]. These emerging hybrid architectures along with high-level synthesis tools allow for a variety of HPC and cloud applications to be accelerated and benefit from the heterogeneous mix of CPUs and FPGAs, targeting increased performance and systems utilization.

HW/SW partitioning is an NP-hard problem by nature. Emerging HPC and cloud applications are larger and often

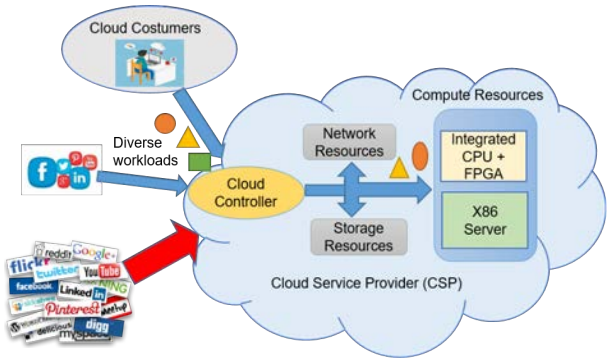

Fig. 1. Emerging CPU-FPGA hybrid architecture in a cloud data center.

process a large set of data, which increase the search space and make the problem more complex compared to the traditional ESs design space. As such, an agile and efficient partitioning algorithm is crucial to improve the overall system performance. Genetic Algorithm (GA) [7] and Particle Swarm Optimization (PSO) [8] are efficient population-based stochastic optimization algorithms. Discrete binary PSO (DPSO) was introduced to solve binary problems such as HW/SW partitioning [9]. GA and DPSO has been proposed and used for different HW/SW partitioning studies [10]–[12]. However, in a large, complex search-space these algorithms might be trapped in a local optima and suffer from premature convergence [1], [13], [14]. In addition, DPSO speed and solution cost depend mainly on its parameters, such as inertia value, cognitive parameter, and social parameter. Existing HW/SW partitioning studies that used the DPSO algorithm, used generally accepted values for these parameters.

In this paper, we extended GA and DPSO using distributed greedy local search mechanisms to mitigate the premature convergence of the algorithms. Moreover, we investigated the effect of DPSO parameters on its speed and solution cost and tuned the DPSO parameters using machine learning. The contributions of this paper are as follows:

- Partitioned real-world OpenCL applications such as $k-means$ clustering algorithm, $Canny$ edge detection algorithm, and Advanced Encryption Standard (AES) algorithm on one of the state-of-the-art CPU-FPGA architectures, Intel HARP v2.
- Developed a cost function that combines execution time, energy consumption and FPGA resource utilization and measured it for each .
  - Measured the execution time of applications on an Intel HARP v2 node for both CPU and FPGA.
  - Estimated the energy consumption of applications considering published power consumption of different FPGA structures, and technology scaling.
- Mitigated the problem of premature convergence of the GA and the DPSO algorithms using distributed greedy local search mechanisms.
- Tuned the parameters of DPSO using Artificial Neural Network (ANN).

This paper is organized as follows: Section II presents the related work. Section III discusses the PSO and the GA algorithms, and the HW/SW partitioning problem. Section IV describes our methodology and experimental setup. Section V discusses the DPSO algorithm variations. The results are presented and discussed in section VI. Finally, Section VII concludes the paper.

II. Related Work

HW/SW partitioning techniques are classified mainly into two main categories: manual and automatic. In manual partitioning, programmers partitions the source code into SW and HW components [15]. They depend on their understanding of the application's source code and the system's architecture. As such, manual partitioning is slow and can result in unoptimized partitioning, depending on the size of applications and experience of programmers/engineers. Automated partitioning techniques, which utilize optimization algorithms to partition an application into SW and HW components, is desired, especially with the increasing size of applications today.

There are three main categories of automatic partitioning algorithms: exact, meta-heuristic, and hybrid algorithms. Using exact algorithms, an application is normally represented as a graph or a Finite State Machine (FSM). Then, an exact algorithm partitions the graph in a way that minimizes a linear objective function [16]. The objective function could be performance or energy consumption. This model results in an optimal solution; however, it's predominant drawback is time complexity when exploring a complex design space. This leaves the exact algorithms inefficient when exploring large design spaces [17].

Meta-heuristic (MH) algorithms, on the other hand, explore the search space intelligently to find sub-optimal solutions. As these algorithms do not exhaustively explore the search space, they offer no guarantees of finding the optimum solution. They are faster than the exact algorithms and can find an approximate solution in a reasonable time. In hybrid algorithms, a combination of exact/MH or MH/MH algorithms is used to achieve an efficient partitioning in a reasonable time [10].

There exist many different optimization algorithms and works for the HW/SW partitioning problem. As such, it is almost impossible to compare and contrast all of them. In this work, we focus on GA and DPSO algorithms and their variations as they have been used for different partitioning problems [2], [11], [12]. Table I summarizes various combinations of DPSO and GA with other exact and MH algorithms. It also shows the efficiency of our variations of the DPSO, which are LPSO and APSO, over the original algorithm in terms of the quality of the resultant partitioning solutions and the algorithm execution time.

