HiMA: A Fast and Scalable History-based Memory Access Engine for Differentiable Neural Computer

Yaoyu Tao, Zhengya Zhang
{taoyaoyu,zhengya}@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

ABSTRACT

Memory-augmented neural networks (MANNs) provide better inference performance in many tasks with the help of an external memory. The recently developed differentiable neural computer (DNC) is a MANN that has been shown to outperform in representing complicated data structures and learning long-term dependencies. DNC's higher performance is derived from new history-based attention mechanisms in addition to the previously used content-based attention mechanisms. History-based mechanisms require a variety of new compute primitives and state memories, which are not supported by existing neural network (NN) or MANN accelerators. We present HiMA, a tiled, history-based memory access engine with distributed memories in tiles. HiMA incorporates a multi-mode network-on-chip (NoC) to reduce the communication latency and improve scalability. An optimal submatrix-wise memory partition strategy is applied to reduce the amount of NoC traffic; and a two-stage usage sort method leverages distributed tiles to improve computation speed. To make HiMA fundamentally scalable, we create a distributed version of DNC called DNC-D to allow almost all memory operations to be applied to local memories with trainable weighted summation to produce the global memory output. Two approximation techniques, usage skimming and softmax approximation, are proposed to further enhance hardware efficiency. HiMA prototypes are created in RTL and synthesized in a 40nm technology. By simulations, HiMA running DNC and DNC-D demonstrates 6.47× and 39.1× higher speed, 22.8× and 164.3× better area efficiency, and 6.1× and 61.2× better energy efficiency over the state-of-the-art MANN accelerator. Compared to an Nvidia 3080Ti GPU, HiMA demonstrates speedup by up to 437× and 2,646× when running DNC and DNC-D, respectively.

CCS CONCEPTS

• **Computer systems organization → Neural networks**; **Data flow architectures**; **Special purpose systems**.

KEYWORDS

differentiable neural computer, memory-augmented neural networks, memory access engine

MICRO '21, October 18–22, 2021, Virtual Event, Greece

ACM ISBN 978-1-4503-8557-2/21/10…$15.00
https://doi.org/10.1145/3466752.3480052

ACM Reference Format:
Yaoyu Tao, Zhengya Zhang. 2021. HiMA: A Fast and Scalable History-based Memory Access Engine for Differentiable Neural Computer. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '21), October 18–22, 2021, Virtual Event, Greece.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3466752.3480052

1 INTRODUCTION

The application of neural networks (NNs) have grown extensively to many practical problems such as natural language processing (NLP) [8], speech recognition [12] and computer vision (CV) [15]. In the case of NLP, the improvements come from the sequence modeling capabilities of recurrent NNs (RNNs) [11] such as long short-term memory (LSTM) [35] or gated recurrent units (GRU) [6]. However, the performance of RNNs is limited by how long memories can persist, because the dynamic states are intrinsically embodied within the network. RNNs become less effective in tasks like question answering (QA) [34] where relevant information for the correct answers could be far away from where the questions are asked. This motivated the development of memory-augmented NNs (MANNs), a fully differentiable model that contains an isolated external memory module that NNs can learn to store to and read from when computing predicted outputs.

Such MANNs include memory network (MemNet and MemN2N) [34, 38], dynamic memory network (DMN) [19], neural Turing machine (NTM) [13] and differentiable neural computer (DNC) [14]. Compared to traditional RNN/LSTM or recently developed Transformer [36], MANNs outperform in tackling long-term dependency problems and find many applications not only in NLP, but also in graph modeling [2, 25], navigation [14, 32] and reinforcement learning [3, 7, 10]. Specifically, DMN uses a GRU as the memory component. MemNet/MemN2N uses an external addressable memory; however, neither memory content nor access history is considered in the addressing. NTM enhances the performance by using content-based soft write and read. An NTM can infer simple algorithms such as copying or sorting. Subsequently, DNC extends NTM by incorporating history-based attention mechanisms that consider historical events when accessing external memory. This allows DNC to achieve better performance than NTM in handling long-term dependencies [14]. However, the enhanced performance of DNC comes at a high computational cost, complex memory operations, and specifically history-based attention mechanisms.

NN accelerators [1, 5, 20, 21, 26, 31] cannot run DNC due to the lack of capability to handle elaborate memory operations. Compared to NN accelerators that store weights, perform convolutions and accumulate partial sums, DNC accelerators need to support more complex and diverse workloads, including new primitives like sorting and matrix transpose and a variety of new state memories

to store access history that do not exist in NNs. MANN accelerators for MemNets or NTM [17, 27, 28, 33] also cannot run DNC due to the lack of support of DNC's unique sorting primitive and new state memories. The only way to run DNC using an existing accelerator is to have it attached to a general-purpose CPU or GPU, which is unlikely to deliver a high efficiency.

NTM accelerators only support content-based attention mechanisms. Specifically, X-MANN [28] implements external memory using resistive crossbars. The performance gain relies on emerging devices that are not widely available. The recently developed MANNA [33] proposed a network-on-chip (NoC) architecture for NTM. MANNA's distributed architecture provides more memory bandwidth and compute parallelism, but its H-tree NoC still incurs a traffic bottleneck when running DNC's history-based attention mechanisms. DNC accelerators have been developed recently [4, 30]. In [4], the efficiency mainly comes from analog-based processing elements such as analog-to-digital converters (ADCs), which are more sensitive to variations and noise, and less portable between process technologies. These designs all followed a centralized architecture for memory access and compute, which could lead to poor scalability when memory size increases. Operations may need to be serialized, degrading both speed and latency.

We present HiMA, a History-based Memory Access engine to efficiently accelerate DNC memory operations. To the best of our knowledge, HiMA is the first distributed, tiled architecture that supports all DNC features. History-based attention mechanisms introduce a variety of new primitives and state memories, requiring access to various memories concurrently and incurring complex traffic in an NoC architecture. HiMA focuses on distributed processing of DNC and optimizing NoC traffic to enhance scalability. We summarize the contributions of this work as follows:

- *Scalable Multi-Mode NoC.* We study the DNC computation and memory access profile, especially history-based attention mechanisms. Based on the analysis, a multi-mode NoC is designed to adapt to DNC's traffic profile, improving both traffic latency and scalability.
- *Optimized Memory Partition.* Conventional row or column-wise partition is suboptimal for DNC's new state memories. We consider both content-based and history-based mechanisms and propose a submatrix-wise partition to reduce NoC traffic.
- *Distributed and Efficiency-Enhanced Kernels.* Both memory and memory operations are distributed to tiles using a new distributed DNC (DNC-D) model. The distributed computational kernels minimize the NoC traffic and provide a higher parallelism. We propose local-global two-stage sort, usage skimming, and softmax approximation to reduce the complexity and improve the computational efficiency.

Prototypes of HiMA are implemented in RTL and synthesized in a 40nm technology. HiMA running DNC and DNC-D demonstrates up to 6.47× and 39.1× improvements in speed, 22.8× and 164.3× improvements in area efficiency, and 6.1× and 61.2× improvements in energy efficiency, respectively, over MANNA, the state-of-the-art MANN accelerator for NTM. Compared to an Nvidia 3080Ti GPU, HiMA shortens the inference time by up to 437× and 2,646× when running DNC and DNC-D.

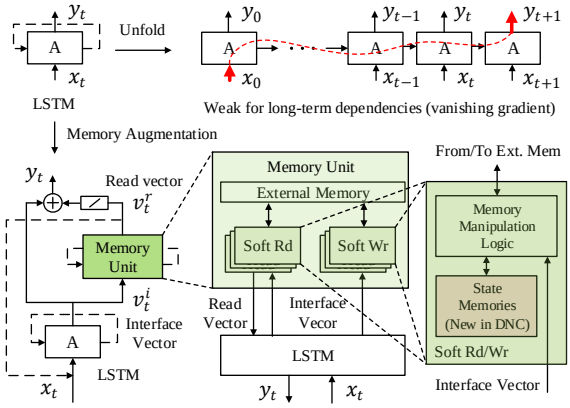


Figure 1: LSTM and memory-augmented neural network.

