RNA: A Reconfigurable Architecture for Hardware Neural Acceleration

Fengbin Tu, Shouyi Yin, Peng Ouyang, Leibo Liu, Shaojun Wei
Tsinghua National Laboratory for Information Science and Technology
Institute of Microelectronics, Tsinghua University, Beijing, P.R.China
Email: tfb13@mails.tsinghua.edu.cn

***Abstract*—As the energy problem has become a big concern in digital system design, one promising solution is combining the core processor with a multi-purpose accelerator targeting high performance applications. Many modern applications can be approximated by multi-layer perceptron (MLP) models, with little quality loss. However, many current MLP accelerators have several drawbacks, such as the unbalance of their performance and flexibility. In this paper, we propose a scheduling framework to guide mapping MLPs onto limited hardware resources with high performance. The framework successfully solves the main constraints of hardware neural acceleration. Furthermore, we implement a reconfigurable neural architecture (RNA) based on this framework, whose computing pattern can be reconfigured for different MLP topologies. The RNA achieves comparable performance with application-specific accelerators and greater flexibility than other hardware MLPs.**

I. Introduction

With the slowing down of Dennard Scaling, the energy problem has become an important issue for digital systems. An attractive solution to this problem is utilizing a heterogeneous architecture with the core processor and a few accelerators. The core processor executes the basic computation and schedules the whole system's resources. The computational intensive applications are assigned to the specialized accelerators, which balance performance with power efficiency.

More application-specific accelerators would expand the application scope, but they might not be used simultaneously, resulting both area and power inefficiency. One of the best alternatives is a reconfigurable accelerator targeting a broad range of applications. This accelerator can be reconfigured by the core processors to perform different algorithms.

However, different algorithms have their own characteristics of data dependencies, core arithmetic units and memory access patterns. It is difficult to use a uniform architecture to cover a very broad application scope. Many emerging high performance applications, such as image recognition, data mining and signal processing, can tolerate a certain degree of inaccurate computing [1].

The neural network is proved to be a practical tool for approximation and prediction. A recent work [2] has demonstrated that some applications of the PARSEC benchmarks suite [3] can be approximated by neural network model with little loss of the quality. In Esmaeilzadeh *et al.*'s work [4], some regions of imperative code are annotated and then approximated with multi-layer perceptrons (MLPs).

The MLP is one of the most widely used neural networks for approximation. Theoretically, any function can be approximated to a certain accuracy with the MLP [5]. Therefore, many works have been done on designing a neural accelerator for MLPs. Kim *et al.* [6] implemented a SIMD neural accelerator for image processing. Temam [7] designed an accelerator with the spatial expansion style, targeting the UCI machine learning repository [8]. In their designs, a MLP is the spatial or temporal combination of basic processing elements (PEs). Though they give support for different MLPs, these works mainly have three drawbacks:

- The largest MLP size supported is often limited by the PE's attributes and the system architecture, so they can't deal with too large MLPs without any extension of PEs.
- A neuron is usually assigned to only one PE. If neurons are fewer than PEs, the resource can't be fully used, which might lead to lower performance and efficiency.
- When the accelerator is frequently invoked, the high latency would become a big concern. Their designs just implement hardware MLP acceleration, but don't have a reasonable scheme to guarantee the best performance for any MLPs.

In this work, each layer of the MLP is abstracted into several multi-layer loops and then mapped to the hardware successively. We propose a scheduling framework, where the loop's computation pattern is reshaped to enable different sized MLPs to execute on the limited resources efficiently. The framework is well designed to solve the main constraints of hardware MLP acceleration. Furthermore, we design an architecture called RNA, to accelerate the different loops provided by the proposed framework. The RNA is reconfigurable for PEs' arithmetic functions, accumulation times, output unit types and the communication patterns, thus providing support for different MLP topologies.

II. General Views on MLPs

The MLP is the most widely used neural network model for approximation and prediction. Back propagation (BP) based learning algorithms can be conducted on MLPs to easily adjust them to different approximation tasks. Therefore, we focus our discussions on how to effectively map different MLPs onto limited hardware resources. In this section, we analyse the data flows in MLPs as multilayer loops and explain how to reorganise the loops to achieve both high flexibility and performance.

