Research on Binary Neural Network Acceleration Methods Based on ARM+FPGA Platform (Post-print)

Authors: Sun Xiaohui, Song Qingzeng, Jin Guanghao, Wenchao Jiang

Date: 2019-01-03T00:00:00+00:00

Abstract

Currently, existing convolutional neural networks, due to their complex architectures and dependence on large-scale datasets, are unable to satisfy the computational performance requirements and energy consumption constraints of certain practical applications or computing platforms. Targeting these applications or platforms, this work investigates binarization algorithms based on ARM+FPGA platforms and designs a binary neural network that reduces data storage demands and decreases computational complexity. Within the ARM+FPGA platform implementation, the conversion of convolutional multiply-accumulate operations into XNOR logical operations and popcount operations improves overall computational efficiency and reduces energy and resource consumption. Furthermore, according to the data storage characteristics of binary neural networks, a novel row-processing improvement algorithm is proposed, which enhances network throughput. In conclusion, this implementation demonstrates superiority over existing FPGA neural network acceleration methods in terms of GOPS, energy efficiency, and resource efficiency.

Full Text

Research on Binary Neural Network Acceleration Method Based on ARM+FPGA Platform

Sun Xiaohui[1], Song Qingzeng[1], Jin Guanghao[1] , Jiang Wenchao[2]
[1]School of Computer Science & Software Engineering, Tianjin Polytechnic University, Tianjin 300387, China; [2]School of Computers, Guangdong University of Technology, Guangzhou 510006, China

Abstract: Existing convolutional neural networks face difficulties in meeting the computational performance requirements and energy consumption constraints of certain practical applications or computing platforms due to their

complex structures and reliance on large datasets. Targeting these applications and platforms, this paper investigates binarization algorithms on ARM+FPGA platforms and designs a binary neural network that reduces data storage requirements and computational complexity. During implementation on the ARM+FPGA platform, convolutional multiply-accumulate operations are converted into XNOR logic operations and popcount operations, improving overall computational efficiency while reducing energy and resource consumption. Additionally, based on the characteristics of data storage in binary neural networks, a novel row-processing optimization algorithm is proposed to enhance network throughput. Overall, this implementation outperforms existing FPGA-based neural network acceleration methods in terms of GOPS, energy efficiency, and resource efficiency.

Key words: binary neural network; FPGA; XNOR; row-processing algorithm

0 Introduction

Convolutional neural networks (CNNs) represent one of the most important deep learning algorithms today, achieving excellent results in computer vision [1], machine translation [2], speech recognition [3], face detection [4], and other domains. As CNNs and their improved variants continue to evolve, they impose increasingly high demands on storage capacity and computational power, limiting their deployment on embedded platforms with constrained power and performance budgets. Consequently, researchers have proposed various model compression algorithms and techniques [5–7] that reduce CNN computational intensity and parameter capacity at the cost of tolerable accuracy degradation. Among these, one of the most significant approaches replaces high-precision floating-point numbers with low-precision fixed-point numbers—for instance, using 8-bit fixed-point instead of 32-bit floating-point reduces storage requirements to one-quarter of the original. The extreme case compresses 32-bit floating-point numbers into binary fixed-point values (1 bit), known as binary neural networks (BNN) [6]. Compared to traditional CNNs, BNNs not only dramatically reduce memory capacity requirements but also alter traditional convolution computation patterns, lowering computational complexity.

Binary neural networks are particularly suitable for implementation on ARM+FPGA architectures. After binarizing both parameters and feature maps, multiply-accumulate operations can be replaced with simpler XOR logic and popcount operations, significantly reducing FPGA resource consumption. For BNNs, specialized pipeline structures can be designed on FPGAs to further accelerate computation. Meanwhile, portions of the network unsuitable for FPGA acceleration can execute on the ARM processor, ensuring implementation flexibility. This paper primarily investigates how BNNs can be deployed on ARM+FPGA hardware platforms using the Cifar-100 dataset, designing and implementing a BNN accelerator. The accelerator employs HLS [8] design

methodology for greater flexibility and shorter development cycles. Testing was conducted across different development boards, with comparisons made against other platforms in terms of speed, accuracy, and power consumption. Experimental results demonstrate that the proposed accelerator offers substantial advantages.

