2024 25th International Symposium on Quality Electronic Design (ISQED) | 979-8-3503-0927-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISQED60706.2024.10528742

An FPGA-based Max-K-Cut Accelerator Exploiting Oscillator Synchronization Model

Mohammad Khairul Bashar[1]†, Zheyu Li[2]†, Vijaykrishnan Narayanan[2], Nikhil Shukla[1]*
[1]University of Virginia, VA, USA, [2]Pennsylvania State University, PA, USA
†Equal Contribution, *Email: ns6pf@virginia.edu

***Abstract*— In this work, we demonstrate a one of its kind FPGA-based compute engine that uses new computational models inspired by the synchronization dynamics of coupled oscillators to solve the general form of the computationally intractable Max-K-Cut combinatorial optimization problem (COP). Prior work on developing oscillator-inspired models for solving COPs, namely oscillator Ising machines, only directly map the MaxCut (K=2) problem. Solving other COPs (e.g., K>2) using such models entails graph decomposition and the use of auxiliary variables that effectively increase the graph size that must be solved by the hardware. This not only increases the computation time but also degrades the solution quality. In contrast, our model offers a generalized formulation that can directly solve the general form of the Max-K-Cut problem for any value of K without the need for auxiliary variables. Subsequently, by mapping the models on an FPGA platform in a way that exploits its fine-grained parallelism, we accelerate the Max-K-Cut problem (K=2, 3, and 4 shown here) on graphs with up to 10,000 nodes. When benchmarking against the state-of-the-art simulated bifurcation machine (SBM) that only uses the Ising model, our implementation offers an average 17× speedup for the Max-2-Cut and up to 390× average speedup for the Max-3-Cut and Max-4-Cut problems for similar solution quality in all cases.**

Keywords—combinatorial optimization, Ising machine, FPGA, oscillator, Max-K-Cut

I. Introduction

Combinatorial optimization problem (COP) is a subset of optimization problems that entail finding the optimal value of a function in a discrete or combinatorial domain. COPs find extensive applications in various fields ranging from VLSI design to neural network training. From a computational standpoint, many COPs remain computationally intractable, requiring exponentially increasing computational resources with increasing problem size [1]. Examples of such NP-hard COPs include the Traveling Salesman Problem, Max-K-Cut problem to name a few. The NP-hard Max-K-Cut problem, the focus of the present work, is defined as the challenge of dividing a graph into K partitions such that the weight sum of the edges crossing different subsets is maximized (Fig. 1). When K=2, the Max-K-Cut problem transforms into the archetypal MaxCut problem. While various computational models and design approaches have been investigated for solving MaxCut (K=2), it has limited direct applications. In contrast, the general Max-K-Cut problem (K>2)-the focus of the present work- finds direct practical applications in fields such as protein interaction analysis [2], wireless communication [3], and scheduling [4] among others.

The approaches to solving such COPs can be classified into two broad classes: Methods and algorithms designed to yield exact solutions, and approximate solvers. Owing to the fundamental NP-hard complexity of such COPs, the former approach typically results in exponentially increasing time-to-solution and /or memory requirements. Consequently, even COPs of small size become impractical to solve using exact solvers. The only practical alternative is to use approximate solvers such as heuristics. While such methods cannot guarantee an exact solution, they promise significant speedup. However, purely algorithmic heuristic approaches also face challenges in solving intractable COPs. The tradeoff between speedup and solution quality can be substantial. Furthermore, such heuristics are extremely sensitive to the nature of the input problems and typically need extensive parameter optimization. Consequently, there is active research interest in exploring alternate computational approaches and models to accelerate such problems [5].

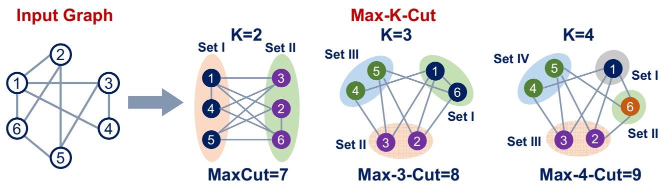


Figure 1: Illustration of the Max-K-Cut problem (for k=2, 3, and 4) for a representative 6-node graph. While prior work on physics-inspired computing has focused on accelerating MaxCut (K=2), here we develop, for the first time, an FPGA accelerator for the general case of Max-K-Cut (K≥2).

Physics inspired computational paradigms such as Ising machines, based on the Ising model, represent a promising approach to solving COPs [6], [7]. The underlying idea behind this method is that the natural energy minimization in the physical systems finds a natural analogue to the minimization of the objective function that defines a COP. Consequently, as the system evolves towards the ground state by minimizing its energy, it naturally solves the COP. As an archetypal example of such mapping, the Ising Hamiltonian given by $H = -\sum_{\mathrm{i,j,i<j}}^{N} J_{\mathrm{ij}} s_{\mathrm{i}} s_{\mathrm{j}}$ [8] can *directly* map the objective function of the MaxCut problem using the following relationship: edge weight between node i and j, $w_{\mathrm{ij}} = -J_{\mathrm{ij}}$. Here, the i$^{\mathrm{th}}$ spin s_{i} ($s_{\mathrm{i}} \in \{-1, +1\}$) maps the i$^{\mathrm{th}}$ node of the input graph. Such spin assignments to the nodes divide them into two sets that yield a MaxCut solution. Furthermore, such Ising machines, that minimize the Ising Hamiltonian, can be mapped to the dynamics of coupled electronic oscillators [8]-[10], qubits [11] etc. There have been many demonstrations that have shown the promise of such systems in solving the MaxCut problem. As elucidated in the following section, this physics inspired approach has been exploited either by developing the actual physical implementation of the system (e.g., a network of coupled oscillators [9],[10]) or by developing emulators (e.g., SBM: simulated bifurcation machine [12]) that use the computational models inspired by the physics of such systems. While the former approach which entails an application specific implementation promises larger speedups, scaling such designs are challenging. In contrast, the latter approach essentially aims to use the physics-based approach as a computational model to solve the COP and has been shown to be much more scalable as well as offer substantial speedup.

