Design Space Exploration of Memory Controller Placement in Throughput Processors with Deep Learning

Ting-Ru Lin[1], Yunfan Li[2], Massoud Pedram[1], Fellow, IEEE, and Lizhong Chen[2], Senior Member, IEEE

Abstract—As throughput-oriented processors incur a significant number of data accesses, the placement of memory controllers (MCs) has a critical impact on overall performance. However, due to the lack of a systematic way to explore the huge design space of MC placements, only a few ad-hoc placements have been proposed, leaving much of the opportunity unexploited. In this paper, we present a novel deep-learning based framework that explores this opportunity intelligently and automatically. The proposed framework employs a genetic algorithm to efficiently guide exploration through the large design space while utilizing deep learning methods to provide fast performance prediction of design points instead of relying on slow full system simulations. Evaluation shows that, the proposed deep learning models achieves a speedup of 282X for the search process, and the MC placement found by our framework improves the average performance (IPC) of 18 benchmarks by 19.3% over the best-known placement found by human intuition.

Index Terms—Interconnection networks, memory controllers, deep learning, design space exploration

1 Introduction

The unprecedented success of deep learning has spurred researchers to revisit conventional approaches to tackle difficult problems in various application domains. A challenge recently conquered by deep learning is mastering the game of Go that requires searching $\sim 250^{150}$ possible sequences of move solutions [1], [2]. Such superior capability in searching through large solution spaces has great potential in optimizing computer architecture [3], but significant research is needed to make the idea tangible in specific architectural problems. In this work, we take the initiative to explore innovative application of deep learning in optimizing memory controller (MC) placements in throughput-oriented processors.

While the MC placement for many-core CPUs has been well studied, only very limited work has been conducted to investigate the impact of the MC placement for throughput processors (e.g., GPGPUs, many-core accelerators, etc.) [7]. Fig. 1 depicts a typical throughput processor, where computing core (CC) nodes and memory controller (MC) nodes are connected by a network-on-chip (NoC) through on-chip routers and network interfaces (NIs). A CC node may further include a cluster of small processing elements, and a MC node includes a shared L2 cache bank and a memory controller that interfaces with off-chip DRAM. Unlike CPUs, the CC nodes in throughput processors rarely communicate among themselves. Instead, most of the on-chip traffic is directly between the CC nodes and MC nodes. Due to the quantity difference between the two types of nodes, a unique many-to-few-to-many (M2F2M) traffic pattern is identified [6]. As all the off-chip data is supplied through the MC nodes, the placement of memory controllers is critical to application performance.

A recent work [7] shows that, a MC placement optimized specifically for M2F2M in throughput processors can outperform the best MC placements proposed for CPUs (e.g., edge, diamond, top-bottom) [4] by a large margin. This is achieved by using a bottom MC placement together with a technique called virtual channel (VC) monopolization. In this technique, if a physical link carries only one type of traffic (i.e., request or reply), all the VCs in the link are allocated to that type instead of strictly reserving VCs for both traffic types. While that work demonstrates promising results in optimizing MC placements for throughput processors, the developed placement as well as all the previously proposed MC placements in other works represent merely a few design points in the huge solution space of MC placements. Thus, much needed is a fundamentally different approach that can explore the entire space automatically, rather than proposing a few ad-hoc placements based on human intuition.

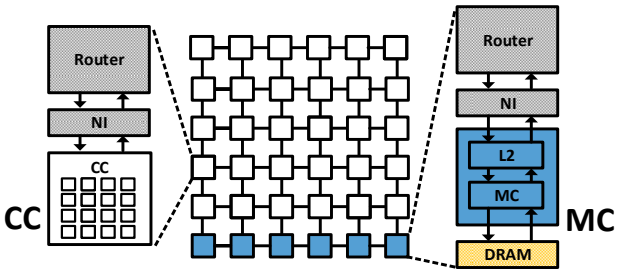

Fig. 1: A typical tile-based throughput processor.