1 Related Work

In FPGA-based CNN acceleration optimization, convolution operations can be transformed into more efficient matrix multiplication operations, with primary conversion algorithms including GEMM transformation [9] and Winograd transformation [10]. Additionally, fast Fourier transform [11] can optimize convolutions by converting time-domain operations to frequency-domain multiplication. Beyond computational optimization, data transmission paths can be optimized through array contraction [12], SIMD [13], loop optimization [14, 15], and design space exploration [16] to reduce inter-data transmission latency.

In addition to computational transformation and data path optimization, CNN acceleration can also be achieved through weight pruning [17] to increase model sparsity and reduce operation counts. Convolutional kernels can also employ low-rank approximation [18] to reduce the number of multiplications during inference, achieving optimization by decreasing overall computational workload.

Beyond the aforementioned optimization methods, FPGA implementations can utilize approximate computing techniques—including fixed-point arithmetic [19], dynamic fixed-point arithmetic [20], binary quantization [5, 21, 22], pseudo-binary quantization [23, 24], and stochastic computing [25]—to reduce data precision and accelerate network operations. These methods not only lower computational complexity but also fully leverage the potential of FPGA fixed-point computation.

2.1 Network Structure

A binary neural network is a model compression approach for CNNs that binarizes both weight parameters and feature maps. Initial work focused on weight binarization [5] to reduce parameter storage, while subsequent research [6] further compressed feature maps, reducing both storage requirements and computational complexity while providing solutions for backpropagation in BNN training. Binarization in BNNs maps weights and activations to +1 and -1. Reference [5] proposed two algorithms: stochastic and deterministic binarization. Although stochastic binarization better reflects real-world scenarios, it requires generating random bits during quantization, which is unfavorable for hardware implementation. Therefore, this paper adopts the following deterministic binarization method:

$$b = \text{sign}(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

where b represents the binarized data (weights/feature maps) and x is the original value. To reduce distribution information loss during network transmission caused by binarization operations, a new hierarchical structure called batch normalization (BN) [26] is added to the network architecture:

$$y = \gamma \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

where x and y represent the input and output data, respectively; μ and σ^2 are the mean and variance values of each feature map; γ and β are parameters obtained through network training; and ϵ is a small value to ensure the denominator is non-zero during computation. The batch normalization layer accelerates network training while ensuring the stability of original data distribution information. During network inference, all parameters are fixed, so implementation only requires applying Equation (2) to each pixel of the input feature maps, with each feature map having uniquely corresponding batch normalization parameters.

The structural definitions of BNN and CNN are compared in Figure 1. The CNN layer sequence consists of convolution, bias, pooling, and activation functions. The BNN removes the bias layer, applies pooling directly after convolution, and adds a BN layer after pooling to reduce feature map distribution information loss. Except for the first layer' s input data, all layer inputs and outputs after the binary activation function are binary.

2.2 Network Model

The binary neural network model designed in this paper includes six convolutional layers and three fully connected layers. Each convolutional and fully connected layer is followed by a BN layer and activation layer, while pooling layers are added after every two convolutional layers (i.e., after the 2nd, 4th, and 6th convolutional layers). Convolutional kernels are 3×3 in size, and pooling layers use 2×2 max sampling with a stride of 2. The first convolutional layer receives original image data as floating-point type, which differs from other layers, while weight parameters for all layers are binary. Detailed parameter settings for each layer are shown in Table 1, where Conv denotes convolutional layers and FC (full connect) denotes fully connected layers. Table 1 specifies the number of input and output feature maps for each layer and calculates the memory occupied by outputs and weights, with each data point stored using 1 bit. BN layer parameters obtained after training are floating-point data that produce intermediate floating-point values after processing convolved data, which are finally converted to binary output through the activation function.

2.3 Model Training

This paper implements the network model defined in Table 1 using the Theano and Lasagne deep learning frameworks [6]. The Cifar-100 dataset [27] is used for training and validation, comprising 60,000 32×32 RGB images including real-world objects such as birds, cats, airplanes, cars, and frogs, with pixel values represented as integers from 0 to 255. These images are divided into 50,000 training samples and 10,000 test samples, with each set containing 100 categories further subdivided into 20 super-classes. Each image carries one fine label and one coarse label.