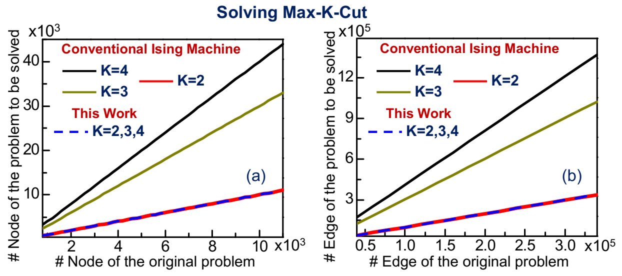


Figure 2: The variation in (a) the number of nodes and (b) the number of edges of the graph required to solve the Max-K-Cut problem, as a function of the corresponding quantities in the original input graph. It can be observed that, in the case of our approach, these quantities coincide with the original graph quantities as no conversion is required.

However, one of the long-standing limitations of this paradigm is that such physics inspired methods lack flexibility, and only COPs whose objective functions are exactly equivalent can be *directly* mapped. For example, in the case of Ising machines, only the specific case of the MaxCut problem (K=2) can be directly solved. Solving the general Max-K-Cut problem (K>2) entails transforming or decomposing the problem that is compatible with Ising model by introducing additional auxiliary variables (nodes) and edges. Consequently, the actual graph that must be mapped to the hardware is significantly larger than the input problem size [6], [8]. This not only degrades the time-to-solution and solution quality but also increases the energy consumption of the hardware (physical implementation or emulator). Fig. 2 compares the size of the actual graph that must be solved by the hardware as a function of the size of the input problem for K=2, 3, and 4 in the Max-K-Cut problem. For example, to solve the Max-3-Cut using an Ising machine, a 10,000-node graph needs to be converted to a 30,000-node graph [6], [8] that will then be mapped to the Ising machine.

Therefore, in this work, we present a novel physics-based computational model that is implemented on FPGA to solve the general Max-K-Cut problem (for any K) without incurring any additional (auxiliary) nodes i.e., the size of the input problem is exactly the same as that of the input graph. The contributions of this work can be listed as shown below:

- We implement novel computational models inspired by the synchronization dynamics of oscillators that solve the general form of the Max-K-Cut problem. Unlike the conventional Ising machine-based solvers / annealers, our approach can directly solve the Max-K-Cut without expanding the problem size. Mapping the Max-K-Cut problem to an Ising machine entails the use of additional auxiliary variables (nodes) that are not required in our approach.
- We develop an FPGA accelerator (AWS F1 instance) that exploits design techniques such as sparse matrix random access parallelization and design an efficient dataflow architecture to accelerate the oscillator synchronization-based computational models.
- By leveraging the inherent parallelism in the computational models and the FPGA implementation, we demonstrate the solutions to the Max-K-Cut problem (K=2,3,4) on graphs up to 10,000 nodes with speedups ranging from 17× - 390× over a state-of-the-art Ising machine-based accelerator, while maintaining similar solution quality.

The paper is divided into 5 sections. Prior related work is described in the following section. Subsequently, Section III delves into the proposed approach, providing comprehensive details regarding the computational model and the FPGA design. Results and performance benchmarking are discussed in section IV. Section V summarizes the key accomplishments.

II. Related Work

In the past, many approaches have leveraged the idea of the energy minimization in physical systems to solve hard COPs including simulated annealing [13], simulated bifurcation [12], synchronized oscillators [8]-[10], [14], quantum annealing [11], artificial neural networks [15], *p*-bit-based methods [16], and coherent Ising machines [17] among others. Simulated annealing (SA) was one of the early approaches to be investigated. One of the primary challenges of this approach is the sensitivity of the parameters (e.g., schedule for annealing and update) to the input problem [18]. Recently, a state-of-the-art simulated bifurcation-based algorithm was proposed to solve the MaxCut problem (SBM) [12]. There have been various FPGA- and GPU-based implementations for these physics-inspired algorithms. A few examples include: (a) Tatsumura *et al.* [19], and Yoshimura *et al.* [20], showcased FPGA-based implementations of the SBM and SA, respectively, capable of solving the MaxCut problems on graphs with up to 4096 nodes. (b) Cook *et al.* [21], demonstrated GPU-based Metropolis annealing to solve the MaxCut (i.e., only for K=2). Besides FPGA and GPU-based implementations, ASIC implementations such as an SRAM based CMOS annealing processor IC [22] and stochastic cellular automata-based annealing processor [23] have also been developed to solve the MaxCut problem.

Complementing the digital accelerators, analog demonstrations that rely on the physics of the hardware have also been developed. For example, the following works [10], [14], [24] developed coupled oscillator based Ising machines exploiting the fact that the dynamics of a network of coupled oscillators under second harmonic injection can minimize the Ising Hamiltonian. Using the same principle, other hardware platforms such as optoelectronic oscillators (Coherent Ising Machines [17]) and p-bits [16] have also been showcased.

While all these approaches bring forth their own innovations, they are inherently limited to solving binary optimization. Solving other problems (e.g., Max-K-Cut, K>2) using the above methods will involve the need for graph transformation and the use of additional auxiliary variables – a limitation that we aim to overcome in this work. Efforts on developing physics (energy minimization)-inspired approaches for solving COPs beyond the MaxCut e.g., the Max-K-Cut problem, have been very sparse, and have generally relied on traditional approximation algorithms. Our work is inspired by [25] which extends the theory of Ising machines to the more general Potts model. Therefore, this work represents a unique direct implementation of a novel physics-inspired algorithm on a hardware accelerator for solving the general form of the Max-K-Cut problem.