Furthermore, our study handled real application graphs as compared to existing work that considered random graphs [16], [18]. Finally, the majority of research studies focused on optimizing a single objective function, for instance, reducing the execution time [19], boosting throughput [20], reducing the energy consumption [21], and decreasing HW area [22], while some studies considered two objectives [18], [23]. Our work targets optimizing performance, power consumption, and FPGA area by calculating a cost function.

III. Background: PSO and GA in the context of HW/SW partitioning

In this section, we discuss the DPSO and GA algorithms in the context of HW/SW partitioning. We also formulate the HW/SW partitioning problem as a single-optimization function that combines different objectives (weighted sum of different metrics) in one cost function.

A. Particle Swarm Optimization Algorithm

Particle swarm optimization is a stochastic optimization technique developed by Kennedy and Eberhart [8]. They later

TABLE I
PSO AND GA IN HYBRID HW/SW PARTITIONING ALGORITHMS

Hybrid Partitioning approach	Publication	Contribution
GA & PSO	[2]	Generates a better quality solution compared to PSO and faster than GA. Slow exploration of design space compared to PSO.
FCM & PSO	[10]	Generates better partitioning solution in a shorter time compared to PSO and FCM. Applicable to both binary and extended partitioning but there is No consideration for communication cost between HW and SW.
TS & PSO	[2]	Combines the parallel nature of PSO with the memory feature of TS to reduce the PSO run-time for large graphs.
BB & PSO	[24]	Exploits the efficiency of PSO to speed up the partitioning process, BB generates more accurate partitioning decision compared to the proposed algorithm.
FEO & PSO	[25]	The authors a conformist PSO (CPSO) to avoid trapping in a local minimum and enhance search diversity. In order to improve the quality of the CPSO output, they combined the CPSO with fireworks explosion operations(FEO) which stimulates the swarm to traverse disparate regions looking for an optimal solution.
GA & TS	[26]	the authors used the TA as a local search technique with GA. This combination of TS and GA is one variation of Memetic Algorithm (MA). They demonstrate the robustness of the algorithm and its ability to generate a better quality solution as the expense of the execution time.
APSO	our approach	The operational parameters of the PSO are artificially tuned to improve the convergence speed and the quality of the partitioning solution.
LPSO	our approach	greedy search technique is used to mitigate trapping into local optimum. The further searching extension improves the accuracy/quality of the partitioning decision.

modified it to work with discrete binary problems [9], called PSO for the rest of the paper. The algorithm mimics the social behavior of animal herds and bird flocks in search for food. In order to search the space effectively, each particle has two attributes; the particle position X_i and velocity V_i.

Each particle uses equation 1 to update its velocity and equation 2 to update its position. Table II defines the parameters used in equations 1, and 2, where k indicates the number of iterations.

$$V_{k+1}^i = W_k V_k^i + c_1 r_1 (P_k^i - X_k^i) + c_2 r_2 (P_k^g - X_k^i) \quad (1)$$

$$X_{k+1}^i = X_k^i + V_{k+1}^i \quad (2)$$

The acceleration parameters of PSO, which include $c1$, $c2$, and w, play a significant role in balancing exploration and exploitation of the design space. Hence, these parameters significantly impact the convergence speed and accuracy of PSO [27], [28]. Moreover, these parameters might vary from one application to another. However, research studies that utilized PSO have used rule-thumbed (or generally accepted)

TABLE II
PSO PARAMETERS' DEFINITION

Parameter	Definition
c1	cognitive parameter (self confidence constant)
c2	social parameter (swarm confidence constant)
r1,r2	perturbation factors
W_k	inertia weight
X_i^k	particle position
V_k	particle velocity
P_k^i	particle best known position
P_k^g	swarm best known position

acceleration parameters. For instance, existing HW/SW partitioning work have chosen constant values in the range [0.4 - 0.9] for w [27], and in the range [0 - 2] for $c1$ and $c2$ [11], [25], [28]–[30]. The constant parameters were chosen based on accepted values used in the literature and not optimized for the HW/SW partitioning problem.

B. HW/SW Partitioning Problem Formulation

Mathematically, the HW/SW partitioning problem could be formulated as a single objective function or a combination of single-objective functions. The mathematical formulation depends on the system's ultimate objective and describes the output of the system. It could be either a minimization or a maximization problem. In SOO, as the name suggests, there is one goal to optimize. However, in MOO, the objective function has more than a single objective to optimize. Adding more objectives increases the complexity of the optimization problem. In addition, these objectives can be conflicting, so a trade-off often has to be made. The following shows different objective functions starting from a constrained single objective function to multi-objective formulation.

$$\begin{aligned} \min_x \quad & F(x) = w_1 f_1(x) + w_2 f_2(x), ... + w_M f_M(x) \\ \text{s.t.} \quad & f_i(x) \leq b_i \forall i \in M \end{aligned} \quad (3)$$

Each of these single objective functions ($f_1(x)$ to $f_M(x)$) represents one cost criterion which is execution time , energy consumption, or resource utilization (HW area). w_1, w_2, etc represent the weights of the each optimization objective and b_i represents a constraints on the objective, if applicable.