2 BACKGROUND

Recurrent neural networks (RNNs) [29] such as LSTMs [16] extend feedforward DNNs by introducing recurrent connections, thereby allowing the networks to store dynamic states across iterations of inputs. Let x_t be the input and y_t be the output at time t, an LSTM is composed of a chain of LSTM cells, denoted by A, as shown in Figure 1. The introduction of dynamic state has benefited domains which require remembering of event sequences, such as QA in NLP. However, the amount of information that can be stored in the network is bounded by the size of the underlying network. Therefore LSTM lacks the scalability to handle complicated sequence events with long-term dependencies.

To address the scaling problem of LSTM, MANNs have been proposed as shown in Figure 1. A memory unit is connected to an NN (typically an LSTM) and the external memory[1] can be accessed by attention-based mechanisms through soft read and soft write heads. Specifically, at time t the LSTM sends an *interface vector* v_t^i to the memory unit and receives a *read vector* v_t^r from the memory unit. In this way, the dynamic state can be explicitly decoupled from the neural network. NTM [13] is a MANN that outperforms LSTM by employing content-based soft write and read to access the memory, a form of attention mechanism. However, memory access history is completely discarded in memory slot selection and weighting. DNC [14] extends NTM and the memory is accessed based on both memory content and memory access history. History-based attention mechanisms allow DNC to achieve much better performance than NTM, but also introduce complex memory manipulations and a variety of new state memories as highlighted in Figure 1.

2.1 Differentiable Neural Computer

In this work, we focus on DNC memory unit where elaborate memory operations take place. The DNC inference dataflow and mathematical descriptions of the operations are shown in Figure 2. Compared to other variants of MANNs such as NTM or MemNet, DNC is the only model that incorporates history-based memory access. The NN (e.g., LSTM) sends an input to the memory unit, known as the *interface vector* v^i, including the necessary access information

[1]In the context of MANN, an external memory refers to a memory external to the NN (e.g., LSTM) that stores data.

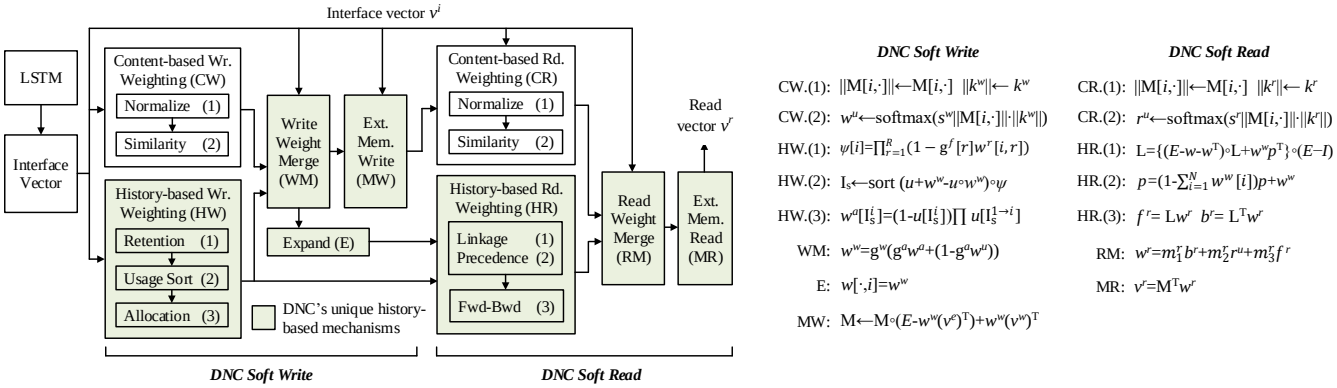


Figure 2: DNC inference dataflow: history-based and content-based memory access in DNC memory unit.

such as write key k^w, read key k^r or write vector v^w. The memory unit returns the *read vector* v^r. Suppose the memory M is modeled as a $N \times W$ matrix ($N > W$) and the number of read heads (i.e., number of parallel reads) is R, we provide a brief operational explanation of the DNC soft write and soft read.

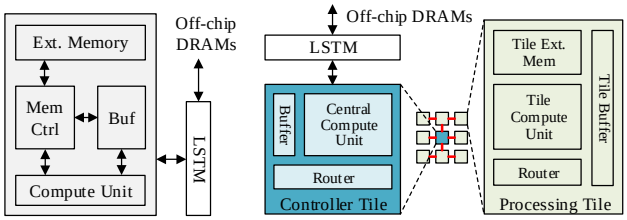
2.1.1 Soft Write. As illustrated in Figure 2, soft write is done in two steps: 1) compute write weighting w^w, and 2) memory write, i.e., apply the weighting w^w to the values (write vector v^w and erase vector v^e) and write them to memory.

In DNC, the write weighting is a combination of *content-based weighting* and *history-based weighting*. The content-based weighting is inherited from NTM, and it is based on the similarity to the write key. Mathematically, the memory entries $M[i,\cdot]$ and the write key k^w are first normalized, and the similarity between the two is computed as the content-based write weighting w^u.

The history-based weighting is brand new in DNC. DNC enhances the selection of memory cells by biasing towards those that are most recently read from (based on read weighting w^r from the previous time step), least recently written to (based on write weighting w^w from the previous time step), or deemed inconsequential (based on free gate g^f). The history-based weighting is computed in three steps: 1) the retention vector ψ is first calculated based on the free gate g^f and the read weighting w^r; 2) the *usage vector* u is updated based on the retention vector and the write weighting w^w, and then sorted; and 3) the history-based write weighting w^a is computed by accumulating the product of the sorted usage. The content-based weighting w^u and the history-based weighting w^a are combined to obtain the write weighting w^w.

2.1.2 Soft Read. Soft read is done in two steps: 1) compute read weighting w^r, and 2) memory read, i.e., apply the weighting w^r to memory M to obtain the read vector v^r.

Similar to soft write, soft read combines both content-based weighting and history-based weighting. The content-based read weighting r^u is computed in the same way as the content-based write weighting. The history-based read weighting is computed in three steps: 1) the write weighting w^w is first expanded to an $N \times N$ matrix to derive linkage matrix L. The *linkage* tracks the order in which memory locations are written to; 2) the precedence vector p is updated to track the degree each memory entry is most recently written to; and 2) a *forward and backward pass* is used to merge the read weighting w^r from the previous time step with the linkage matrix L, as well as the content-based read weighting r^u to update the read weighting w^r.



(a) Centralized architecture [4] (b) H-tree NoC architecture [33]

Figure 3: Centralized-memory architecture [4] and tiled architecture [33] for accelerating MANN's memory unit.

2.2 State-of-the-art MANN Accelerators

Conventional NN or matrix multiplication accelerators do not fully support DNC because they lack the primitives including sorting and matrix transpose, and miss a specialized memory unit to support a variety of state memories. MANN accelerators [4, 17, 27, 28, 30, 33] have been proposed for MANN's memory unit. The memory unit receives an input interface vector from an NN accelerator that executes LSTM inference. In performing the inference, the NN accelerator may communicate with off-chip DRAMs.

Here we focus on NTM and DNC accelerators that support both soft write and soft read and omit simpler accelerators that do not come with such support. Some NTM and DNC accelerators use a centralized-memory architecture [4, 28, 30] as shown in Figure 3(a). In particular, [4, 28] improve the memory access efficiency by introducing in-memory compute through resistive crossbar or analog operations. However, the centralized memory ultimately limits the bandwidth and parallelism available, and the emerging devices and custom mixed-signal circuits are not yet practical for sufficiently large memory sizes.

MANNA [33] introduces the first tiled NoC architecture as shown in Figure 3(b) to solve the bandwidth and parallelism limitation.

