NUMA-Caffe: NUMA-Aware Deep Learning Neural Networks

PROBIR ROY, College of William and Mary
SHUAIWEN LEON SONG, Pacific Northwest National Laboratory and College of William and Mary
SRIRAM KRISHNAMOORTHY and ABHINAV VISHNU, Pacific Northwest National Laboratory
DIPANJAN SENGUPTA, Intel Labs
XU LIU, College of William and Mary

Convolution Neural Networks (CNNs), a special subcategory of Deep Learning Neural Networks (DNNs), have become increasingly popular in industry and academia for their powerful capability in pattern classification, image processing, and speech recognition. Recently, they have been widely adopted in High Performance Computing (HPC) environments for solving complex problems related to modeling, runtime prediction, and big data analysis. Current state-of-the-art designs for DNNs on modern multi- and many-core CPU architectures, such as variants of Caffe, have reported promising performance in speedup and scalability, comparable with the GPU implementations. However, modern CPU architectures employ *Non-Uniform Memory Access* (NUMA) technique to integrate multiple sockets, which incurs unique challenges for designing highly efficient CNN frameworks. Without a careful design, DNN frameworks can easily suffer from long memory latency due to a large number of memory accesses to remote NUMA domains, resulting in poor scalability. To address this challenge, we propose NUMA-aware multi-solver-based CNN design, named *NUMA-Caffe*, for accelerating deep learning neural networks on multi- and many-core CPU architectures. NUMA-Caffe is independent of DNN topology, does not impact network convergence rates, and provides superior scalability to the existing Caffe variants. Through a thorough empirical study on four contemporary NUMA-based multi- and many-core architectures, our experimental results demonstrate that NUMA-Caffe significantly outperforms the state-of-the-art Caffe designs in terms of both throughput and scalability.

CCS Concepts: • **General and reference** → *Performance*; • **Theory of computation** → *Shared memory algorithms*; • **Computer systems organization** → *Neural networks*;

Additional Key Words and Phrases: Deep learning, neural network, NUMA, stochastic gradient descent

ACM Reference format:
Probir Roy, Shuaiwen Leon Song, Sriram Krishnamoorthy, Abhinav Vishnu, Dipanjan Sengupta, and Xu Liu. 2018. NUMA-Caffe: NUMA-Aware Deep Learning Neural Networks. *ACM Trans. Archit. Code Optim.* 15, 2, Article 24 (June 2018), 26 pages.
https://doi.org/10.1145/3199605

This research is supported by National Science Foundation under Grant No. 1618620. This research is also supported by Machine Learning Initiative at the Pacific Northwest National Laboratory. The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under Contract No. DE-AC05-76RL01830.
Authors' addresses: P. Roy and X. Liu, Department of Computer Science, The College of William and Mary, McGlothlin-Street Hall 117, Williamsburg, VA 23185; emails: {proy, xl10}@cs.wm.edu; S. L. Song, S. Krishnamoorthy, and A. Vishnu, High Performance Computing Group, Pacific Northwest National Laboratory (PNNL), Richland, WA 99352; emails: {shuaiwen.song, sriram, abhinav.vishnu}@pnnl.gov; D. Sengupta, Parallel Computing Laboratory, Intel Labs, 2200 Mission College Blvd., Santa Clara, CA 95054-1549; email: dipanjan.sengupta@intel.com.

https://doi.org/10.1145/3199605

1 INTRODUCTION

Deep learning frameworks, such as Caffe [34], TensorFlow [5], and CNTK [62], require a substantial amount of computation resources to keep the model training time reasonable. With the prevalence of powerful multi-core- and many-core-based architectures in modern HPC systems, researchers have attempted to apply HPC resources to accelerate deep neural networks' (DNNs') long training processes. Existing approaches have focused on developing GPU-centric approaches [16, 21, 28, 51] to speed up DNN training due to its massive parallelism and architecture-specific optimization libraries (e.g., cuBLAS and cuDNN [13]). However, there are several limitations hindering GPU scalability as an effective training platform: (i) small resident memory that most real-world data input cannot fit in, (ii) lack of generality in optimizations, (iii) difficulty in porting across architectures, and (iv) strong bias towards certain applications [55]. This makes widely available CPU-based multi- and many-core architectures a strong alternative for training complex neural networks. This effort has been widely supported by both academia and industry, e.g., Microsoft's Project Adam [14] and *spg-CNN* [52]. The main reason to use CPU-based platforms is that they are easily accessible and can achieve several teraflops, perfect for training medium and moderately large networks within a reasonable time. Additionally, recent research on CPU architectures [19, 55] has shown significant training speedups, even at the level of performance as those achieved by GPU optimizations. Thus, tackling CPU-based machine learning challenges is both important and necessary.

However, the state-of-the-art Convolution Neural Network (CNN) designs often lack sophistication in considering complex memory hierarchies of modern multi-core and many-core architectures and their impact on throughput and scalability. For instance, modern multi-core and many-core architectures have employed non-uniform memory access (NUMA) techniques to scale memory bandwidth. A NUMA architecture can scale to thousands of cores, such as SGI UV [53] and HP Integrity Superdome X [24]. A typical NUMA architecture integrates multiple microprocessors, and each has its memory physically attached. Any two microprocessors use a high-speed interconnect (e.g., Intel QuickPath [17] and AMD's HyperTransport [23]) to exchange data. Cores in each microprocessor can access the local memory directly attached to the same microprocessor and also remote memory attached to other microprocessors. To simplify the discussion for the systems in which NUMA effects may exist in both main memory and last-level caches, we refer to cache/memory along with all CPU cores that can access a memory partition with uniform latency as a NUMA node. Local and remote NUMA nodes have significant differences in accessing latency and available bandwidth. Typically, a remote access has lower bandwidth and incurs more latency than a local access. In NUMA architectures, multithreaded programs may suffer from significant performance degradation if they access data in remote memory with high frequency. Since modern supercomputers are commonly built on a large amount of multi-socket-based computing nodes, identifying and resolving scaling issues originating from remote NUMA communication is evident for scaling up deep learning algorithms.

Through a detailed performance scalability study, we find that current multi- and many-core CPU-based CNN designs do not take NUMA effects into consideration, causing significant scalability issues on today's NUMA architectures. Take two variants of a popular CNN framework—Caffe as an example, the original Berkley Vision Lab version [34] and the newly released Intel-Caffe [19], which is highly accelerated by Intel Math Kernel Library 2017 [32] and batch-level parallelization [55]—have largely focused on the layer-level parallelism and memory optimizations inside a NUMA node. However, according to our observations on four contemporary architectures, neither of the two designs scales well across NUMA nodes (Section 3). This is because worker threads spawned for both batch-level and BLAS-level parallelization run on cores in different NUMA

nodes, which incur significant latency due to remote memory access and remote bandwidth contention.

To scale CNN training in NUMA architectures, explicitly co-locating data with its computation is essential for avoiding remote accesses and bandwidth contention. A possible solution is to use processes rather than threads. Under that design, the approach becomes equivalent to distributed memory DL implementations such as S-Caffe [7], FireCaffe [30], Chainer [56], MaTEx [57], PowerAI [15], CNTK [62], PaddlePaddle [4], and Caffe2 [26]. However, this approach has the limitation that it consumes large memory, which is unaffordable for machines with limited memory space such as Intel Knights Landing (KNL). To illustrate the increase of memory usage with the increase of MPI processes, we configure Intel-Caffe at a various number of MPI processes, train Alexnet on Imagenet dataset on 56 core Intel Broadwell machine, and collect memory usage using smem [1]. We have found that the total in-process private memory usage can increase up to 6.3× as the number of process grows from 1 to 56 during strong scaling. On upcoming massively multi-threaded architectures, a process-based solution would be increasingly problematic. Thus, it is necessary to maintain a threaded implementation of CNN training frameworks. However, a CNN usually consists of multiple layers and complex input/output data structures, resulting in different data access patterns. Reorganizing the data layouts at the beginning of each layer is costly and can easily surpass the benefit from minimizing remote accesses.

To address these challenges in the current Caffe designs, we propose *NUMA-Caffe*, a NUMA-aware multi-solver-based design for CNN acceleration on NUMA architectures. It transforms the layer-level data-parallelized Caffe to support NUMA-aware training with multi-level hierarchical parallelism. NUMA-Caffe is independent of DNN topology, impact-free on network convergence, and offers both superior throughput and scalability to the existing designs.

While data-parallel schemes to eliminate scalability bottleneck have been well studied in distributed [5, 22] and multi-GPU [12]-based CNN-training; none of them have studied the NUMA-scalability issue in the CPU-based system. Distributed CNN training on clouds suffers from high communication overhead due to the distributed file system and remote procedure call; comparatively, CPU-based NUMA systems can leverage the fast inter-solver communication through shared memory. On the other hand, due to micro-architectural differences between CPU and GPU, it is necessary to characterize the performance of CNN training on non-accelerated NUMA systems and evaluate the optimization. In summary, this article makes the following contributions:

- We conduct a thorough empirical characterization of NUMA effects on state-of-the-art Caffe designs, and discover the root cause of their scalability issues and its corresponding sources (Section 3).
- With the design challenges in mind, we propose the methodology and design details of *NUMA-Caffe*, a NUMA-aware multi-solver-based design for CNN acceleration on modern CPU-based multi- and many-core architectures (Section 4).
- We evaluate NUMA-Caffe with multiple training datasets and DNN topologies on four mainstream NUMA architectures (i.e., *AMD Opteron, Intel Sandy Bridge, Intel Broadwell, and Intel Knights Landing*). Experimental results demonstrate that our design achieves significant improvement over the state-of-the-art Caffe designs, regarding both overall training throughput and memory scalability (Section 5). Optimization by NUMA-Caffe is orthogonal to other CPU-specific Caffe optimizations (i.e., cache blocking, etc.) and is thus complementary to BVLC-Caffe and Intel-Caffe implementation.

2 BACKGROUND

In this section, we briefly discuss the basics of general *Artificial Neural Networks* (ANNs), CNNs, and their state-of-the-art implementation *Caffe* [34], and the current training/execution for CNNs.

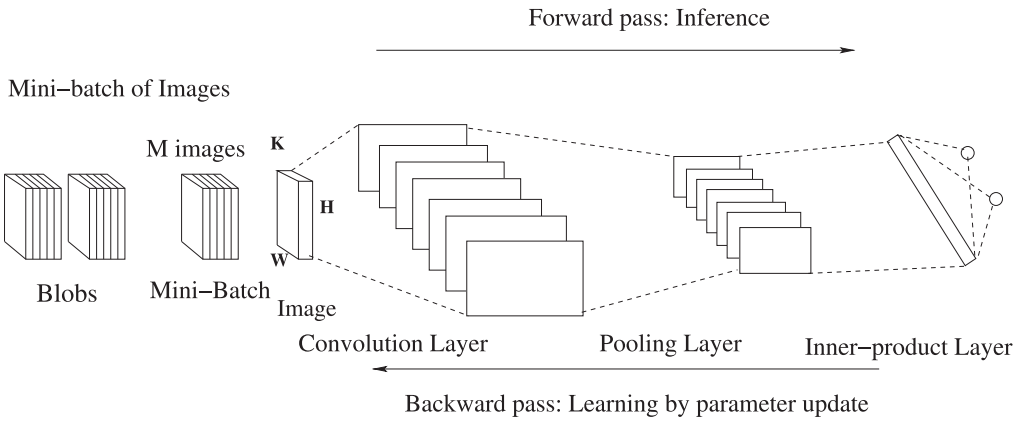

Fig. 1. Data structures and layer computation in Caffe.