IV. METHODOLOGY

This section describes the different benchmarks we targeted, the software and hardware tools we used and the methodology that we followed for this study.

A. CPU-FPGA Hardware

We used Intel's second generation of the Hardware Research Acceleration Program and infrastructure (HARP v2) [5]. Each node in HARP consists of an Intel Broadwell Xeon CPU (14 cores) that is integrated with Intel Arria 10 GX 1150 FPGA into a multi-chip package. The CPU and the FPGA are connected through two PCIe Gen3x8 and one Quick Path Interface (QPI). The traffic between the CPU and the FPGA is distributed over these channels based on link utilization, using

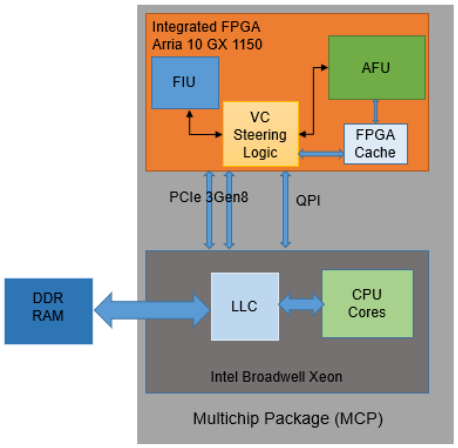

Fig. 2. HARP Architecture.

a Virtual Channel (VC) streaming unit as shown in Figure 2. The FPGA fabric consists of the FPGA's Interface Unit (FIU) and an Accelerator Functional Unit (AFU).

B. Applications

We modified/developed three Open Computing Language (OpenCL) applications to partition and run on a HARP v2 node. The applications are chosen to cover different real-world applications such as machine learning, image processing, and data security.

- k-means Clustering: This is an unsupervised clustering algorithm that takes a data set where each data point consists of N-dimensional observations. The algorithm divides the data set into K clusters depending on a certain similarity criterion, such as Euclidean distance. The algorithm is implemented in OpenCL [31].
- Advanced Encryption Standard (AES): Sharing and virtualization of cloud resources have enabled a tremendous number of customers to access a vast range of cloud services. However, this sharing has posed a security challenge for Cloud Service Providers (CSPs) [32]. AES is a symmetric block cipher encryption algorithm that is extensively used in the cloud.
- Canny Edge Detection Algorithm: Visual social media is exporting billions of images to the cloud daily [33]. Image processing algorithms, which are used to process these images, are data and compute-intensive. One of these algorithms is Canny edge detection. In our previous work, we accelerated *Canny* through a collaborative execution between CPU and FPGA [15].

C. Modeling applications and the objective function

Each application is modeled as a control graph. To generate a graph for each benchmark application, we utilized the LLVM compiler and toolchain to build the application graphs. We used the front end of the LLVM *Clang* to generate an IR code from the OpenCL code. Then we converted the IR code into CFG using a LLVM optimizer and analyzer. To evaluate the cost of each node of a graph, we need to assign cost parameters. These parameters include the execution time or latency of the node in SW, the latency in HW, SW energy consumption, HW energy consumption and HW area. In this study, we measured/estimated the five parameters that determine the total cost of each node. We measured the SW latency SW_L and the HW latency HW_L on a HARP node in μseconds. We estimated SW energy consumption SW_{En} in μ joules of each node using the Thermal Dissipate Power (TDP) document of the CPU and the latency. Moreover, we estimated HW energy consumption HW_{En} in μ joules by estimating the energy consumption of the different HW resources of the FPGA. We reused the measured energy consumption of these individual HW resources such as Adaptive Logic Modules (ALMs), Digital Signal Processors (DSPs), and registers from literature [34], [35]. We scaled the energy to reflect nanometer technology differences among the literature results (90 nm and 65 nm) to our FPGA (20 nm). To scale the results accurately, we built a Multi-variate Linear Regression (MLR) model to predict a proper scaling factor of energy among the different nanometer technologies. We trained and tested the MLR model on a data set generated using the Cacti 6.0 simulator. Finally, we used Intel's Offline Compiler resources utilization report that shows the number of Adaptive Logic Modules (ALMs), DSPs, and registers for FPGA utilization.

For each application, we divided the graph of each application into nodes. Each node is a set of Basic Blocks (BBs). The reason behind this grouping of BBs is to assign each node with a more accurate measure/estimate of the cost parameters. When assigning cost at the BB level, most of the cost parameters' values are not accurate. For instance, we cannot accurately measure the latency, or energy consumption at the level of one BB because it is very small in size, consisting of few instructions. As such, actual time measurements on the real hardware were not accurate. We used higher granularity than one BB by grouping from three to ten BBs, calling each group a *component*. Tables III, IV, V show the components of *k-means*, *Canny*, and *AES* respectively and the cost parameters of each component. Eventually, we have a control-flow graph for each application that consists of a set of components and each component has five cost parameters. We used equation 3 as a cost function that calculates a weighted sum of latency, energy consumption and FPGA utilization, with weights of 1, 0.6, and 0.3 respectively. The solutions are bounded by the CPU latency and energy consumption, and FPGA area.

