Deffe: A Data-Efficient Framework for Performance Characterization in Domain-Specific Computing

Frank Liu
Oak Ridge National Laboratory
liufy@ornl.gov

Narasinga Rao Miniskar
Oak Ridge National Laboratory
miniskarnr@ornl.gov

Dwaipayan Chakraborty
Oak Ridge National Laboratory
chakrabortyd@ornl.gov

Jeffrey S. Vetter
Oak Ridge National Laboratory
vetter@computer.org

ABSTRACT

As the computer architecture community moves toward the end of traditional device scaling, domain-specific architectures are becoming more pervasive. Given the number of diverse workloads and emerging heterogeneous architectures, exploration of this design space is a constrained optimization problem in a high-dimensional parameter space. In this respect, predicting workload performance both accurately and efficiently is a critical task for this exploration. In this paper, we present **Deffe**: a framework to estimate workload performance across varying architectural configurations. Deffe uses machine learning to improve the performance of this design space exploration. By casting the work of performance prediction itself as transfer learning tasks, the modelling component of Deffe can leverage the learned knowledge on one workload and "transfer" it to a new workload. Our extensive experimental results on a contemporary architecture toolchain (RISC-V and GEM5) and infrastructure show that the method can achieve superior testing accuracy with an effective reduction of 32-80× in terms of the amount of required training data. The overall run-time can be reduced from 400 hours to 5 hours when executed over 24 CPU cores. The infrastructure component of Deffe is based on scalable and easy-to-use open-source software components.

CCS CONCEPTS

• **Computer systems organization → Reduced instruction set computing**; **Heterogeneous (hybrid) systems**; • **Computing methodologies → Modeling methodologies**; **Simulation evaluation**; **Neural networks**; *Classification and regression trees*; *Support vector machines.*

KEYWORDS

Workload Characterization, Machine Learning, Transfer Learning, Multichannel Convolution, RISC-V

ACM Reference Format:
Frank Liu, Narasinga Rao Miniskar, Dwaipayan Chakraborty, and Jeffrey S. Vetter. 2020. Deffe: A Data-Efficient Framework for Performance Characterization in Domain-Specific Computing. In *17th ACM International Conference on Computing Frontiers (CF '20), May 11–13, 2020, Catania, Italy.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3387902.3392633

CF '20, May 11–13, 2020, Catania, Italy

ACM ISBN 978-1-4503-7956-4/20/05…$15.00
https://doi.org/10.1145/3387902.3392633

1 INTRODUCTION

As the Moore's law and Dennard scaling come to end, domain-specific computing is poised to provide viable and effective solutions to meet performance and energy requirements of computing workloads [13]. Traditional general-purpose processor cores are optimized for balanced performance on a wide spectrum of workloads. While run-time configurable logic such as FPGA has demonstrated impressive performance improvements for certain workloads, its rigid hardware substrate limits its applicable scope. By optimizing for a specific set of workloads (domain-specific) and taking the software-hardware co-design approach (with domain-specific language), domain-specific computing and domain-specific architecture can provide order-of-magnitude performance improvement compared to general-purpose cores. On the other-hand ASICs can provide best in performance but suffer from lack of programmability and adaptability. Additionally, the increasingly popular open-source hardware movement, with projects like the RISC-V ISA [30] standard and its hardware implementations [2], have provided a readily accessible platform for architectural explorations.

Computer architecture exploration can be cast as a constrained optimization problem in a high-dimensional design parameter space. With a given suite of workloads, the trade-off has to be made among key objective functions such as performance, energy consumption (correlated to operational cost) and area (correlated to manufacturing cost). In domain-specific architecture search, one of the key steps is the performance prediction of a specific kernel or application by using a cycle-accurate full system simulator [6, 16]. The prediction is not only essential in exploring the design space of architecture parameters, but also capable of identifying unexplored regions in the architecture parameter space for a given workload. This capability is crucial for dynamic runtime reconfiguration.

One of the challenges of the combined architecture parameter and workload characteristic search is the sheer size of the search space. For example, the top portion of Table 1 shows the available architecture parameter space of RISC-V architecture, which translates into over 46,000 potential parameter configurations. If the workload configurations, e.g., matrix size in matrix multiplication, are also considered, the size of the search space goes up drastically. Depending on the type of simulation, hardware used and workload configuration, the simulation time for each configuration can take from tens (10^1) to thousands (10^3) of seconds. It is obvious

that a full-factorial exploration of all architecture and workload characteristics is too time-consuming to be feasible.

Table 1: Architecture design parameters of RISC-V (first seven rows) and workload parameters of four workloads

Component / Kernel	Parameter	Parameter Type	Possible Values
L1-D-Cache	Size (KB)	Architecture	1, 2, 4, 8, 16, 32
L1-D-Cache	Ways	Architecture	1, 2, 4, 8
L1-I-Cache	Size (KB)	Architecture	1, 2, 4, 8, 16, 32
L1-I-Cache	Ways	Architecture	1, 2, 4, 8
L2-Cache	Size (KB)	Architecture	1, 2, 4, 8, 16, 32, 64, 128, 256
L2-Cache	Ways	Architecture	2, 4, 8
Cache-line	Size (B)	Architecture	16, 32, 64
K-means	Number of objects	Kernel-Runtime	Range: [100 - 50000]
Matrix Multiplier	Matrix size (Square Matrix)	Kernel-Runtime	Range: [16 - 256]
Needleman-Wunsch	Matrix size (Square Matrix)	Kernel-Runtime	Range: [128 - 2048]
Back Propagation	Layer size	Kernel-Runtime	Range: [16 - 4096]

The focus of this work is Deffe (Data-Efficient Framework for Exploration), a data-efficient framework to estimate the performance of workloads under various architectural configurations. Deffe consists of two major components: a distributed computing infrastructure to emulate the workload performances (e.g., estimated number of CPU cycles) of given workloads under various architectural configurations, and a comprehensive workload performance macro-model built on contemporary machine learning techniques. More specifically, the distributed computing infrastructure in Deffe ensures that sufficiently large samples can be computed within a given wall clock time, which is essential for the construction of a macro-model of the performance. In the performance macro-model aspect, we utilize the contemporary multichannel convolutional neural networks, which is capable of "learning" the intricate correlations among various design parameters. These data-driven (or "learned") correlations can be readily explored by using the transfer learning technique to construct new performance models with much smaller amounts of simulation data points.