III. The proposed Solver

A. Computational Model

The objective function of the Max-K-Cut problem can be expressed as,

$$H = -\sum_{\mathrm{i,j,i<j}}^{N} J_{\mathrm{ij}}.\,Re\left(e^{i\mathrm{f}(\Delta\phi_{\mathrm{ij}})} s_{\mathrm{i}} s_{\mathrm{j}}^{*}\right) \tag{1}$$

Where, Re represents real part, i represents imaginary unit ($i = \sqrt{-1}$), i represents ith index, $s_{\mathrm{i}} = e^{i\phi_{\mathrm{i}}}$; $\phi_{\mathrm{i}} \in \{0, \frac{2\pi}{K}, \frac{4\pi}{K} \dots\}$. s_{i} represents the ith state variable that corresponds to an individual node in the graph. s_{i} can be considered analogous to the Ising spin except with K number of states. $J_{\mathrm{ij}} = -w_{\mathrm{ij}}$ (w_{ij}: edge weight). The function $\mathrm{f}(.)$ is defined as,

$$\mathrm{f}(\Delta\phi_{\mathrm{ij}}) = \lim_{\sigma\to 0} \sum_{k=1}^{K-1} \left(\left((2k-1)\pi - \frac{2k\pi}{K}\right).\,e^{-\left(\frac{\left(\Delta\phi_{\mathrm{ij}} - \frac{2k\pi}{K}\right)^2}{2\sigma^2}\right)} + \left(\frac{2k\pi}{K} - (2k-1)\pi\right).\,e^{-\left(\frac{\left(\Delta\phi_{\mathrm{ij}} + \frac{2k\pi}{K}\right)^2}{2\sigma^2}\right)} \right) \tag{2}$$

and essentially makes the coupling coefficient sensitive to the phase. The phase configuration that yields the minimum value of H corresponds to the solution of the Max-K-Cut problem [25]. Moreover, Eq. (1) can be minimized by a coupled oscillator system whose energy function (E) and dynamics ($^{\mathrm{d}\phi_{\mathrm{i}}}/_{\mathrm{d}t}$) can be described by:

$$E\big(\phi(t)\big) = -\frac{KC_1}{2} \sum_{\mathrm{i,j,\,j\neq i}}^{N} J_{\mathrm{ij}} \cos\left(\Delta\phi_{\mathrm{ij}} + \mathrm{f}(\Delta\phi_{\mathrm{ij}})\right) - \sum_{\mathrm{i=1}}^{N} C_S \cos(K\phi_{\mathrm{i}}(t)) \tag{3}$$

$$\frac{\mathrm{d}\phi_{\mathrm{i}}(t)}{\mathrm{d}t} = -C_1 \sum_{\mathrm{j=1,\ j\neq i}}^{N} J_{\mathrm{ij}} \sin\left(\Delta\phi_{\mathrm{ij}} + \mathrm{f}(\Delta\phi_{\mathrm{ij}})\right) - C_S \sin(K\phi_{\mathrm{i}}(t)) \tag{4}$$

C_1 represents the coupling strength among the oscillators and C_S represents the strength of the Kth harmonic signal injected into the system. The ground state (global minima) of Eq. (3) can be shown to be equivalent to the global minima of objective function for the Max-K-Cut problem (Eq. (1)). The system of equations in Eq. (4) describes the corresponding dynamics which detail how the system evolves towards the ground state. It can be observed that Eq. (4) represents a unique set of dynamics to solve the general form of the Max-K-Cut problem without requiring any additional variables.

B. Numerical Implementation

The computational model, represented by Eq. (4), is solved using a stochastic differential equation (SDE) framework developed on an FPGA platform. The SDE kernel adds noise to the dynamics. The presence of noise helps the system escape from local minima (corresponding to sub-optimal solutions) in the high dimensional phase space and help improve the solution quality while incurring minimum performance penalty. Hence, the dynamics can be written as,

$$\frac{\mathrm{d}\phi_{\mathrm{i}}(t)}{\mathrm{d}t} = \mathrm{g}(.) + \mathrm{d}w_t \tag{5}$$

Where, $\mathrm{d}w_t$ describes the Weiner process [26] that introduces stochasticity into the dynamics. g(.) is the right-hand side of Eq. (4). Additionally, we use a tanh(.) that augments the phase dynamics as used in earlier works [27], [28]. The numerical integration technique used to solve the system of equations

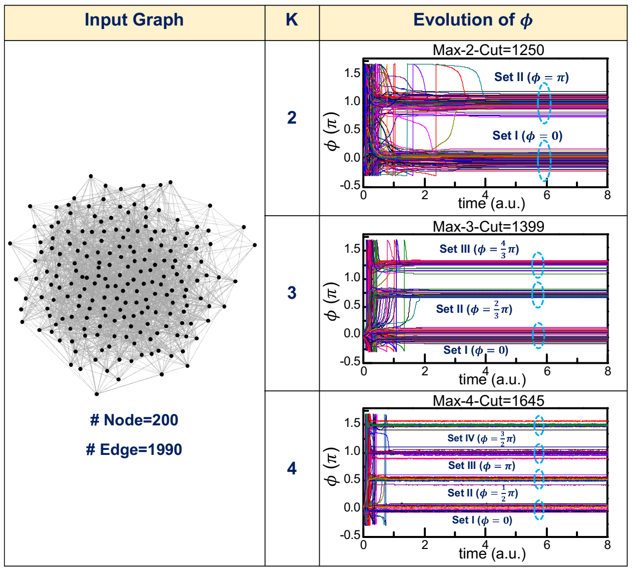


Figure 3: Illustration example showing the Max-K-Cut solution obtained using the proposed FPGA accelerator for a 200-node graph. The solutions are calculated for K=2, K=3, and K=4.

described in Eq. (5) is a simple trapezoidal integrator [29]. Solving Eq. (5) yields the time evolution of the oscillator phases (ϕ). Upon achieving a steady state, the dynamics create K number of partitions which correspond to the K vertex clusters created by the Max-K-Cut. Fig. 3 shows the Max-K-Cut (K=2, 3, and 4) solution for a representative 200 node graph obtained using the proposed platform.