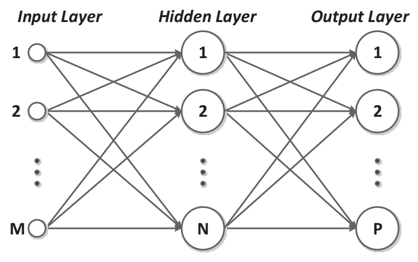


Fig. 1. A typical 3-layer MLP.

A. Loop Representations for MLPs

As shown in Fig.1, a typical 3-layer MLP ($M \times N \times P$) contains two layers of computation: the hidden layer and the output layer. As for the hidden layer, each of the N neurons accumulates the weighted M inputs and then generates the result through a sigmoid function. Since the sigmoid only operates in the end, the accumulation consumes the most execution time. In software, this accumulation process can be represented as the loop below (called Full Parallelism, FP):

$$\begin{array}{l} for(i=0; i<M; i++)\{ \\ \quad S_{1,i+1} \overset{p}{=} S_{1,i} + D_{i+1} \cdot W_{1,i+1}; \\ \quad S_{2,i+1} \overset{p}{=} S_{2,i} + D_{i+1} \cdot W_{2,i+1}; \\ \quad \ldots \\ \quad S_{N,i+1} \overset{p}{=} S_{N,i} + D_{i+1} \cdot W_{N,i+1}; \} \end{array} \tag{1}$$

, where the partial sums are stored in the matrix $S_{N\times(M+1)}$ (One row for each neuron). D_M stores the M inputs and $W_{N\times M}$ is the corresponding weight matrix. The operator "$\overset{p}{=}$" means that the equations complete in parallel, because neurons in the same layer have no data dependence between each others.

B. Scheduling Constraints

The main problem is how to map a loop like (1) onto hardware resources. In most works, the hardware MLP is implemented by numbers of PEs with certain interconnections. In this section, we simplify the PE as an arithmetic logical unit (ALU) that can switch to an adder or multiplier. The ALU has two output modes: the direct mode and sigmoid mode. The PEs' interconnections can be optimized based on our scheduling framework (more details will be illustrated in Section IV). This scheduling problem has three constraints, which are not well solved by many hardware MLPs:

- **Scalability Constraint**: The number of PEs (N_{pe}) is limited by the system size, so the FP fashion like (1) becomes impossible in hardware, if the neuron's number (N) is bigger than the maximum that the system can support.
- **Real-time Constraint**: The number of inputs (M) varies in different tasks. If M is a big number, it's almost impossible to store all the input data in the on-chip memory, especially in a real-time system. Therefore, to avoid too much latency, one data has to enter the system only once.
- **Utilization Constraint**: If the neurons are relatively fewer, some PEs might not be utilized, thus leading to a loss of performance and efficiency.

III. Mapping MLPs to Hardware

In the FP method, a neuron's computation can be completed with only two PEs with a software pipelining method(as shown in Fig.2(a)). These two PEs are configured as a multiplier and an adder respectively. The input data is multiplied by its weight and then added with the last partial sum in the next cycle. The total process would consume $M+1$ cycles. If we directly apply this computation pattern to a layer of N neurons, the scalability's bottleneck would be the total number of PEs (**Scalable Constraint**). On the other hand, the input data enters the system only once (**Real-time Constraint**), so this data has to stay in the system before all the neurons complete its accumulation. In this case, the computation shouldn't be directly mapped to the hardware. Therefore, a new computation pattern need to be introduced to overcome the problems. Fortunately, the inputs are independent and their order doesn't change the final result, so it's possible to reorganise the loop for the two scheduling constraints. In this section, we ignore the exact connections between PEs and propose two extensional methods to adjust the loop for different cases.

A. Neuron Extension

In one extensional method, called Neuron Extension (NE), the input data is stored in the multiplier and successively serves for N neurons in a pipeline. No data would come in before the current input data's accumulation finishes. As shown in Fig.2(b), the data is sent to n neurons at the same time , and n is the maximum number of parallel neurons supported. If we assume N to be a multiple of n, the loop (1) can be reorganised as:

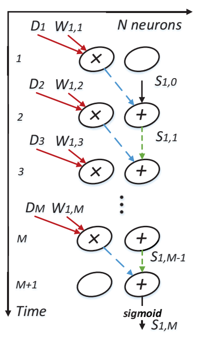
$$\begin{array}{l} for(i=0; i<M; i++) \\ \quad for(j=1; j<=N/n; j++)\{ \\ \quad\quad S_{(n-1)j+1,i+1} \overset{p}{=} S_{(n-1)j+1,i} + D_{i+1} \cdot W_{(n-1)j+1,i+1}; \\ \quad\quad S_{(n-1)j+2,i+1} \overset{p}{=} S_{(n-1)j+2,i} + D_{i+1} \cdot W_{(n-1)j+2,i+1}; \\ \quad\quad \ldots \\ \quad\quad S_{nj,i+1} \overset{p}{=} S_{nj,i} + D_{i+1} \cdot W_{nj,i+1}; \} \end{array} \tag{2}$$

In NE, n neurons compute concurrently and the computation time is reduced to be $\frac{1}{n}MN+1$. Though the parallelism is still limited by the hardware resources, the software pipelining guarantee its high flexibility for different MLP sizes.

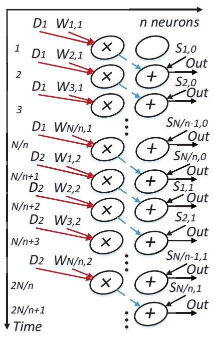
B. Computation Extension

The special designed pipeline makes NE well suited for a large MLP size. However, if $N < n$, NE would degrade to FP, which can't take full use of all the PEs (**Utilization Constraint**). Instead of mapping neurons like NE, we can unroll the accumulations onto the PEs, which is called Computation Extension (CE). Since the inputs and partial sums have no data dependence, several multipliers can operate together, whose corresponding loop representation is:

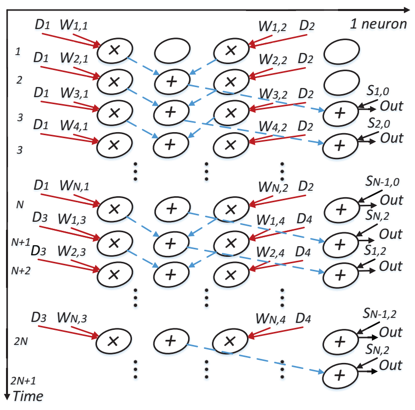
$$\begin{array}{l} for(i=0; i<M/m; i++) \\ \quad for(j=1; j<=N; j++)\{ \\ \quad\quad S_{j,i+m} \overset{p}{=} S_{j,i} + \sum_{k=1}^{m} D_{i+k} \cdot W_{j,i+k}; \} \end{array} \tag{3}$$



(a) FP



(b) NE



(c) CE

Fig. 2. Scheduling methods.

The parameter m is assumed to be the maximal number of parallel multipliers and it takes $\lceil \log_2 \text{m} \rceil$ cycles to merge these results. Fig.2(c) shows an example ($m = 2$) of CE: the software pipelining technique similar to NE is used again to shorten the execution time. The computation costs $\frac{1}{m}MN + \lceil \log_2 \text{m} \rceil + 1$ cycles, which is close to that of NE's.

C. Scheduling Framework

The three scheduling methods for MLPs are concluded in TABLE I, where CPC and MAC are short for computation cycles and memory access cycles. FP has no memory accesses, but its application scope is limited by the system size ($N < n$). NE is FP's extension designed for large MLPs, but it has to store and load partial sums every cycle. CE is able to deal with multiple MLP topologies with high performance close to NE. Since some partial sums are merged directly in CE, it requires fewer memory accesses during computation.

In this section, we propose a scheduling framework based on these three methods, to achieve the highest performance for any MLPs. We use the total execution time (TET) as the metric for performance. The TET is made up of three parts : the configuration cycles (CFC), the actual computation cycles (CPC) and the extra cycles for memory accesses (MAC). The whole MLP's TET is the sum of each layer's TET:

$$TET = CFC + CPC + MAC \tag{4}$$

TABLE I. Three Scheduling Methods

Methods	CPC	MAC	PEs	Scope
FP	$M + 1$	0	$2N$	$N \leq n$
NE	$\frac{1}{n}MN + 1$	$2(M-1)N$	$2n$	$N > n$
CE	$\frac{1}{m}MN + \lceil \log_2 \text{m} \rceil + 1$	$2(\frac{1}{m}M - 1)N$	$2m$	All

The scheduling framework we propose is illustrated in Algorithm 1. The algorithm takes the MLP's topology, m and n as inputs. Since the MLP's computation actually starts from the hidden layer, the layer's label l is initialised to be 2. The algorithm's output is a scheduling table (ST), which indicates the scheduling method of each layer. Since the MLP executes layer by layer, we choose the method with the shortest TET for each layer separately and obtain the whole schedule by combining them.