To summarize, the main contributions of this paper are as follows.

- A workload performance model based on convolutional neural networks and transfer learning techniques. By combining architecture variables and workload characteristics and casting the performance modeling problem as transfer learning tasks in machine learning, our approach can achieve comparable accuracy without a large number of training samples;
- A distributed infrastructure to collect a large number of samples for performance modeling. By using scalable open-source software components, we can readily employ the infrastructure on platforms from lab-wide clusters to large-scale HPC clusters;
- Deffe framework, the implementation of the performance modeling method on the distributed compute infrastructure, which we plan to release for public use.
- An evaluation of Deffe on a contemporary, realistic exploration of RISC-V designs using GEM5 simulation.

The organization of the remaining part of this paper is as follows: in Section. 2, we describe the background of the topic and the related work. In Section. 3, we describe the details of our approach. In Section. 4, we describe the experimental results, followed by conclusion and future work in Section. 5.

2 BACKGROUND AND RELATED WORK

Knowing its potential performance is crucial when design a computer system. A common technique is to estimate the performance of a set of workloads, assuming they would be carried out on the system being designed. This task can be considered as a specialization of workload characterization, which has been a classic research topic for decades[9]. Some examples of more recent work include [4, 15, 20, 28, 32], which explored the research topics such as performance-power trade-offs, performance estimates of big data systems, and workload characterization of heterogeneous systems.

2.1 Machine Learning in Performance Characterization

Machine learning techniques have been widely used in architecture and system research in various contexts. For example, in an early work [7], a fully connected neural network was used in resource management; in [17], machine learning models were developed for online performance prediction. In [21], a customized tree classifier was used to predict performance metrics for task scheduling. In [5], machine learning models were used to predict potential performance gains on GPUs) of parallel programs. The authors of [29] explored the idea of using machine learning techniques, such as PCA (Principal Component Analysis) and neural networks, to explore various options of cache configurations. On the other hand, the authors of [18] used convolutional neural networks to characterize on-chip traffic patterns in GPGPUs. The authors of [14] presented the idea of using a fully connected neural network to learn the performance characterization of various workloads. Our machine learning framework is influenced by their work, but we have made some distinct advancements by leveraging contemporary deep learning architecture and learning techniques. The outcome is a much more robust and data-efficient learning framework, which we will explain in detail in Section 3.

2.2 RISC-V and Gem5

RISC-V [30] is an open-source ISA (Instruction Set Architecture) and is available with open-source software ecosystem such as compilers (GCC, LLVM), simulators (GEM5), emulators (QEMU) and operating systems (Fedora Linux, Embedded RTOS). It has been well received in industry and academia for both embedded and HPC. Researchers in industry and academia have provided many variants of open source RISC-V cores. *Rocket* [2] is a family of popular RISC-V cores provided by University of California, Berkeley. They have been implemented with hardware construction language *Chisel* [3] and have provided customization of various architecture parameters. As an example, the Rocket-chip RISC-V core has two levels of caches: L1 and L2. L1 caches for instructions (L1I) and data (L1D) are separated. All caches in the Rocket chips have configurable architecture parameters, such as *N-ways*, *size* and *cache line size*. These parameters can impact the performance of kernel execution on RISC-V significantly, and they can lead to heterogeneous RISC-V cores. Exploration and fine-tuning of these architecture parameters

is a problem of design space exploration for the target kernels. The architecture components of the RISC-V system is shown in Figure 1.

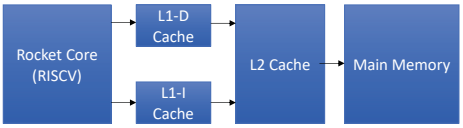


Figure 1: Main components of RISC-V architecture

We use RISC-V as the test architecture of our performance characterization framework. Our rationale of using RISC-V is due to its openness and rapid adoption. However our framework is quite general in that it can be deployed for other architectural or system-level exploration work.

The performance characterization of specific kernels for the RISC-V processor can be obtained either by running them on real-chip boards (e.g., HiFive Unleashed) or evaluation boards (e.g., Xilinx VUP9P FPGA), or by using cycle-accurate simulators. Real circuit boards are not suitable for the design space exploration problems because we need to fine-tune architectures and explore various architecture parameters. HDL simulators such as Verilator can provide a cycle-accurate behavior model in C++ or SystemC for the given verilog files. However, they are very slow in simulation, which runs at ~10 kHz. Emulators such as QEMU can provide functionality of the RISC-V cores with very fast emulation, but they do not provide performance cycles. FireSim [16] is a cycle-accurate FPGA-accelerated system-level simulation platform, running on Amazon AWS cloud FPGA resources. It can run the simulations at the rate of ~150 MHz. However, it requires FPGA resources and engineering effort to implement. The GEM5 simulator [6] can provide a near cycle-accurate behavior of RISC-V with parameterized system architecture components. The detailed CPU behavioral models implemented for RISC-V [25] in GEM5 can provide near 90% cycleaccuracy when compared to Verilators and FPGA soft cores (FireSim), and they can run at a simulation speed of ~10 MHz. Hence, we have selected GEM5 simulator for performance characterization of kernels.

3 FRAMEWORK FOR PERFORMANCE CHARACTERIZATION

In this section we describe Deffe, a general framework for performance characterization. Deffe has two components: a workload performance model based on contemporary neural network structure and transfer learning technique, and a scalable computing infrastructure that can collect sufficiently large training/testing data. Although our proposed machine learning performance model has superior data efficiency (i.e., it requires less data to train), we believe an efficient performance simulation infrastructure is still needed because architecture exploration can be carried out at various granularities (e.g., binary translation, RTL simulation or even ASIC synthesis flow), depending on the level of explorations required. Some of the simulation flows can still be quite time-consuming[25]. On the other hand, the accuracy and generalization of machine-learning-based methods are closely related to the amount of the available training data. Hence the availability of a scalable infrastructure is still a necessary basic component.

3.1 Distributed Simulation Infrastructure

The full space exploration algorithm will provide the cost metrics for all combinations of design point parameters. In the case of discrete design parameters, if the search method requires exploration of N parameters, each with M_i values, the total design points in the search space will be $C = \prod_i^N M_i$. It is obvious that the total number is exponential with respect to the search space dimension. Exhaustive search of all possible design points may be good when the number of parameters is small. Otherwise it is impractical for high-dimensional design space exploration.