C. *Architecture of the FPGA Implementation*

Since the conventional energy minimization-based computing approaches such as Ising machines are unable to *directly* solve the Max-K-Cut problem, here we design an efficient FPGA platform to accelerate the computational model for solving the Max-K-Cut problem described above. Our design strategies are governed by three main objectives: 1) exploiting a maximum level of parallelism; 2) consuming a reasonable amount of hardware resources; 3) ensuring flexibility so that it supports the general case of the Max-K-Cut for any graph of arbitrary size *without FPGA reconfiguration*. The details of the FPGA system design are described below:

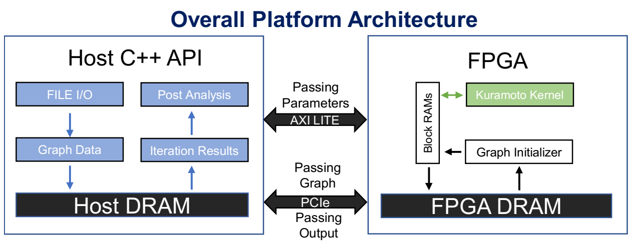


Figure 4: Block diagram depicting the architecture of the proposed FPGA-based Max-K-Cut solver.

Fig. 4 shows the high-level block diagram of the accelerator. The computational model is implemented on the FPGA using two major kernels: (a) Graph Initializer; and (b) Kuramoto Kernel. Initially, the graph data in CSR (compressed row storage) format, is stored in the host DRAM, and is subsequently transferred to the FPGA DRAM through PCIe. Additionally, a set of control registers are accessible between the host and the FPGA through the AXI-lite kernel interface which allows dynamical modification of the essentials such as the graph size, K (in Max-K-Cut), number of iterations without the need for reconfiguring the FPGA platform each time. Once the graph data is transferred to the FPGA, the Graph Initializer kernel processes the data and transfers it to the high-bandwidth on-chip Block RAMs which allow parallel access. Thus, the off-chip DRAM is only read once. After initialization, the Kuramoto Kernel (details in the following subsections) is responsible for performing the computation. Table I lists key parameters used in the design.

TABLE I. PARAMETER USED IN THE FPGA IMPLEMENTATION

Parameters	Definition	Parameter values used in the evaluation
N	Size of the graph	Input problem dependent (up to 10,000)
K (Max-K-Cut)	Number of partitions required from the Cut	Input problem dependent (2, 3, 4)
X	Size of each segment in Block RAM	80
n	Number of bits used to represent the fractional part in standard fixed-point format	19 bits (Fractional part)

D. *Graph Initializer*

Since most practical graphs have limited connectivity (within 5-10%) [30], using a full matrix representation is inefficient since both the computation and the storage space will be wasted on zero entries. Our platform adopts the standard CSR format for sparse matrix representation and re-organizes it using the Graph Initializer to enable row-wise fine-grained parallelism in FPGA. As shown in Fig. 5, the Graph Initializer reads the raw sparse matrix from the FPGA DRAM and transforms it to non-overlapping segments of size X. Each

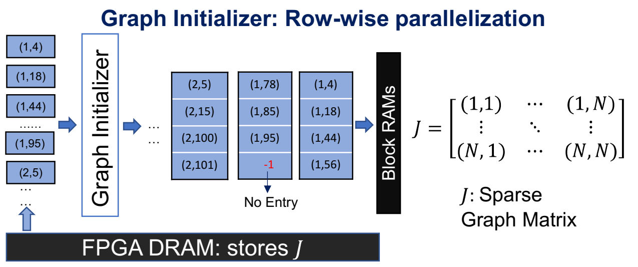


Figure 5: Role of the Graph Initializer which transforms the sparse graph matrix (J) in raw format (Column, Row) to non-overlapping Block RAM segments with size X. Empty entries are represented as -1.

segment stores a fixed number of non-zero indices in a way that all indices are from the same row of the sparse graph matrix. In case of any empty indices, a -1 value is inserted. When transferring segments to Block RAMs, each segment is fully partitioned which allows parallel access to all indices within one segment every cycle.