Table 1: Analysis of DNC Kernels

Type	Category	Kernel Name	Key Primitives	Ext. Mem Access	State Mem Access	Total NoC Traffic
Access Kernels	Content-based Weighting	Normalize	inner-prod	$O(NW)$	$O(W)$	$O(N_tN)$
		Similarity	inner-prod	$O(NW)$	$O(W)$	$O(N_t)$
	Memory Access	Memory Write	el-add/sub/mult, outer-prod	$O(NW)$	$O(N)$	$O(N_tN)$
		Memory Read	transpose, mat-vec mult	$O(NW)$	$O(N)$	$O(N_tNW)$
State Kernels (*New in DNC*)	History-based Write Weighting	Retention	el-mult, vec acc-prod	No	$O(RN)$	No
		Usage	el-add/sub/mult	No	$O(N)$	No
		Usage Sort	sort (Section 4.3)	No	$O(N)$	$O(N)$
		Allocation	vec acc-prod	No	$O(N)$	$O(N_t)$
		Wr. Weight Merge	el-add/sub	No	$O(N)$	No
	History-based Read Weighting	Linkage	mat expand, outer-prod, el-add/sub/mult	No	$O(N^2)$	$O(N_tN)$
		Precedence	el-add, vec acc-sum	No	$O(N)$	$O(N_t)$
		Forward-backward	transpose, mat-vec mult	No	$O(N^2)$	$O(N_tN^2)$
		Rd. Weight Merge	el-add	No	$O(RN)$	No

Manna contains two types of tiles: processing tile (PT) which includes external memory sub-banks and the associated compute units, and controller tile (CT) which includes top-level processing units to distribute information to the PTs and collect results from the PTs. Designed for NTM, Manna cannot run DNC due to the lack of support for new primitives like sorting and new state memories for maintaining access history. Manna's H-tree NoC also becomes less efficient in carrying inter-tile communication when the PT count increases. The inefficiency is exacerbated by DNC's history-based attention mechanisms that inject complex traffic patterns onto the NoC, limiting its scalability as seen in Figure 5(d).

3 ANALYSIS OF DNC KERNELS

We first analyze DNC's memory access and computational profile, followed by a simulation study of DNC running bAbI dataset [37] on CPU and GPU.

3.1 Theoretical Kernel Analysis

Table 1 lists DNC's computational kernels with their corresponding primitives, associated memory access complexity and the NoC traffic condition when mapped to a tiled architecture. Recall that the external memory M is modeled as a $N \times W$ matrix ($N > W$) and the number of read heads is R. Let N_t be the number of tiles in a tiled architecture. We categorize DNC kernels into two types: 1) state kernels for maintaining memory states and determining how the external memory is accessed, and 2) access kernels that do not maintain states and perform the actual access to the external memory. NTM only needs access kernels, while DNC requires a variety of new state kernels to support history-based mechanisms. These new state kernels impose critical challenges:

- *Computation*: Kernels such as usage sort and forward-backward require compute-intense large-scale data sorting of $O(N\log N)$ complexity (assume merge sort) or matrix-vector multiplication of $O(N^2)$ complexity, which can be major bottlenecks.
- *Memory Access*: Kernels such as linkage and forward-backward require $O(N^2)$ accesses to the new linkage memory, which easily surpass the memory access by the access kernels used by other MANNs like NTM.
- *NoC Traffic*: In a tiled architecture, some kernels such as linkage rely on inter-tile traffic. The amount of NoC traffic can be as high as $O(N_tN^2)$ depending on state and external memory partitions, and the traffic pattern is non-uniform over time and space due to the different primitives and state memories involved.

The challenges require designing an efficient NoC and memory organization, maximizing distributed processing, and providing efficient computational kernels such as sort.

3.2 Kernel Runtime Analysis

We simulated DNC inference on an Nvidia 3080Ti GPU and an Intel Core i7-9700K CPU using the bAbI dataset [37] in NLP. The bAbI dataset consists of 20 tasks. Each task is independent of the others, tests one aspect of an intended NLP behavior and includes as many as 10,000 QA examples. It is the only publicly available and practically meaningful dataset to date to demonstrate DNC's performance. In our experiments, we ran tasks in the bAbI dataset and recorded the average runtime. Figure 4 captures runtime breakdown of DNC kernels in different categories. The average GPU inference time is 5.16 ms/test, 2.12× faster than the 10.94 ms/test inference time on the CPU. On both the GPU and the CPU, the NN (an LSTM) takes less than 5% of the total runtime, while the memory unit takes more than 95% of the total runtime. *It highlights the need of a memory access engine for DNC.* Additional insights can be derived from the kernel runtime shown in Figure 4.

- History-based write weighting, including retention, usage sort and allocation, accounts for 72% of the runtime on the GPU. A possible explanation is that GPU is not well suited for speeding up large-scale sorting.

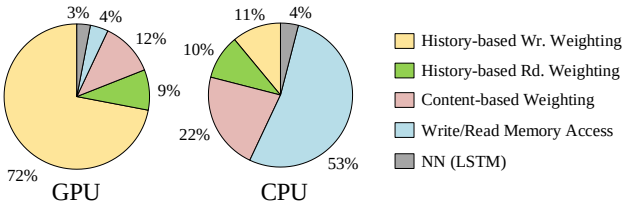


Figure 4: Kernel runtime breakdown on CPU/GPU for the bAbI dataset. The external memory size is $N \times W = 1024 \times 64$ and the LSTM is 1-layer of size 256.

- History-based read weighting, including linkage and precedence, relies on vector and matrix operations that can be extensively parallelized by the GPU. This part uses only 9% of the GPU's runtime.
- Content-based write/read weighting, including normalization and similarity, involves many multiply-accumulate (MAC) and softmax operations. It costs 12% of the runtime on the GPU and 22% on the CPU, but it is not the dominant part on either platform.
- Memory write and read[2] are much faster on the GPU (4% of the runtime) than on the CPU (53% of the runtime), because they are dominated by parallelizable weighting operations using MAC arrays.

The results show that improving the DNC performance requires both optimized computational kernels such as sort and highly parallel matrix operations. Another consideration is that GPU and CPU follow a centralized-memory architecture and high premiums in power and area are paid in sustaining a high-bandwidth interface and a versatile memory hierarchy. In designing an accelerator targeting high performance, high energy efficiency and low cost, a distributed, tiled architecture is the preferred approach.

4 HIMA ARCHITECTURE DESIGN

HiMA is a memory access engine that follows a distributed, tiled architecture that consists of one CT and many PTs with an NoC linking the tiles. Based on kernel analysis and simulation results, we summarize HiMA's architectural design goals:

- *Scalable and versatile NoC*: A fixed NoC is sub-optimal for the complex NoC traffic by DNC's unique state kernels. The H-tree NoC in [33] suffers from traffic saturation with more than 8 tiles as shown in Figure 5(d). A more scalable NoC that supports DNC's versatile kernels is desired.
- *Efficient memory partition*: DNC's external memory is large, and its state memories can be even larger. For example, the linkage matrix requires a memory of $N \times N$ ($N > W$). A strategy is needed to partition large external and state memories and distribute them to tiles with the goal of minimizing the NoC traffic amount when executing DNC kernels.
- *Distributed compute kernels*: DNC's kernels are ideally distributed to the tiles. The sort kernel is especially important. It is a performance bottleneck and it is not supported by existing NN/MANN accelerators [1, 5, 17, 20, 21, 26–28, 31, 33]. The goal is to distribute such kernels along with memory to tiles, while minimizing the NoC traffic amount.

[2]In the context of DNC, memory write and memory read are not simply write to or read from memory. Applying write weighting to data before write to memory and applying read weighting to data read from memory are the dominant operations in memory write and memory read, respectively.

4.1 Scalable Multi-Mode NoC

Reconfigurable NoCs have been proposed for DNN accelerators for different tensor sizes. Specifically, Maeri [22] and Herald [21] employ a multi-layer binary tree as show in Figure 5(a) with configurable interconnects between adjacent sub-trees at each level. These designs are suitable for DNN's reduction, collection and multi-cast dataflows. They are however sub-optimal for DNC's diverse traffic patterns, especially for transpose and matrix-vector multiplication that require communications between distant tiles. Data transfer between two distant tiles may need to pass through their mutual root tile, which can become a traffic congestion point that drastically increases the inter-tile communication latency. The H-tree NoC demonstrated by Manna [33] was designed for access kernels that require only two types of inter-tile communication: 1) broadcast of interface vectors from CT to PTs, and collection of read vectors from the PTs; and 2) transfer of submatrices of the external memory and partial sums between PTs for transpose and matrix-vector multiplication. In implementing the access kernels, the H-tree NoC does not present a bottleneck up to 16 tiles [33].

For history-based memory access, the new state kernels introduce far more inter-tile traffic and a diverse traffic profile. To see the suitability of the H-tree NoC and the multi-layer binary tree, we mapped DNC to a tiled architecture utilizing these two NoCs. To check scalability, we simulated the speedup by increasing the number of PTs. Here we assume ideal CT and PTs where the memory bandwidth or the computational parallelism do not present bottlenecks, and ideal routers that can handle any traffic congestion by stalling. Figure 5(d) shows that the speedup starts to saturate beyond the 8 tiles for both NoCs. The H-tree NoC [33] requires traffic between two tiles to go through their mutual root tile, as show in Figure 5(b), resulting in traffic congestion at highest root node for the distant pairs of PTs. The binary-tree NoC [22] is an enhanced H-tree by additional interconnects between adjacent sub-trees at each level, as shown in Figure 5(a). It outperforms the H-tree slightly, but its scalability still saturates at a low level. We analyze the traffic profile of DNC primitives and the suitable NoC topologies.