2.1 Basics of ANNs

An ANN comprises a series of stacked-up layers, each of which contains a group of *neurons*. A neuron $L_{h+1}[i]$ in each layer applies a non-linear function (Ψ) to a linear combination of the outputs of neurons in the preceding layer (L_h) using a set of weights (W):

$$L_{h+1}[i] = \Psi\left(\sum_j L_h[j] \times W[i,j]\right). \tag{1}$$

The common execution method of ANNs is Stochastic Gradient Descent (or SGD), which conducts first forward propagation [14, 55] to compute the network's output of an input example and then performs backward propagation to calculate the error gradients of the weights. After that, SGD produces delta weights via multiplying error gradients of weights and their corresponding inputs. In other words, delta weights are the updates that SGD applies to the existing weights for model updating. The overall training process of ANNs is to repeat the procedure above for a new input example with the updated model.

2.2 CNN and Caffe

A CNN is a subclass of traditional ANNs where each of its neurons only connects to a small region (or its local surroundings) of the previous layer. Another significant difference is that neurons of CNNs' layers form *feature maps* and all the neurons in a feature map will apply the same feature to their corresponding inputs. One of the most popular and widely used CNN training infrastructures today is Caffe, originally developed by Berkeley Vision and Learning Center (BVLC) [34] for community-based deep learning missions. Caffe simulates DNNs as a network of computing units. These computing units, referred to as "layers," take a dataset as input, perform a specific set of operations, and then pass the output data to the next layer. Figure 1 shows its data structures and layer computation in Caffe. As discussed previously, a set of layers stack one upon another and perform sequential training for a deep network. This process is referred to as "*forward and backward pass.*" Caffe represents all data in the network as "blobs," ranging from a batch of images to model parameters and gradient data generated during passes. Layers communicate and store data through blobs. Blob is implemented as an N-dimensional array and allocated with a contiguous chunk of memory. A blob contains a batch of M images, which will take a continuous space of M × channel K × height H × width W.

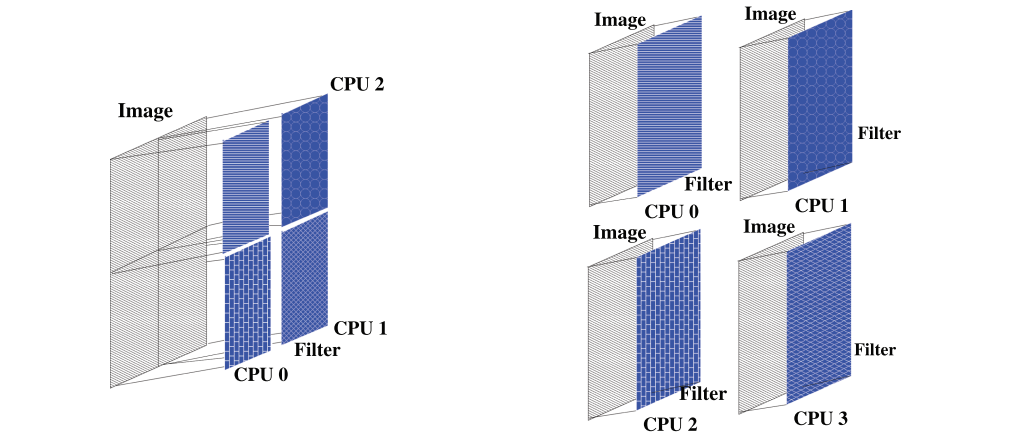

Fig. 2. BLAS-level and batch-level parallelization.

Among the layers shown in Figure 1, the convolution layer is the most significant layer in a CNN training process [34, 55]. Convolution layers extract features from image data by convolving feature vectors on input samples one-by-one using BLAS libraries. Generally, BLAS libraries parallelize this operation by creating multiple worker threads, each of which operates on a small section of that image. This is commonly referred as *BLAS-level parallelism*. A recent study [55] has shown that, due to a small image size, BLAS operations cannot efficiently parallelize convolution layers in Caffe. A *batch-level parallelization* is then proposed by this study: each batch is parallelized using OpenMP pragma at coarse-grained level instead of fine-grained BLAS level. This method significantly improves parallel efficiency of the convolution layers in Caffe. Figure 2 illustrates both BLAS-level and batch-level parallelization. However, due to the sequential nature of the blobs and non-uniformity of memory access, neither BLAS-level nor batch-level parallelization can resolve the issue that parallel threads may cross NUMA boundary when performing memory access. So far, none of the recent state-of-the-art CNN frameworks (e.g., References [14, 19, 34, 52, 55]) have focused on such system aspects of DNN design nor explored the potential NUMA scalability problems when training DNNs. This motivates us to investigate the performance impact of NUMA effects on different Caffe designs, especially when a large number of NUMA nodes are utilized for computation.

2.3 CNN Training and Execution

The current training process for CNN is similar to what was described in Section 2.1. Caffe applies a mini-batch gradient descent algorithm where it updates model parameters after training on a mini-batch of N samples. These mini-batches reduce the divergence of network training by a frequent update of model parameters. From the performance perspective, mini-batch gradient descent helps efficient computation of convolution. For CNN execution, current designs such as Caffe follow a two-step process [52]: Unfolding+Parallel_MM. First, the input activation vector is unfolded into a matrix. Then, a matrix-matrix multiplication is performed, where one matrix contains the layer's weights and the other contains the unfolded activations.

Table 1. Architecture Configurations of the Testbeds under Evaluation

Processor Type	#Nodes	#Cores/ NUMA	L1 data cache	L2	L3	Memory/ NUMA
Sandy Bridge: Intel Xeon[1] E5-4650 2.70GHz	4	8	32KB private	256KB private	20MB shared	64GB
KNL: Intel Xeon Phi 7210 1.30GHz	4	16	32KB private	1024KB private	-	4GB HBM 24GB DDR4
Broadwell: Intel Xeon E7-4830v4 2.00GHz	4	14	32KB private	256KB private	35MB shared	251GB
AMD Opteron Processor 6168 800MHz	8	6	64KB private	512KB private	5MB shared	16GB

3 PERFORMANCE IMPACT OF NUMA ON CNN DESIGNS: OBSERVATIONS AND ANALYSIS

To analyze the NUMA impact on the scalability of current CNN designs, we use popular Caffe designs to showcase the exploration. The designs under consideration for evaluation include the BVLC Caffe [34], the coarse-grain parallelized Caffe based on BVLC [55] (i.e., the batch-level parallelization discussed in Section 2.2), and the newly released Intel-Caffe [19] for optimizing CNN training on newer multi- and many-core architectures, especially benefiting the Intel family platforms (e.g., Broadwell and Intel Knights Landing), using fine-grained architecture-specific or customized tuning (e.g., efficient cache utilization and vector processing extension). Upon further investigation, Intel-Caffe already includes the coarse-grain parallelization and its related optimizations described in Reference [55]. Therefore, we select Intel-Caffe as the state-of-the-art design for our evaluation in this article. Furthermore, we apply the newest and most optimized Intel Kernel Library (Intel MKL 2017) [32] for all the Caffe designs under evaluation.

We conduct the evaluation of memory scalability and NUMA effects on Caffe designs using four contemporary NUMA-based multi- and many-core architectures: Intel Sandy Bridge (four NUMA nodes), AMD Opteron (eight NUMA nodes), Intel Broadwell (four NUMA nodes), and Intel Knights Landing (four NUMA nodes). The Intel KNL is configured to SNC-4 mode and on package high-bandwidth memory (HBM) is set as flat mode. While all the plots are drawn without explicitly using HBM, we report the effect of HBM in the evaluation section. The detailed architecture configurations of all four machines can be found in Table 1. Two training datasets are selected for the evaluation in this section: CIFAR-10 [36] and Imagenet [37]. CIFAR-10 is a CNN performing imaging classification for a computer vision dataset with 10 object classes, each having 6,000 32×32 color images. We apply a Caffe implementation of Alex's cuda-convnet model [37] to train CIFAR-10. The other training set we select is Imagenet, which is a large database of images for visual recognition research. We refer to ILSVRC 2012 dataset, which consists of more than 1 million images belonging to 1,000 categories. Several models have been proposed to train the Imagenet dataset. In this article, we analyze a popular model based on Alexnet [37], consisting of 25 layers:

[1]Intel and Xeon are trademarks of Intel Corporation in the U.S. and/or other countries.

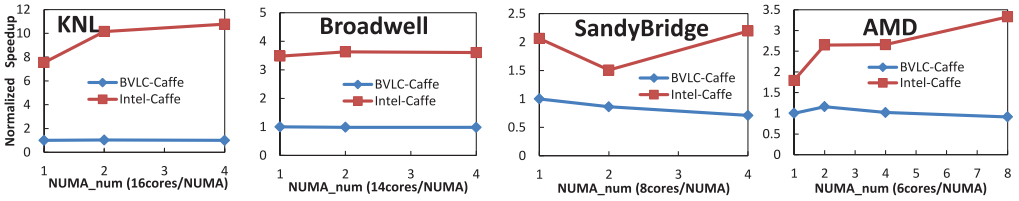

Fig. 3. Normalized throughput of training Alexnet on four mainstream architectures with increasing number of NUMA nodes. Data points are normalized to BVLC-Caffe at one NUMA node.

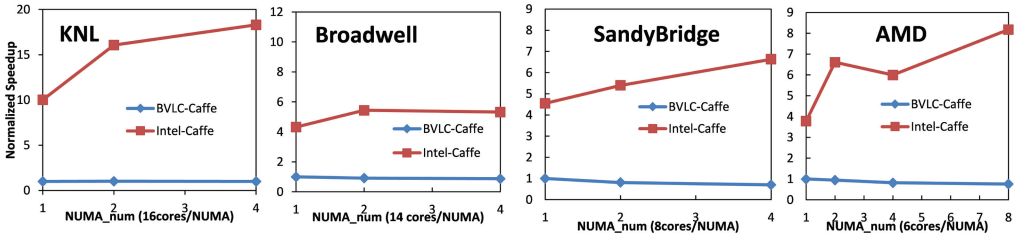

Fig. 4. Normalized throughput of training CIFAR-10 on four mainstream architectures with increasing numbers of NUMA nodes. Data points are normalized to BVLC-Caffe at one NUMA node.

5 convolution and 3 inner-product layers. For a fair comparison, we keep the network models the same for different Caffe designs when training the two datasets. The experiments are run with *strong scaling* by maintaining the batch size constant to 128 and 256 for CIFAR-10 and Alexnet, respectively.