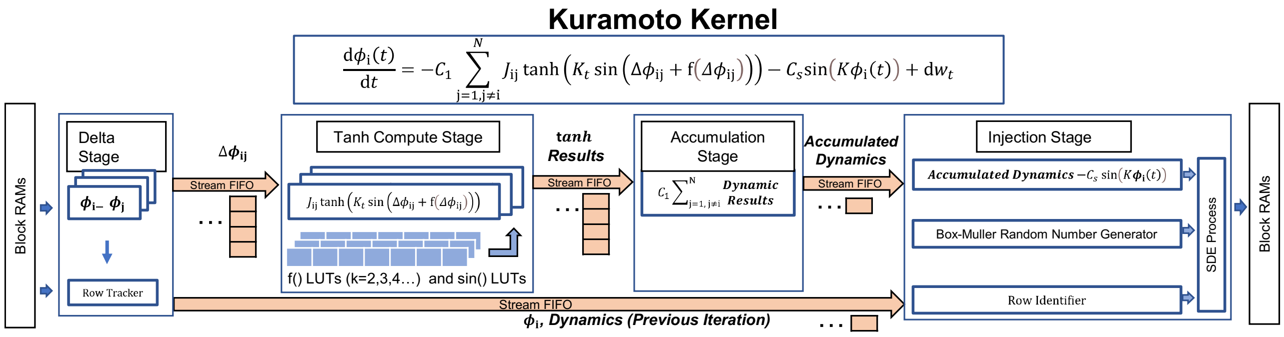

Kuramoto Kernel
$$\frac{d\phi_i(t)}{dt} = -C_1 \sum_{j=1,j\neq i}^{N} J_{ij} \tanh\left(K_t \sin\left(\Delta\phi_{ij} + f(\Delta\phi_{ij})\right)\right) - C_s \sin(K\phi_i(t)) + dw_t$$
Block RAMs
Delta Stage
$\phi_i - \phi_j$
Row Tracker
$\Delta\phi_{ij}$
Stream FIFO
Tanh Compute Stage
$J_{ij} \tanh\left(K_t \sin\left(\Delta\phi_{ij} + f(\Delta\phi_{ij})\right)\right)$
f() LUTs (k=2,3,4…) and sin() LUTs
tanh Results
Accumulation Stage
$C_1 \sum_{j=1,j\neq i}^{N}$ Dynamic Results
Accumulated Dynamics
Injection Stage
Accumulated Dynamics $-C_s \sin(K\phi_i(t))$
Box-Muller Random Number Generator
Row Identifier
SDE Process
Block RAMs
ϕ_i, Dynamics (Previous Iteration)

Figure 6: Block diagram showing the architecture of the Kuramoto Kernel. Kuramoto Kernel fetches one segment every cycle from Block RAMs. Each segment has X (=80 used here) non-zero entries. The Kuramoto Kernel then uses X non-zero entries every cycle in a fully streaming dataflow fashion.

E. Kuramoto Kernel

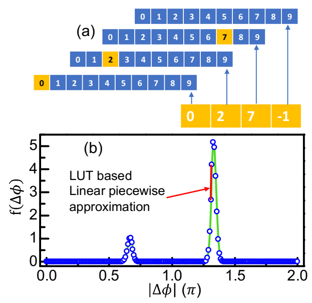
Fig. 6 shows the architecture of the Kuramoto Kernel. The kernel is designed using a streaming dataflow architecture, where inputs and outputs of each stage are connected with stream FIFOs. Thus, all stages are operating concurrently as soon as the input data is available. All Block RAMs accesses are done only on the first and the last stage, thus greatly reducing the control overhead and routing challenges. Depending on the available resources, the throughput of the kernel can be scaled by customizing the parallelism of each stage and the stream FIFO width. All stages, subfunctions within the stages, and stream FIFO are designed and implemented using Vitis HLS 2021.2.

*1) **Delta Stage:*** The Delta Stage (Fig. 6) is responsible for Block RAM reading and generating primary inputs, specifically $\Delta\phi_{ij}$, for upcoming stages. It first reads the Block RAM for the ϕ vector generated from the previous iteration and the non-zero indices stored in non-overlapping segments, as discussed before. In each cycle, one segment of non-zero indices from the Block RAM is accessed. Subsequently, X (=80 in our design) number of primary inputs $\Delta\phi_{ij}$ are generated parallelly using the subtraction units implemented in this stage. The Row Tracker unit detects the row number of the current segment and sends dynamics (i.e., g(.) in equation (5)) calculated during the prior iteration along with ϕ to the Injection Stage.

Sparse Matrix Random Access Parallelization: It can be observed from the dynamics in equation (5) that only the non-zero indices of $\Delta\phi_{ij}$ need to be computed. Assuming each segment has X parallel accessible indices and ϕ is a vector of size N (N: size of the input graph), then an X to N crossbar, and a fully partitioned ϕ array is required to support X parallel access. Obviously, this method is challenging to scale as the graph size grows. While previous efforts have used banked row buffer strategy [31], the latency cannot be guaranteed in such schemes due to bank conflict. Here we design a method where we maintain X individual copies of the ϕ vector to support X parallel random accesses. Thus, no memory partition and crossbar are required, and no bank conflict will occur in any case. As new ϕ values are generated in each iteration, we simultaneously update X copies. Details are depicted in Fig. 7a. The rationale behind utilizing this approach of maintain multiple copies of the ϕ vector is that present FPGA designs have abundant Block RAM which can be usefully exploited to achieve speed up in the memory access.

*2) **Tanh Compute Stage:*** *The Tanh Compute Stage* supports X number of inputs in each cycle to avoid any stalls in dataflow. X copies of all the subfunctions inside the Tanh Compute stage are instantiated to facilitate parallel computing.

Compute Logic Optimization: To minimize the hardware resource consumption while maintaining enough parallelism, we re-formulate the f(.) function as a piecewise linear approximation where all the entry points are pre-stored in RAM-based LUTs (Look Up Table) (Fig. 7b). Computing f(.) function for different K cuts only requires switching to different dedicated LUTs. For trigonometric functions such as sin(.), we design a customized hybrid CORDIC [32] pipeline where the first 10 bits of the result are obtained by directly looking at the pre-stored RAM-based LUT, thus, eliminating the first 10 pipeline stages. Consequently, hardware resource consumption is reduced in all categories as shown in Fig. 7c (maintaining similar accuracies). With the native implementation, the f(.)



Resource Utilization (Scaled to X=80)	**f() func (LUT)**	Native f func()
Block RAMs	0.3k (5%)	2.4k (40%)
DSP	0.3k (3%)	11k (110%)
Flip Flops	54k (2%)	910k (34%)
Logic Slices	69k (5%)	1081k (78%)

(c)

Resource Utilization (Scaled to X=80)	**Hybrid Sin()**	Native Sin()
Block RAMs	0.3k (5%)	0 (0%)
DSP	0.4k (4%)	0.4k (4%)
Flip Flops	76k (3%)	108k (5%)
Logic Slices	276k (23%)	544k (46%)

Figure 7: (a) Depiction of the sparse matrix random access parallelization technique utilized to parallelly update state variables (ϕ). (b) Piecewise linear approximation of the f(.) function used in the computational model. (c) Resource utilization breakdown in the implementation of the resource hungry f(.) function and the sine function.

function alone will require more than available (110%) DSP resources when X is scaled to 80.