To address this need, we propose a deep-learning based framework that overcomes two major challenges that have hindered such exploration before. The first one is the enormous solution space. For instance, placing 6 MCs and 30 CCs in a 6x6 chip has nearly 2 million (1,947,792) combinations. This number grows super-linearly as the numbers of MCs and CCs increase. The second challenge is the time-consuming performance evaluation of each design point. To assess the "goodness" of a specific MC placement, cycle-accurate simulations over a set of benchmarks are conventionally needed, which takes days if not weeks. This considerably limits the total number of design points that can be assessed. The proposed framework employs a heuristic search algorithm to efficiently guide exploration through the large design space while utilizing deep learning to provide fast performance prediction of design points. Evaluation shows that, the MC placement found by the proposed

[1]*The authors are with University of Southern California, Los Angeles, CA 90007. Email: {tingruli, pedram}@usc.edu*
[2]*The authors are with Oregon State University, Corvallis, OR 97331. Email: {liyunf, chenliz}@oregonstate.edu*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2019.2905587, IEEE Computer Architecture Letters

framework is 19.3% better than the best-known placement in terms of system performance (IPC). Furthermore, the proposed deep learning model achieves 282X speedup for the search process. To our knowledge, this is the first work that intelligently and automatically explores the MC placement design space for throughput-oriented processors.

2 Proposed Approach

2.1 Overview

The proposed deep-learning based framework is depicted in Fig 2. Design space search is guided by the SGA (Simple Genetic Algorithm) in the dotted blue box. Each design point is a MC placement, which is encoded as a gene sequence, e.g., a 36-bit vector for a 6x6 mesh with "1"s and "0"s indicating the positions of MCs and CCs, respectively. A set of random genes are fed into the SGA as the first generation of MC placements. The SGA mimics natural evolution by *evaluating* a generation of genes, *selecting* the good ones, and performing *crossover* and *mutation* to obtain the next generation of genes. The process can be repeated a number of times to explore better genes. The final genes (i.e., MC placements) produced by the SGA are validated by running benchmarks on a cycle-accurate, full system simulator (GPGPU-Sim) to get the corresponding application performance under those MC placements. As a critical step in the exploration, a deep neural network (DNN) model is used to evaluate the fitness of the genes (i.e., the performance of the employed MC placements), as shown in the dotted red box. This avoids using slow full system simulations to evaluate every design point in the large space. The remainder of this section explains why the DNN and the SGA are selected for this task as opposed to choosing other algorithms/models, and the detailed implementation and optimization of the proposed framework.

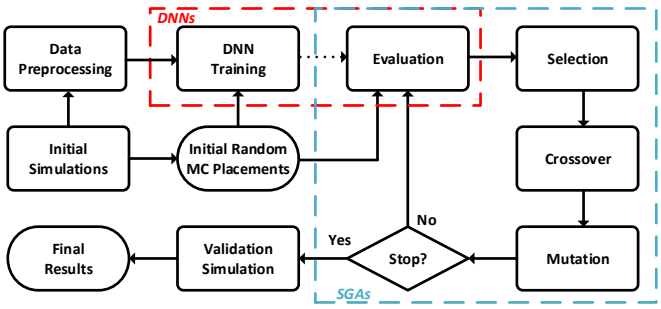

Fig. 2: Proposed deep-learning and SGA based framework.

2.2 DNN for Evaluating Design Points

Need for Non-linear Prediction: We start with assessing whether application performance can be predicted straightforwardly from the MC placement using simple linear models. To verify that, 35 random placements are selected and simulated. We perform traffic flow analysis that examines the load on each physical channel when the flows of packets between all pairs of MC and CC are superposed. We use the average instruction-per-cycle (IPC) of 18 benchmarks as the application performance. The correlation coefficient is -0.10 between the maximum channel load and application performance, and -0.33 between average hop count and application performance. The low degree of correlation coefficients prompts us to adopt deep learning methods for non-linear prediction without the need for explicit equations or strong assumptions.