3.1 Observations on Scalability

To observe Caffe's scalability, we measure the throughput (iterations/sec) of the two different Caffe designs (i.e., BVLC-Caffe and Intel-Caffe) with increasing numbers of NUMA nodes on the four architectures listed in Table 1. Figures 3 and 4 demonstrate their performance scalability (in terms of speedup of throughput) for strong scaling scenario across NUMA nodes. We have the following observations:

(1) The BVLC-Caffe performs much worse than Intel-Caffe at every scaling point. This is because BVLC-Caffe only applies BLAS-level parallelism while Intel-Caffe adds another level of parallelism, batch-level parallelism (depicted in Figure 2 and Reference [55]), for further optimizing parallel efficiency and memory access pattern. Although both designs utilize the newest math kernel library (MKL2017), Intel-Caffe enables additional architecture-specific optimizations such as cache blocking. This performance superiority of Intel-Caffe over BVLC-Caffe is reflected in both scalability and single NUMA node performance discrepancy.
(2) Both BVLC-Caffe and Intel-Caffe suffer from bad scalability on all four architectures. BVLC-Caffe either does not scale or scales worse than a single NUMA node scenario. Even with a superior parallelization scheme and additional platform-specific optimizations, Intel-Caffe only scales slightly better than BVLC-Caffe before reaching to two NUMA nodes. However, its scalability drops significantly beyond two NUMA nodes.
(3) Design implementation, training dataset and its network topology, and system architecture all impact Caffe's scalability. For instance, design and training input shape Caffe's

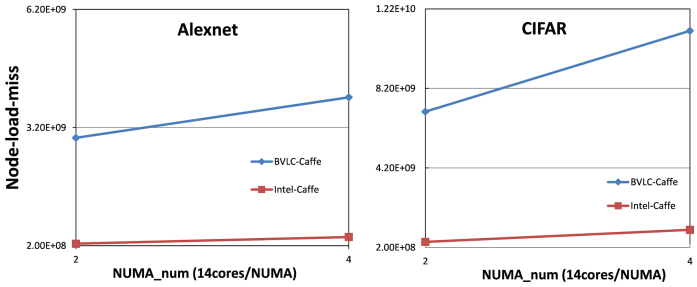

Fig. 5. Node-load-miss when training (a) Alexnet and (b) CIFAR on Intel Broadwell platform.

Table 2. Memory Access Latency Collected from PEBS Sampling for 800 Iterations of CIFAR-10(Unit of Latency is CPU Cycles)

# of NUMA	Avg DRAM latency	Scaling factor	Avg memory latency	Contribution of DRAM latency (%)
1 NUMA	13.6	1.00	69.2	19.66
2 NUMA	20.0	1.47	153.2	13.03
4 NUMA	64.5	4.75	173.7	37.15

memory access pattern and parallelization at each layer. System architecture choice decides specific tuning opportunities and overall systematic constraints such as memory bandwidth and access latency. Due to the variation in CPU capability and memory access pattern, convolution algorithms scale differently with the number of NUMA nodes. If NUMA overhead exceeds computational capability, a drop of throughput will be observed, e.g., for Sandy Bridge and AMD cases in Figure 3.

All the evidence above seems to suggest the state-of-the-art Caffe designs suffer from significant scalability issues when increasing the number of NUMA nodes. Since the experiments are conducted under strong scaling scenario, we focus on investigating memory-level scalability issues. Next, we apply system-level profiling tools to quantify the bottlenecking effects so as to explore the potential root cause of Caffe's scalability issue on multi-core architectures.

3.2 Identifying Root Cause of Caffe's Scalability Bottleneck

According to the observations from Figures 3 and 4, a strong possibility that results in such scalability problems for Caffe designs is NUMA factor. Thus, we first leverage the hardware performance event counters to investigate if remote NUMA accesses are frequent. We use the event *node-load-miss* as the evaluation metric, which counts memory loads that fetch data from cache and DRAM residing in remote NUMA domains. Figure 5(a) and (b) show the *node-load-miss* with the growing NUMA counts on Intel Broadwell platform for both datasets. We can clearly observe that for both cases remote memory access increases. BVLC-Caffe's remote memory access increases even more significantly than Intel-Caffe. This access intensity gap between the two designs is mainly caused by different memory access patterns with or without coarse-grain parallelization. These two figures indicate that current Caffe designs might not have taken NUMA scalability into consideration.

To further quantify the NUMA effects, we apply the Precise Event-Based Sampling (PEBS) [33] provided by Intel Broadwell and Sandy Bridge to estimate the DRAM access latency in cycles. Table 2 shows the sampling-based DRAM access latency and its contribution to the total memory access latency for CIFAR-10 on Broadwell. The scaling factor is the ratio that normalizes average

Table 3. Average Bandwidth Utilization of CIFAR10 Training by IntelCaffe

Package	Total, GB/sec	Read, GB/sec	Write, GB/sec
package 0	12.598	5.795	6.803
package 1	0.728	0.337	0.391
package 2	1.023	0.569	0.454
package 3	0.653	0.301	0.352

DRAM latency against one NUMA scenario. We can clearly observe that, under strong scaling, the fraction of memory access time spent on DRAM has increased significantly due to frequent long-latency accesses when increasing the number of NUMA nodes, i.e., from 20% to 37%. This observation suggests that NUMA factor plays a significant role in bottlenecking the scalability of contemporary Caffe designs and it needs to be significantly reduced for achieving better scalability.

Table 3 shows the average bandwidth of four packages (or NUMA domains) on the Sandy Bridge machine while training CIFAR-10 network using Intel-Caffe. We collected the bandwidth using vTune [18]. Although the network is trained with full capacity on four packages of the Sandy Bridge (all 32 cores), only one package consumes 83.98% of total read and write bandwidth. One possible option to distribute the bandwidth utilization all over the four packages is to use Linux command-line scheduler for scheduling memory placement. For instance, *numactl* runs processes with a specific NUMA scheduling or memory placement policy. "*numactl—-interleave*" can set a "memory interleave policy." Memory pages will be allocated using round robin on nodes. When memory cannot be allocated on the current interleaving target, it falls back to other nodes. Although this simple scheme can help improve memory bandwidth utilization, it fails to maintain data-locality in the NUMA nodes; resulting in complex memory access patterns that bottleneck performance due to many long random remote memory accesses. For instance, we use Intel-Caffe to train GoogleNet [54] on Intel Sandy Bridge and AMD Opteron platforms using "*numactl—interleave,*" and its performance appears to be much worse than our proposed optimization, i.e., 1.4× and 1.6× slower throughputs, respectively. Similar phenomena is observed for training other networks on different architectures. This observation suggests that a more sophisticated design to provide focused NUMA optimization is highly desirable.

3.3 Source of NUMA Effects in Caffe

We further analyze the source of NUMA communication that is bottlenecking Caffe's scalability. To pinpoint the problematic code regions that cause significant NUMA access overhead, we performed data-centric analysis provided by HPCToolkit [6] (e.g., mapping activities to the corresponding function or instruction regions) to trace every *node-load-miss* of CIFAR-10 trained by Intel-Caffe. Table 4 summarizes the code regions that are causing significant REMOTE_DRAM accesses. We can observe that, although all these layers are susceptible to NUMA-induced overhead, convolution layer consumes 51.5% of the total remote DRAM accesses while performing matrix-multiplication (not including *im2col_cpu*). Regarding runtime, convolution layer is also the most significant and occupies the majority of the runtime. Additionally, 5.4% of the remote DRAM access occurs when pooling layer performs non-linear downsampling.

Table 4 reflects how a CNN is mapped to the NUMA domains in the current Caffe designs. During the network initialization, Caffe initializes queues along with several temporary buffers to hold input images and intermediate results to pass to the next layer. During memory allocation, no explicit NUMA affinity is maintained. If no thread binding is applied, OS will schedule a thread to any

Table 4. Node-Load-Miss Breakdown at Function Level When Training CIFAR-10 Network by Intel-Caffe

Layer	Function	% NodeLoadMiss
Convolution Layer	weight_cpu_gemm	36.1%
	forward_cpu_gemm	11.4%
	im2col_cpu	5.7%
	backward_cpu_gemm	4.0%
Pooling Layer	Backward_cpu	3.3%
	Forward_cpu	2.0%
ReLU Layer	Backward_cpu	3.7%

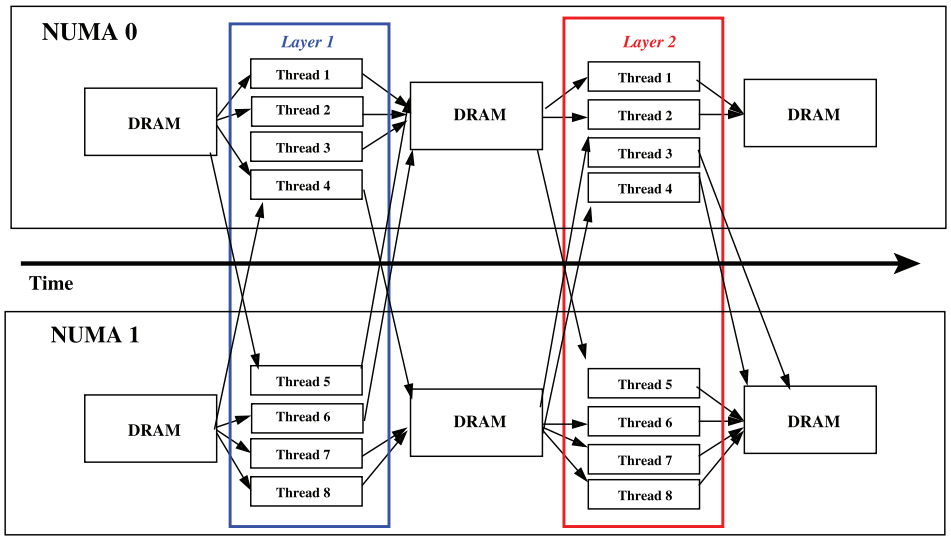

Fig. 6. NUMA communication in current Caffe designs.

random CPU core of the packages. As a result, memory gets allocated to the package where the first touch occurs without any explicit control. During the training phase, Caffe runs data-parallel layers without any expectation of data locality. Both BLAS-level parallelism and coarse-grain batch-level parallelism spread worker threads to available CPU resources. As current implementations of Caffe do not consider NUMA boundaries, worker threads reach across NUMA nodes. But their corresponding input image batches and output may not reside in the same NUMA domain. This also happens to all the temporary buffers created and initialized outside of the OpenMP regions. Figure 6 illustrates the thread mapping of coarse-grain parallelized Caffe (e.g., Intel-Caffe) on a multi-NUMA machine where each OpenMP thread in a layer is mapped to a core statically. Both fine-grain and coarse-grain implementations have to cross the NUMA boundary to fetch or write data to the remote memory domains, drastically increasing memory access latency under limited bandwidth.