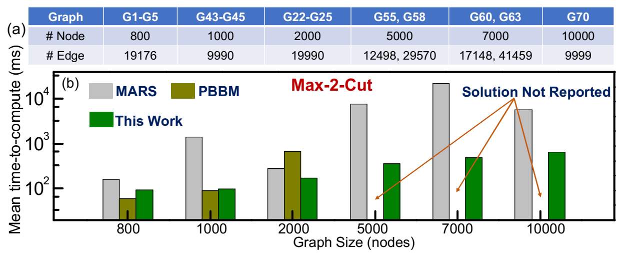
*3) **Accumulation Stage:*** The Accumulation Stage is pipelined to support the accumulation of X inputs in each cycle so that the dataflow does not stall in any case. All segments in the same row will be accumulated to produce one Accumulated Dynamic (Fig. 6).

*4) **Injection Stage:*** Depending on the number of segments initialized for each row, an Accumulated Dynamic is received by the Injection Stage every few cycles. The ϕ vector from the current iteration is then read from the stream FIFO and used to calculate the injection term ($\sin(K\phi_i(t))$). Additionally, the ϕ vector along with the accumulated dynamics from the previous iteration (read from the stream FIFOs) are used as inputs for the SDE process. The normal distributed stochastic noise required for the SDE Process is generated using a standard Box-Muller Random Generator [33].

IV. RESULTS

A. Computing Max-K-Cut

We evaluate the performance of our implementation using instances from the G-Set benchmark database. The database contains hard non-planar random graphs with a broad size range allowing us to solve graphs with sizes ranging from 800 to 10,000 nodes (specified in Fig. 8a). The oscillator dynamics are evaluated for 4000 iterations (epochs). During evaluation, we set the FPGA frequency to 100 MHz. We first compare our results (mean computation time) for the MaxCut case with two other *GPU-based implementations*, namely the MARS (Mean-field



(a)

Graph	G1-G5	G43-G45	G22-G25	G55, G58	G60, G63	G70
# Node	800	1000	2000	5000	7000	10000
# Edge	19176	9990	19990	12498, 29570	17148, 41459	9999

Figure 8: Comparison of our approach for the baseline MaxCut (K=2) with prior works. (a) Graph instances from the G-Set database used for benchmarking. (b) Comparison of the mean computation time for the GPU-based MARS algorithm and the GPU-based PBBM algorithm. Our approach exhibits solution quality comparable to that of the MARS approach. PPBM did not report average solution quality.

Annealing from a Random State) algorithm [34] and PBBM (Population Based Boltzmann Machine) algorithm [35] that have also evaluated problems from the G-Set database. Furthermore, the FPGA-based implementations for solving MaxCut demonstrated in [36] and [37] have only focused on simpler planar and toroidal graphs. Additionally, they do not address the general case of the Max-K-Cut problem for K>2 cases. Hence, We have not included these results in the benchmarking. From Fig. 8, it can be observed that our approach provides 18× and 2× mean speedup, respectively, compared to the MARS and the PBBM approach, while providing similar solution quality (>98.5 %); here, solution quality is defined as the ratio of the obtained solution to the best-known cut [38]. We note that none of the GPU, FPGA, and ASIC based annealing approaches reported *direct* implementation (i.e., without preprocessing and auxiliary variables) of the broader Max-K-Cut (K>2) problem. Current methods have to rely on transforming the Max-K-Cut problem to a binary optimization form (QUBO: quadratic unconstrained binary optimization) entailing additional nodes (axillary variables) so that it can be mapped to an Ising machine. We now compare our direct implementation (using the new models) with the Ising machine implementation for the Max-K-Cut (after the problem transformation). We use the GPU-based simulated bifurcation (Ising) machine (SBM; from Toshiba and available on AWS [39]) for this comparison. While we experimentally evaluate the Max-3-Cut and Max-4-Cut solutions on the SBM, we use the SBM-based results reported for the MaxCut (K=2) [38].

Fig. 9a presents a comparison of the cumulative computation time between our approach and the state-of-the-art GPU-based SBM approach [38] for solving the archetypal Max-2-Cut problem over the G-set graphs (Fig. 8a). While for smaller graphs, computation time from our approach is comparable with the SBM approach, it provides a 20x speedup over the SBM for graphs exceeding 5000 nodes. However, we note that this comes at the cost of a small degradation in solution quality. The average solution produced by our approach is within 98.5% of the best-known solutions for the Max-2-Cut instances tested whereas the average SBM solutions are within 99.9% of the best-known solutions.

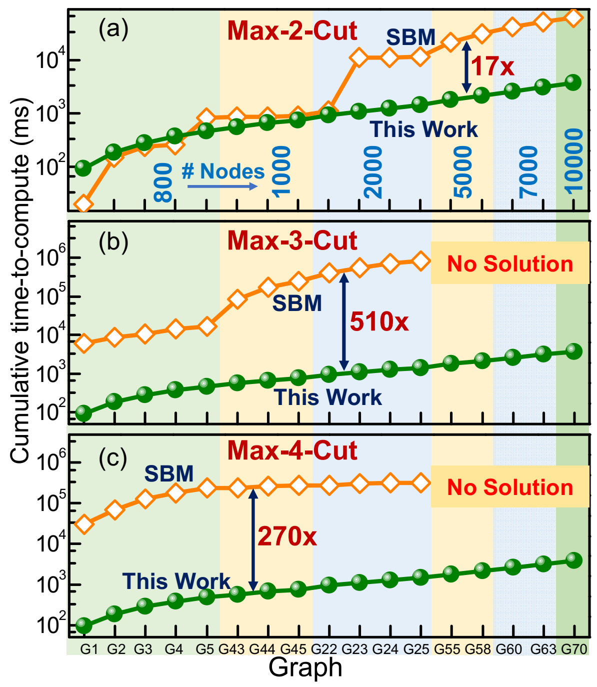


Figure 9: Cumulative time-to-compute for solving the (a) Max-2-Cut, (b) Max-3-Cut, (c) Max-4-Cut on the G-set graph instances and their comparison with state-of-the art SBM approach. In all cases, we maintain a high mean accuracy exceeding >95% of the best-known solution.