Algorithm 1 Scheduling Framework for MLPs

```
Require: $MLP, m, n$
Ensure: $ST$
 1: $l \leftarrow 2$
 2: while $l \leq MLP.Layers$ do
 3:     $M \leftarrow MLP.Layer(l).Inputs$
 4:     $N \leftarrow MLP.Layer(l).Neurons$
 5:     if $N > n$ then
 6:         $ST.Layer(l) \leftarrow argmin(TET_{NE}, TET_{CE})$
 7:     else
 8:         $ST.Layer(l) \leftarrow argmin(TET_{FP}, TET_{CE})$
 9:     end if
10:     $l \leftarrow l + 1$
11: end while
```

IV. Hardware Implementation

In the framework we have proposed, the only hardware constraint is the number of PEs. However, the exact hardware architecture will influence the parameter m and n , or even the computation of TET. In this section, we design an architecture that supports the proposed scheduling framework and reduces TET to the minimum level.

A. PE Design for High Parallelism

For a loop with finite steps, increasing the degree of parallelism (the number of parallel operators, $2n = N_{pe}$) will achieve shorter execution time. Since NE is specially targeted for large MLP sizes, it would be very meaningful if this process can be further accelerated. The pipeline in NE is actually an accumulation structure composed of two PEs. In our previous discussions, all the PEs are assumed to be ALUs without special functions inside. Since each of them already have a multiplier and an adder inside, there is some space for more parallelism. As shown in Fig.3, we design an accumulation structure in it. By changing $C0$ and $C1$, the PE performs the multiplier, adder or accumulation function. The PE's output type is controlled by $C2$. Therefore, the NE/FP's pipeline can be built on the single PE independently, thus achieving the maximal parallel degree ($2n = 2N_{pe}$).

B. Interconnection Design for Low Latency

Unlike NE, CE's PEs have frequent data communications, which needs extra interconnections of PEs to support. Fig.4(a)

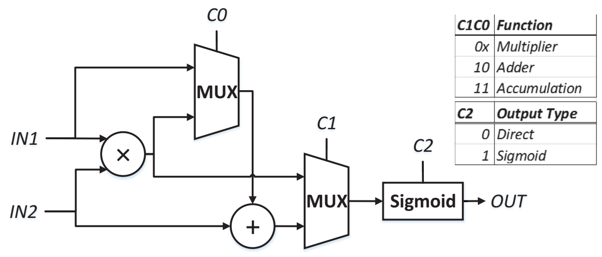
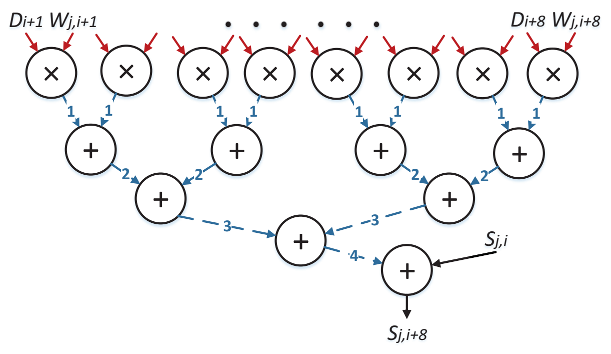
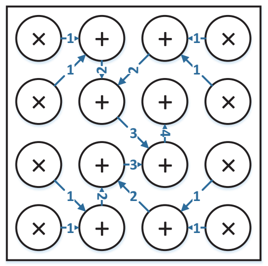


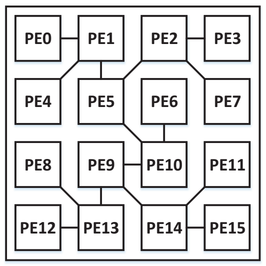
Fig. 3. PE design.



(a) CE ($m = 8$)



(b) CE's data flows



(c) PEA

Fig. 4. An example of a 4×4 PEA.

is an example of a typical CE's kennel ($m = 8$), which occupies totally 16 PEs. The data flows are reshaped as a 4×4 style (Fig.4(b)). The arrows and numbers between PEs indicate the directions and orders of data flows, which can be directly mapped to the interconnections of a PE array (PEA), as shown in Fig.4(c). This distinctive structure is very different from the traditional mesh designs. In a mesh-PEA, all the PEs have full connections with their four direct neighbors. The communications between diagonal neighbors need an extra neighbor PE as an router, which would cost one cycle's latency. In our design, diagonal neighbors are able to communicate through direct connections without any latency. Meanwhile, the removal of the unnecessary connections saves chip area and routing power consumption.