3.4 NUMA-Aware Design: Challenges

To eliminate significant NUMA overhead in the current Caffe designs, one possible solution is to maintain NUMA locality between data and worker threads within each layer, requiring us to split

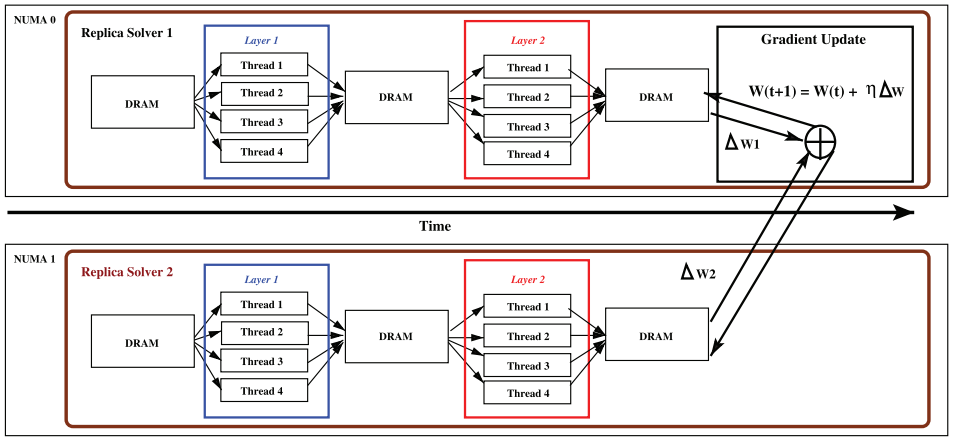

Fig. 7. NUMA-Caffe Design: a hierarchical mapping of parallel regions on NUMA architecutes to eliminate NUMA communication overhead.

data buffers and initialize them to NUMA domains. However, this fine-grain data-partitioning may pose significant overhead and is difficult to effectively eliminate NUMA communication. Recall the description of the basic data structures in Caffe in Section 2.2. The dimension of a blob structure is not the same throughout the network. Based on initial batch size and layer construct, input and output blob can vary. For instance, pooling layer reduces the spatial size of a blob to decrease the number of parameters. As a result, subsequent convolution layers will have a reduced kernel size. Explicit maintenance of NUMA private data localization and tagging them to worker threads during the forward and backward pass on every layer may incur significant overhead. Moreover, it is difficult to eliminate remote NUMA memory access through static data partitioning due to irregular access patterns for different layers. For example, a data-parallel implementation of a fully connected inner-product layer still requires accessing parameters residing in the remote NUMA domains.

4 DESIGN METHODOLOGY

To address the challenges above, we propose a highly-efficient NUMA-aware CNN training framework for CPU-based multi- and many-core architectures, named *NUMA-Caffe*. As discussed previously, data needs to be co-located with its computation to avoid frequent remote NUMA accesses. A straightforward solution is to replace threads with processes (e.g., MPI processes) for ensuring data and computation co-location. However, processes consume more memory in comparison to threads [66]. Thus, a process per core or NUMA domain is not applicable to machines with limited memory space, such as Intel Knights Landing. Thus, NUMA-Caffe still keeps the pure threaded implementation within a single machine as the existing Caffe variants.

However, unlike the existing Caffe variants, NUMA-Caffe adopts a hierarchical parallelism to assign work across threads and distribute threads to different NUMA domains. Figure 7 overviews the NUMA-aware design for NUMA-Caffe. On the algorithm level, NUMA-Caffe decomposes the work into two levels of parallelism. The whole dataset is first decomposed into disjoint segments for computation and distribute to different NUMA domains to benefit from NUMA node-level parallelism. NUMA-Caffe then further partitions the dataset in each NUMA domain to enjoy the thread-level parallelism inside a NUMA node. Section 4.1 describes this NUMA-aware algorithm in NUMA-Caffe.

After that, NUMA-Caffe maps the algorithm to an arbitrary NUMA machine for both computation and data. NUMA-Caffe employs a more flexible thread-handling scheme than the one used in existing Caffe variants that are based on OpenMP environmental variables. Section 4.2 elaborates on the thread-mapping approach. Finally, Section 4.3 discusses the issues raised in NUMA-Caffe implementation.

4.1 NUMA-Caffe Algorithm

Caffe employs a mini-batch-based stochastic gradient for training optimization: mini-batch Standard Gradient Descent (SGD) can perform training on independent data samples and then update the network parameters at the end of every N training samples. Given this nature of the SGD training algorithm, we can confine the computation on disjoint input datasets to enable network-level parallelism. As illustrated in Figure 7, NUMA-Caffe replicates the network across different NUMA domains, known as different solvers. Inside each solver, NUMA-Caffe performs the computation based on the local data with the fine-grained parallelism on the layer level as the existing Caffe variants do. Finally, all solvers synchronize at the end of each iteration to update the gradient to guarantee the correctness of the algorithm.

Algorithm 1 elaborates on the NUMA-aware design. The code region marked in black color is the algorithm used in the latest Caffe implementation—Intel-Caffe, while the blue code lines are the adaptation added by NUMA-Caffe. From the algorithm, we can see that NUMA-Caffe introduces a new parallel loop nest at `line 4` to enable the work decomposition across NUMA domains. Threads in each NUMA domain inheri the computation from the original algorithm and perform layer-level parallelism independently as shown from `line 5-11`. After all the NUMA domains finish processing the batch, a synchronization is needed to gather all the results at the end of each iteration, as shown at `line 12-13`. Next, we highlight two features in this algorithm and prove its correctness.

ALGORITHM 1: Algorithm for NUMA Optimization

```
repeat
    for each miniBatch b ∈ B do
        for each NUMAnode n, Fork Solver do
            for each Layer l ∈ L do
                Do Forward pass with b(n);
            end
            for each Layer l ∈ L do
                Do backward pass with b(n);
                update weight value w(n) for b(n);
            end
            Accumulate weight value, W=W + w(n);
        end
    end
until until satisfied with training;
```

—*Parallel SGD Solvers.* Training independent solvers is an SPMD technique. More specifically, multiple solver instances compute layer output and gradients of their own dataset and then work towards the convergence of a common neuron model. Within each layer, SGD solvers are trained independently on different mini-batches. As a result, each solver computes slightly different model parameters (e.g., weight and bias) compared to another solver running in parallel. The layer

operations during a forward and backward pass of a solver are completely independent of those of another solver.

—*Reduction of Gradients.* As all the SGD solvers work toward a common goal—convergence of the network, timely gradient update is important for the network to get experience from all the parallel solvers. In NUMA-Caffe, we use synchronous gradient update, under which, co-running solvers block/wait at a barrier for other solvers to finish an epoch and then perform a global reduction of the gradients. Note that the asynchronous update is not chosen because it can create divergence and inaccuracy to the final network since slower solvers may work on outdated parameters. We will demonstrate that the reduction overhead in NUMA-Caffe is negligible compared to the major NUMA-induced latency in the existing NUMA-oblivious Caffe solutions (see Section 5).

4.1.1 Algorithm Correctness Proof. Under an unchanged set of training parameters (e.g., batch size), NUMA-Caffe does not change the original mini-batch SGD but only applies hierarchical data-parallelism. The entire batch (b) is split into n kernels, each with a batch size of b/n. *Theorem 1* demonstrates that our algorithm's result is theoretically equivalent to that of the original algorithm used in BVLC and Intel-Caffe.

Theorem 4.1. *Batch gradient descent on a single compute node is equivalent to batch gradient descent on multiple nodes performed by averaging the gradient updates.*

Proof. Let n be the number of compute nodes and nk the batch size.

Set $x_1, \ldots, x_{nk}$ to be the samples in a batch. Batch gradient descent computes the gradient updates for each one, $\nabla_{x_1}, \ldots, \nabla_{x_{nk}}$ without performing a weight update and then applies the update for

$$\frac{1}{nk} \sum_{i=1}^{nk} \nabla_{x_i}.$$

We split the samples among the nodes. How this is done does not matter, but for simplicity, we will set $x_1, \ldots, x_k$ to node 0, $x_{k+1}, \ldots, x_{2k}$ to node 1, and so on. Then, the gradient update computed by node r is

$$\frac{1}{k} \sum_{i=1}^{k} \nabla_{x_{i+r}}.$$

The average is then

$$\begin{aligned} \frac{1}{n} \sum_{r=0}^{n-1} \frac{1}{k} \sum_{i=1}^{k} \nabla_{x_{i+r}} &= \frac{1}{nk} \sum_{r=0}^{n-1} \sum_{i=1}^{k} \nabla_{x_{i+r}} \\ &= \frac{1}{nk} \sum_{i=1}^{nk} \nabla_{x_i}, \end{aligned}$$

which is precisely the number computed on a single node. □

We also conduct empirical validation to confirm the output correctness of our algorithm through empirical evaluation. We evaluate the accuracy of NUMA-Caffe in comparison to BVLC-Caffe. As an example here, we report NUMA-Caffe's accuracy for training CIFAR-10 network. We configure NUMA-Caffe to use eight SGD solvers on the AMD node. Figure 8 compares the output accuracy of these two Caffe designs. From the data, we find that NUMA-Caffe does not suffer from accuracy reduction. After 40,000 iterations, NUMA-Caffe's percentage of error is within the bound of 1.24%. Experiments on other networks also give the same conclusion.

Throughout the article, we measure performance and accuracy under strong scaling with fixed batch size for fair comparison. Due to fixed batch size, we do not observe accuracy loss. On the

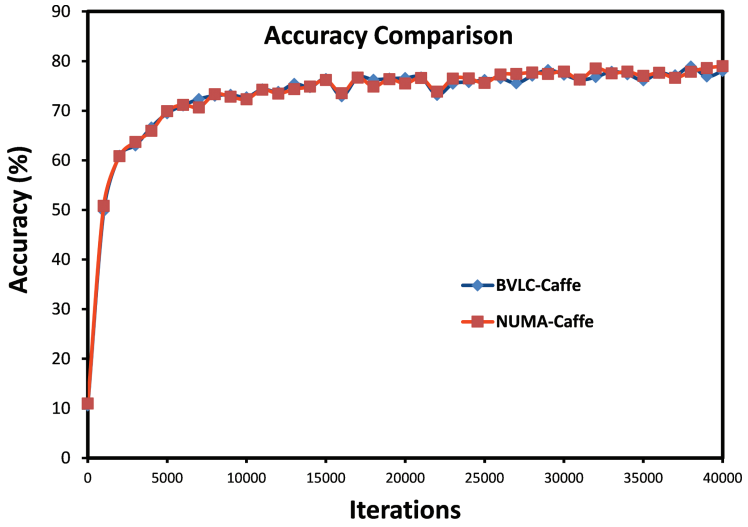

Fig. 8. Empirical validation on the accuracy of our algorithm using CIFAR-10.

other hand, several recent studies have proposed schemes for large batch trainings without deviating the accuracy during weak scaling; Goyal et al. [26] proposed the scheme of changing the learning rate, which overcomes this challenge for distributed systems with a broad range of mini-batch sizes. You et al. [61] applied batch normalization to large batch sizes (8,192–32,768), resulting in no accuracy loss. We use a batch size of 64–256, implying that our solution can scale 128–512 compute nodes, providing no accuracy loss.

4.2 Hierarchical Computation and Data Mapping Strategy

The intrinsic two-level parallelism of NUMA-Caffe can be naturally mapped to threads running on NUMA architectures. Binding these threads to cores is necessary but challenging. As existing Caffe implementations are based on OpenMP and MKL, KMP_AFFINITY environmental variables can bind threads to cores according to thread creation order. However, threads in nested parallel regions are created in an unpredicted order, so NUMA-Caffe cannot rely on these environmental variables.

From implementation perspective, NUMA-Caffe's hierarchical thread mapping is performed at three levels: (1) At top level, NUMA-caffe creates a pthread-based SGD solver for each NUMA domain. (2) At the next hierarchical level, each pthread solver parallelizes the batch-level loop operations of the solver using OpenMP threads. These intra-solver OpenMP threads maintain the NUMA-affinity of their parent pthread solver via cpu-bind. (3) At the lowest level of the hierarchy, each OpenMP thread performs single-threaded BLAS operations through MKL threads. These MKL threads maintain the affinity of their parent OpenMP threads through CPU-binding.