Proposed DNN Model: The key idea of the proposed framework is to use deep learning for expedited performance comparison between genes. This is made possible by the observation that precise full system performance of different MC placements is not needed. Rather, the performance prediction only needs to be precise enough to tell the *relative* fitness of the two genes under comparison.

To select a specific DNN model, we notice that the task of relative MC locations analysis for system performance is very similar to that of image spatial analysis for image segmentation. Thus, convolutional neural networks (CNNs), which are most suitable for handling images, should take priority over other deep learning methods. Fig 3 shows the architecture of the proposed deep neural network model, which is based on Autoencoder for its robustness to redundant features and noises. It takes a MC placement and several performance indicating features to predict the full system performance (IPC). Specifically, the input of Encoder 1 is a boolean matrix of a MC placement that is converted directly from the gene vector. To increase prediction accuracy, Encoder 2 is added to take four statistical features, which are obtained by performing traffic flow analysis under each MC placement to reflect its impact on the NoC. The features are: 1) the maximal channel load, 2) whether the channel in 1) can be monopolized, 3) average hop count, and 4) the total number of monopolizable VCs. The first and second statistical features reflect NoC throughput, the third indicates NoC latency, and the fourth shows the potential for throughput increase. Additionally, the input also includes benchmark and kernel identification. Feature Fuser [8] is then used to combine outputs from two encoders into one input of the Decoder, thereby forming a complete Autoencoder. Fig 3 caption provides the details of each component and layer.

Data Preprocessing and DNN Training: To generate the training data for the proposed DNN model, we have conducted full system simulations of 18 benchmarks, repeated with 35 random MC placements, and collected the IPC values at the per kernel basis. To prevent training data bias caused by some benchmarks having much greater numbers of kernels than others, the data from up to 10 kernels of each benchmark is used. To increase sensitivity of the DNN model for each benchmarks, the selected kernels have the largest IPC difference within a benchmark. For efficient convergence, the IPC of placement i given benchmark b and kernel k is normalized between $(-1, 1)$ by calculating

$$\mathrm{IPC}^{norm}_{b,k,i} = \frac{\mathrm{IPC}_{b,k,i} - (\mathrm{IPC}_{b,k,max} + \mathrm{IPC}_{b,k,min})/2}{(\mathrm{IPC}_{b,k,max} - \mathrm{IPC}_{b,k,min})/2} \quad (1)$$

where

$$\mathrm{IPC}_{b,k,max} = \max_i \mathrm{IPC}_{b,k,i} \quad (2)$$

$$\mathrm{IPC}_{b,k,min} = \min_i \mathrm{IPC}_{b,k,i} \quad (3)$$

With 35 placements and 102 kernels, there are 3,570 normalized simulation data. The batch size for training is 16. The popular Adam optimization is adopted for training:

$$\theta_{t+1} = \theta_t - \eta * f(\Delta\theta_t, \Delta\theta_{t-1}) \quad (4)$$

where θ_t and $\Delta\theta_t$ are the parameters of a DNN model and the gradient of the parameters at the t-th iteration respectively. η (=0.001) is the learning rate [8]. Following the 5-fold cross-validation to avoid overfitting [9], we partition the training data into 5 portions, and use 4 portions for

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2019.2905587, IEEE Computer Architecture Letters

training and 1 portion for validation in a rotating fashion for 5 times. One trained DNN from each fold is selected to form an ensemble of 5 DNNs as a predicative model for performance evaluation.