C. RNA Design

We implement a reconfigurable neural architecture by expanding the PEA with an online reconfiguration mechanism to support the scheduling framework. As illustrated in Fig.5, the RNA is composed of the FIFO interface, PEA, data memory and controller.

The FIFO interface is the communication interface between the RNA and external, including 1 configuration FIFO (CFIFO), 1 data FIFO (DFIFO) and 16 Weight FIFOs (WFIFOs).

The PEA is the core arithmetic component for neural acceleration. Each PE loads the MLP's input data from the DFIFO and weights from their own WFIFO. The data memory is divided into 16 memory banks for the PEs, where the intermediate results are stored. Since the hidden layer's outputs will be used for all the neurons of the next layer, the data memory is specially designed to open full accesses for all the PEs.

The controller schedules the whole system based on the configurations read from the CFIFO. The configuration contains the information needed to implement the current scheduling method, including validation, memory addresses, data path controls and so on. The controller's four stages (*Load Configuration, Load Data, Compute* and *Store Data*), are design as a pipeline. Thus, configurations are loaded every cycle and the CFC and MAC are hidden in the pipeline's execution. In our design, TET is reduced to the minimum level:

$$TET = CPC + STG - 1 \tag{5}$$

, where STG stands for the number of the controller's stages and $STG - 1$ equals to the cycles for filling the pipeline.

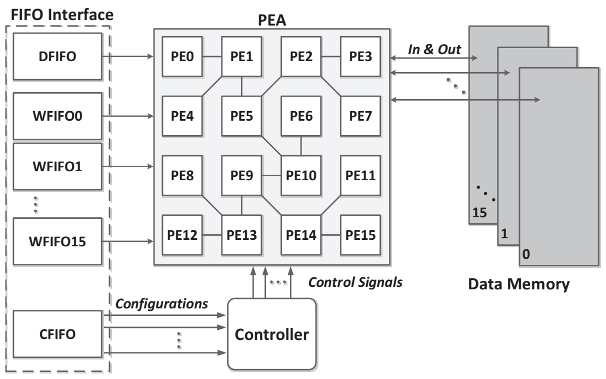


Fig. 5. The RNA's architecture.

V. Experimental Results

We have implemented a RNA, with a 4×4 PEA, to test the scheduling framework's flexibility and performance. Our design is synthesized using TSMC 65nm technology. The two important parameters in the scheduling framework can be calculated: $m = 8, n = 16$. We conduct three sets of experiments based on the RNA: We first approximate some applications with the RNA, and compare the approximation errors with software implementations; We change the RNA's scheduling framework to CE and pure FP/NE, to illustrate the proposed framework's effectiveness; Finally, we compare the RNA's flexibility and performance with some application-specific accelerators and other neural accelerators.

TABLE II. THE RNA'S FLEXIBILITY FOR DIFFERENT MLPS

App.	Topology[8]	Scheduling	SW. Error[8]	HW. Error
fft	$1 \to 4 \to 4 \to 2$	$F \to F \to F$	7.22%	9.34%
inversek2j	$2 \to 8 \to 2$	$F \to C$	7.50%	10.56%
jmeint	$18 \to 32 \to 8 \to 2$	$N \to F \to C$	7.32%	8.83%
jpeg	$64 \to 16 \to 64$	$F \to N$	9.56%	11.84%
kmeans	$6 \to 8 \to 4 \to 1$	$F \to C \to C$	6.18%	10.21%
sobel	$9 \to 8 \to 1$	$F \to C$	3.44%	5.72%

A. Flexibility for Different MLPs

We train some MLPs to approximate the *fft*, *inversek2j*, *jmeint*, *jpeg*, *kmeans* and *sobel* applications [8], based on the topologies given in Esmaeilzadeh *et al.*'s work [4]. They have approximated some high performance applications with MLP models and achieved comparable accuracy. We map our trained MLPs to the RNA and use the same error metrics to test the hardware approximation errors (shown in TABLE II). The column $Scheduling$ lists the scheduling method for each layer (from the second layer), while F, N, C stand for FP, NE, CE separately.