TABLE III
COMPONENTS COST OF K-MEANS ALGORITHM.

Component	L_{SW}	L_{HW}	A_{HW}	E_{SW}	E_{HW}
C1	2354	100	213600	235400	310
C2	10	3	213600	1000	310
C3	1313939	14154	209328	131393900	250
C4	985455	10615	209328	98545500	270
C5	1642424	17692	222144	164242400	260
C6	656969	7077	170880	65696900	249
C7	1970908	7077	222144	197090800	260

V. DISCUSSION

A. LPSO: PSO-based greedy distributed local search

To mitigate the premature convergence (converging to a local optimal solution) of the PSO algorithm, we implemented

TABLE IV
COMPONENTS COST OF *Canny* EDGE DETECTION ALGORITHM.

Component	L_{SW}	L_{HW}	A_{HW}	E_{SW}	E_{HW}
C1	50	42	170880	500	350
C2	50	42	170880	500	350
C3	258	48	209328	25800	360
C4	250	48	209328	25000	350
C5	311	132	213600	31100	280
C6	311	132	213600	31100	280
C7	187	79	209328	18700	280
C8	437	185	222144	43700	280
C9	400	194	222144	40000	280
C10	714	339	226416	71400	420
C11	135	100	209328	13500	320
C12	135	100	209328	13500	320

TABLE V
COMPONENTS COST OF ADVANCED ENCRYPTION STANDARD ALGORITHM.

Component	L_{SW}	L_{HW}	A_{HW}	E_{SW}	E_{HW}
C1	6	7	209328	630	300
C2	50	21	128160	525	180
C3	34	14	85440	3400	180
C4	30	11	205056	3000	170
C5	17	6	170880	1700	160
C6	34	20	128160	3400	160
C7	29	13	85440	2900	170
C8	7	9	209328	700	300
C9	6	7	209328	600	300
C10	50	21	128160	5000	180
C11	34	14	85440	3400	180
C12	58	22	128160	5800	170
C13	38	14	85440	3800	160
C14	17	6	205056	1700	160
C15	30	11	209328	3000	170
C16	7	9	209328	700	300

a distributed greedy local search at the end of each iteration of the PSO algorithm. For each particle in the swarm, we locally search for a neighboring particle that has a lower cost. If we find such a particle, we replace the original particle. Otherwise, we leave the original particle. We apply the local search around all particles in the swarm and at the end of each iteration. The modified algorithm either outperforms the original PSO (in most cases) or gives similar results.

B. Memetic algorithm (MA): GA-based local search

GA also suffers from the premature convergence when applied to HW/SW partitioning. Hence, we implemented a greedy local search around each chromosome in the population at the end of each generation. This search aims at finding a better quality partitioning solution around each solution in the population. If the local search finds a better solution, we replace the original chromosome with the neighboring chromosome at a lower cost.

C. APSO: Artificially-tuning PSO parameters using machine learning

Most of the literature used acceptable (fixed) parameter values for the PSO algorithm. In this work, we use a machine learning technique to optimize the parameters, increase the accuracy of the algorithm and reduce its execution time. We specifically use the Artifical Neural Network (ANN) algorithm to tune the PSO parameters.

To train the ANN algorithm to tune the PSO acceleration parameters, we generated a data set from the aforementioned applications. The data set size is 3200 records for each application with a total of 9600 records for the three applications. Each record in the dataset consists of source-level characteristics of the application, such as the number of *for loops* and *if statements*, Intermediate Level (IR) level characteristics such as the number of BBs, and static and dynamic characteristics (e.g. the number of branch instructions, the number of branches mis-predictions and Cycles Per Instruction (CPI)). In addition, it includes the number of Compute Units (CUs) that represents the replication of the OpenCL kernel in hardware.

The cost of the resulting partitioning decision and the PSO execution time were calculated using a range of possible acceleration parameters. For instance, we varied the inertia weight (w) in the range [0 - 4] with a step size of 0.5 and the acceleration coefficients ($c1$ & $c2$) in the range [0 - 10] with a step size of 0.5. We noticed that different values of the acceleration parameters and inertia value affect the cost of PSO solution and its execution time.

The ANN algorithm consists of two hidden layers with (16 - 32) nodes each. We used 1000 epochs to forward and backward propagation to adjust the weights. Adaptive Moment Estimation (Adam) optimizer is used, among all other optimizers, since it combines good features of other optimizers such as of Adadelta and RMSprop. We used cross validation and evaluated the ANN algorithm using Root Mean Squared Error (RMSE) as a loss function.