However, we believe it is still necessary to construct a scalable compute infrastructure to enable timely exploration of large design points. The rationale is that any macro-modeling approach depends on the local curvature of the objective function. Hence it is difficult to predict the minimal number of sampling points a priori. By building a scalable computing infrastructure, we have the flexibility to build accurate macro-models by using larger sample size without sacrificing the accuracy due to limited simulation time.

To this end, we constructed the computing infrastructure by using DAKOTA[1] with SLURM[31] as the back end. DAKOTA is a generic framework for Uncertainty Quantification (UQ) and Design Space Exploration (DSE) for scientific discovery and engineering design. It provides configurability of design space parameters, cost metrics, and an execution environment for each design point to obtain the cost metrics. It also provides flexibility to evaluate the batch of jobs for design point executions by means of high-level configurable parameters. SLURM, a highly efficient and highly configurable cluster management tool, is available on almost 60% of TOP500 HPC clusters. The integration of the two packages is straightforward. The result is an easy-to-use, highly configurable and scalable platform to explore large design space. Figure. 2 shows the diagram of the integrated compute infrastructure.

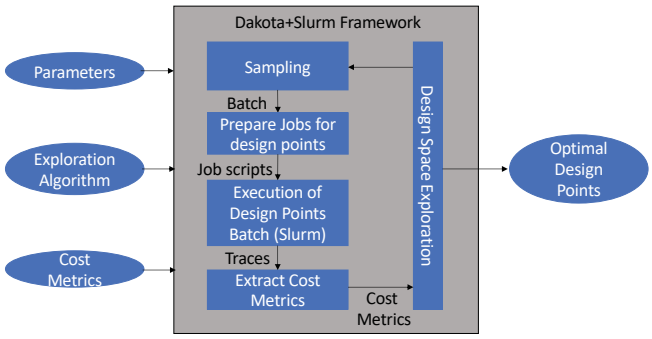


Figure 2: Diagram of integrated Dakota-Slurm infrastructure

3.2 Performance Modeling by Machine Learning

Using machine learning techniques for performance modeling has been explored before. Random Forest[8] is a popular choice due to its simplicity and theoretically lower bound on accuracy. Another option is to use an artificial neural network[14] due to its capacity to map arbitrary complex nonlinear functions. In this work, we use a neural network as the theoretical framework. However, different from previous practice, such as [14], which uses fully connected

network structure, here we use a convolutional neural network (CNN).

3.2.1 Convolutional Network Structure. The CNN structure is the basic building block for many machine learning applications, such as computer vision, voice recognition, and natural language processing. CNN's are different from fully connected (FC) neural networks in that they require fewer parameters (weights) and are more efficient to run. Conceptually, CNN is quite simple. In a 1D case, a CNN is simply a convolution operator with a given window size (e.g., 3) followed by a nonlinear function (e.g., sigmoid function $S(x) = 1/(1+e^{-x})$ or hyper-tangent function $S(x) = \tanh(x)$ as the activation function). However, in machine learning, two additional techniques make the CNN very powerful: (1) CNN can have multiple layers, which effectively cascade multiple convolutions together, and each layer may have multiple "channels," which are effectively parallel convolutions applied to the same input space, and (2) The convolution kernels (or weights) are trained from known data by backpropagation. The overall outcome is that multiple layers of CNN can effectively *discover* the hidden and complex correlations among different input signals. In this work, we use the network structure shown in Figure. 3

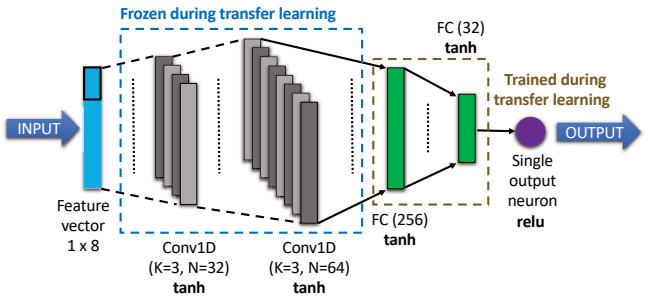


Figure 3: Convolutional neural network model used for performance modeling. The input feature is the concatenation of architectural configurations and workload characteristics. Dropout(not shown) is applied to the FC layers.

The first two layers are convolutional layers, each with kernel width (or *stride*) of 3. The number of channels are 32 and 64, respectively. The purpose of those two convolutional layers is to model the correlations among the input signals. The last two layers of the proposed network are fully connected. To improve the generalization of the network, we deployed dropout[27]. During training, the technique randomly discards a percentage of the connections of the FC layers (e.g., 50%) in backpropagation, to prevent the model from over-fitting.

We explicitly include workload characterization (e.g., the number of objects in k-means clustering) as part of the input signal. The rationale is that when treating the problem size as part of input to the CNN model, we can rely on the learning capacity of CNN to discover the correlation of performance, even across multiple workloads. For example, if certain architectural configuration is slow for a large k-means clustering run, it is likely that it is also slow for a large matrix-matrix multiplication run.

3.2.2 Transfer Learning. An important benefit of using CNN structure is that we can exploit transfer learning techniques in machine learning[22]. The concept of transfer learning can be summarized as follows[22]:

Given a *source* domain D_S and associated learning task T_S, transfer learning deals with learning a new task T_T on a *target* domain D_T, where $D_T \neq D_S$ or $T_T \neq T_S$.

In our application, we explicitly train a CNN network to predict the performance of one workload (e.g., matrix-matrix multiplication) as the source task. We can then use the transfer learning technique to train a different model which can predict performance of a different task (e.g., predict the performance of K-means workload). The benefit is that transfer learning from a pretrained model in domain D_S is usually much faster to train on target domain D_T than training a different model from scratch. A more important benefit for us is that transfer learning requires much fewer training samples while still achieve good accuracy. The intuitive explanation is that the task on the target domain is somehow close to the source domain, and hence transfer learning can leverage learned knowledge to make learning the new task easier.

Deploying transfer learning in CNN structures is widely supported by many deep learning frameworks. In one approach, when training the CNN network for a target task, we can initialize the weights (especially those of the convolutional layers) from those of the source task, which effectively provides a good starting point for the target task CNN (instead of initializing them with random values). Another approach is to fix the CNN weights of the target task with those of the source task and to only allow the weights in the FC layers to be trained. In our study, we choose the latter. Transfer learning is difficult to apply for FC layers because they are translation invariant, and hence FC layers do not have the internal structure for transfer learning to leverage.