- Interface vector broadcast, read vector collection and sorting require only CT-PT traffic. A star NoC is the most suitable, where all PTs are connected directly to the CT with a distance of 1 hop. However, the CT needs a complex router and can become a traffic congestion point with increasing PT count, limiting scalability.
- Accumulation of products or sums and vector inner product require sending accumulated results from one PT to the next PT. A ring NoC is the most suitable.
- Matrix transpose requires transferring on-tile submatrices to other tiles along diagonals, as shown in Figure 5(c). A diagonally-connected NoC is suitable.
- Matrix vector multiplication and vector outer product require each tile to send its local submatrices to all other tiles

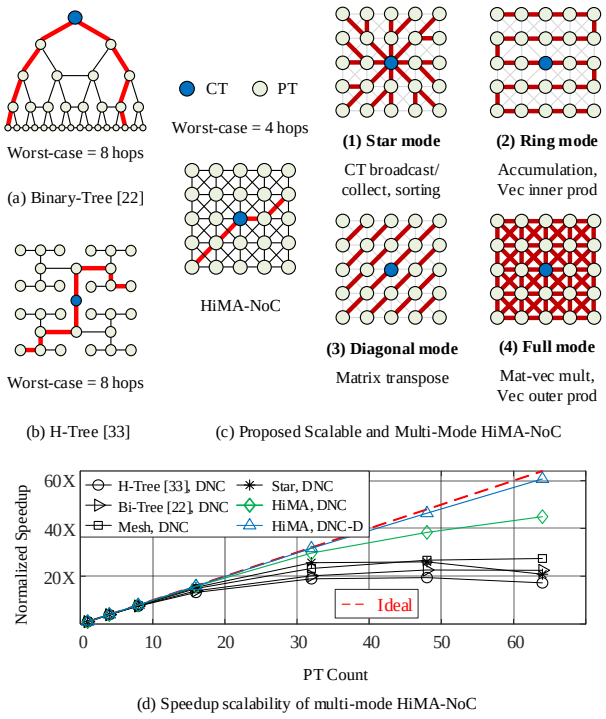


Figure 5: Scalable and Multi-Mode HiMA-NoC.

for computation. A full-duplex mesh NoC is the most suitable. However, the scalability of a full-duplex mesh is even worse than the star NoC, because every tile, not only CT, can become a traffic congestion point.

The analysis shows that a fixed NoC topology does not meet DNC's diverse traffic profile. We propose a multi-mode NoC, named HiMA-NoC, to shorten the transfer distance, reduce the traffic congestion and enhance the scalability. Figure 5(c) shows an example HiMA-NoC for 5×5 tiles. It is made by adding diagonal connections in a mesh NoC. The worst-case inter-tile transfer distance is kept to 4 hops in the 5×5 example. Compared to a fixed NoC that improves traffic conditions for some primitives but worsens for others, HiMA-NoC can be configured in run time to efficiently support different traffic patterns through multi-mode routers (Section 6). Using the same simulation setup, HiMA-NoC provides a more scalable speedup than the fixed H-tree, mesh or star NoC, as shown in Figure 5(d). Note that HiMA-NoC does not reduce the amount of traffic, but enhances the tile-to-tile communication latency.

4.2 Submatrix-wise Memory Partition

DNC's external memory and state memories need to be partitioned and distributed to the tiles. The partition affects the available access bandwidth, the achievable compute parallelism, and the amount and the patterns of inter-tile data communication. SIMBA[31], a state-of-the-art distributed NN accelerator, distributes weights to tiles and efficiently supports convolution and FC workloads for DNNs. MANNA [33] partitions external memory row-wise so that each PT receives N/N_t rows of the external memory, where N_t is the number of PTs. MANNA does not support state memories.

HiMA's memory partitions are designed for external memory and new state memories that are nonexistent in NNs or other variants of MANNs like NTM. DNC requires access to various memories concurrently, and the traffic patterns are non-uniform depending on the primitives that are running. There need to be more considerations on memory partition to reduce the amount of traffic.

4.2.1 External Memory Partition. The external memory is accessed by the access kernels, first in computing content-based weighting including normalization and similarity, followed by memory write or read. Figure 6(a) illustrates three possible external memory partitions and the inter-tile traffic patterns in computing content-based weighting for a small example of $N_t = 4$ tiles.

The row-wise partition of the external memory eliminates inter-tile transfer for normalization because normalization is computed on a row of memory, which are stored in the same PT. When calculating similarity, one PT produces only a partial sum (psum), and it collects the psums from the rest of $N_t - 1$ PTs to compute the global sum, followed by scaling and softmax. The softmax result is then distributed to the $N_t - 1$ PTs. Hence the number of inter-tile transfers is $2(N_t - 1)$. Alternatively, if we follow the column-wise partition of the external memory, normalization requires $2N(N_t - 1)$ inter-tile transfers, but similarity can be computed locally.

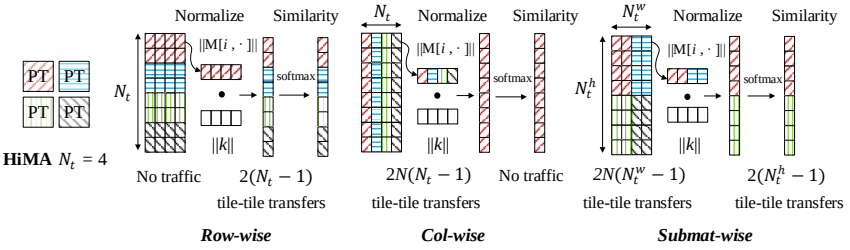
The row-wise and column-wise partitions can be viewed as special cases of a generalized submatrix-wise partition where the external memory is divided into N_t^h block rows and N_t^w block columns, where $N_t = N_t^h \times N_t^w$. As shown in Figure 6(a), using submatrix-wise partition, normalization and similarity calculations cost $2N(N_t^w - 1)$ and $2(N_t^h - 1)$ inter-tile transfers, respectively. Based on Eq. (1) and given $N \gg N_t$, to minimize the inter-tile traffic, $N_t^w = 1$ and $N_t^h = N_t$. In other words, *the row-wise partition of the external memory costs the minimum inter-tile traffic in computing content-based weighting.*

$$\underset{N_t^h, N_t^w}{\operatorname{argmin}} \left(2N(N_t^w - 1) + 2(N_t^h - 1) \right) \tag{1}$$

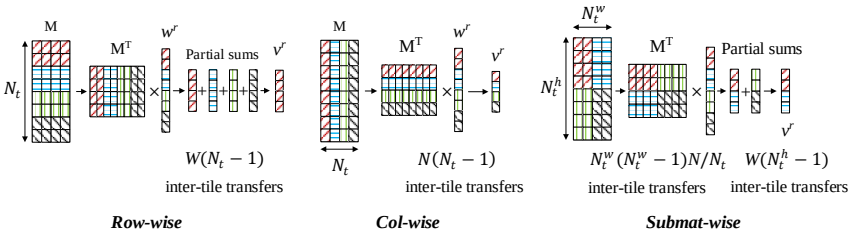
The memory write to the external memory requires element-wise operations that can be executed locally by PTs in parallel. The memory read from the external memory requires inter-tile traffic to support matrix transpose and matrix-vector multiplication. Figure 6(b) illustrates the inter-tile transfer patterns. Similarly, the optimal partition for matrix transpose and matrix-vector multiplication can be derived as Eq. (2).

$$\underset{N_t^h, N_t^w}{\operatorname{argmin}} \left(\frac{N_t^w (N_t^w - 1) N}{N_t} + W(N_t^h - 1) \right) \tag{2}$$

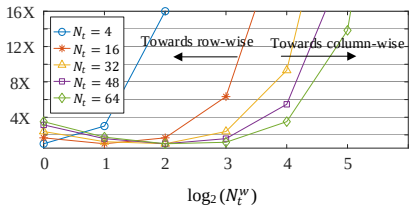
Figure 6(c) illustrates the external memory partition choices and the impact on inter-tile traffic of memory read kernel for a $N \times W = 1024 \times 64$ used in running the bAbI dataset. The five sets of curves correspond to different number of tiles N_t, and they cover a range of N_t^w choices. Due to the quadratic dependence on N_t^w, N_t^w should generally be kept low to reduce the inter-tile transfers. Therefore, *the row-wise partition of the external memory is advantageous for minimizing the inter-tile traffic in memory read.*