VI. RESULTS

This section presents and discusses the results of the different partitioning algorithms: GA, MA, PSO, APSO, and LPSO in terms of the partitioning cost and the partitioning latency. The experiments were conducted using different numbers of iterations (10, 30, 60) and for ten different sizes of the population (10-100) with a step of 10. Because the aforementioned optimization algorithms depend on randomly generated seed variables and population, we ran each experiment 10 times and recorded the average to observe the accuracy of these algorithms. For each pair of an iteration number and a population size, 10 different randomized populations were generated and then used to run the different optimization algorithms. The experimental parameters of GA and DPSO are shown in Table VI.

Figures 3, 4, 5 shows the partitioning cost using the different optimization algorithms, GA, MA, discrete PSO, APSO and LPSO. Figure 3 shows the partitioning cost of the *k-means* benchmark using different sizes of the population and 10, 30, 60 iterations. As shown, APSO results in an average percent reduction in the cost of 4.2% and up to 10%. This is because the APSO parameters are optimized for this benchmark in a way that guides the swarm to find a more accurate solution. LPSO also outperforms PSO as it conducts a more extensive

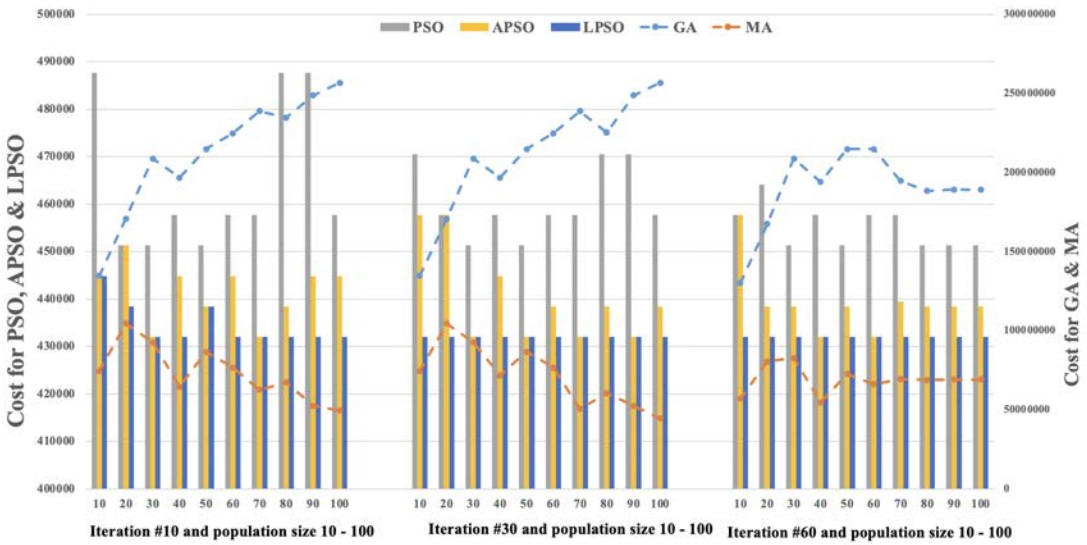

Fig. 3. Partitioning cost of GA, MA, PSO, APSO and LPSO using 10, 30, and 60 iterations and different sizes of population for k-means algorithm.

TABLE VI
PSO AND GA EXPERIMENTAL PARAMETERS.

Parameter	Value
PSO	
Population size	10, 20, 30, 40, 50, 60, 70, 80, 90, 100
Particle size	7, 12, 16
Ma. no. of iterations	10, 30, 60
Cognitive value	2
Social value	2
Inertia weight	1 - 0.3
GA	
Population size	10, 20, 30, 40, 50, 60, 70, 80, 90, 100
Chromosome dimensions	7, 12, 16
Max. no. of iterations	10, 30, 60
Mating pool size	3, 6, 8
Cross over rate	0.5
Mutation	single point

greedy search than PSO in the neighborhood of the leading particles in the swarm. It results in up to 11.4% reduction in the cost and an average of 5.9% cost reduction. The average here is taken for all iterations/population sizes.

On the other hand, MA improves the fitness of the solution compared to GA by up to 82.6% and 63.6% on average. This is due to the ability of MA to improve the divergence of the population and replace some solutions in the population with lower cost solutions. The GA algorithm results in a lower partitioning cost when the size of the population is ten. For the same number of iterations, increasing the size of GA population affect the quality of the solution negatively. The reason is a larger population needs more iterations to reach a better solution.

Figure 4 shows the partitioning cost of the *Canny* benchmark using different sizes of the population and 10, 30, 60 iterations. APSO reduces the cost by 30.4% on average and up to 62.8%. LPSO also outperforms PSO by up to 55.4% and an average of 25.4% cost reduction. The average cost of APSO and LPSO is comparable. On the other hand, MA reduces the cost of the solution by up to 26.3% and an average of 18.7% compared to GA.

Figure 5 shows the partitioning cost of the *AES* using different sizes of the population and iterations. APSO outperforms PSO with up to 17.5% and an average of 4.4% cost reduction. LPSO also reduces the cost over PSO by up to 23.4% and an average of 5.1%. However, these algorithms give the same level of cost when the population size is larger than 60. On the other hand, MA improves the solution cost compared to GA with up to 40% and an average of 33% reduction of the cost, and gives better results for all different sizes of the population.