The binary neural network model employs exponentially decaying learning rates, uses the ADAM method to minimize hinge loss, sets batch size to 50, and utilizes batch normalization to accelerate training. The last 5,000 samples from the training set are used as validation data. Training runs for 500 epochs, with final recognition results shown in Table 2 . The total weight size in the final network is 13.37 MB, including convolutional layer weights.

3.1 Hardware-Software Partitioning

Compared with traditional FPGA design patterns, ARM+FPGA architecture not only shortens development cycles but also allows consideration of performance, integration, and flexibility during design, reducing cost and power consumption while enhancing features and performance.

When designing the neural network forward inference process for ARM+FPGA architecture, the primary computational stages are convolution and fully connected operations. To enable each platform component to achieve optimal performance, the hardware-software partitioning implements the main network operations within the FPGA, while the ARM processor preloads and allocates input images and layer parameters and controls the FPGA initialization phase. The system hardware-software partitioning is illustrated in Figure 2 [Figure 2: see original paper]. The hardware logic portion primarily consists of compute units, on-chip memory, DMA, and a finite state machine controller, while the system processing portion mainly includes the CPU and memory controller.

In the hardware logic portion, DMA primarily preloads weight data into on-chip memory during computation and writes final prediction results back to off-chip memory. On-chip memory mainly stores result data after compute unit operations and saves data read by DMA. Within compute units, optimization measures accelerate network operations during data-weight computation and output final network predictions.

The system processing portion first reads input feature maps and layer parameter data stored in off-chip memory into memory, allocates parameters according to each layer' s hyperparameters, then transfers the allocated data and control information to FPGA internal memory. As shown in Table 1, feature maps and weights occupy different memory proportions. Storing these parameters alone

within the FPGA would consume significant memory resources. To maximize memory utilization while ensuring optimal system performance, improvements were made to the storage of intermediate data generated during FPGA computation, with detailed descriptions provided in Section 4.2. Using this approach, the system processing portion can sequentially transfer all or part of the feature map and parameter data to FPGA on-chip memory according to network layer execution order, ensuring data transfer efficiency.

3.2 Hardware Design

Based on the hardware-software partitioning results and considering different data types and computational workloads, the compute unit is divided into three main modules to process convolution and fully connected operations: a fixed-point convolution module, a binary convolution module, and a fully connected module. Each module processes corresponding feature map data according to the network layer structure and generates final predictions.

The fixed-point convolution module primarily processes the original image data input to the first layer. Within this module, a line buffer receives the input 3-channel 32×32 images and converts corresponding pixel values into 20-bit fixed-point data. Since the input weights are binary, sign flipping replaces multiplication during computation. This method employs fully parallel processing of the three-channel data, adding input data to three buffers where the convolution kernel slides each cycle, computing a 3×3×3 convolution whose results pass through batch normalization and binarization to produce one output bit per cycle. As the first convolution layer occupies minimal runtime, excessive hardware resources are not allocated for its acceleration.

The binary convolution module constitutes the most important component of the compute unit, processing the remaining binary convolutional layers and occupying the majority of system runtime. Therefore, this module must maintain high throughput and efficient resource utilization when handling input feature maps of different sizes. To perform convolution operations more effectively within the module, multiple rows of input data must be buffered for simultaneous access. However, standard line buffers can only store single-row data per cycle and handle fixed buffer widths, causing buffer underutilization and throughput loss when input data width is smaller than buffer width. To fully utilize hardware resources and resolve line buffering issues, this paper designs a variable-width line buffer, as shown in Figure 3 [Figure 3: see original paper]. Compared to conventional buffers, the variable-width line buffer solves hardware resource underutilization problems while outputting and caching one new feature map row per cycle. During binary convolution computation, input feature maps are first read from on-chip memory and reorganized according to feature map width before loading into the line buffer. The convolution module generates operation results by sliding across the line buffer and outputs their accumulated sum to the buffer as input for the next convolution.

The fully connected module performs final fully connected calculations and returns prediction results. During execution, it bitwise XNORs feature map data with weight data through loop unrolling, then uses popcount to sum the result bits. Similar to the binary convolution module' s intermediate data processing, it accumulates convolution results in a buffer and applies binarization after processing all inputs to output final results.