For this kind of hierarchical thread mapping, NUMA-Caffe employs its own thread scheduler. First, NUMA-Caffe extracts the NUMA topology using hwloc [8] to understand the mapping between cores and NUMA domains. NUMA-Caffe assigns a unique ID for each NUMA domain and provides an equation to iterate all the core IDs that belong to this NUMA domain. Second, NUMA-Caffe records the ID of a NUMA domain when it allocates a solver for it. Third, for all the threads spawned within a solver, NUMA-Caffe binds all these threads to the NUMA domain allocated for the solver by explicitly iterating all the core ID in this NUMA domain and calling

`sched_setaffinity`. With this approach, NUMA-Caffe can automatically distribute all of its threads to appropriate cores in any given NUMA architecture without any manual interference.

Besides binding threads to cores, NUMA-Caffe also needs to guarantee the data are placed in appropriate NUMA domains. Linux employs "first touch" policy by default, that is, allocating memory pages in a NUMA domain if a thread from this NUMA domain initializes these pages. To leverage this first touch policy, NUMA-Caffe spawns a helper thread per solver to initialize all the data, such as the queues and temporal buffers used in this solver. NUMA-Caffe's thread scheduler and helper threads guarantee the data and computation are co-located in the same NUMA domain automatically.

—*Mapping Gradient Update to NUMA Architectures.* As discussed previously, NUMA-Caffe applies a synchronized gradient update at the end of each iteration. Synchronizing all threads from different NUMA domains may incur many remote accesses. To maintain a balanced remote NUMA memory bandwidth during the update, we create a tree topology of the NUMA domains. This could be more effective than the flat structure when the number of NUMA nodes is very large. For n NUMA nodes, we create a $n - ary$ tree of solvers, which maintains a list of parent and children. At the end of each iteration, solvers follow tree hierarchy for gradient reduction. For example, the parent solvers wait for their child solvers to reach the barrier. Once notified by their child solvers, a parent solver reduces the gradient by accumulating its own gradient with its child solvers' gradients and notifies its parent. In this way, the root solver eventually computes the gradient of the total network. After that, the root solver computes the updated weights for the next iteration and child solvers to copy the new weight blobs to their individual local NUMA domains.

4.3 Discussion

There are several advantages of our NUMA-Caffe design. First, it provides automatic data localization and eliminates all the major remote memory access within data parallel layers. Second, it is network agnostic, as it does not statically split data structures within each layer and maintain data-thread affinity between NUMA nodes. This design can be applied to any neural network topology. This also removes the burden to design and test new networks to accommodate NUMA effects and enables instantaneous scaling. Third, due to data encapsulation, this design is easier to be adopted in any object-oriented DNN implementation since all the buffers as well as global control structures are private to the respective parallel solver instance without requiring any explicit affinity maintenance. Finally, our design is convergence-invariant, which means it does not affect the network convergence rate.

5 EVALUATION AND ANALYSIS

We evaluate[2] NUMA-Caffe on four modern NUMA architectures: Intel Sandy Bridge, AMD Opteron Magny-Cours, Intel® Knights Landing, and Intel Broadwell. Table 1 shows the hardware configurations of these machines. To evaluate our NUMA-Caffe, we choose two state-of-the-art Caffe designs for comparison: BVLC-Caffe [34] and Intel-Caffe [19]. Their background is introduced in Section 3. For a valid comparison, all three Caffe designs are compiled with `g++ 4.8.5 -O3` and linked to Intel MKL 2017 for optimized BLAS operations on all the testing machines. We

[2]Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations, and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information, go to http://www.intel.com/performance.

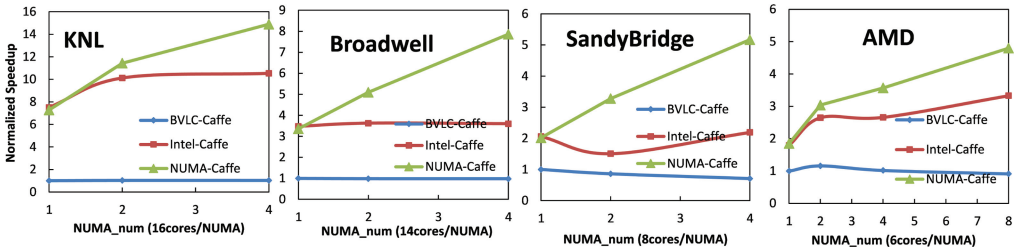

Fig. 9. Normalized throughput of training Alexnet on four architectures using three Caffe designs with increasing numbers of NUMA nodes. Data points are normalized to BVLC-Caffe at one NUMA node.

disable SMT in all of our Intel platforms because we find that SMT does not show benefit to all three Caffe designs.

We configure the three Caffe implementations to train two different networks. First, we use the ImageNet dataset to train the AlexNet network with an exponential learning policy. The initial learning rate, the momentum, and the weight decay are set to *0.01, 0.9, and 0.0005*, respectively. AlexNet has a default batch size of 256. Second, we use the CIFAR-10 dataset to train the CIFAR network with *0.9* momentum and *0.004* weight decay. We set the initial learning rate as 0.001. CIFAR has a batch size of 128. These configuration parameters are either the default provided by Caffe or chosen by experienced users.

We employ the following comparison strategy: (1) We measure the end-to-end of execution time and show that NUMA-Caffe has the best throughput and scalability across multiple NUMA nodes for most cases (Section 5.1) and (2) We collect NUMA-related events via hardware performance counters to highlight that NUMA-Caffe successfully removes most of the remote accesses, which leads to its high performance and superior scalability (Section 5.2).

5.1 Performance and Scalability Measurement

To quantitatively evaluate different Caffe design performance, we measure the training throughput, which is defined as the number of training iterations completed per second. The measurement excludes all the intermediate validation and snapshot steps. A higher throughput indicates better performance. For scalability evaluation, we have all Caffe variants keep input batch size constant and train it with more cores (known as strong scaling). We report the speedups that are normalized against the training throughput of BVLC-Caffe running on all the cores of a single NUMA node of each testing machine.

5.1.1 Evaluation on AlexNet network. Figure 9 shows the normalized throughputs of all Caffe variants trained on our AMD Opteron, Intel Sandy Bridge, Intel® KNL and Intel Broadwell machines. From the figure, we can see that BVLC-Caffe has the worst performance, Intel-Caffe provides a 3-10× speedup over BVLC-Caffe across different machines, and NUMA-Caffe shows the best performance—up to a 14× speedup over BVLC-Caffe and a 2.4× speedup over Intel-Caffe when using all the cores across all NUMA nodes. Particularly, NUMA-Caffe achieves a significant speedup on Intel KNL and Broadwell, which are among the latest architectures released by Intel. These two architectures support MKL 2017 using advanced vectorization units, such as AVX2 and AVX512. They are sensitive to the NUMA overhead because long latency memory accesses along with a high bandwidth requirement can significantly degrade the performance of vectorization. We also evaluate NUMA-Caffe and Intel-Caffe on KNL with HBM enabled. For both Caffe implementations, we place the data in HBM first and then in DRAM if HBM is full. We find that NUMA-Caffe

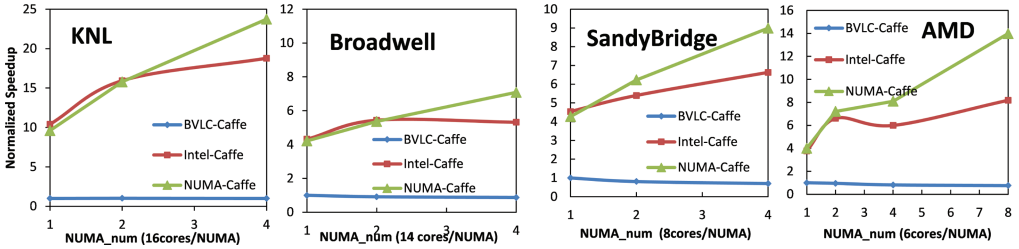

Fig. 10. Normalized throughput of training CIFAR-10 on four architectures using three Caffe designs with increasing number of NUMA nodes. Data points are normalized to BVLC-Caffe at one NUMA node.

achieves a 1.29× speedup over Intel-Caffe. Moreover, NUMA-Caffe gains a 2× speedup over the DDR4-only mode of itself.

For performance scalability, BVLC-Caffe does not scale across NUMA nodes at all; Intel-Caffe has a slightly better scaling than BVLC-Caffe but still does not scale in most architectures; NUMA-Caffe has the best scalability. As BVLC-Caffe depends on MKL for the parallelism, it inherits the problem that MKL owns: not a NUMA-aware design. Thus, BVLC-Caffe does not show any speedup when running on Intel KNL and Broadwell, and even worse performance on AMD Opteron and Intel Sandy Bridge. In contrast, Intel-Caffe employs coarse-grained-layer-level parallelism. Its parallelization and synchronization overhead is largely reduced compared to BVLC-Caffe, causing less traffic across NUMA nodes. Thus, for some architectures, such as AMD Opteron, Intel-Caffe shows moderate scalability. However, Intel-Caffe does not use a NUMA-aware design, so it does not scale in most architectures. Unlike the two, NUMA-Caffe shows the near-linear scalability across all the machines. It is mainly due to its NUMA-aware algorithm and the thread scheduler to co-locate computation and data to avoid most remote NUMA accesses. In Section 5.2, we will show that NUMA-Caffe effectively removes most of the remote accesses.

5.1.2 Evaluation on CIFAR Network. Figure 10 shows the performance and scalability of the three Caffe designs on our four testing platforms. Overall, BVLC-Caffe has the lowest throughput, while Intel-Caffe and NUMA-Caffe have comparable throughputs when using two NUMA nodes. It is worth noting that NUMA-Caffe has an increasing speedup over Intel-Caffe when running on more NUMA nodes. For example, the speedup of NUMA-Caffe over Intel-Caffe is from 1.15× to 1.36× when using two and four NUMA nodes in the Intel Sandy Bridge machine. Furthermore, when training on Intel Broadwell machine, NUMA-Caffe achieves a 1.4× speedup on two NUMA nodes and gains the highest improvement of 2.2× on four NUMA nodes. Through explicitly using HBM on Intel KNL, NUMA-Caffe runs 1.06× faster than Intel-Caffe. Unlike Alexnet, both NUMA-Caffe and Intel-Caffe gains a small speedup of 1.12× and 1.33×, respectively, over the DDR4-only mode, when explicitly using HBM. This observation indicates that CIFAR network is less memory bandwidth bound compared to AlexNet. From the evaluation on all four platforms, NUMA optimization of CIFAR network results in relatively lower speedup than AlexNet.

For scalability, BVLC-Caffe does not show any scalability, which is in expectation because of the frequent synchronization and high parallelization overhead at the fine-grained MKL level. Intel-Caffe shows moderate scalability, and NUMA-Caffe has a slight improvement over Intel-Caffe. Moreover, NUMA-Caffe introduces some management overhead, such as forking solvers and thread scheduling. This overhead cannot be easily amortized in the CIFAR network training. Therefore, NUMA-Caffe shows less throughput at some points compared to Intel-Caffe, such as running on one NUMA node of Intel Sandy Bridge and Intel KNL.