The number of iterations and the population size that results in the lowest cost vary from one optimization algorithm to another. It also varies based on the graph size of the application. Our results show that the best iteration/populations for APSO is 10/50 and 10/80 for LPSO. LPSO requires a larger population than the PSO as LPSO performs an extensive local search around the swarm best solution, which requires more particles to perform the search. The best number of iterations for MA is 20 with a population between 90 and 100, while the GA algorithm favors an iteration number of 30 with a low population of 10 considering our three applications. This is because MA performs an extensive local search compared to GA, and that requires more individuals.

To verify the accuracy of one of our partitioning estimated cost and execution time with the actual measurements on hardware, we implemented two solutions on a HARP node and measured the actual execution time. The results show an overhead in the execution time of 3% and 2.9% for the two experiments for Canny, which is due to communication.

Finally, we measured the execution time of the optimization algorithms. The APSO algorithm has the lowest execution time among all other optimization algorithms for all different num-

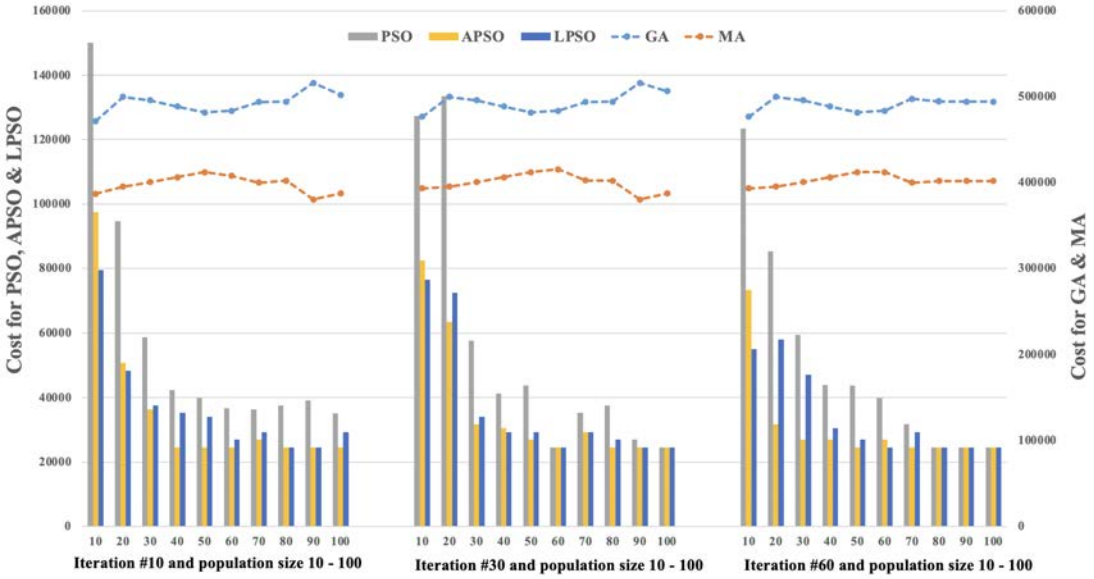

Fig. 4. Partitioning cost of GA, MA, PSO, APSO and LPSO using 10, 30, and 60 iterations and different sizes of population for Canny algorithm.

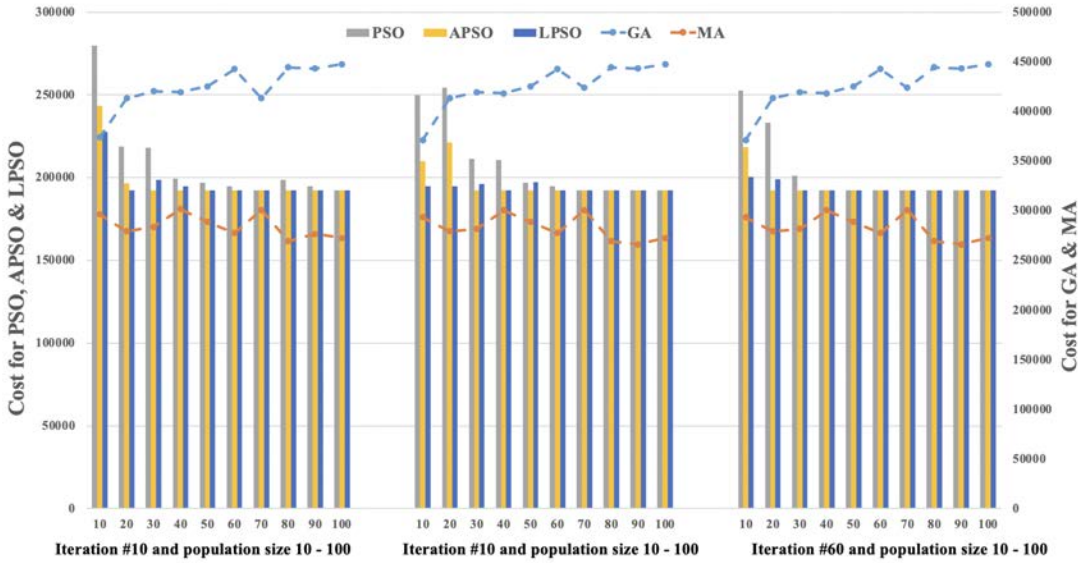

Fig. 5. Partitioning cost of GA, MA, PSO, APSO and LPSO using 10, 30, and 60 iterations and different sizes of population for AES algorithm.