Regarding on-chip memory in the hardware logic portion, since output feature maps between adjacent upper and lower layers must be stored simultaneously, this design implements two 128 KB buffers based on the maximum parameter quantity per layer from Table 1 to store intermediate computation results, using round-robin access to fully utilize memory resources and reduce off-chip data transfers.

3.3 Software-Hardware Interface

This paper uses Xilinx SDSoC as the primary design tool for the BNN application. In the BNN accelerator' s hardware-software partitioning, the software portion mainly controls overall program execution and reads data from off-chip memory into memory, while the hardware portion performs massive parallel computation tasks. Since weights and feature maps are stored and accessed differently in the hardware portion, a compromise interface implementation is adopted.

First, to reduce frequent data access between software and hardware during computation, the system processing portion transfers data to hardware memory through value passing rather than address passing. Although this consumes some memory space, it saves interaction time overhead.

To ensure efficient data transfer, weight data must be stored in contiguous physical memory space and received via FIFO streams. This data reception method better matches the characteristic of sequentially reading weight data layer by layer. After determining the corresponding interface data transfer mode, SDSoC generates component units with ends corresponding to the PS and accelerator, where the intermediate controller is the configured data mover.

Finally, SDSoC synthesizes the C/C++ code portions marked for hardware implementation in the software program into RTL hardware implementation, generating PS-PL memory data transfer modules and required DMA modules through specified interface directives. The program ensures that generated intermediate data is stored in on-chip memory and guarantees network throughput by continuously arranging neural network data and weights, with off-chip data transfers only occurring during the first and last calls to compute units and weight loading.

Experimental Setup

Experiments evaluate the proposed design on ZedBoard and ZCU102. ZedBoard represents a low-cost Xilinx Zynq-7000 SoC containing a 7Z020 FPGA and dual-core ARM Cortex-A9 embedded processor, while ZCU102 serves as a high-end Zynq UltraScale+ MPSoC containing a ZU9EG FPGA and embedded processors including quad-core ARM Cortex-A53, dual-core ARM Cortex-R5, and a graphics processing unit.

This paper uses Xilinx SDSoC 2017.4 as the primary design tool, employing Vivado HLS and Vivado to compile C/C++ code into FPGA executable format. The CPU in the system architecture shown in Figure 2 corresponds to the Cortex-A9 embedded processor on ZedBoard and the Cortex-A53 processor on ZCU102. Compute units in the hardware logic are primarily implemented using FPGA internal LUTs, with each compute unit instantiating different configurable logic blocks and input-output units, while on-chip memory corresponds to buffer storage resources such as BRAM, FIFO, and RAM.

During implementation, the three convolutional computation functions in the compute unit are encapsulated separately. Corresponding to the overall network inference process, appropriate functions are called based on each layer' s operational relationships. Data transfer between functions occurs on on-chip memory storing each layer' s feature map results. After synthesizing the overall network operation logic through SDSoC, the corresponding hardware logic structure on FPGA is generated.

The implemented design is also compared with two server-grade computing platforms and one embedded platform: an Intel Xeon E5-2640 multi-core processor (CPU), an NVIDIA Tesla K80 (GPU), and an embedded NVIDIA Jetson TX2 (mGPU). CPU and GPU baselines are adjusted from code provided in references, calling respective optimized libraries during computation.

Device power during BNN operation is obtained through power monitors. Idle power on ZedBoard and ZCU102 is 4.4 W and 23.4 W, respectively, with maximum operating power of 4.7 W and 23.6 W, respectively, demonstrating very low dynamic power consumption during FPGA operation. Throughput (giga-operations-per-second, GOPS) counts total addition and multiplication operations; in BNN, each binary XOR, flip, and addition is counted as one operation.

4.1 Accuracy Analysis

A complex factor in binary neural network models is the interaction between binarized data participating in convolution operations and edge padding. Binarized neural networks quantize each activation value to -1 or +1, but each input feature map is zero-padded at the edges, meaning convolutions can encounter up to three values: -1, 0, and +1. The Ternary-NN [24] network addresses this situation by dividing activation values into three parts, eliminating edge padding effects compared to binary data. However, BNNs may require two data bits to

store operators for these three values. This paper uses +1 padding to retrain the network, eliminating zero points and establishing a truly binary CNN. This +1 padded BNN achieves 58.74% accuracy during training and 57.24% accuracy in C/C++ FPGA implementation, only slightly worse than the original. For FPGA implementation, +1 padding better fits the network design, but considering hardware resource savings, zero padding is still adopted. Table 2 compares network model test results on the same Cifar-100 test dataset.