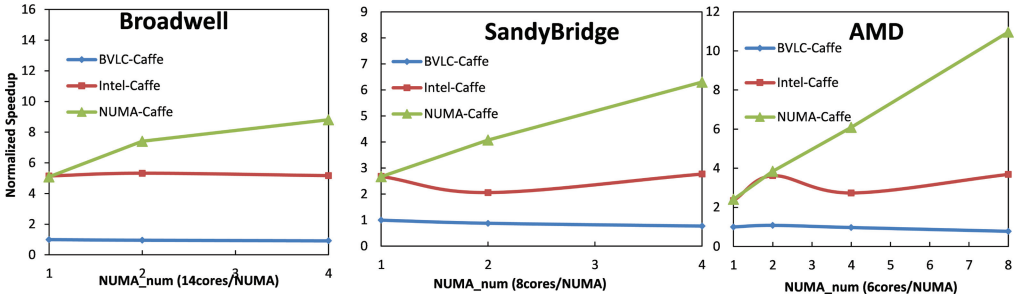

Fig. 11. Normalized throughput of training GoogLeNet on two architectures using three Caffe designs with increasing number of NUMA nodes. Data points are normalized to BVLC-Caffe at one NUMA.

5.1.3 Evaluation on GoogLeNet Network. Figure 11 summarizes the performance of a GoogLeNet model training on three platforms and three variants of Caffe implementations. From the figure, BVLC-Caffe has the worst performance among all the platforms while Intel-Caffe enjoys speedup of 3.6×, 5.6×, and 4.75× on four-NUMA Sandy Bridge, Broadwell and eight-NUMA AMD, respectively. NUMA-Caffe achieves the best performance with 8× and 2× speedup on Sandy Bridge compared to BVLC-Caffe and Intel-Caffe. On the eight-NUMA AMD machine, NUMA-Caffe supersedes Intel-Caffe and BVLC-Caffe by 3× and 14×, respectively. A similar trend is observed on Broadwell machine where NUMA-Caffe achieves 1.7× and 9.6× speedup compared to Intel-Caffe and BVLC-Caffe. On the KNL machine, both Intel-Caffe and NUMA-Caffe suffer from the bottleneck at the input buffer due to the slower HDD disks on our server node. For this reason, we do not report GoogLeNet results on KNL machine here.

Among the three testing platforms, BVLC-Caffe shows no scalability at all due to poor CPU and memory utilization. Intel-Caffe shows moderate scalability on AMD and better scalability on Broadwell. NUMA-Caffe shows the best scalability on AMD machine where it scales 1.1×, 2.2×, and 3× more than Intel-Caffe on two, four, and eight NUMA nodes, respectively.

5.1.4 Comparison to GPU-Based Implementation. We further train the three DNNs on single NVIDIA Tesla K40c GPU with BVLC-Caffe. We compare the performance of GPU-based training with the Intel-Caffe and NUMA-Caffe training on Sandy Bridge with four NUMA nodes. Both AlexNet and GoogLeNet training by BVLC-Caffe on GPU show superior performance than Intel-Caffe and NUMA-Caffe training. BVLC-Caffe trains AlexNet 3.2× and 1.38× faster on GPU compared to Intel-Caffe and NUMA-Caffe training. For GoogLeNet training, BVLC-Caffe on GPU gains 2.5× and 1.09× speedup compared to Intel-Caffe and NUMA-Caffe training on Intel Sandy Bridge. Both AlexNet and GoogLeNet training is computationally expensive. NVIDIA Tesla K40c takes advantage of higher FLOPs/cycle count compared to Intel Sandy Bridge and achieves superior performance while training these networks. On the contrary, CIFAR training on Intel-Caffe and NUMA-Caffe achieves 1.7× and 2.3× speedup compared to the training by BVLC-Caffe on NVIDIA GPU. Due to smaller datasets and shallower CIFAR network, GPU training is bottlenecked by communication overhead.

5.1.5 Summary of the Findings. We summarize our findings from the experiments as follows:

—BVLC-Caffe with fine-grained MKL-level parallelism only does not show any scalability across NUMA nodes with the given two networks and four NUMA architectures.
—Intel-Caffe shows moderate scalability, and the scalability on CIFAR is better than AlexNet. This shows that deeper networks are more sensitive to NUMA latency.

Table 5. Percentage of Remote Memory Access Reduction of Intel-Caffe and NUMA-Caffe Compared to BVLC-Caffe While Training on Four-NUMA Broadwell

Models	Intel-Caffe	NUMA-Caffe
CIFAR	90.18%	96.05%
AlexNet	89.37%	92.09%
GoogLeNet	83.73%	91.47%

Table 6. Balanced Package Bandwidth Utilization of CIFAR Training by NUMA-Caffe on Sandy Bridge

Package	Total, GB/sec	Read, GB/sec	Write, GB/sec
Package 0	3.69	1.99	1.70
Package 1	3.37	1.76	1.61
Package 2	3.80	1.96	1.83
Package 3	3.40	1.76	1.64

—NUMA-Caffe demonstrates the best throughput and scalability in most cases. However, the NUMA-aware design and mapping are not free. NUMA-Caffe may introduce additional overhead, which if not amortized by the beneficial computation, will slightly degrade the overall performance.

5.2 Detailed Analysis with NUMA-Related Events

To evaluate our NUMA-aware optimizations in NUMA-Caffe, we further collect NUMA-related events from hardware performance counters using Linux Perf [41] on Intel Broadwell. We glean two events: `node-load` and `node-load-miss`, which indicate local and remote NUMA node accesses, respectively. We compare all three Caffe designs using a derived metric: remote memory access per iteration, which is the absolute remote memory accesses averaged across iterations.

Table 5 presents the reduction of remote memory access by Intel-Caffe and NUMA-Caffe during CIFAR training on Broadwell. Compared to the Intel-Caffe designs, NUMA-Caffe achieves the highest, 96% reduction of remote accesses on four NUMA nodes. To further understand the root causes of remote accesses in NUMA-Caffe, we associate the metrics with source code. We find that over 71% of the remote accesses occur during gradient and weight update between parallel SGD solvers. This communication across NUMA nodes guarantees the correctness of NUMA-Caffe. The remaining 29% of remote access is caused while populating the minibatch buffers from image database. Thus, NUMA-Caffe can eliminate most of the unnecessary remote NUMA accesses. Furthermore, Figure 12 plots the layer level speedup of CIFAR training on Intel-Caffe and NUMA-Caffe compared to BVLC-Caffe. Compared to Intel-Caffe, NUMA-caffe achieves the best speedup of 1.44×, 2.9×, and 1.24× on conv1, conv2, and conv3 layer, respectively.

To evaluate the bandwidth usage across different NUMA nodes, we measure the bandwidth consumption for NUMA-Caffe when training the CIFAR network in Table 6. From the table, we can see that NUMA-Caffe achieves balanced bandwidth usage, effectively avoiding bandwidth contention. We further compare the memory access and thread synchronization overhead of CIFAR training on Sandy Bridge. NUMA-Caffe achieves 7× less memory access latency and 1.12× less wait and spin time for CIFAR training compared to Intel-Caffe. This proves that NUMA-aware

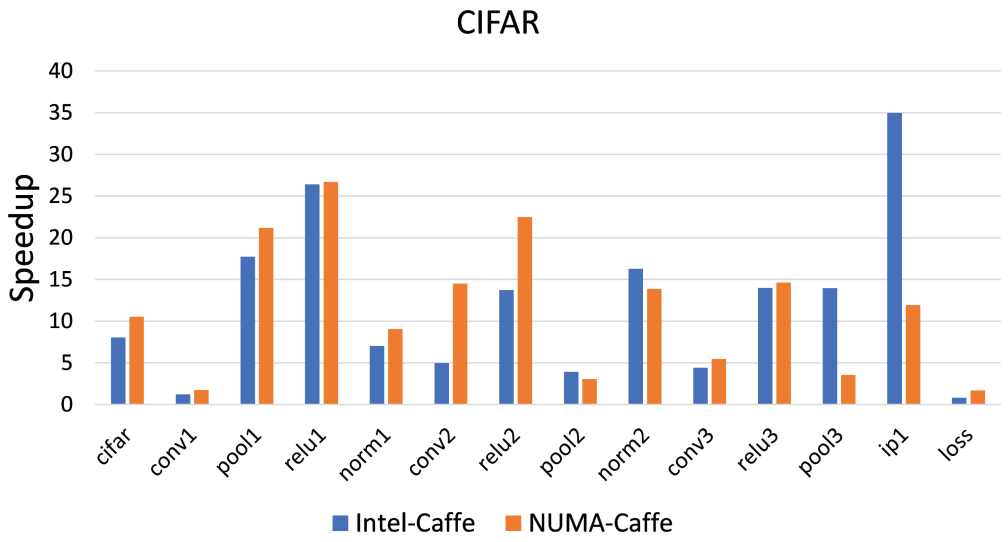

Fig. 12. Layer-level speedup of CIFAR training by Intel-Caffe and NUMA-Caffe on Sandy Bridge. Runtime is compared with BVLC-Caffe and measured on four NUMA nodes.

optimization achieves speedup by reducing both memory reference overhead and synchronization overhead.

For AlexNet and GoogLeNet training with ImageNet data, we observe a similar behavior as for CIFAR. Table 5 shows that, compared to Intel-Caffe, NUMA-Caffe achevies the highest 92.09% and 91.47% reduction of remote accesses during AlexNet and GoogLeNet training on four NUMA nodes. NUMA-Caffe achieves 12× and 13× less memory access latency for AlexNet and GoogLeNet training, respectively, compared to Intel-Caffe. Moreover, NUMA-Caffe has the least wait and spin time compared to Intel-Caffe. NUMA-Caffe observes 1.56× and 1.19× less synchronization overhead compared to Intel-Caffe training on four-NUMA Intel Sandy Bridge. Figure 13 plots the layer-level speedup of AlexNet training on four NUMA nodes by Intel-Caffe and NUMA-Caffe compared to BVLC-Caffe. All the most significant layers achieve the best performance after NUMA optimization. Similar layer-level speedup is observed for GoogLeNet training. Figure 14 shows the speedup of the most significant 16 layers of GoogLeNet that contributed 80% of the total runtime. NUMA-Caffe shows the best performance to optimize the most significant layers compared to Intel-Caffe.

While NUMA-Caffe effectively reduces a significant amount of remote NUMA accesses, it cannot totally eliminate them. At the end of each forward-backward pass, solvers of NUMA-Caffe update their gradients. This gradient update involves obtaining weight matrix located on remote NUMA domains. Moreover, actively waiting on lock words by threads across NUMA domains requires remote NUMA accesses. These remote accesses are necessary to guarantee the correctness of the algorithm.