3.2.3 Intelligent Incremental Sampling. One remaining question is how to determine the number of samples needed to build the performance model. From a data-efficiency perspective, we want the smallest possible number of samples. On the other hand, the smaller sample size may have a negative impact on the accuracy of the trained model. We propose an incremental approach to overcome this problem. A diagram illustrating the concept is shown in Figure 4. We start with a predefined number of samples. The samples are split into a training set and a validation set. The training set is used to train the model; the validation set is used for cross-validation. Since there is no overlap between the training and validation set, the cross-validation provides an independent metric on the generalization and accuracy of the trained model. In the next step, a percentage of the validation set is added to the training set (the percentage is 100% as shown in Figure 4). We further randomly choose a new batch from the remaining search space, and augment the validation set. The step is repeated with larger and larger training set. We continue the process until the accuracy of the trained model stops improving. The premise of this approach is that as the randomly chosen training set grows, it will eventually cover the the support of the modeling problem, at which point adding more training samples will not contribute to further improvement of the accuracy.

3.2.4 Other Considerations. The simulated CPU cycles have a much wider range than the output of a neural network. We explored two different approaches to accommodate the wide output range: (1) we normalized the CPU cycles by taking their logarithmic values, effectively reducing their range, and (2) we inserted an exponential function into the output of the neural network before

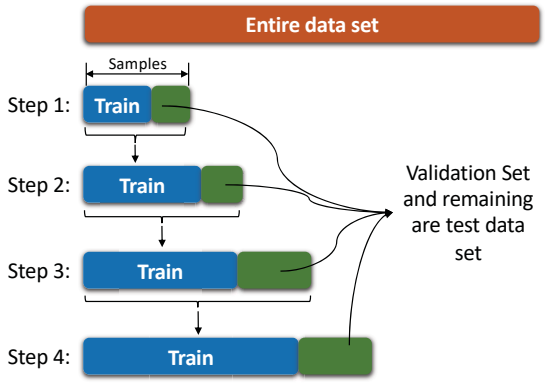


Figure 4: Incremental sampling illustrated. Only four step are shown, although in practice more steps may be necessary.

computing the loss function. The exponential function in the latter makes the training more sensitive to the noise of the stochastic gradient descent algorithm; hence it takes longer to converge. Given the efficiency of contemporary GPUs, this is not a big issue. In the first case, since the simulated CPU cycles have been normalized by a logarithmic function, the loss function (LOSS) we used is effectively $\frac{1}{N}\sum_{i=1}^{N}\frac{|y'_i - ln(y_i)|}{ln(y_i)}$ where y' is the output of the machine learning model and y is the ground truth. We can also compute the relative error by directly comparing the loss of the non-normalized data $\frac{1}{N}\sum_{i=1}^{N}\frac{|e^{y'_i} - y_i|}{y_i}$, which we named EXPLOSS. We compare the performance of Deffe model and baseline models in both cases.

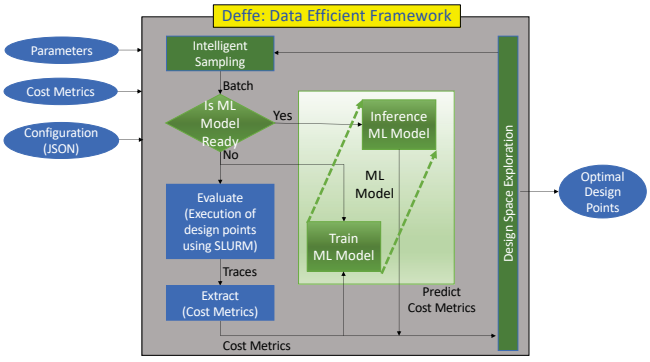


Figure 5: Structure of the Deffe framework with efficient sampling. The blue blocks are compute infrastructure components; the green blocks are modeling components.

3.2.5 Overall Framework. The overall Deffe framework is shown in Figure. 5. Once deployed, the Deffe framework can be used to facilitate other applications, such as design space exploration of architecture parameters, or runtime optimization of configurable logic.

4 EXPERIMENTAL RESULTS

The workload prediction accuracy of the proposed machine learning model was compared with three baseline machine learning regression methods: LARS-Lasso Regression (LLR) [12], Support Vector Regression (SVR) [11], and Random Forest Regression (RFR) [19]. The reason for choosing those three methods is that empirically they achieve the best accuracy among other traditional machine learning methods. We experimented with both baseline models and the proposed machine learning model on both normalized CPU-cycle output (by logarithmic function) and non-normalized output.

4.1 Experiment Setup

Parameters for the experiments include both kernel run-time parameters from benchmarks and RISC-V system architecture parameters. The benchmarks considered for the experiments are *Back-propagation (backprop)*, *KMeans clustering (k-means)*, and *Needleman-Wunsch (nw)* kernels, which are obtained from the Rodinia benchmark suite [10], along with the *matrix multiplication (matmul)* kernel. The selected benchmarks have run-time kernel parameters that affect the run-time performance of the kernel. Although only one run-time kernel parameter is considered for the validation of the proposed method, it can be extended to a larger set of run-time kernel parameters.

The RISC-V system architectural parameters along with the run-time kernel parameters used for the validation are shown in Table 1. The architectural parameters considered for the validation have discrete integer values. The permutation of all possible architectural parameter values yields over 46,000 different configurations, which will be multiplied with range of kernel run-time parameter values. For example, the k-means kernel will lead to 2.3 billion permutations. If the kernel is simulated for each permutation, it takes 1 second on average to get its workload characterization cost metrics (CPU cycles). It would require ~73 years to compute the cost metrics for all permutations on a single machine if this approach was followed. The cost metric (CPU cycles) for each permutation is captured with the GEM5 simulator in batch processing mode, run in a Dakota+Slurm environment, which consists of 20 Intel Xeon nodes, each with 20 CPU cores.

The performance model is implemented in Python, with the option of using Keras-Tensorflow [26] or PyTorch [23] as the underlying deep learning framework. The baseline models are implemented in Python by scikit-learn [24]. We use a NVIDIA Tesla V100-DGXS for both training and inference. However, the inference can be easily carried out on CPUs for easy deployment. The optimizer that we have used in deep learning were Adam for Keras-Tensorflow and SGD for PyTorch. We did not see a clear difference between the two optimizers.