bers of iterations and population sizes. This is because APSO has intelligently-tuned the acceleration parameters. APSO is faster than PSO by up to 29%. On the other hand, PSO has lower execution time than GA, MA, and LPSO. This is due to the slow genetic operators that are used in GA and MA and the local search extensions used in MA and LPSO. We also found that increasing both the population size and the number of iterations increase the execution time of the partitioning algorithms as both require more computation and communication among the population's individuals during each iteration.

VII. Conclusion

HW/SW co-design has been used to fulfil system design requirements and achieve its constraints. The growing complexity of the design space and the emergence of FPGA in high-performance and cloud computing systems have demanded more efficient partitioning solutions. In this work, we considered real-world application implemented using OpenCL, including *K-means*, *Canny* edge detector, and *AES* algorithms. We partitioned the applicatios into components, and we used the Intel's HARP-v2 infrastructure to measure/estimate the execution time, energy consumption and FPGA utilization of each component, and to validate our results.

In addition, we used a cost function that combines different objectives such as the execution time, the HW area, and the energy consumption. We targeted PSO and GA population-based optimization algorithms, which have demonstrated their efficiency in HW/SW partitioning. However, these algorithms

suffer from premature convergence. Also, the speed of convergence and solution cost of PSO depend on its acceleration parameters. Most researcher use PSO with rule-thumbed parameters. These parameters might need to be changed from one problem to another to produce better solutions. In this study, we tuned these parameters using a neural network algorithm that produces a PSO variation that is more accurate than PSO by up to 62.8% and faster by up to 29%. Moreover, we extended the PSO and the GA algorithms with a distributed greedy local search mechanisms that mitigate the premature convergence. The local-search-based PSO improves the accuracy of PSO by up to 55.4% at the expense of the execution time. A GA-based local search technique (MA) was proved to improve the accuracy of MA by up to 82.6% at the expense of the execution time.

VIII. Acknowledgement

We would like to thank the reviewers for their valuable feedback. This work was funded by NSF award no. 1821691, and Intel Inc. equipment access grant to HARP v2.

References

[1] I. Mhadhbi, S. B. Othman, and S. B. Saoud, "A comprehensive survey on hardware/software partitioning process in co-design," *International Journal of Computer Science and Information Security (IJCSIS)*, vol. 14, no. 3, 2016.

[2] Y. Wu, H. Zhang, and H. Yang, "Research on parallel hw/sw partitioning based on hybrid pso algorithm," in *International conference on algorithms and architectures for parallel processing*, 2009, pp. 449–459.

[3] C. Kachris and D. Soudris, "A survey on reconfigurable accelerators for cloud computing," in *26th International conference on field programmable logic and applications (FPL)*, 2016, pp. 1–10.

[4] Amazon EC2 F1 Instances. https://aws.amazon.com/ec2/instance-types/f1/, Accessed Nov. 2021. [Online]. Available: https://aws.amazon.com/ec2/instance-types/f1/

[5] "Hardware Accelerator Research Program." [Online]. Available: https://software.intel.com/en-us/hardware-accelerator-research-program, Accessed Apr. 2020

[6] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim *et al.*, "A cloud-scale acceleration architecture," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 1–13.

[7] G. Li, J. Feng, J. Hu, C. Wang, and D. Qi, "Hardware/software partitioning algorithm based on genetic algorithm." *Journal of Computers*, vol. 9, no. 6, pp. 1309–1315, 2014.

[8] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *MHS. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, 1995, pp. 39–43.

[9] J. Kennedy and R. Eberhart, "A discrete binary version of the particle swarm algorithm," in *IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*.

[10] I. Mhadhbi, S. Ben Othman, and S. Ben Saoud, "An efficient technique for hardware/software partitioning process in codesign," *Scientific Programming*, vol. 2016, pp. 1–11, 2016.

[11] Q. Zhai, Y. He, G. Wang, and X. Hao, "A general approach to solving hardware and software partitioning problem based on evolutionary algorithms," *Advances in Engineering Software*, vol. 159, pp. 1–22, Sept. 2021, 102998. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0965997821000272

[12] R. Akeela and M. P. Krawiec-Thayer, "Efficient hw/sw partitioning of halo: Fpga-accelerated recursive proof composition in blockchain," *Microsystem Technologies*, pp. 1–11, 2021.