4.2 Performance Analysis

This paper' s experiments compare the accelerator' s overall performance with various baselines listed in Table 3 . Since raw throughput is largely limited by device size, power consumption and throughput per watt are also compared. Compared to mGPU, the two FPGA platforms achieve runtime performance improvements of 8× and 29.5×, respectively, with per-watt throughput 14.1× and 9.4× higher than mGPU. Compared to x86 processors, they achieve speedups of 2× and 6.7×, respectively. Compared to GPU, the two designs differ in performance by 18.6× and 5.5×, respectively, but as expected, their power consumption is much lower and per-watt throughput significantly higher. Conv1 represents the first convolutional layer, Conv2-5 represent binary convolutional layers, and FC1-3 are the final fully connected layers. The last row shows each platform' s per-watt efficiency.

4.3 Power Analysis

Due to the relative novelty of BNNs, the Cifar-100 dataset is selected to design corresponding accelerated networks, measuring throughput per resource and throughput per watt as comparison baselines.

Table 4 compares this implementation with state-of-the-art FPGA accelerator implementations found in literature, with corresponding numbers retrieved from the respective papers. Two comparison platforms use the same ZedBoard device, while another comparison uses a larger-capacity FPGA. This paper' s BNN accelerator surpasses the current best FPGA accelerators in throughput while being more resource- and energy-efficient. The final results demonstrate that BNNs are more suitable for FPGA implementation than CNNs, as they can more effectively utilize hardware resources.

5 Conclusion

This paper designs an ARM+FPGA-based binary neural network accelerator that completely implements the forward propagation process for the Cifar-100 dataset. The BNN network' s reduced storage requirements and binary operational characteristics make it highly suitable for FPGA architecture organization. By designing a new accelerator architecture, the accelerator achieves superior area throughput and per-watt throughput compared to existing full-precision networks, significantly improving computational efficiency. Recent

low-precision CNN research continues to progress, achieving near state-of-the-art recognition results on the ImageNet dataset [7, 24]. Future implementation of low-precision CNNs requires further exploration of model compression algorithms and the design of larger, more complex accelerators to meet future demands for massive computation and real-time performance.

References

[1] Farabet C, Poulet C, Han J, et al. CNP: an FPGA-based processor for convolutional networks [C]// Proc of International Conference on Field Programmable Logic and Applications. Piscataway, NJ: IEEE Press, 2009: 32-37.

[2] Devlin J, Zbib R, Huang Zhongqiang, et al. Fast and robust neural network joint models for statistical machine translation [C]// Proc of ACL Conference. 2014: 1370-1380.

[3] Geoffrey H, Li Deng, Dong Yu, et al. Deep neural networks for acoustic modeling in speech recognition [J]. IEEE Signal Processing Magazine, 2012, 29 (6): 82-97.

[4] Garcia C, Delakis M. Convolutional face finder: a neural architecture for fast and robust face detection [J]. IEEE Trans on Pattern Analysis and Machine Intelligence, 2004, 26 (11): 1408-1423.

[5] Courbariaux M, Bengio Y, David J P. BinaryConnect: training deep neural networks with binary weights during propagations [C]// Proc of the 28th International Conference on Neural Information Processing Systems. Cambridge, MA: MIT Press, 2015: 3123-3131.

[6] Courbariaux M, Bengio Y. Binarized neural networks: training deep neural networks with weights and activations constrained to +1 or -1 [J]. arXiv: 1602.02830v1, 2016.

[7] Hubara I, Courbariaux M, Bengio Y, et al. Quantized neural networks: training neural networks with low precision weights and activations [J]. arXiv preprint arXiv: 1609.07061, 2016.

[8] Cong J, Bin Liu, Neuendorffer S, et al. High-level synthesis for fpgas: from prototyping to deployment [J]. IEEE Trans on Computer Aided Design of Integrated Circuits and Systems, 2011, 30 (4): 473-491.