4.2 Validation using a Full Dataset

The first experiment was to demonstrate the accuracy of our proposed machine model. To fully test the accuracy, we have utilized the scalable computing infrastructure of Deffe to generate estimated CPU-cycle data for all four workloads. It took a long runtime to generate data (~1 day per workload on the 20-node cluster). However, we have to emphasis that this is not the operational mode of Deffe, we have simply used the infrastructure to create the data for this study. In a real operation, the performance model construction should take a fraction of the runtime because only a small number of samples are needed.

In this study, we have used a $54-16-30$ split: 54% of data points are used for training, 16% are used for periodical validation during the training process, and the remaining 30% are reserved for testing

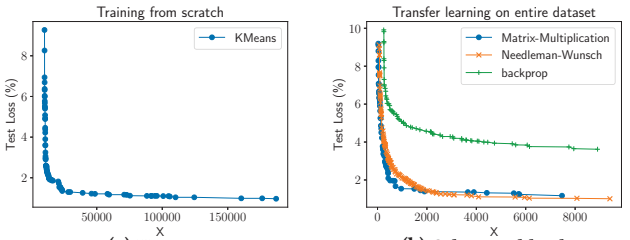
after the training is complete. The losses are reported on the testing data points only.

Table 2: Evaluation results on full dataset. All values are testing losses in percentage. Our CNN-models achieve better accuracy than three baseline models. In the transfer learning column, K-means is trained as the source task.

Kernel	Dataset Size (parameters combi-nation)	Total Simulation Time (hrs) (1 host)	Baseline-Log			Baseline-Real	CNN-Log (4000 epoch)		CNN-Real (NTL: 100k epoch, TL: 8000 epoch)
			Regression	Test Loss (%)		Test Loss (%)	Test Loss (%)		Test Loss (%)
			Type	LOSS	EXPLOSS	EXPLOSS	LOSS	EXPLOSS	EXPLOSS
K-means	152207	9600	LLR	8.4	343.2	114.8	0.2	6.1	2.5
			SVR	4.7	1923.0	306.1			
			RFR	0.2	5.6	5.2			
MatMul	132752	7400	LLR	8.1	194.1	184.2	0.1	3.7	NTL: 2.1 TL: 2
			SVR	2.3	35.4	105.1			
			RFR	2.7	33.0	36.6			
Needleman-Wunsch	56392	6600	LLR	5.4	125.1	50.8	0.1	4.2	NTL: 1.4 TL: 0.9
			SVR	0.5	7.9	117.9			
			RFR	0.3	7.2	6.5			
Back propagation	169266	9000	LLR	10.1	267.0	114.3	0.2	5.1	NTL: 2.1 TL: 3.7
			SVR	1.6	25.3	222.5			
			RFR	1.2	10.5	11.1			

The results are tabulated in Table 2. We have also listed the number of sample points in full-factorial analysis and extrapolated total simulation time if we only use one CPU (instead of a multinode cluster). For baseline comparison, we have included results when the CPU-cycle variable is normalized (Baseline-log) and non-normalized (Baseline-Real). In the first case, the error computation can be carried out using normalized data (named "LOSS") or transformed back to non-normalized (named "EXPLOSS"). For our CNN-based performance model, we have used the same two approaches for accuracy assessment. In the last column, we have also included the accuracy results with and without transfer learning (using the model trained on k-means as the source model). Overall, our CNN model achieves better or comparable in accuracy when the same amount of training data are supplied. We have observed increased loss when computed by transforming the machine learning outputs back to non-normalized values. This is fully understandable for any exponential function. However, it is clear that our CNN-based model behaves better given its better accuracy in the normalized space. Another observation is that when compared with models trained from scratch, the models trained from transfer learning only have small increase in loss, which is in line with observations from other transfer learning applications.

Among the baseline models, RFR has the best overall accuracy, which has been reported by other studies. We observed that the accuracy of RFR on MatMul is considerably worse. Our hypothesis is that this particular workload has certain structure that is not suited for the RFR model. However, we are still investigating the root cause.



(a) K-means (b) Other workloads

Figure 6: Convergence of original training on K-means and applying transfer learning on other models. The number of epochs of transfer learning is an order of magnitude smaller than training from scratch.

To demonstrate the efficiency of transfer learning, we have compared training convergence of a training a model for k-means from scratch as well as convergence using the previously trained k-means model as the source model to the other three workloads, which is shown in Figure 6a. One can observe that the number of epochs (which is closely related to time it take for the model to converge) of the transfer learning is an order of magnitude smaller than training from scratch. This is a clear demonstration that there are close connections among the performance models of different workloads and that transfer learning can benefit from the connections to transfer the learned knowledge from one task to the other.

4.3 Efficient Data Sampling

The efficient sampling with machine learning models can make the overall workload characterization much faster with similar accuracy when compared to complete dataset training. In each step, parameter samples are selected randomly from the given search space based on the method described in Section 3.2.3. The model is trained with these samples until the desired accuracy is met. We conduct three experiments to show the effectiveness of the intelligent sampling method, . But instead of stopping the training until convergence, we have fixed the training to 8000 epochs before augmenting the training data. For testing, we have chosen all of the sampled data which have not been used for training or validation.

4.3.1 Without Transfer Learning. In the first experiment, the proposed machine learning model is trained from the scratch on the given samples each time. In each step, the weights of the model are randomly initialized. This is equivalent to forgoing transfer learning and simply training the model with increasingly more data points. For illustration purpose, we only show the training convergence of backpropagation in Figure 7.

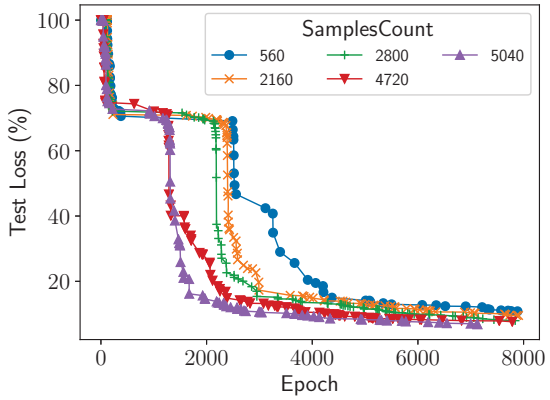


Figure 7: Training from scratch with five sample sizes for "backpropagation." When no prior knowledge is used, testing losses converge much more slowly.