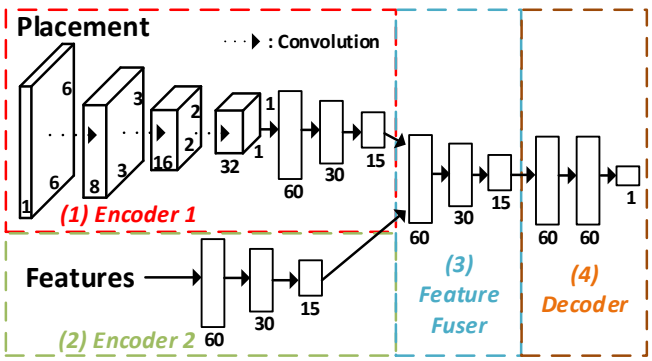

Fig. 3: The proposed DNN model.(1) Encoder 1: there are three convolutional layers, with the kernel size of 3x3x1, 3x3x8 and 2x2x16, respectively. In each convolutional layer, the batch normalization after the convolution normalizes the value distribution and is followed by the max pooling operation of size 2x2x1 without overlapping. After three convolutional layers, a tensor of size 1x1x32 is flattened to a 32x1 vector and used as the input of the subsequent fully connected layer. The number of neurons of fully connected layers is 60-30-15. (2) Encoder 2: the number of neurons after input features is 60-30-15. (3) Feature Fuser: the outputs of both encoders are first concatenated to be a single input of the Feature Fuser. The number of neurons is 60-30-15. (4) Decoder: there are 60-60-1 neurons for generating an output value. The activation function for all the convolutional layers and fully connected layers is the leaky ReLU. The dropout probability is 0.5 for fully connected layers in the Encoder 1, Encoder 2 and Feature Fuser, and 0.1 for the Decoder stage.

2.3 SGA for Searching Design Space

Why SGA: It is crucial to select an efficient searching method that matches the nature of exploring the enormous design space of MC placements. To achieve that, we have calculated the differences ($\geq$0) in statistical features (introduced in the previous subsection using traffic flow analysis) of 1000 placements after a horizontal or vertical shift of one MC location in each placement. The median and standard deviation (std) of the statistical differences of the first, third and fourth statistical feature are 0 (std:11.19), 0.056 (std:0.04) and 2.0 (std:2.52), respectively. The small statistical differences indicate that a MC placement generated from the combination of similar MC placements may exhibit similar traffic characteristics and NoC performance. This observation inspires us to consider genetic algorithms in which offspring generations inherit the characteristics of parent generations for gradual improvement over generations.

SGA Implementation: As shown in Fig 2, the SGA explores the design space by iterating over four phases: evaluation, selection, crossover, and mutation. (1) Evaluation: each generation has 35 genes. To evaluate them, instead of running full system simulations, the proposed DNN model is used as the fitness function to estimate system performance given the MC placement, four statistical features, and benchmark and kernel info. (2) Selection: Based on the evaluation, the MC placements are ranked by their average performance of all the benchmarks, and the top 7 are selected to breed the next generation. Here, the performance of a benchmark is the average performance of each kernel, which in turn is defined relatively as the ratio of its IPC to the IPC of the bottom MC placement (state-of-the-art) [7]. (3) Crossover: A new set of 35 MC placements are generated. In every new placement, the position of each MC randomly comes from one of the parents. (4) Mutation: a MC position may be shifted by one in either the horizontal or the vertical direction with a small probability. This probability decreases exponentially with generations. There is no repetition of 35 MC placements between generations.

3 Results and Analysis

Model Validation: We evaluate 18 workloads from the Rodinia [10] and NVIDIA SDK benchmarks. Simulations are conducted on GPGPU-Sim, a cycle-accurate simulator for throughput processors, augmented with VC monopolization. The key configuration is shown in TABLE 1. During the 5-fold cross-validation, we observe that the training and testing loss curves for different folds are closed to each other, and the loss value converges to less than 0.2 consistently after 3,000 iterations. This loss value is reasonable for 35 MC placements in a generation and is sufficient for our framework. The consistent loss value and the low actual prediction error (12% for the top 5 found placements below) confirm that there is no overfitting. To validate the feasibility of using the DNN ensemble, we conduct full system simulations of some of the top and bottom ranked MC placements at the 10th, 20th, 30th, 40th and 50th generations during the search. The relative comparison of the simulated performance between two placements in each generation are consistent with the predicted performance from the DNN model. These results justify the comparison functionality of the DNN for unseen MC placements.

TABLE 1: Configuration for full system simulation.