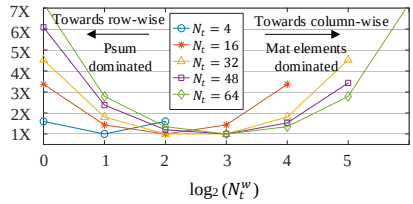
(a) Normalization and similarity in content-based weighting



(b) Transpose and matrix-vector multiplication in history-based weighting and soft read

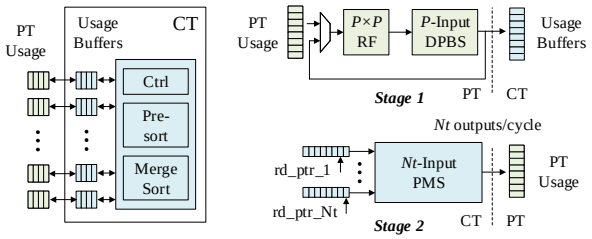


(c) Normalized traffic amount for memory read kernel and external memory partition for various Nt



(d) Normalized traffic amount for forward-backward kernel and submat-wise linkage memory partition for various Nt

Figure 6: External memory partition in a 2×2 tile for (a) content-based weighting, and (b) transpose and matrix-vector multiplication in history-based weighting and soft read; (c) inter-tile traffic for memory read kernel with various external memory partition; (d) inter-tile traffic for forward-backward kernel with various linkage memory partition.



(a) Centralized sort [4] (b) Proposed two-stage usage sort

Figure 7: Two-stage usage sort.

4.2.2 State Memory Partition. The state kernels require a set of state memories: usage, linkage, precedence, write weight and read weight. State memories of size N (usage, precedence, write weight) or $N \times R$ (read weight) can be straightforwardly partitioned to N/N_t or $N/N_t \times R$ parts and distributed to the N_t PTs.

The linkage memory, on the other hand, has a size of $N \times N$. The linkage memory is used by the forward-backward kernel in computing matrix transpose and matrix-vector multiplication. The inter-tile transfer patterns look similar to Figure 6(b), except that the input matrix is $N \times N$ instead of $N \times W$. Similarly, we find the number of inter-tile transfers based on the generalized submatrix-wise partition and formulate the optimization in Eq. (3).

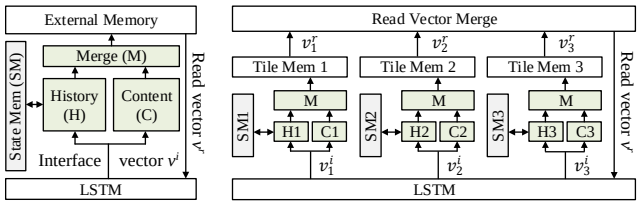
$$\underset{N_t^h, N_t^w}{\operatorname{argmin}} \left(\underbrace{\frac{N_t^h(N_t^h-1)}{N_t} + N_t^w}_{\text{Forward}} + \underbrace{\frac{N_t^w(N_t^w-1)}{N_t} + N_t^h}_{\text{Backward}} \right) \quad (3)$$

The partition choices and the impact on inter-tile traffic of forward-backward kernel are plotted in Figure 6(d) for a $N \times W = 1024 \times 64$ external memory. The results show that both the low-end of N_t^w (corresponds to row-wise partition, or transfer psums) and the high-end of N_t^w (corresponds to column-wise partition, or transfer matrix elements) are suboptimal. The minimum inter-tile traffic is in between. As an example, for $N_t = 16$, the optimal submatrix partition for the linkage memory is $N_t^h \times N_t^w = 4 \times 4$.

4.3 Two-Stage Usage Sort

Usage vector sort is a bottleneck primitive. In HiMA, the usage vector is distributed and stored in parts on the PTs. A conventional solution using centralized merge sort [4], as shown in Figure 7(a), takes $N\log N$ cycles for a length-N usage vector. To achieve a lower latency, we propose a local-global two-stage sort for the distributed tile architecture: 1) a local usage vector of size $n = N/N_t$ is first sorted by each PT, 2) global merge sort by CT to combine N_t sorted local usage vectors.

We illustrate the two-stage sort for an example of $N_t = 4$ tiles and an external memory of $N = 1024$ rows. Each PT keeps a local usage vector of length $n = 256$. In stage 1 in each PT, a local usage vector is reshaped into a $P \times P$ matrix where $P = \lceil\sqrt{n}\rceil = 16$. We apply a fast multi-dimensional sorting algorithm (MDSA) [24] to complete the local sort in only 6 phases. The 2D MDSA sorter is illustrated in Figure 7(b), which is composed of a $P \times P$ register file (RF) and a P-input dual-mode pipelined bitonic sorter (DPBS) [24] supporting both ascending and descending order. The 16-input DPBS can be pipelined into $D_{DPBS} = 5$ stages. A length $n = 256$ local usage vector can be sorted in only $6 \times (P + D_{DPBS}) = 126$ cycles.



(a) Classic DNC (b) Distributed Execution of DNC-D

Figure 8: Illustration of memory operation in DNC and distributed execution in DNC-D.

In stage 2, sorted local usage vectors are sent from PTs to CT for global merge sort. CT utilizes N_t memory banks to store the local usage vectors. We apply an N_t-input parallel merge sorter (PMS) [23] in CT to support N_t outputs per cycle, which are to be written back to the corresponding PTs as shown in Figure 7(b). The pointers are updated to keep track of the status of each memory bank. The 4-input PMS can be pipelined into $D_{PMS} = 7$ stages. The global merge sort for the example $N_t = 4$ takes only $n + D_{PMS} = 263$ cycles. With the proposed local-global two-stage sort, usage sort computation latency is reduced to only $6 \times (P + D_{DPBS}) + n + D_{PMS} = 389$ cycles compared to $N\log N$ cycles for the centralized merge sort.

5 ALGORITHMIC TECHNIQUES

A distributed, tiled architecture provides more opportunities for parallel processing. However, only a subset of DNC primitives like element-wise operations can take the full advantage of distributed processing, while most of the primitives need to operate on the entire external memory or the entire state memories, resulting in excessive traffic and limited hardware scalability. We aim to distribute most of the processing to individual PTs by presenting a distributed version of the DNC model named DNC-D, while minimizing the accuracy loss over DNC.

5.1 Distributed Execution for DNC

In DNC, the LSTM provides an interface vector to the memory unit as the input and receives a read vector as the output. As shown in Figure 8, in DNC-D the LSTM provides a sub interface vector to each distributed PT instead of broadcasting one global interface vector to all the PTs. Soft read and soft write are executed locally on each PT's local portion of the external memory and state memories.

DNC-D could degrade the inference accuracy. To minimize the loss, we introduce a weighted sum to merge the N_t output read vectors v_i^r from the PTs, where $i \in \{1, ..., N_t\}$, and compute the final read vector v^r as output to the LSTM as (4) below:

$$v^r = \sum_{i=1}^{N_t} \alpha_i^r v_i^r, \tag{4}$$

where the trainable weights $\alpha_i^r \in [0, 1]$ are determined by the LSTM. The distributed execution offers several advantages: 1) it eliminates the inter-PT communication, 2) it reduces the computations on PTs related to inter-PT data, and 3) it removes the global sort. Without any inter-PT traffic, HiMA achieves nearly optimal speedup scaling as shown in Figure 5(d). In Section 7, we study the improved hardware efficiency and the accuracy loss of DNC-D.

5.2 Approximation Techniques

We introduce two optional approximation techniques to further reduce the compute complexity.

Usage Skimming: The usage vectors are collected in computing the write allocation. In practice, we observe that the least significant usage entries have little effect on computation of the write allocation. We propose usage skimming to discard the K smallest usage entries. Usage skimming reduces the complexity of usage sort and write allocation proportionally. In Section 7, we evaluate the inference accuracy impact of usage skimming and show the hardware efficiency improvements.