The result shows that the RNA achieves high flexibility for different MLP topologies with acceptable loss of quality. The hardware error rates are a little higher than the software's, mainly because of the fixed-point operations in hardware implementation and the differences in training/testing datasets.

In the application $jpeg$, the output layer has 64 neurons with 16 inputs. Without the proposed scheduling method, a $16-$PE array wouldn't have the flexibility to support such a large MLP. As for other applications, we can see that not all the MLPs use only one scheduling method. The application $jmeint$ even uses three different scheduling methods in its three layers. It means that there's no one scheduling method suited for any MLP topologies, and the mixed method provides much more flexibility.

B. Effectiveness of the Scheduling Framework

To test the effectiveness of our scheduling framework, we change the scheduling methods on the RNA and compare the performance of pure CE, FP/NE with the proposed method. Experiments are conducted on the 4×4 RNA to measure TET that they consume for the same approximation tasks (*fft*, *inversek2j*, *jmeint*, *jpeg*, *kmeans* and *sobel*), as shown in Fig.6. As we have discussed in Section III-C, the single FP or NE don't have the flexibility for arbitrary MLP sizes. In this experiment, they are combined as one method to provide a wide enough application scope.

We can see that CE always has the longest TET among the three scheduling methods. Meanwhile, FP/NE's TET is usually a little worse than the proposed method. It can be concluded that our scheduling framework is quite suitable for different MLP topologies:

- FP/NE is actually a mixed method without CE, so its TET is usually very close to the proposed method's. In our scheduling framework, FP/NE is used to deal with the majority of MLP layers and decides most of the performance, because it is able to parallelly accelerate multiple neurons.
- CE has no advantage over FP/NE if $N > m$, because of its relatively smaller scaling factor (m) than NE's. However, CE's high utilization of PEs helps it to achieve considerable performance on fewer neurons. Typical MLPs usually has multiple inputs and a smaller number of neurons. Therefore, as shown in TABLE II, CE is often used for the output layers as a supplement to FP/CE.

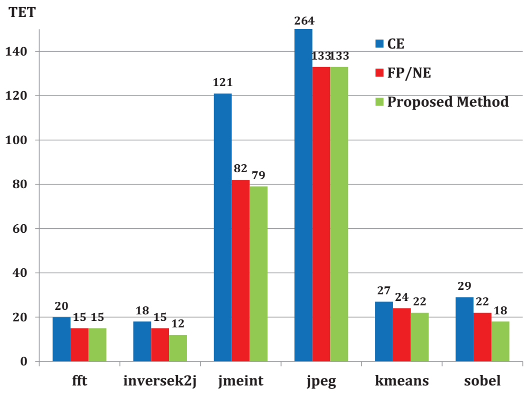


Fig. 6. Performance of different scheduling methods.

TABLE III. COMPARISON WITH OTHER NEURAL ACCELERATORS

Accelerator	Area	Power	Components	App. Scope
ISNN'05 [6]	*	*	$16 \to 16 \to 16$	image recognition
ISCA'12 [7]	$9.02mm^2$	4.70W	$90 \to 10 \to 10$	UCI repository
RNA	$7.24mm^2$	0.96W	16 PEs	much broader

C. Comparisons with Other Accelerators

Our RNA is synthesized using the Synopsys Design Compiler. It is compared with another two works on MLP neural acceleration. Kim *et.al.* [6] implement a 3-layer MLP ($16 \to 16 \to 16$) for image recognition. Similarly, Temam [7] fixes the MLP's topology to $90 \to 10 \to 10$ based on the UCI repository [8]. As for the RNA, its size is close to that of Temam's work [7], but the average power consumption is much lower. The main reason is that the RNA's special PEA design reduces many unnecessary PE connections in traditional mesh designs. Besides, although the RNA has only 16 PEs, it is reconfigurable for any MLP topologies (even bigger than the PEA size). The RNA's application scope is much broader than the mentioned two works.