Next, we evaluate the computation time for the Max3-Cut and Max-4-Cut problems. The comparison of computation times for Max-3-Cut and Max-4-Cut are presented in Fig. 9b and Fig. 9c, respectively. It can be observed that our method achieves a remarkable ~510x and ~270x mean speedup compared to the

SBM in solving the Max-3-Cut and Max-4-Cut problems, respectively (~390x average speedup for the Max-3-Cut and Max-4-Cut combinedly), while maintaining similar solution quality. In our approach, the time-to-compute primarily consists of the Kuramoto Kernel computation time, which accounts for most of the overall time-to-solution. We calculate the K-Cut values in the host system using the Kernel results obtained from the FPGA. Similarly, the SBM computation time consists of only the Ising problem computing time; the pre-processing (conversion to Ising problem) and post-processing (finding K-Cut solution from Ising solution) are performed in the host and their computation time is not added in the overall computation time presented here. Also, we are unable to solve the Max-3-Cut and Max-4-Cut for some of the larger graphs (here, 5000 to 10000 node graphs) using SBM since after converting them to Ising problems, their size exceeds 10,000 nodes that is the maximum limit for the SBM available on AWS.

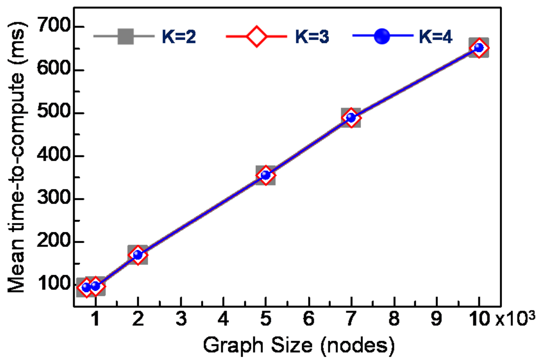


Figure 10: Mean time-to-compute as a function of graph size.

We also analyze how time-to-compute scales with problem size and the value of K. Fig. 10 presents the average time-to-compute as a function of graph size. It can be observed that as the number of nodes increases, the computation time scales linearly, which can be attributed to the parallel access of each graph rows by the FPGA platform (i.e., row-wise parallelization). Most importantly, the computation time does not change with the value of K, unlike prior designs and implementations. It can be attributed to the computational model that facilitates the solution of the Max-K-Cut without increasing the problem size. Furthermore, if K value changes, the only operational change that the FPGA system needs to make is switching the LUT for the f(.) function. Hence, the computation time remains the same regardless of the value of K for a particular graph.

B. *Resource Utilization and Energy Benchmark*

TABLE II. OVERALL RESOURCE UTILIZATION

Resource	**Xilinx VU9P FPGA**	Resource	**Xilinx VU9P FPGA**
Kernel Frequency	100 MHz	DSP	2.7K (38%)
Block RAMs	1.3K (30%)	Flip Flops	470K (19%)
Ultra RAMs	0.5K (50%)	Logic Slices	518K (43%)

Table II presents a detailed overview of the resource utilization in the FPGA implementation. Table III compares the energy consumption between our FPGA implementation (collected using the AWS 'FPGA image describe' command) and the SBM implementation used for benchmarking in this work. While the SBM energy data for the Max-2-Cut has been reported, the energy numbers for K=3, 4 are projected since the SBM energy data (on AWS) is unavailable. It can be observed that our approach not only offers better computational capability but also enables over 8x improvement in the energy consumption / iteration.

TABLE III. COMPARISON WITH OTHER APPROACHES

Approach	Platform	Problem Solved	Energy (mJ) / Iteration (2000 node)		
			Max-2-Cut	**Max-3-Cut**	**Max-4-Cut**
SBM [19]	GPU	MaxCut	3.44 (reported)	10.32 (Projected)	13.76 (Projected)
This Work	FPGA	Max-K-Cut & MaxCut	0.4225 (measured)	0.4225 (measured)	0.4225 (measured)

V. CONCLUSION

In summary, we have demonstrated an FPGA accelerator with the unique capability of solving the general class of Max-K-Cut problems without reconfiguring the FPGA platform. To accomplish this, our approach exploits novel oscillator-synchronization-inspired computational models in combination with the inherent fine-grained parallelism of the FPGA. We showcase the ability of the platform to accelerate graphs up to 10,000 nodes with up to 390x speedup and orders of magnitude reduction in energy consumption. Among the physics-based approaches, our method uniquely stands out in its capability, flexibility, and efficiency, and thus, advances the state-of-art in solving computationally hard COPs.

ACKNOWLEDGEMENT

This work was supported by the National Science Foundation under Grant 2132918.

REFERENCES

[1] C. S. Calude, "Unconventional computing: A brief subjective history," Springer International Publishing, 2017.
[2] S. Choudhury et al., "An Approximation Algorithm for Max k-Uncut with Capacity Constraints," in International Joint Conference on Computational Sciences and Optimization, pp. 934-938, 2009.
[3] J. Fairbrother et al., "A two-level graph partitioning problem arising in mobile wireless communications," Computational Optimization and Applications, vol. 69, pp. 653-676, 2018.
[4] J. E. Mitchell, "Realignment in the national football league: Did they do it right?," Naval Research Logistics (NRL), vol. 50, no. 7, pp. 683-701, 2003.
[5] A. Chen et al., "A survey on architecture advances enabled by emerging beyond-CMOS technologies," IEEE Design & Test, vol. 36, no. 3, pp. 46-68, 2019.
[6] A. Lucas, "Ising formulations of many NP problems," Frontiers in physics vol. 2, p. 5, 2014.
[7] N. Mohseni et al., "Ising machines as hardware solvers of combinatorial optimization problems," Nature Reviews Physics vol. 4, pp. 363-379, 2022.