System Parameters	Details
Computing Core	30 Cores, 1400 MHz
Memory Controller	6 MCs, 924 MHz
Memory Bandwidth	168 GB/s (28 GB/s x 6)
Interconnection	6x6 2D Mesh, 1400 MHz
Routing	DOR with VC monopolization
Virtual Channel	2 VCs/port, 4 flits/VC
Shared Memory	48 KB
L1 Inst. Cache	2 KB (4 sets/4 ways LRU)
L1 Data Cache	16 KB (32 sets/4 ways LRU)
L2 Cache	64 KB per MC (8-way LRU)

MC Placement Results: The design space search starts with a set of random MC placements and gradually introduces new and better placements. The search is terminated after 300 iterations after the results have been stabilized. In Fig. 4(2)-(6), we visualize the top 5 ranked MC placements from the final generation. For easier comparison, Fig. 4(1) shows the bottom MC placement with VC monopolization (the best-known scheme for GPUs [7] due to much higher VC utilization), along with the max channel load (the lower the better), average hop count, and the number of monopolizable VCs. The actual performance of these MC placements from full system simulations is plotted in Fig. 5. Note that the light bars are the average performance of the top 5 placements found by the proposed framework. These placements, which are found by the deep-learning assisted approach, achieve a significant performance improvement of 18.8% (on average) over the state-of-the-art scheme that is found by human observation. Additionally, it is worth mentioning that, the top ranked MC placement in Fig. 4(2)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2019.2905587, IEEE Computer Architecture Letters

achieves 19.3% average IPC improvement. The close performance among the top ranked MC placements demonstrates the effectiveness of our framework in finding multiple good solutions with limited training data.

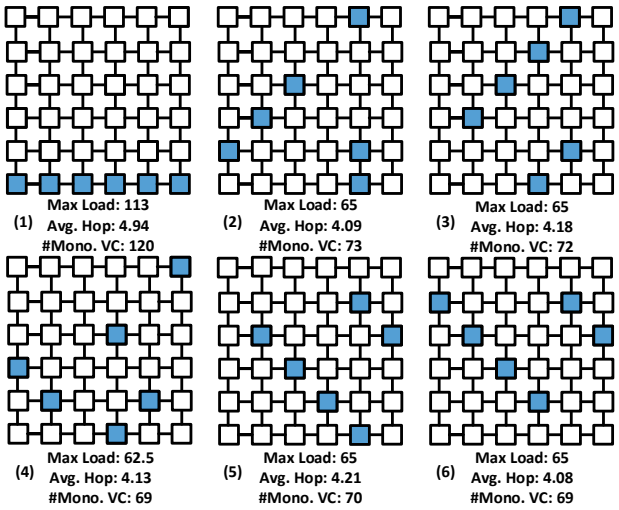

Fig. 4: Comparison of MC placements. (1): State-of-the-art (bottom+VC mono); (2)-(6): from the proposed framework.

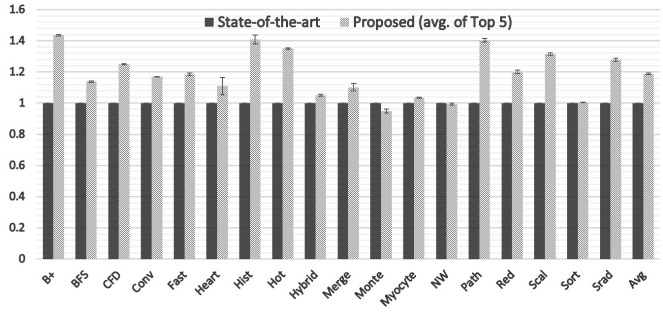

Fig. 5: Performance comparison (IPC) of MC placements.

Further Analysis: Several interesting things can be noticed upon a closer look at the found MC placements, which may provide insights on the attributes of good placements. First, in all the MC placements in Fig. 4(2)-(6), the memory controllers are sparsely distributed. This is consistent with the objective of minimizing the max channel load which directly determines NoC throughput. Second, it can be seen that MCs are often aligned diagonally. This reduces the superposition of heavy traffic flows from the MCs if they are aligned horizontally or vertically. Third, the diagonal lines of MCs are also in parallel, e.g., in (3)(5)(6). Our traffic flow analysis shows that, with the parallel alignment added on top of diagonal positions, there is a larger potential for VCs to be monopolized, particularly for the most congested channels that are the bottleneck of the NoC. The proposed deep-learning based approach is able to balance these potentially contradictory attributes comprehensively and automatically. This can be very useful as more techniques are being proposed to increase optimization dimensions.