Note that the numbers of training points are only fractions of those in the previous section. In the case of largest number of training points (4,720), the training sample size is only 2% of total numbers reported in Table 2. The trend among the training processes of five sampling sizes clearly indicates that it is more difficult for the model to learn from a smaller dataset if no prior knowledge is provided.

4.3.2 With Transfer Learning across Samples. In the second experiment, the transfer learning of a previously trained model on the last set of samples was used for next set of samples. However, the convolution layer weights are not frozen in training. The trend of test losses after each of set of sampling is shown in Figure 8. Each set of samples is trained for 8,000 epochs. However, the epoch number is shown as being continuous in our graphs after each set of samples. It has been observed that the test losses are minimized to ~5% with <3% of samples. The spikes in the losses in the graph are due to destabilized model training in the early epochs with new set of data along with the transfer learning from the previous set of samples. After some number of epochs the training losses are stabilized. Although the losses are minimized to ~5%, each kernel requires its own model and their weights. This can be avoided without conducting a third experiment.

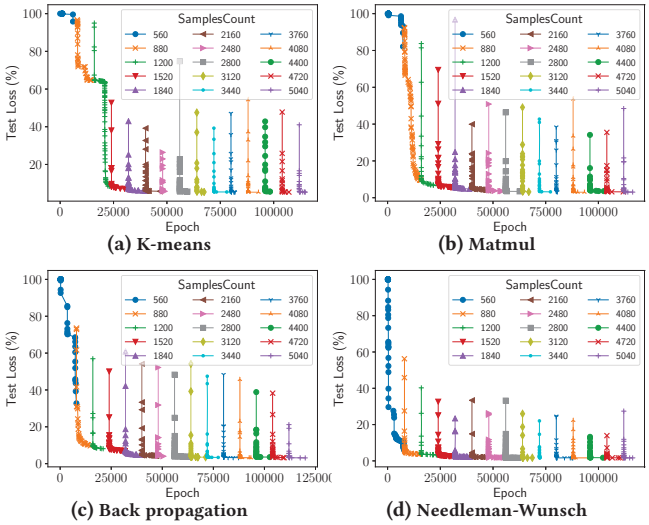


Figure 8: Sampling with transfer learning across samples. The small peaks when new training samples are introduced are typical behavior in training.

4.3.3 With Transfer Learning across Kernels and Samples. In the third experiment, the model trained for one kernel is used as the base for training another kernel. After the transfer learning of the model, the weights of the convolution layers are frozen. Later, the transfer learning of the previously trained model on the last set of samples is used for next set of samples, as mentioned in the previous subsection. The model trained with k-means dataset shares same architecture parameters and can provide some correlation on the workload characterization. Hence, the two convolution layers in the model are frozen in the training process after doing the transfer learning of model weights learned from the k-means dataset. In this approach, we are able to achieve very low losses similar to training on complete datasets, but with less than <3% of its samples. It has been observed that similar accuracy is achieved even with 1,000 epochs per set of samples. The proposed machine learning model is converging fast with transfer learning across kernels and samples.

4.3.4 Summary. We have compared the above three sampling techniques with each other and with the baseline RFR technique, which is the best among the baseline models. The comparison is shown in Table 3. TLS stands for transfer learning of samples and TLK stands

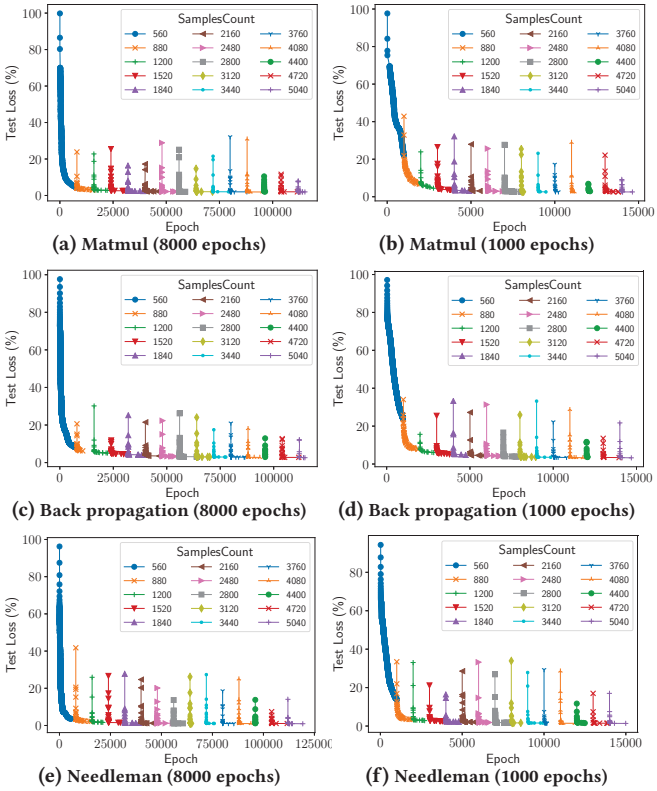


Figure 9: Sampling with transfer learning across kernels and samples. The number of epochs is much smaller than in Figure 8, which is a demonstration of the power of transfer learning.

for transfer learning of kernels. *Without transfer learning* (No TLS, No TLK) results are shown with percentages of samples selection in the complete dataset. All these experiments are conducted with fixed 8,000 epochs for each selection set of samples. It can be seen that ~5% test loss is achieved with < 3% of entire dataset in the *With transfer learning across samples* (TLS - No TLK). *With transfer learning across kernels and samples* (TLS - TLK) has provided similar accuracy as earlier. Moreover, it enables a multiple-kernel learning model for easy training of multiple kernels and extension to new kernels. Because this method requires ~3% of a dataset, the overall simulation time of obtaining comparable modeling accuracy for the selected set of samples is reduced drastically by 32-80×. Because the k-means model has to be trained from scratch, its reported testing loss when the training sample size is small, which is expected.

To be complete, we also tabulated the accuracy of the baseline models. Overall, the accuracy of the baseline models is inferior to the accuracy of models trained by transfer learning. We also observe the large loss on MatMul workload, as we have discussed earlier.