Softmax Approximation: Softmax is a timing-critical compute block. State-of-the-art softmax approximations include look-up-table (LUT) based [18] or piece-wise linear approximation (PLA) based [9] exponential function. The drawback of the LUT-based is that the number of entries in the table increases exponentially with the input bit width. We combine PLA and LUT approaches: we apply the PLA-based approximation with a small number of line pieces, each of which is an affine function with a slope; and we utilize a LUT of affine functions to store the corresponding function parameters. The design costs only 1 multiply and 1 add.

6 HIMA PROTOTYPE IMPLEMENTATION

Putting everything together, the HiMA architecture is depicted in Figure 9. It is composed of a CT with surrounding PTs connected via HiMA-NoC. HiMA incorporates all the architectural features and the optional algorithmic features.

Controller Tile: CT contains the LSTM and it also executes kernels that require global-level processing. An LSTM implementation employed by [33] is used in this work and it handles the LSTM inference and communication with off-chip memories. The CT design is illustrated in Figure 9. It sends the interface vectors to the PTs and collects the read vectors from the PTs through routers. The global usage buffers and merge sorter are employed for the 2nd-stage usage sort. Note that using DNC-D, the distributed DNC model, the 2nd-stage usage sort can be eliminated for smaller area.

Processing Tile: A PT's memory system consists of an external memory bank and state memory banks for linkage, precedence, usage, read weighting and writing weighting. The memory partitions are determined based on the submatrix-wise partitions. PT's compute modules support vector and matrix operations for the primitives outlined in Table 1. Two matrix buffers hold the data for processing from the on-PT memories, the PT router or the interface collector. A matrix buffer loader is used to format and store the data to the corresponding buffers. A matrix-matrix engine (M-M engine) is developed to perform matrix and vector operations. The M-M engine is made of an array of processing elements (PEs) with a configurable processing tree (CPT) to support different sizes of vectors and matrices.

Each PE consists of a small RF to hold the intermediate values. The PE design supports bypass, add, multiply, multiply-then-add or add-then-multiply modes. The CPT consists of multiple stages of compute cells (CPT cells) including adders, multipliers, special function units (SFUs) and bypass routes. It follows a binary tree for reduction and enables faster accumulation. PT also includes a length-N/N_t MDSA sorter for on-tile usage sorting. The proposed

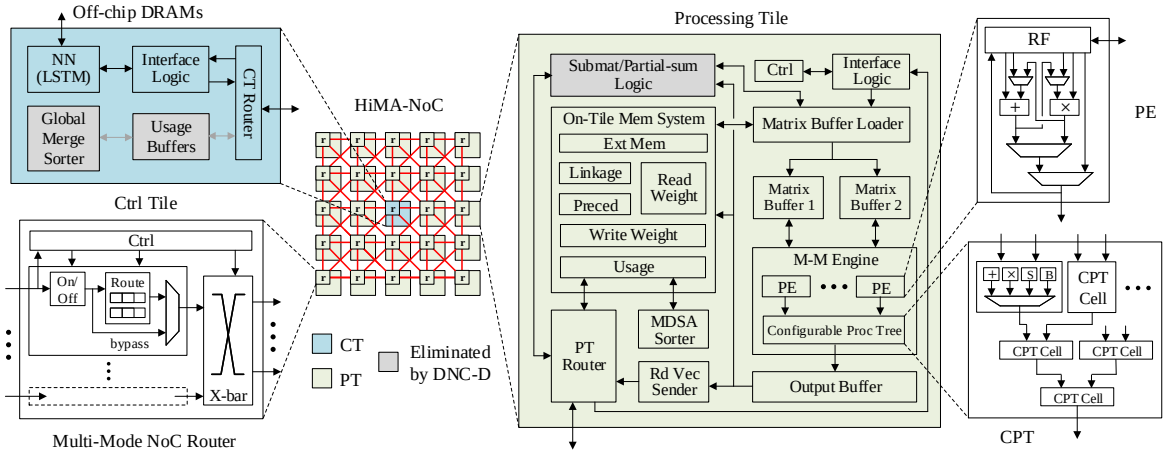


Figure 9: HiMA architecture and CT/PT designs.

architecture provides parallelism through the PE array and the multi-entry RF inside a PE. The size of PE array, the depth of RF inside PE, and the number of CPT stages can be scaled up to provide a higher degree of parallelism.

Multi-Mode NoC Router: Figure 9 illustrates the 8-way multi-mode router that supports different HiMA-NoC modes specified in Figure 5. In addition to the conventional router logic and buffers, input/output ports are controlled by on/off switches to enable traffic only in certain directions. For example, only the east/west ports are enabled for the inner tiles in the ring mode; and only the northeast/southwest ports are enabled in the diagonal mode. Feed-through single-cycle transfer is enabled when the input buffer in the forward direction is empty, bypassing router logic and reducing the latency for non-congested tiles. The multi-mode router is implemented by route LUTs specifically designed to support the proposed modes and a controller that monitors the buffer conditions and generates control signals for each mode.

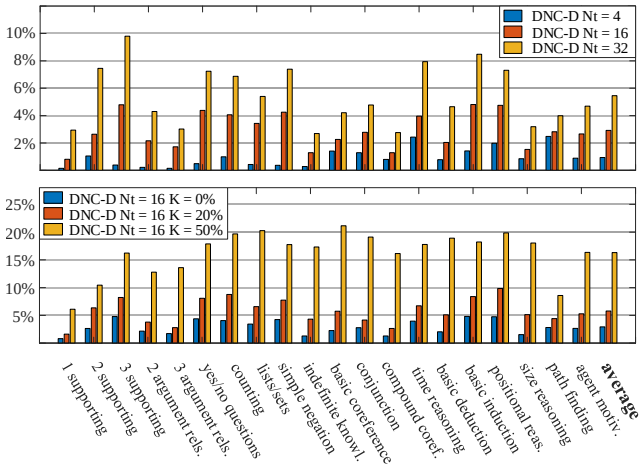


Figure 10: DNC-D inference error over DNC for 20 tasks of the bAbI dataset.

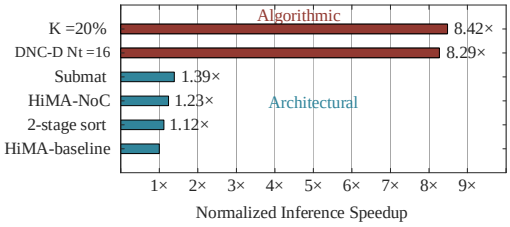
7 EVALUATIONS AND BENCHMARKING

We developed a parameterized RTL simulator for HiMA to evaluate its silicon area, inference speed and power consumption. All designs utilize a 32-bit precision for a fair comparison with state-of-the-art MANN accelerators [4, 33]. We verified the designs against a functional model of DNC in Python at kernel level as well as system level. To estimate area, we synthesized designs at a 500 MHz clock frequency in a 40nm CMOS technology. We used Ansys PowerArtist to obtain power measurements of executing DNC kernels based on switching activities.

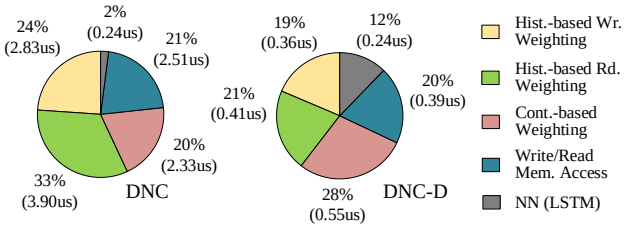
The HiMA-baseline architecture employs the H-tree NoC used in [33]. Proposed architectural features, including HiMA NoC, the optimized submatrix-wise memory partition and the two-stage usage sort, are incorporated in the optimized HiMA architectures. HiMA can be further enhanced by the DNC-D model, the usage skimming and the softmax approximation. Based on the architectural and algorithmic features, we create two HiMA architectural prototypes, HiMA-DNC that runs DNC and HiMA-DNC-D that runs DNC-D. Each prototype is equipped with $N_t = 16$ PTs and 1 CT, and supports an external memory of size up to $N \times W = 1024 \times 64$ for processing the bAbI dataset.