[13] S.-R. Kuang, C.-Y. Chen, and R.-Z. Liao, "Partitioning and pipelined scheduling of embedded system using integer linear programming," in *11th International Conference on Parallel and Distributed Systems, Proceedings.*, 2005, pp. 37–41.

[14] P. Arató, Z. Á. Mann, and A. Orbán, "Algorithmic aspects of hardware/software partitioning," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 10, no. 1, pp. 136–156, 2005.

[15] S. Rahamneh and L. Sawalha, "An opencl-based acceleration for canny algorithm using a heterogeneous cpu-fpga platform," in *IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, pp. 322–322.

[16] W. Jigang, B. Chang, and T. Srikanthan, "A hybrid branch-and-bound strategy for hardware/software partitioning," in *2009 Eighth IEEE/ACIS International Conference on Computer and Information Science*, 2009, pp. 641–644.

[17] W. Zuo, L.-N. Pouchet, A. Ayupov, T. Kim, C.-W. Lin, S. Shiraishi, and D. Chen, "Accurate high-level modeling and automated hardware/software co-design for effective soc design space exploration," in *Proceedings of the 54th Annual Design Automation Conference*, 2017, pp. 1–6.

[18] K. Mourad and R. Boudour, "A modified binary firefly algorithm to solve hardware/software partitioning problem," *Informatica*, vol. 45, no. 7, 2021.

[19] Z. Wang, B. He, and W. Zhang, "A study of data partitioning on opencl-based fpgas," in *25th International Conference on Field Programmable Logic and Applications (FPL)*, 2015, pp. 1–8.

[20] W. Liu, W. Li, P. S. Un, and Y. B. Cho, "High-throughput hw-sw implementation for mv-hevc decoder," in *International SoC Design Conference (ISOCC)*, 2018, pp. 226–228.

[21] K. Neshatpour, M. Malik, M. A. Ghodrat, A. Sasan, and H. Homayoun, "Energy-efficient acceleration of big data analytics applications using fpgas," in *IEEE International Conference on Big Data (Big Data)*, 2015, pp. 115–123.

[22] E. Manor and S. Greenberg, "Efficient hardware/software partitioning for heterogeneous embedded systems," in *IEEE International Conference on the Science of Electrical Engineering in Israel (ICSEE)*, 2018, pp. 1–4.

[23] A. Iguider, A. En-Nouaary *et al.*, "Hw/sw partitioning algorithms for multi-objective optimization in embedded systems," *International Journal of Information Science and Technology*, vol. 2, no. 2, pp. 19–28, 2018.

[24] T. Eimuri and S. Salehi, "Using dpso and b&b algorithms for hardware/software partitioning in co-design," in *Second international conference on computer research and development*, 2010, pp. 416–420.

[25] X. Yan, F. He, N. Hou, and H. Ai, "An efficient particle swarm optimization for large-scale hardware/software co-design system," *International Journal of Cooperative Information Systems*, vol. 27, no. 01, p. 1741001, 2018.

[26] G. Lin, W. Zhu, and M. M. Ali, "A tabu search-based memetic algorithm for hardware/software partitioning," *Mathematical Problems in Engineering*, vol. 2014, 2014.

[27] Y. Shi and R. C. Eberhart, "Parameter selection in particle swarm optimization," in *International conference on evolutionary programming*, 1998, pp. 591–600.

[28] M. Juneja and S. Nagar, "Particle swarm optimization algorithm and its parameters: A review," in *International Conference on Control, Computing, Communication and Materials (ICCCCM)*, 2016, pp. 1–5.

[29] A. Farmahini-Farahani, M. Kamal, S. M. Fakhraie, and S. Safari, "Hw/sw partitioning using discrete particle swarm," in *Proceedings of the 17th ACM Great Lakes symposium on VLSI*, 2007, pp. 359–364.

[30] X.-H. Yan, F.-Z. He, and Y.-L. Chen, "A novel hardware/software partitioning method based on position disturbed particle swarm optimization with invasive weed optimization," *Journal of Computer Science and Technology*, vol. 32, no. 2, pp. 340–355, 2017.

[31] P. Arora, S. Varshney *et al.*, "Analysis of k-means and k-medoids algorithm for big data," *Procedia Computer Science*, vol. 78, pp. 507–512, 2016.

[32] M. Almorsy, J. Grundy, and I. Müller, "An analysis of the cloud computing security problem," *arXiv preprint arXiv:1609.01107*, 2016.

[33] "The State of Video Marketing in 2018 [Infographic]." [Online]. Available: https://www.socialmediatoday.com/news/the-state-of-video-marketing-in-2018-infographic/518339/

[34] P. Bhattacharjee and A. Majumder, "A variation-aware robust gated flip-flop for power-constrained fsm application," *Journal of Circuits, Systems and Computers*, p. 1950108, 2018.

[35] L. A. Montalvo, K. K. Parhi, and J. H. Satyanarayana, "Estimation of average energy consumption of ripple-carry adder based on average length carry chains," in *VLSI signal processing, IX*, 1996, pp. 189–198.