We also compare the RNA with some application-specific accelerators for *fft*, *jpeg*, *kmeans* and *sobel* (TABLE IV). For fairness, we use a new measurement called energy efficiency (uJ^{-1}), defined by the energy consumption on a certain workload. A higher efficiency means a better combination of performance and power. The RNA achieves 5 times the energy efficiency of the *fft* accelerator[9]. However, the RNA performs not that well in the *jpeg* test and thus it's less energy efficient than the *jpeg* accelerator. As for *kmeans* and *sobel*, the RNA's energy efficiency is almost at the same

TABLE IV. Comparison with Application Specific Accelerators

Accelerator	Workload	Area	Eff.	RNA
fft [9]	64 point FFT	$13.48mm^2$	0.488	2.17
jpeg [10]	64×64 image compression	$0.28mm^2$	414	12.8
kmeans [11]	4 clustering for 65536 points	$0.14mm^2$	75	112
sobel [12]	64×64 image edge detection	*	5443	1083

magnitude as the specific accelerators. Overall, the RNA's energy efficiency is comparable with each of the application specific accelerators, but its area cost is even smaller than the sum of them. More importantly, the flexibility of the RNA provides the potential for a much broader application scope than any of these accelerators.

VI. Conclusions

In this paper, we design a reconfigurable architecture RNA targeting MLPs for multi-purpose applications. We deeply explore the inner data flows in MLPs and propose a scheduling framework, where the loop's computation is reorganised to ensure the hardware's flexibility for different MLPs with high performance. The framework is well designed to solve the three main constraints of hardware MLP acceleration, and is proved to be a good combination of high flexibility and performance.

The RNA is designed to accelerate the reorganised loops provided by the proposed framework. The RNA is reconfigurable for PEs' data paths and the communication patterns between them, thus providing support for the scheduling methods FP, NE and CE in the framework. Our design is synthesized using the Synopsys Design Compiler. It is configured to approximate some applications with different MLPs, whose approximating error is close to the software's. The RNA's energy efficiency is comparable with the applications-specific accelerator, while targeting a much broader application scope.

Acknowledgment

This work is supported in part by the China Major S&T Project (No.2013ZX01033001-001-003), the International S&T Cooperation Project of China grant (No.2012DFA11170), the Tsinghua Indigenous Research Project (No.20111080997) and the NNSF of China grant (No.61274131).

References

[1] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "Enerj: approximate data types for safe and general low-power computation," in *Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation*. ACM, pp. 164–174.

[2] T. Chen, Y. Chen, M. Duranton, Q. Guo, A. Hashmi, M. Lipasti, A. Nere, S. Qiu, M. Sebag, and O. Temam, "Benchnn: On the broad potential application scope of hardware neural network accelerators," in *Workload Characterization (IISWC), 2012 IEEE International Symposium on*. IEEE, pp. 36–45.

[3] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, pp. 72–81.

[4] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, pp. 449–460.

[5] T. M. Mitchell, "Machine learning. 1997," *Burr Ridge, IL: McGraw Hill*, vol. 45, 1997.

[6] D. Kim, H. Kim, H. Kim, G. Han, and D. Chung, *A SIMD neural network processor for image processing*. Springer, 2005, pp. 665–672.

[7] O. Temam, "A defect-tolerant accelerator for emerging high-performance applications," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*. IEEE Computer Society, pp. 356–367.

[8] A. Asuncion and D. Newman, "UCI machine learning repository," 2007. [Online]. Available: http://archive.ics.uci.edu/ml

[9] K. Babionitakis, K. Manolopoulos, K. Nakos, D. Reisis, N. Vlassopoulos, and V. A. Chouliaras, "A high performance vlsi fft architecture," in *Electronics, Circuits and Systems, 2006. ICECS'06. 13th IEEE International Conference on*. IEEE, pp. 810–813.

[10] Z. Qihui, C. Jianghua, Z. Shaohui, and M. Nan, "A vlsi implementation of pipelined jpeg encoder for grayscale images," in *Signals, Circuits and Systems, 2009. ISSCS 2009. International Symposium on*. IEEE, pp. 1–4.

[11] T.-W. Chen, C.-H. Sun, H.-H. Su, S.-Y. Chien, D. Deguchi, I. Ide, and H. Murase, "Power-efficient hardware architecture of k-means clustering with bayesian-information-criterion processor for multimedia processing applications," *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, vol. 1, no. 3, pp. 357–368, 2011.

[12] N. Kazakova, M. Margala, and N. G. Durdle, "Sobel edge detection processor for a real-time volume rendering system," in *Circuits and Systems, 2004. ISCAS'04. Proceedings of the 2004 International Symposium on*, vol. 2. IEEE, pp. II–913–16 Vol. 2.