6 RELATED WORK

DNN Frameworks. Depending on the specific underlying architecture, different forms of data-parallelism have been applied for training a DNN. For example, recent DNN research has been largely focusing on GPU-powered approaches [16, 21, 28, 51] due to its massive parallelism and architecture-specialized acceleration libraries (e.g., cuBLAS and cuDNN [13]). Although

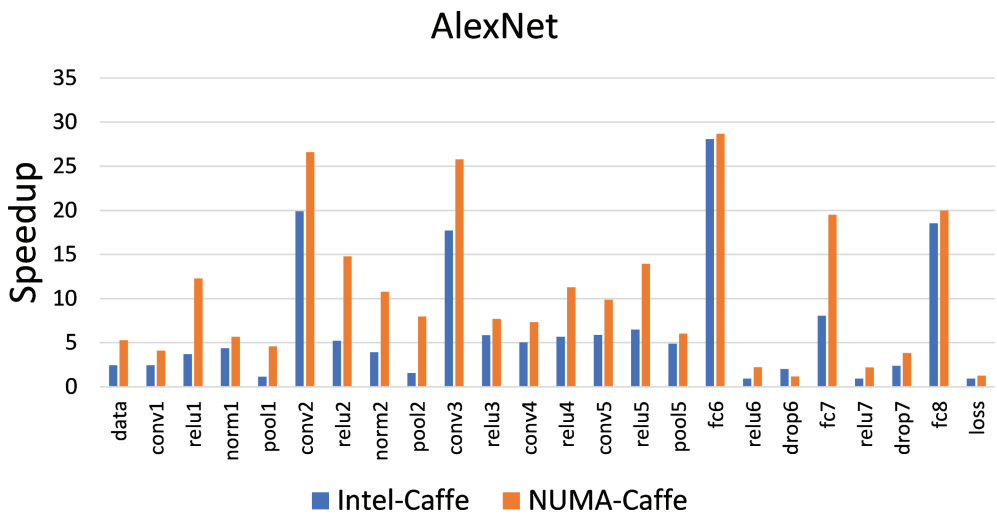

Fig. 13. Layer-level speedup of AlexNet training by Intel-Caffe and NUMA-Caffe on Sandy Bridge. Runtime is compared with BVLC-Caffe and measured on four NUMA nodes.

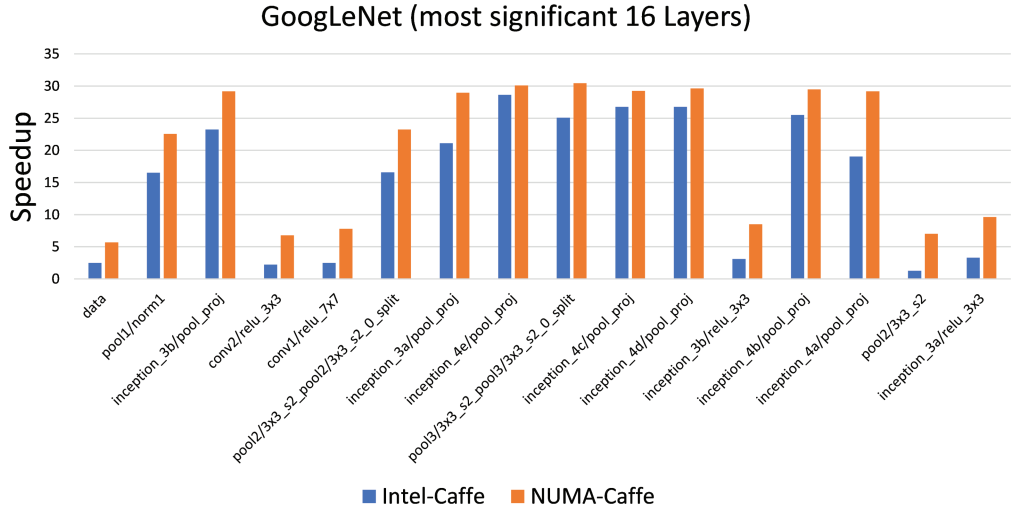

Fig. 14. Layer-level speedup of GoogLeNet training by Intel-Caffe and NUMA-Caffe on Sandy Bridge. Runtime is compared with BVLC-Caffe and measured on four NUMA nodes.

GPU-accelerated DNNs have reported significant speedup, there are still two major limitations that hinder scalability: (1) many real-world data models cannot be effectively processed due to GPUs' memory constraints and (2) major performance improvement of GPU-based DNNs comes from the specialized libraries, which are not general, hard to port and pose significant bias for training certain types of applications [55].

At node-level, several CPU-based frameworks that target accelerating complex networks on multi- and many-core architectures have been proposed. For instance, popular CNN-based frameworks, including BVLC [34], coarse-grain parallelized Caffe [55] and Intel Caffe [19], have significantly accelerated CNNs for shared-memory multi-core systems. This line of research is very

promising because the results in References [19, 55] have demonstrated that their multi-core-based approaches have achieved a similar level of performance as those obtained by the GPU-optimized CNNs, but without GPUs' major limitations. However, when scaling up to a large number of NUMA nodes, their performance is still hindered by the significant overhead caused by frequent remote memory access. Similarly, the recent *spg-CNN* [52] provides a scheduler for automatic code generations, optimizing for sparsity and spatial reuse in convolutions. However, *spg-CNN* has not taken NUMA effects on modern architectures into consideration to conduct optimization for addressing this important system-level issue. To tackle this problem, we propose *NUMA-Caffe*, a NUMA-aware multi-solver design for CNN acceleration. More specifically, we transform the layer-level data-parallelized Caffe to support NUMA-aware training with multi-level hierarchical parallelism. Our design is independent of CNN topology, impact-free on network convergence rate, and provides further scalability to the existing multi-core-based CNN frameworks.

Several data-parallelism oriented proposals have studied DNN training from the perspectives of distributed-memory environment [39, 48, 50]. Data-parallel methods on distributed-memory environments replicate the model and launch them on separate node/machine with different mini-batches and reconstruct the model parameters through gradient update. Other than data-parallelism, model-parallelism-based approaches also perform well for training large models on distributed systems where communication is a major bottleneck [22, 35]. It scales training by partitioning a model itself and executing parts of it across different nodes. This approach helps reduce communication overhead while maintaining data locality within each node. The Distbelief [22] framework introduced both data-parallelism and model-parallelism in CPU clusters where it asynchronously updates models after distributed training. Tensorflow [5] and Mxnet [12] apply similar multi-solver-based data and model-parallelism for distributed training on heterogeneous systems including multi-GPU and mobile devices. Facebook [59] adopts both data-parallelism and model-parallelism to train multi-GPU machines. FireCaffe [30] uses data-parallelism and introduce reduction trees instead of a parameter server to scale DNNs on multi-GPU clusters. At node-level, BVLC-Caffe [34] adopts the similar data-parallelism for multi-GPU training. These map-reduce like DNN training frameworks preserve the data-locality and at the same time, reduce inter-node communications; this inherent capability inspired us to choose model-replication based data-parallel mechanism for optimizing NUMA-bottlenecks. Furthermore, we apply batch-level data parallelism for intra-node parallelism. Although we adopt an approach that is well explored in the distributed and multi-GPU systems, the goal of this article is orthogonal to the prior work. First, cloud-based implementations incur large network communication overhead due to distributed file systems and remote procedure call; comparatively, NUMA-Caffe can take advantage of the fast inter-node communication of shared memory systems. Second, although multi-GPU systems observe NUMA behaviors, due to micro-architectural differences it is important to study various DNN NUMA characteristics on CPU-based systems. Moreover, NUMA-Caffe enables other NUMA-aware optimizations of CPU systems (i.e., lock cohorting). To our knowledge, NUMA-Caffe is the first to quantify the CPU-based NUMA-scalability challenges of various DNNs.

Locality-Aware Analysis and Application Optimization. Locality-aware thread placements have been explored to improve data sharing in parallel programs [44, 45, 46]. Furthermore, NUMA-aware locality models are proposed for thread and data placement [43]. The recent development of NUMA analysis tools [38, 42] have provided a better understanding of NUMA bottlenecks in different types of applications. With the help of this development, NUMA-aware library [9], compiler [47], OS [47], and application optimizations [60, 63, 65] have also emerged. Reference[65] optimizes graph analytics for NUMA-based architectures. NUMA-aware Phoenix [60] explores the MapReduce scalability on NUMA systems. Dimmwitted [63] studies the challenges of scaling statistical analytics on NUMA-based shared memory systems and proposes three optimization

tradeoffs. Inspired by some of the valuable experience from these research, we have extensively characterized three state-of-the-art multi- and many-core-based CNNs and proposed design to significantly address their scalability limitations caused by the negative NUMA effects.

Distributed Memory DNN Optimizations. Several researchers have proposed designing distributed memory DL algorithms and implementations. A few prominent approaches include S-Caffe [7], FireCaffe [30], Chainer [56], MaTEx [57], PowerAI [15], CNTK [62], PaddlePaddle [4], and Caffe2 [26]. These approaches leverage HPC interconnects by using MPI [25, 27], and provide overlap of communication with computation. While one MPI process per NUMA domain can reduce NUMA problems, exascale systems may have 1K–10K threads per node and likely would not have sufficient memory to allow one MPI rank per NUMA domain. There is a significant body of research on parameter-server-based solutions including GeePS [20], Petuum [58], MXNET [12], Poseidon [64], Parle [10], ProjectAdam [14], CaffeonSpark [3], SparkNet [49], DSSTNE [2], and others [11, 22, 29, 31]. Yet, parameter-server-based solutions have convergence issues and require warm start [40]—which are not attractive for large datasets such as considered in this article.

7 CONCLUSIONS

In this article, we proposed a novel NUMA-aware framework for training CNNs on modern CPU-based multi- and many-core-based architectures. We first conducted a thorough empirical evaluation of NUMA impact with several existing Caffe designs running on four contemporary NUMA architectures, pinpointed the root cause of their scalability issues, and analyzed its sources. Based on these insights and a detailed discussion on design challenges, we proposed the design methodology and implementation of our NUMA-aware multi-solver CNN acceleration framework, named NUMA-Caffe. NUMA-Caffe's design choice is orthogonal to other CPU-specific Caffe optimizations (i.e., cache blocking). As a result NUMA-Caffe provides complementary benefits to BVLC-Caffe and Intel-Caffe. Finally, we evaluated our design with multiple input datasets and CNN topologies on various NUMA architectures. Results demonstrate that our design outperforms the state-of-the-art CNN solutions in both throughput and scalability.

REFERENCES

[1] 2013. smem memory reporting tool. Retrieved January 21, 2018 from https://www.selenic.com/smem.

[2] 2016. Amazon DSSTNE github project. Retrieved January 21, 2018 from https://github.com/amzn/amazon-dsstne.

[3] 2016. CaffeOnSpark github project. Retrieved January 21, 2018 from https://github.com/yahoo/CaffeOnSpark.

[4] 2016. PaddlePaddle. Retrieved January 21, 2018 from https://github.com/paddlepaddle/paddle.

[5] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv:1603.04467*.

[6] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent. 2010. HPCTOOLKIT: Tools for performance analysis of optimized parallel programs http://hpctoolkit.org. *Concurrency and Computation: Practice &. Experience—Scalable Tools for High-End Computing* 22, 6 (April 2010), 685–701.

[7] Ammar Ahmad Awan, Khaled Hamidouche, Jahanzeb Maqbool Hashmi, and Dhabaleswar K. Panda. 2017. S-Caffe: Co-designing MPI runtimes and caffe for scalable deep learning on modern GPU clusters. In *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'17)*. ACM, New York, NY, 193–205.

[8] Francois Broquedis, Jérôme Clet-Ortega, Stéphanie Moreaud, Nathalie Furmento, Brice Goglin, Guillaume Mercier, Samuel Thibault, and Raymond Namyst. 2010. Hwloc: A generic framework for managing hardware affinities in HPC applications. In *Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP'10)*. IEEE Computer Society, Washington, DC, 180–186.

[9] François Broquedis, Nathalie Furmento, Brice Goglin, Pierre-André Wacrenier, and Raymond Namyst. 2010. ForestGOMP: An efficient OpenMP environment for NUMA architectures. *Int. J. Parallel Program.* 38, 5–6 (2010), 418–439.