5 CONCLUSION AND FUTURE WORK

In this paper, we have presented Deffe, a general framework for workload performance prediction in domain-specific computing. The modeling component of the framework is built on a CNN and the transfer learning technique. The modeling component is supported by the infrastructure component, which is based on a

Table 3: Testing accuracy comparison of transfer learning models with Random Forest. Overall CNN models from transfer learning have superior accuracy than RF. All numbers are percentages.

Samples Count	Average Total Simulation Time (hrs) (1 host)	Baseline – Random Forest Regression Test Loss (%)				TLS & No TLK - Test Loss (%) Epochs-8000 / Sample				TLS & TLK - Test Loss (%) Epochs-8000 / Sample		
		K-means	Matmul	NW	Backprop	K-means	Matmul	NW	Backprop	Matmul	NW	Backprop
560	45	8.7	33.3	9.7	12.1	95.8	82.1	6.1	32.7	4.32	3.5	8.2
880	71	7.1	33.1	6.7	11.3	64.7	9.4	3.6	9.7	3.0	2.2	6.2
1200	96	6.8	33.5	6.5	11.5	8.1	6.5	3.3	8.1	2.8	1.6	4.4
1520	121	6.8	34.5	6.4	11.2	6.7	5.4	2.4	6.7	2.4	1.5	4.0
1840	147	6.7	34.0	6.3	10.9	6.0	4.0	2.2	4.7	2.3	1.3	3.5
2160	172	6.2	32.1	6.4	10.8	5.4	4.0	2.1	4.2	2.2	1.2	3.2
2480	198	6.3	32.5	6.4	11.1	5.4	3.5	1.8	4.1	2.1	1.1	3.1
2800	224	6.2	35.3	6.4	10.8	5.3	3.4	1.7	3.8	2.0	1.1	3.0
3120	250	6.0	33.0	6.4	10.8	5.3	3.1	1.7	3.5	2.0	1.0	2.9
3440	275	6.0	33.7	6.4	10.8	5.3	3.1	1.7	3.3	1.9	1.0	2.8
3760	300	6.0	32.8	6.4	10.8	5.3	3.0	1.6	3.2	1.9	1.0	2.7

scalable and easy-to-use evaluation software environment. Extensive experimental results show that the proposed model can achieve less than 5% testing loss with less than 3% of the training samples. The drastic reduction in the need for training samples is equivalent to reducing the sampling evaluation by 32-80×. The proposed framework can be readily deployed for other applications.

For future work, we plan to further improve Deffe before releasing it for public use. We also plan to use the framework for other architecture and system exploration research.

ACKNOWLEDGMENTS

This manuscript has been co-authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under contract number DE-AC05-00OR22725. This research was supported in part by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. Additionally this research was supported by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the US Department of Energy.

REFERENCES

[1] Brian M Adams. [n.d.]. DAKOTA, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: version 5.0 user's manual. ([n. d.]).

[2] Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A. Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. 2016. *The Rocket Chip Generator.* Technical Report UCB/EECS-2016-17. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[3] Jonathan Bachrach, Huy Vo, Brian C. Richards, Yunsup Lee, Andrew Waterman, Rimas Avizienis, John Wawrzynek, and Krste Asanovic. 2012. Chisel: constructing hardware in a Scala embedded language. In *The 49th Annual Design Automation Conference 2012, DAC '12, San Francisco, CA, USA, June 3-7, 2012*, Patrick Groeneveld, Donatella Sciuto, and Soha Hassoun (Eds.). ACM, 1216–1225. https://doi.org/10.1145/2228360.2228584

[4] Prasanna Balaprakash, Darius Buntinas, Anthony Chan, Apala Guha, Rinku Gupta, Sri Hari Krishna Narayanan, Andrew A Chien, Paul Hovland, and Boyana Norris. 2013. Exascale workload characterization and architecture implications. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 120–121.

[5] Ioana Baldini, Stephen J Fink, and Erik Altman. 2014. Predicting gpu performance from cpu runs using machine learning. In *2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing*. IEEE, 254–261.

[6] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, and et al. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 1–7.

[7] Ramazan Bitirgen, Engin Ipek, and Jose F Martinez. 2008. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *2008 41st IEEE/ACM International Symposium on Microarchitecture*. IEEE, 318–329.

[8] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[9] Maria Calzarossa and Giuseppe Serazzi. 1993. Workload characterization: A survey. *Proc. IEEE* 81, 8 (1993), 1136–1150.

[10] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. Lee, and K. Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*. 44–54.

[11] Harris Drucker, Christopher JC Burges, Linda Kaufman, Alex J Smola, and Vladimir Vapnik. 1997. Support vector regression machines. In *Advances in neural information processing systems*. 155–161.

[12] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. 2004. Least angle regression. *Annals of Statistics* 32 (2004), 407–499.

[13] John L Hennessy and David A Patterson. 2019. A new golden age for computer architecture. *Commun. ACM* 62, 2 (2019), 48–60.

[14] Engin Ipek, Sally A McKee, Karan Singh, Rich Caruana, Bronis R de Supinski, and Martin Schulz. 2008. Efficient architectural design space exploration via predictive modeling. *ACM Transactions on Architecture and Code Optimization (TACO)* 4, 4 (2008), 1–34.

[15] Zhen Jia, Lei Wang, Jianfeng Zhan, Lixin Zhang, and Chunjie Luo. 2013. Characterizing data analysis workloads in data centers. In *2013 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 66–76.

[16] Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee, Nathan Pemberton, Emmanuel Amaro, Colin Schmidt, Aditya Chopra, et al. 2018. FireSim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 29–42.

[17] Jiangtian Li, Xiaosong Ma, Karan Singh, Martin Schulz, Bronis R de Supinski, and Sally A McKee. 2009. Machine learning based online performance prediction for runtime parallelization and task scheduling. In *2009 IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE, 89–100.

[18] Yunfan Li, D Penney, Abhishek Ramamurthy, and Lizhong Chen. 2019. Characterizing On-Chip Traffic Patterns in General-Purpose GPUs: A Deep Learning Approach. In *International Conference on Computer Design (ICCD)*.

[19] Andy Liaw, Matthew Wiener, et al. 2002. Classification and regression by randomForest. *R news* 2, 3 (2002), 18–22.

[20] Maria Malik, Setareh Rafatirad, and Houman Homayoun. 2018. System and architecture level characterization of big data applications on big and little core server architectures. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)* 3, 3 (2018), 1–32.