7.1 Inference Accuracy

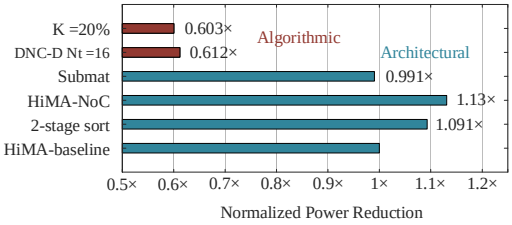
To evaluate the inference accuracy of the DNC-D model, we performed simulations using the bAbI dataset and report the error rates over DNC across 20 benchmark tasks in Figure 10. The error rate of the DNC-D model increases with the number of distributed tiles N_t. If N_t is capped at 32, the average error rate of DNC-D is kept below 6% over DNC. With a usage skimming rate of $K = 20\%$ and $N_t = 16$, DNC-D demonstrates an error rate of 5.8% higher than DNC. Further increasing the skimming rate to 50% increases the error rate above 15% over DNC. The proposed algorithmic features trade inference accuracy for a higher hardware efficiency. One can select N_t based on the accuracy tolerance. Parameters used in approximations can be selected based on simulations.



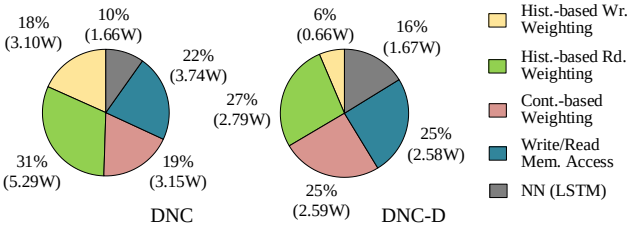
(a) Speedup breakdown with architectural and algorithmic optimizations



(b) Kernel runtime breakdown of HiMA running DNC and DNC-D



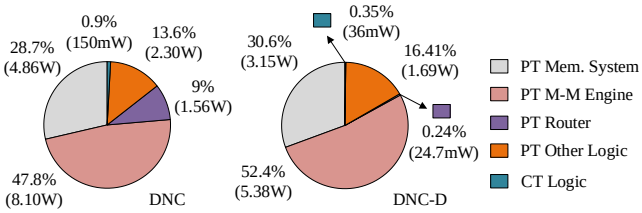
(c) Power reduction with architectural and algorithmic optimizations



(d) Kernel power breakdown of HiMA running DNC and DNC-D

Area (mm^2)	HiMA baseline	HiMA DNC	HiMA DNC-D
PT	4.92	5.01	4.22
PT Mem	2.07	2.07	1.53
CT	0.43	0.52	0.18
Total	79.14	80.69	67.71
Power (W)	16.80	16.96	10.28

(e) HiMA silicon area and power with a 40nm CMOS technology



(f) Module power breakdown of HiMA running DNC and DNC-D

Figure 11: HiMA speed, silicon area and power (N_t = 16).

7.2 Inference Speed

Figure 11(a) itemizes the inference speedup after steps of architectural optimizations over a HiMA-baseline (N_t = 16): 1) the two-stage sort provides a 1.12× speedup over the HiMA-baseline; 2) replacing the H-tree NoC in the HiMA-baseline by the multi-mode HiMA-NoC reduces the communication latency and improves the inference speed to 1.23× over the baseline. The improvement is mainly due to run time savings of traffic-intensive kernels involving matrix transpose and matrix-vector multiplications, such as linkage, forward-backward and memory read; 3) applying the submatrix-wise partition increases the inference speed to 1.39× over the baseline, where the speedup is mainly attributed to the reduced traffic amount. These improvements are based on architectural features only. The architecturally optimized HiMA-DNC achieves an inference time of 11.8 μs per test. Figure 11(b) shows the kernel run time breakdown in executing DNC. History-based write weighting and read weighting are the most significant, taking 24% and 33% of the run time, respectively.

To further improve the speed, we can apply DNC-D with distributed execution (N_t = 16). HiMA-DNC-D achieves a 8.3× inference speedup over the baseline as shown in Figure 11(a). The improvement is due to several factors: 1) the elimination of all inter-PT traffic, 2) the computation reduction on PTs, and 3) the elimination of global usage sort. Applying a K = 20% usage skimming and the softmax approximation increases the inference speedup to 8.4× over the baseline. The architecturally and algorithmically optimized HiMA-DNC-D with K = 20% usage skimming shortens the inference time to 1.95 μs per test. As shown in Figure 11(b), the run time for history-based write weighting and read weighting in DNC-D are reduced by 87% and 89% compared to the run time in DNC, respectively.

7.3 Silicon Area and Power

Both HiMA-DNC and HiMA-DNC-D prototypes contain N_t = 16 PTs, and they implement all the architectural features. Additionally, HiMA-DNC-D employs a simpler PT and CT due to the elimination of the inter-PT communication and the associated global processing. Figure 11(e) compares the silicon area and power consumption of HiMA-DNC and HiMA-DNC-D to HiMA-baseline. HiMA-DNC has a PT area of 5.01 mm^2. The architectural features cost an overhead of 1.8% for the PT over the baseline PT. PT's memory system occupies 2.07 mm^2, including an external memory of 16.4 KB, a linkage memory of 262 KB and multiple 256 B state memories. The linkage memory and the external memory account for 81.3% and 4.8% of the PT memory area, respectively.

Figure 11(c) itemizes the power impact of architectural features: 1) the two-stage sort adds 9% power over the baseline due to the introduction of local sorters in each PT; 2) adopting the multi-mode HiMA-NoC increases the power by another 4%; 3) applying the submatrix-wise partition reduces the total power to 0.9% below the baseline, where the power saving comes from the reduced data

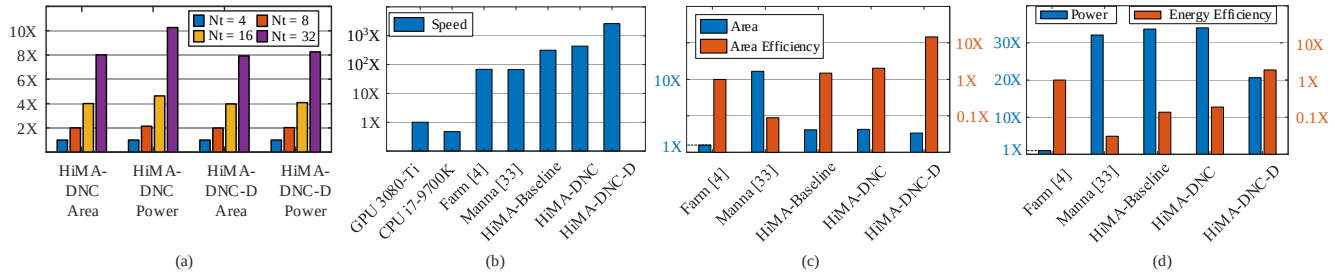


Figure 12: (a) Area and power scalability of HiMA-DNC and HiMA-DNC-D to support more tiles and larger external memory; (b)-(d) speed, area and power comparison of HiMA ($N_t = 16$) with state-of-the-art MANN accelerators and GPU/CPU (area efficiency is measured by throughput/area, and energy efficiency is measured by throughput/power).

movement. In all, HiMA-DNC consumes 16.96 W for running a complete DNC inference.

HiMA-DNC-D has a smaller on-PT linkage memory and the centralized sorter is eliminated in CT. It results in a reduced PT area of 4.22 mm^2 and a reduced CT area of 0.18 mm^2. HiMA-DNC-D employs a simpler router that only supports CT-PT traffic as DNC-D eliminates all inter-PT traffic. HiMA-DNC-D uses 16.1% less silicon area and consumes 39.4% less power than HiMA-DNC.

Figure 11(d) and Figure 11(f) show the kernel and module power breakdown. Notably, DNC-D reduces the power of history-based write weighting by 79% due to the elimination of global usage sort in CT and the usage transfers between CT and PTs. DNC-D also cuts 98.4% of the router power because of the elimination of all inter-PT traffic. Since DNC-D allows PT to compute based only on local memories, the computation and traffic reduction result in power savings across all relevant kernels and modules.

HiMA can be scaled up with more tiles to support a larger external memory and a higher degree of parallelism. As shown in Figure 12(a), the power of HiMA-DNC grows super-linearly with N_t mainly because of the increased traffic and the related computations on each PT, while DNC-D improves the power scalability close to the ideal (linear) scaling.

7.4 Comparison with State-of-the-Art Designs

Figure 12(b) compares HiMA's performance to the state-of-the-art MANN accelerators as well as an Nvidia 3080-Ti GPU and an Intel Core i7-9700K CPU. The speedup is normalized to the GPU. Figure 12(c) and Figure 12(d) compare HiMA's area and power to the MANN accelerators. GPU and CPU are omitted in area and power comparisons since it would be unfair to compare area and power of an accelerator to general-purpose computing platforms. The area and power are normalized to Farm [4]. The area is also normalized based on each design's process technology.