[9] Bottleson J, Sungye K, Andrews J, et al. ClCaffe: OpenCL accelerated caffe for convolutional neural networks [C]// Proc of IEEE International Parallel and Distributed Processing Symposium Workshops. 2016.

[10] Andrew L, Gray S. Fast algorithms for convolutional neural networks [J]. arXiv e-print, arXiv: 1509.09308.

[11] Jong H K, Mudassar B, Taesik N, et al. Design of an energy-efficient accelerator for training of convolutional neural networks using frequency-domain computation [C]// Proc of the 54th Annual Conference on Design Automation. 2017.

[12] Chakradhar S, Sankaradas M, Jakkula V, et al. A dynamically configurable coprocessor for convolutional neural networks [J]. ACM SIGARCH Computer Architecture News, 2010, 38 (3): 247-257.

[13] Li Huimin, Fan Xitian, Jiao Li, et al. A high performance FPGA-based accelerator for large-scale convolutional neural networks [C]// Proc of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. 2016: 26-35.

[14] Wei Xuechao, Yu C H, Zhang Peng, et al. Auto-mated systolic array architecture synthesis for high throughput CNN inference on FPGAs [C]// Proc of Annual Conference on Design Automation. 2017: 1-6.

[15] Derrien S, Rajopadhye S. Loop tiling for reconfigurable accelerators [C]// Proc of International Conference on Field Programmable Logic and Applications. 2001: 398-408.

[16] Williams S, Waterman A, Patterson D. Roofline: an insightful visual performance model for multicore architectures [J]. Communications of the ACM, 2009, 52 (4): 65-76.

[17] Han S, Pool J, Tran J, et al. Learning both weights and connections for efficient neural network [C]// Advances in Neural Information Processing Systems. 2015: 1135-1143.

[18] Sironi A, Tekin B, Rigamonti R, et al. Learning separable filters [J]. IEEE Trans on Pattern Analysis and Machine Intelligence, 2015, 37 (1): 94-106.

[19] Farabet C, Poulet C, LeCun Y, et al. CNP: an FPGA-based processor for convolutional networks [C]// Proc of International Conference on Field Programmable Logic and Applications. 2009: 1689-1699.

[20] Williamson D. Dynamically scaled fixed point arithmetic [C]// Proc of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing Conference. 1991: 315-318.

[21] Rastegari M, Ordonez V, Redmon J, et al. XNOR-net: imagenet classification using binary convolutional neural networks [C]// Proc of European Conference on Computer Vision. 2016: 525-542.

[22] Umuroglu Y, Nicholas J F, Gambardella G, et al. FINN: a framework for fast scalable binarized neural network inference [C]// Proc of ACM/SIGDA International Symposium on Field Programmable Gate Arrays. 2017: 65-74.

[23] Hubara I, Courbariaux M, Bengio Y. Binarized neural networks [C]// Advances in Neural Information Processing System. 2016: 4107-4115.

[24] Li Fengfu, Zhang Bo, Liu Bin. Ternary weight networks [J]. arXiv e-print, arXiv: 1605.04711, 2016.

[25] Ren Ao, Li Zhe, Ding Caiwen, et al. SC-DCNN: highly-scalable deep convolutional neural network using stochastic computing [C]// Proc of International Conference on Architectural Support for Programming Languages and Operating Systems. 2017: 405-418.

[26] Ioffe S, Szegedy C. Batch normalization: accelerating deep network training by reducing internal covariate shift [J]. arXiv eprint, arXiv: 1502.03167, 2015.

[27] Krizhevsky, Hinton G. Learning multiple layers of features from tiny images [D]. [S.l.]: University of Toronto, 2009.

[28] Suda N, Chandra V, Dasika G, et al. Throughput-optimal OpenCL based FPGA accelerator for large-scale convolutional neural networks [C]// Proc of ACM/SIGDA ISFPGA. 2016: 16-25.

[29] Qiu Jiantao, Wang Jie, Yao Song, et al. Going deeper with embedded FPGA platform for convolutional neural network [C]// Proc of the 26th International Conference on Field Programmable Logic and Applications. 2016: 1-9.

[30] Venieris S I, Bouganis C S. fpgaConvNet: a framework for mapping convolutional neural networks on FPGAs [C]// Proc of IEEE FCCM. 2016: 40-47.

Note: Figure translations are in progress. See original paper for figures.