Speedup: To search from the initial random MC placements to the final solutions through 300 SGA generations, the proposed framework only needs to evaluate 10,500 MC placements out of 1,947,792 candidates, which is equivalent to a 185X reduction. On top of that, for the DNN model, it takes 280 hours to collect simulation/training data on a linux workstation with 2.67GHz Xeon(R) CPU E7-8873, 2 hours for the training and 5-fold cross-validation, and 15 hours for evaluating 10,500 placements, totaling 297 hours (or 12.37 days). It would have taken 84,000 hours (or 3,500 days) if the 10,500 placements were evaluated in the conventional full system simulations on the same machine. Thus, the speedup from adopting DNN alone is 282X, which is quite significant.

Comparison with Other Works: In the first work that explores MC placement for CPUs [4], several MC placements (e.g., edge, diamond, top-bottom) are proposed by observational methods and a genetic algorithm is employed to verify that diamond performs the best for CPUs. However, the fitness function in that work considers only the max channel load while ignoring monopolizable VCs and hop count. Consequently, the best MC placement found by our approach achieves 36.4% higher IPC than edge, 27.8% higher than diamond, 20.3% higher than top-bottom, and 19.3% higher than bottom+mono in throughput processors. Interestingly, the best placement for CPUs (diamond) performs worse than top-bottom in throughput processors.

4 Conclusion

This work explores the use of deep learning in optimizing MC placements in throughput processors. We propose a framework that integrates deep learning with a genetic algorithm to address the challenges of enormous design space and slow evaluation of design points. The proposed framework is able to find a MC placement that improves system performance by 19.3% over the best-known placement for throughput processors, while offering insights on good MC placements. Moreover, the proposed DNN model achieves 282X speedup for the search process by providing expedited performance estimation. The substantial benefits from this work exemplify the viability of utilizing deep learning methods in improving computer architecture.

Acknowledgment

We appreciate Shao-Hua Sun's assistance in DNN development. This research is supported, in part, by the National Science Foundation grants #1566637, #1619456, #1619472 and #1750047, and Software and Hardware Foundations.

References

[1] D. Silver, A. Huang, C. J. Maddison, and et al., "Mastering the game of Go with deep neural networks and tree search," in *Nature*, 2016.
[2] D. Silver, J. Schrittwieser, K. Simonyan, and et al., "Mastering the game of Go without human knowledge," in *Nature*, Oct., 2017.
[3] T.S. Karkhanis, J.E. Smith, "Automated design of application specific superscalar processors: an analytical approach," in *ISCA*, 2007.
[4] D. Abts, N.D. Enright Jerger, J. Kim, "Achieving predictable performance through better memory controller placement in many-core CMPs," in *ISCA*, 2009.
[5] O. Kayiran, A. Jog, M. T. Kandemir, and C. R. Das, "Neither More Nor Less: Optimization Thread-level Parallelism for GPGPUs," in *PACT*, Sept., 2013.
[6] A. Bakhoda, J. Kim, and T. M. Aamodt, "Throughput-Effective On-Chip Networks for Manycore Accelerators," in *MICRO*, Dec., 2010.
[7] H. Jang, J. Kim, P. Gratz, K. H. Yum, and E. J. Kim, "Bandwidth-Efficient On-Chip Interconnect Designs for GPGPUs," in *DAC*, June, 2015.
[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.
[9] E. Ipek, S. A. McKee, B. de Supinski, M. Schulz, and R. Caruana,"Efficiently exploring architectural design spaces via predictive modeling," in *ASPLOS*, Oct., 2006.
[10] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer S.-H. Lee and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *IISWC*, Oct., 2009.