Farm achieves a 68.5× faster speed over the GPU. Farm's faster speed is mainly attributed to its small memory size (up to $N = 256$) and mixed-signal designs. However, Farm's centralized-memory architecture is not scalable to a larger size to support practical problems and the mixed-signal computation is not yet feasible at a large enough scale. The 16-tile NTM accelerator Manna [33] utilizes an H-tree NoC. It achieves a similar speedup as Farm, but it costs 11× area and 32× power to support 20× larger external memory than Farm. Manna still cannot run DNC due to the lack of support for history-based memory access.

HiMA-baseline uses the same H-tree NoC as Manna and it supports DNC's history-based memory access. It has a 4× larger external memory than Farm while using only 3.16× the area of Farm. HiMA-baseline consumes a higher power than Manna to support DNC's history-based mechanisms. HiMA-DNC achieves a 1.39× faster speed over HiMA-baseline thanks to the architectural features. The overhead of the architectural features is almost negligible, which explains why HiMA-DNC uses similar area and power as HiMA-baseline. HiMA-DNC-D takes advantage of the DNC-D model to increase the speed by 8.4× over HiMA-baseline and reduces the area by 14.4% and power by 38.8% over HiMA-baseline. Compared to Manna that was designed in a 15nm technology, the 40nm HiMA-DNC-D demonstrates 39.1× faster speed, 164.3× better area efficiency and 61.2× better energy efficiency.

8 CONCLUSION

We present HiMA, a distributed, tile-based accelerator, to efficiently speed up history-based memory access for advanced MANN models like DNC. A multi-mode NoC is designed to support different traffic patterns and improve latency and scalability. Submatrix-wise memory partition is developed to minimize the amount of data movements. To achieve better hardware efficiency, we leverage the tiled architecture to design a two-stage usage sort. To fundamentally improve the efficiency of HiMA's distributed architecture, we distribute not only memory, but also memory operations to the tiles in the form of a new DNC-D model. The HiMA compute kernels can be further optimized by skimming insignificant usage entries and applying an efficient approximation to the softmax function.

We create two HiMA architectural prototypes: HiMA-DNC that runs DNC and HiMA-DNC-D that runs DNC-D. The results show that HiMA-DNC and HiMA-DNC-D achieve 6.47× and 39.1× higher speed, 22.8× and 164.3× better area efficiency, and 6.1× and 61.2× better energy efficiency than Manna, the state-of-the-art tiled MANN accelerator for NTM. Compared to an Nvidia 3080Ti GPU, HiMA-DNC and HiMA-DNC-D outperform by up to 437× and 2,646× in speed, respectively.

ACKNOWLEDGMENTS

This work was supported in part by NSF CCF-1900675.

REFERENCES

[1] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. 2016. Fused-layer CNN accelerators. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–12.

[2] Antreas Antoniou, Amos Storkey, and Harrison Edwards. 2017. Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340* (2017).

[3] Diana Borsa, Bilal Piot, Rémi Munos, and Olivier Pietquin. 2017. Observational learning by reinforcement learning. *arXiv preprint arXiv:1706.06617* (2017).

[4] Nagadastagiri Challapalle, Sahithi Rampalli, Nicholas Jao, Akshaykrishna Ramanathan, John Sampson, and Vijaykrishnan Narayanan. 2020. FARM: A flexible accelerator for recurrent and memory augmented neural networks. *Journal of Signal Processing Systems* (2020), 1–15.

[5] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, et al. 2018. Serving dnns in real time at datacenter scale with project brainwave. *iEEE Micro* 38, 2 (2018), 8–20.

[6] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*.

[7] Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. 2018. Model-based reinforcement learning via meta-policy optimization. In *Conference on Robot Learning*. PMLR, 617–629.

[8] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural Language Processing (Almost) from Scratch. *J. Mach. Learn. Res.* 12 (Nov. 2011), 2493–2537.

[9] Xiao Dong, Xiaolei Zhu, and De Ma. 2019. Hardware Implementation of Softmax Function Based on Piecewise LUT. In *2019 IEEE International Workshop on Future Computing (IWOFC)*. 1–3.

[10] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. 2017. Stabilising experience replay for deep multi-agent reinforcement learning. In *International conference on machine learning*. PMLR, 1146–1155.

[11] Alex Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850* (2013).

[12] A. Graves, A. Mohamed, and G. Hinton. 2013. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. 6645–6649.

[13] Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401* (2014).

[14] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature* 538, 7626 (2016), 471–476.

[15] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[16] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[17] Hanhwi Jang, Joonsung Kim, Jae-Eon Jo, Jaewon Lee, and Jangwoo Kim. 2019. Mnnfast: A fast and scalable system architecture for memory-augmented neural networks. In *Proceedings of the 46th International Symposium on Computer Architecture*. 250–263.

[18] Ioannis Kouretas and Vassilis Paliouras. 2019. Simplified hardware implementation of the softmax activation function. In *2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST)*. 1–4.

[19] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. 2016. Ask me anything: Dynamic memory networks for natural language processing. In *International conference on machine learning*. 1378–1387.

[20] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. 2019. Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 754–768.

[21] Hyoukjun Kwon, Liangzhen Lai, Tushar Krishna, and Vikas Chandra. 2019. Herald: Optimizing heterogeneous dnn accelerators for edge devices. *arXiv preprint arXiv:1909.07437* (2019).

[22] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '18)*. New York, NY, USA, 461–475.

[23] Susumu Mashimo, Thiem Van Chu, and Kenji Kise. 2017. High-Performance Hardware Merge Sorter. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 1–8. https://doi.org/10.1109/FCCM.2017.19

[24] A. Norollah, D. Derafshi, H. Beitollahi, and M. Fazeli. 2019. RTHS: A Low-Cost High-Performance Real-Time Hardware Sorter, Using a Multidimensional Sorting Algorithm. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 7 (2019), 1601–1613. https://doi.org/10.1109/TVLSI.2019.2912554

[25] Junhyuk Oh, Valliappa Chockalingam, Honglak Lee, et al. 2016. Control of memory, active perception, and action in minecraft. In *International Conference on Machine Learning*. PMLR, 2790–2799.

[26] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. 2017. Scnn: An accelerator for compressed-sparse convolutional neural networks. *ACM SIGARCH Computer Architecture News* 45, 2 (2017), 27–40.

[27] Seongsik Park, Jaehee Jang, Seijoon Kim, and Sungroh Yoon. 2019. Energy-Efficient Inference Accelerator for Memory-Augmented Neural Networks on an FPGA. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1587–1590.

[28] Ashish Ranjan, Shubham Jain, Jacob R Stevens, Dipankar Das, Bharat Kaul, and Anand Raghunathan. 2019. X-MANN: A crossbar based architecture for memory augmented neural networks. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.

[29] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature* 323, 6088 (1986), 533–536.

[30] Akane Saito, Yuki Umezaki, and Makoto Iwata. 2017. Hardware accelerator for differentiable neural computer and its fpga implementation. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*. 232–238.

[31] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, Stephen G. Tell, Yanqing Zhang, William J. Dally, Joel Emer, C. Thomas Gray, Brucek Khailany, and Stephen W. Keckler. 2019. Simba: Scaling Deep-Learning Inference with Multi-Chip-Module-Based Architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 14–27. https://doi.org/10.1145/3352460.3358302

[32] Abhik Singla, Sindhu Padakandla, and Shalabh Bhatnagar. 2019. Memory-based deep reinforcement learning for obstacle avoidance in UAV with limited environment knowledge. *IEEE Transactions on Intelligent Transportation Systems* (2019).

[33] Jacob R Stevens, Ashish Ranjan, Dipankar Das, Bharat Kaul, and Anand Raghunathan. 2019. Manna: An accelerator for memory-augmented neural networks. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 794–806.

[34] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. 2015. End-to-end memory networks. In *Advances in neural information processing systems*. 2440–2448.

[35] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. 3104–3112.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762* (2017).

[37] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. 2015. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698* (2015).

[38] Jason Weston, Sumit Chopra, and Antoine Bordes. 2014. Memory networks. *arXiv preprint arXiv:1410.3916* (2014).