[21] Andréa Matsunaga and José AB Fortes. 2010. On the use of machine learning to predict the time and resources consumed by applications. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE, 495–504.

[22] Sinno Jialin Pan and Qiang Yang. 2009. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2009), 1345–1359.

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035.

[24] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.

[25] Alec Roelke and Mircea R Stan. 2017. Risc5: Implementing the RISC-V ISA in gem5. In *First Workshop on Computer Architecture Research with RISC-V (CARRV)*.

[26] Pramod Singh and Avinash Manure. 2020. Introduction to TensorFlow 2.0. In *Learn TensorFlow 2.0*. Springer, 1–24.

[27] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* 15, 1 (2014), 1929–1958.

[28] Silvio Stanzani, Raphael Cóbe, Jefferson Fialho, Rogério Iope, Marco Gomes, Artur Baruchi, and Júlio Amaral. 2018. Towards a Strategy for Performance Prediction on Heterogeneous Architectures. In *International Conference on Vector and Parallel Processing*. Springer, 247–253.

[29] Ruben Vazquez, Ann Gordon-Ross, and Greg Stitt. 2019. Machine Learning-based Prediction for Dynamic, Runtime Architectural Optimizations of Embedded Systems. In *2019 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*. IEEE, 1–7.

[30] Andrew Waterman. 2016. *Design of the RISC-V Instruction Set Architecture*. Ph.D. Dissertation. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-1.html

[31] Andy B Yoo, Morris A Jette, and Mark Grondona. 2003. Slurm: Simple linux utility for resource management. In *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 44–60.

[32] Feng Zhang, Jidong Zhai, Bingsheng He, Shuhao Zhang, and Wenguang Chen. 2016. Understanding co-running behaviors on integrated CPU/GPU architectures. *IEEE Transactions on Parallel and Distributed Systems* 28, 3 (2016), 905–918.

A ARTIFACT DESCRIPTION

A.1 Abstract

This artifact contains all components of Deffe presented in paper titled Deffe: A Data-Efficient Framework for Performance Characterization in Domain-Specific Computing, by F. Liu, N.R. Miniskar, D. Chakraborty and J.S. Vetter.

A.2 Description

A.2.1 Check-list (artifact meta information).

- **Algorithm:** machine learning; convolutional neural network
- **Program:** Python and Bash
- **Dependencies:** Python, Keras, Tensorflow, Scikit, GEM5
- **Run-time environment:** Linunx such as Ubuntu 18.04.4 LTS
- **Hardware:** x86_64, may require GPU and SLURM
- **Output:** Workload execution times along with prediction accuracy
- **Experiment workflow:** Untar the package, run the applications and check the outputs.
- **Publicly availability:** Yes.

A.2.2 How software can be obtained (if available). The source code and some pretrained models of Deffe can be found at the following DOI:

`https://DOI.org/10.5281/zenodo.3749419`

under a permissive UT-Battelle license.

A.2.3 Hardware dependencies. Although the program can run on a single x86_64 CPU, the runtime can be very long. To generate training data using GEM5, a cluster of x86_64 nodes managed by SLURM are recommended. It is recommended to train machine learning models on GPU's.

A.2.4 Software dependencies. SLURM environment is recommended to manage the x86_64 cluster to generate training datasets. GEM5 is required to compute estimated CPU cycles for each kernel in workload benchmark. Keras + Tensorflow or PyTorch are required to inference machine learning model, using pretrained models. To train the machine learning models, Keras + Tensorflow, or pyTorch are required. To use GPU for training, Tensorflow-gpu is required.

A.2.5 Datasets. Pretrained machine learning models are available in the packages. For example,

`example/experiments/transfer_learning_samples/exp/kmeans.hdf5`

contains pretrained ML model. Other pretrained models available in subdirectories under

`example/experiments`

A.3 Installation

(1) Download the Deffe software from the given DOI
(2) Untar the software and set the environment variable.

```
$ cd $HOME
$ tar -xzvf $DOWNLOAD_DIRECTORY/deffe.tar.gz
$ cd deffe
$ source setup.source
```

A.4 Experiment workflow

The Deffe package contains a README file, with many details on how to use Deffe to model workload performances. It also contains the steps on how to train ab initio machine learning models using Deffe , as well as how to extend the functionality of Deffe.

The top-level program has a built-in help function:

```
$ cd example;
$ python3 ../framework/run_deffe.py -h
$ cd ..;
```

For example, to use transfer learning from a model trained using Kmeans kernel for kernel "Matmul", one can use the following command:

```
$ source setup.source
$ cd example/experiments;
$ cd transfer_learning_samples_across_kernels;
$ cd log/matmul
$ python3 $DEFFE_DIR/framework/run_deffe.py \
 -config $DEFFE_DIR/example/config_matmul_tl_samples.json \
  -icp ../../kmeans.hdf5 \
  -only-preloaded-data-exploration \
  -epochs 1000 \
  -batch-size 256 \
  -train-test-split 1.0 \
  -validation-split 0.23
```

After training, the following command can be used to generate the testing accuracy statistics:

```
$ python3 $DEFFE_DIR/framework/run_deffe.py \
    -model-extract-dir checkpoints \
    -config $DEFFE_DIR/example/config_matmul.json  \
    -only-preloaded-data-exploration \
    -train-test-split 1.0 \
    -validation-split 0.23 \
    -load-train-test \
    -loss custom_mean_abs_exp_loss \
    -model-stats-output test-output-exploss.csv
```

or

```
$ python3 $DEFFE_DIR/framework/run_deffe.py \
  -model-extract-dir checkpoints \
  -config $DEFFE_DIR/example/config_matmul.json \
  -only-preloaded-data-exploration \
  -train-test-split 1.0
  -validation-split 0.23 \
  -load-train-test \
  -loss custom_mean_abs_log_loss\
  -model-stats-output test-output-logloss.csv
```

A.5 Evaluation and expected result

The evaluation results can be found in the output log file, in the format of ".csv" file. A snippet of the output file is shown below:

```
Epoch, TrainLoss, ValLoss, TestLoss, Step, TrainCount, ValCount
12, 0.0079, 0.004, 0.0036964816898546666, 2, 880, 202
896, 0.0036, 0.0017, 0.001683124783852085, 7, 2480, 570
845, 0.0042, 0.0018, 0.0017624508281644177, 5, 1840, 423
......
```