[8] T. Wang, and J. Roychowdhury, "OIM: Oscillator-based Ising machines for solving combinatorial optimisation problems." UCNC 2019.

[9] S. Dutta et al., "Experimental demonstration of phase transition nano-oscillator based Ising machine," in 2019 IEDM, pp. 37-8. IEEE, 2019.

[10] A. Mallick et al., "Overcoming the accuracy vs. performance trade-off in oscillator ising machines," in 2021 IEDM, pp. 40-2. IEEE, 2021.

[11] T. Albas et al., "Adiabatic quantum computation," RMP vol. 90, p. 015002, 2018.

[12] H. Goto et al., "Combinatorial optimization by simulating adiabatic bifurcations in nonlinear Hamiltonian systems," Science advances vol. 5, p. eaav2372, 2019.

[13] D. Bertsimas et al., "Simulated annealing," Statistical science vol. 8, pp. 10-15, 1993.

[14] M. K. Bashar et al., "Experimental demonstration of a reconfigurable coupled oscillator platform to solve the Max-cut problem," IEEE JXCDC vol. 6, pp. 116-121, 2020.

[15] A. Fischer, and C. Igel, "An introduction to restricted Boltzmann machines," In 17th Iberoamerican Congress, CIARP 2012, pp. 14-36, 2012.

[16] N. A. Aadit, et al., "Massively parallel probabilistic computing with sparse Ising machines." Nature Electronics vol. 5, pp. 460-468, 2022.

[17] T. Honjo et al., "100,000-spin coherent Ising machine," Science advances vol. 7, p. eabh0952, 2021.

[18] Computational issues of simulated annealing, retrieved on May 21, 2023. https://www.pks.mpg.de/tisean/TISEAN_2.1/docs/surropaper/node18.html

[19] K. Tatsumura et al., "FPGA-based simulated bifurcation machine," in 2019 FPL, 59-66. IEEE, 2019.

[20] C. Yoshimura et al., “Implementation and evaluation of FPGA-based annealing processor for Ising model by use of resource sharing,” Int. J. Net. and Com., vol. 7, pp.154-172, 2017.

[21] C. Cook et al., “GPU-based ising computing for solving max-cut combinatorial optimization problems,” Integration, vol. 69, pp.335-344, 2019.

[22] T. Takemoto, et al., "2.6 A 2× 30k-spin multichip scalable annealing processor based on a processing-in-memory approach for solving large-scale combinatorial optimization problems," ISSSCC, 2019.

[23] K. Yamamoto et al., "7.3 STATICA: A 512-spin 0.25 M-weight full-digital annealing processor with a near-memory all-spin-updates-at-once architecture for combinatorial optimization with complete spin-spin interactions." ISSCC, 2020.

[24] H. Lo et al., "An Ising solver chip based on coupled ring oscillators with a 48-node all-to-all connected array architecture," Nature Electronics, pp. 1-8, 2023.

[25] A. Mallick et al., “Computational Models Based on Synchronized Oscillators for Solving Combinatorial Optimization Problems,” Physical Review Applied, vol. 17, p.064064, 2022.

[26] T. Szabados, "An elementary introduction to the Wiener process and stochastic integrals." arXiv preprint arXiv:1008.1510, 2010.

[27] T. Wang, and J. Roychowdhury, "Oscillator-based Ising machine," arXiv preprint arXiv:1709.08102, 2017.

[28] M. K. Bashar et al., "Dynamical System-based Computational Models for Solving Combinatorial Optimization on Hypergraphs," IEEE JxCDC vol. 9, pp. 21-28, 2023.

[29] Trapezoidal rule (differential equations), Retrieved at May, 10, 2023. At: https://en.wikipedia.org/wiki/Trapezoidal_rule_(differential_equations)

[30] G. Melancon, "Just how dense are dense graphs in the real world? A methodological note," in Proceedings of the 2006 AVI workshop on BEyond time and errors: novel evaluation methods for information visualization, pp. 1-7. 2006.

[31] J. Fowers et al., "A High Memory Bandwidth FPGA Accelerator for Sparse Matrix-Vector Multiplication," 2014 FCCM, pp. 36-43, 2014.

[32] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," ACM/SIGDA sixth international symposium on FPGA, pp. 191-200. 1998.

[33] D-U. Lee et al., "A hardware Gaussian noise generator using the Box-Muller method and its error analysis," IEEE Trans. Comput. vol. 55, pp. 659-671, 2006.

[34] A. Yavorsky et al., "Highly parallel algorithm for the Ising ground state searching problem," arXiv preprint arXiv:1907.05124, 2019.

[35] S. F. Mousavi, “A GPU-Accelerated Population-Based Boltzmann Machine for Solving Combinatorial Optimization Problems,” Diss. University of Toronto (Canada), 2020.

[36] Y. Zou, and M. Lin, "Massively simulating adiabatic bifurcations with FPGA to solve combinatorial optimization," in Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 65-75, 2020.

[37] H. Gyoten, M. Hiromoto, and T. Sato, “Area efficient annealing processor for ising model without random number generator,” IEICE TRANSACTIONS on Information and Systems 101, pp. 314-323, 2018.

[38] Benchmarking the MAX-CUT problem on the Simulated Bifurcation Machine, retrieved on May 10, 2023. At: https://medium.com/toshiba-sbm/benchmarking-the-max-cut-problem-on-the-simulated-bifurcation-machine-e26e1127c0b0

[39] Simulated Bifurcation Machine. At: https://aws.amazon.com/marketplace/pp/prodview-f3hbaz4q3y32y?ref=_ptnr_mdm