[10] Pratik Chaudhari, Carlo Baldassi, Riccardo Zecchina, Stefano Soatto, and Ameet Talwalkar. 2017. Parle: Parallelizing stochastic gradient descent. *arXiv:1707.00424*.

[11] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. 2016. Revisiting distributed synchronous SGD. *arXiv:1604.00981*.

[12] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv:1512.01274* (2015).

[13] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. CUDNN: Efficient primitives for deep learning. *arXiv:1410.0759*.

[14] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project Adam: Building an efficient and scalable deep learning training system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI'14)*. USENIX Association, Broomfield, CO, 571–582.

[15] Minsik Cho, Ulrich Finkler, Sameer Kumar, David Kung, Vaibhav Saxena, and Dheeraj Sreedhar. 2017. PowerAI DDL. *arXiv:1708.02188*.

[16] Dan Claudiu Cireşan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. 2010. Deep, big, simple neural nets for handwritten digit recognition. *Neural Comput.* 22, 12 (Dec. 2010), 3207–3220.

[17] Intel Corporation. 2009. Intel QuickPath Interconnect. Retrieved January 21, 2018 from http://www.intel.com/content/www/us/en/io/quickpath-technology/quickpath-technology-general.html.

[18] Intel Corporation. 2010. Intel VTune Performance Analyzer. Retrieved January 21, 2018 from http://software.intel.com/en-us/intel-vtune.

[19] Intel Corporation. 2017. Intel distribution of Caffe*. https://github.com/intel/caffe.git.

[20] Henggang Cui, Hao Zhang, Gregory R. Ganger, Phillip B. Gibbons, and Eric P. Xing. 2016. GeePS: Scalable deep learning on distributed GPUs with a GPU-specialized parameter server. In *Proceedings of the 11th European Conference on Computer Systems (EuroSys'16), London, United Kingdom, April 18-21, 2016*. ACM, 4:1–4:16.

[21] G. E. Dahl, Dong Yu, Li Deng, and A. Acero. 2012. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Trans. Audio, Speech, Lang. Process.* 20, 1 (Jan. 2012), 30–42.

[22] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. 2012. Large scale distributed deep networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems—Volume 1 (NIPS'12)*. Curran Associates Inc., 1223–1231.

[23] Advanced Micro Devices. 2009. AMD HyperTransport Technology. Retrieved January 21, 2018 from http://www.amd.com/en-us/innovations/software-technologies/hypertransport.

[24] Hewlett Packard Enterprise. 2017. HP Integrity Superdome X. Retreived January 21, 2018 from http://www8.hp.com/h20195/v2/GetPDF.aspx/c04383189.pdf.

[25] Al Geist, William Gropp, Steve Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, William Saphir, Tony Skjellum, and Marc Snir. 1996. MPI-2: Extending the message-passing interface. In *Proceedings of the 2nd International Euro-Par Conference on Parallel Processing (I Euro-Par'96)*, Luc Bougé, Pierre Fraigniaud, Anne Mignotte, and Yves Robert (Eds.). Springer, Berlin, 128–135.

[26] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch SGD: Training imagenet in 1 hour. *arXiv:1706.02677*

[27] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. 1996. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing* 22, 6 (1996), 789–828.

[28] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Process. Mag.* 29, 6 (2012), 82–97.

[29] Qirong Ho, James Cipar, Henggang Cui, Jin Kyu Kim, Seunghak Lee, Phillip B. Gibbons, Garth A. Gibson, Gregory R. Ganger, and Eric P. Xing. 2013. More effective distributed ML via a stale synchronous parallel parameter server. In *Proceedings of the 26th International Conference on Neural Information Processing Systems—Volume 1 (NIPS'13)*. Curran Associates Inc., 1223–1231.

[30] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, and Kurt Keutzer. 2016. Firecaffe: Near-linear acceleration of deep neural network training on compute clusters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2592–2600.

[31] Ayae Ichinose, Atsuko Takefusa, Hidemoto Nakada, and Masato Oguchi. 2017. Pipeline-based processing of the deep learning framework caffe. In *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication (IMCOM'17)*. ACM, New York, NY, Article 97, 97:1–97:8 pages.

[32] Intel. 2017. Intel Math Kernel Library 2017. Retrieved January 21, 2018 from https://software.intel.com/en-us/articles/intel-math-kernel-library-intel-mkl-2017-getting-started.

[33] Intel Corporation. 2010. *Intel 64 and IA-32 Architectures Software Developer's Manual*, Vol. 3B: System Programming Guide, Part 2, No. 253669-032 (June 2010).
[34] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093*.
[35] Alex Krizhevsky. 2014. One weird trick for parallelizing convolutional neural networks. *arXiv:1404.5997* (2014).
[36] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning Multiple Layers of Features From Tiny Images*. Technical Report. Department of Computer Science, University of Toronto.
[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems—Volume 1 (NIPS'12)*. Curran Associates Inc., 1097–1105.
[38] Renaud Lachaize, Baptiste Lepers, and Vivien Quéma. 2012. MemProf: A memory profiler for NUMA multicore systems. In *Presented at 2012 USENIX Annual Technical Conference (USENIX ATC'12)*. 53–64.
[39] John Langford, Alexander Smola, and Martin Zinkevich. 2009. Slow learners are fast. *arXiv:0911.0491*.
[40] Mu Li, David G. Andersen, Alexander Smola, and Kai Yu. 2014. Communication efficient distributed machine learning with the parameter server. In *Proceedings of the 27th International Conference on Neural Information Processing Systems—Volume 1 (NIPS'14)*. MIT Press, Cambridge, MA, 19–27.
[41] Linux. 2015. Perf: Linux profiling with performance counters. Retrieved January 21, 2018 from https://perf.wiki.kernel.org/index.php/Main_Page.
[42] Xu Liu and John Mellor-Crummey. 2014. A tool to analyze the performance of multithreaded programs on NUMA architectures. In *Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'14)*. ACM, New York, NY, USA, 259–272.
[43] Hao Luo, Jacob Brock, Pengcheng Li, Chen Ding, and Chencheng Ye. 2016. Compositional model of coherence and NUMA effects for optimizing thread and data placement. In *Proceedings of the 2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'16)*. IEEE, 151–152.
[44] Hao Luo, Chen Ding, and Pengcheng Li. 2014. Optimal thread-to-core mapping for pipeline programs. In *Proceedings of the ACM SIGPLAN Workshop on Memory System Performance and Correctness*.
[45] Hao Luo, Pengcheng Li, and Chen Ding. 2015. *Parallel Data Sharing in Cache: Theory, Measurement and Analysis*. Technical Report URCS 994. Department of Computer Science, University of Rochester.
[46] Hao Luo, Pengcheng Li, and Chen Ding. 2017. Thread data sharing in cache: Theory and measurement. In *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, 103–115.
[47] Zoltan Majo and Thomas R. Gross. 2012. Matching memory access patterns and data placement for NUMA systems. In *Proceedings of the 10th International Symposium on Code Generation and Optimization*. ACM, 230–241.
[48] Gideon Mann, Ryan McDonald, Mehryar Mohri, Nathan Silberman, and Daniel D. Walker. 2009. Efficient large-scale distributed training of conditional maximum entropy models. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems (NIPS'09)*. Curran Associates Inc., 1231–1239.
[49] Philipp Moritz, Robert Nishihara, Ion Stoica, and Michael I. Jordan. 2015. SparkNet: Training deep networks in spark. *arXiv:1511.06051*.
[50] Feng Niu, Benjamin Recht, Christopher Re, and Stephen J. Wright. 2011. HOGWILD! A lock-free approach to parallelizing stochastic gradient descent. In *Proceedings of the 24th International Conference on Neural Information Processing Systems (NIPS'11)*. Curran Associates Inc., 693–701.
[51] Rajat Raina, Anand Madhavan, and Andrew Y. Ng. 2009. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML'09)*. ACM, New York, NY, 873–880.
[52] Samyam Rajbhandari, Yuxiong He, Olatunji Ruwase, Michael Carbin, and Trishul Chilimbi. 2017. Optimizing CNNs on multicores for scalability, performance and goodput. In *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 267–280.
[53] SGI. 2015. SGI UV The World's Most Powerful In-Memory Supercomputers. Retrieved August 01, 2014 from https://www.sgi.com/products/servers/uv.
[54] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*. 1–9.
[55] Marc Gonzalez Tallada. 2016. Coarse grain parallelization of deep neural networks. In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'16)*. ACM, New York, NY, Article 1, 12 pages.
[56] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. 2015. Chainer: A next-generation open source framework for deep learning. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in the 29th Annual Conference on Neural Information Processing Systems (NIPS'15)*, Vol. 5.

[57] Abhinav Vishnu, Charles Siegel, and Jeffrey Daily. 2016. Distributed tensorflow with MPI. *arXiv:1603.02339* (2016).
[58] Eric P. Xing, Qirong Ho, Wei Dai, Jin-Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. 2015. Petuum: A new platform for distributed machine learning on big data. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'15)*. ACM, New York, NY, USA, 1335–1344.
[59] Omry Yadan, Keith Adams, Yaniv Taigman, and Marc'Aurelio Ranzato. 2013. Multi-GPU training of convnets. *arXiv:1312.5853* .
[60] Richard M. Yoo, Anthony Romano, and Christos Kozyrakis. 2009. Phoenix rebirth: Scalable MapReduce on a large-scale shared-memory system. In *Proceedings of the IEEE International Symposium on Workload Characterization, 2009 (IISWC'09)*. IEEE, 198–207.
[61] Yang You, Igor Gitman, and Boris Ginsburg. 2017. Scaling SGD batch size to 32k for ImageNet training. *arXiv: 1708.03888*.
[62] Dong Yu, Adam Eversole, Mike Seltzer, Kaisheng Yao, Oleksii Kuchaiev, Yu Zhang, Frank Seide, Zhiheng Huang, Brian Guenter, Huaming Wang, Jasha Droppo, Geoffrey Zweig, Chris Rossbach, Jie Gao, Andreas Stolcke, Jon Currey, Malcolm Slaney, Guoguo Chen, Amit Agarwal, Chris Basoglu, Marko Padmilac, Alexey Kamenev, Vladimir Ivanov, Scott Cypher, Hari Parthasarathi, Bhaskar Mitra, Baolin Peng, and Xuedong Huang. 2014. *An Introduction to Computational Networks and the Computational Network Toolkit*. Technical Report MSR-TR-2014–112. Microsoft Research.
[63] Ce Zhang and Christopher Ré. 2014. DimmWitted: A study of main-memory statistical analytics. *Proceedings of the VLDB Endowment* 7, 12 (Aug. 2014), 1283–1294.
[64] Hao Zhang, Zhiting Hu, Jinliang Wei, Pengtao Xie, Gunhee Kim, Qirong Ho, and Eric Xing. 2015. Poseidon: A system architecture for efficient GPU-based deep learning on multiple machines. *arXiv:1512.06216*.
[65] Kaiyuan Zhang, Rong Chen, and Haibo Chen. 2015. NUMA-aware graph-structured analytics. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'15)*. ACM, New York, NY, 183–193.
[66] Xiaomin Zhu, Junchao Zhang, Kazutomo Yoshii, Shigang Li, Yunquan Zhang, and Pavan Balaji. 2015. Analyzing MPI-3.0 process-level shared memory: A case study with stencil computations. In *Proceedings of the 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'15)*. IEEE, 1099–1106.

Received July 2017; revised February 2018; accepted March 2018