NoC-enabled Software/Hardware Co-Design Framework for Accelerating *k-mer* Counting

Biresh Kumar Joardar*, Priyanka Ghosh*, Partha Pratim Pande*, Ananth Kalyanaraman*, Sriram Krishnamoorthy†

*School of EECS, Washington State University
Pullman, WA 99164, U.S.A.
{biresh.joardar, priyanka.ghosh, ananth, pande}@wsu.edu

†HPC Group, Pacific Northwest National Laboratory
Richland, WA 99352, U.S.A.
sriram@pnnl.gov

ABSTRACT

Counting *k-mers* (substrings of fixed length *k*) in DNA and protein sequences generate non-uniform and irregular memory access patterns. Processing-in-Memory (PIM) architectures have the potential to significantly reduce the overheads associated with such frequent and irregular memory accesses. However, existing *k-mer* counting algorithms are not designed to exploit the advantages of PIM architectures. Furthermore, owing to thermal constraints, the allowable power budget is limited in conventional PIM designs. Moreover, *k-mer* counting generates unbalanced and long-range traffic patterns that need to be handled by an efficient Network-on-Chip (NoC). In this paper, we present an NoC-enabled software/hardware co-design framework to implement high-performance *k-mer* counting. The proposed architecture enables more computational power, efficient communication between cores/memory – all without creating a thermal bottleneck; while the software component exposes more in-memory opportunities to exploit the PIM and aids in the NoC design. Experimental results show that the proposed architecture outperforms a state-of-the-art software implementation of *k-mer* counting utilizing Hybrid Memory Cube (HMC), by up to 7.14X, while allowing significantly higher power budgets.

KEYWORDS

Manycore, M3D, Thermal, PIM, *k-mer* counting, Co-design

1 INTRODUCTION

Analysis of biomolecular data such as DNA and proteins has been one of the primary drivers of scientific discovery in biological sciences. From a computational perspective, these biomolecules can be represented as strings (equivalently, sequences). Hence, sequence analysis occupies a significant portion of many bioinformatics workflows. One such operation is *k-mer* counting, where the goal is to determine the counts of all distinct fixed length substrings of length *k* in a large collection of input sequences. Computing *k-mer* abundance profiles is often necessary for several bioinformatics applications e.g., de novo genome assembly, repeat identification, etc.

Challenges: From a software perspective, implementing *k-mer* counting in a resource- and time-efficient manner is a challenging task. Existing software-only solutions for efficient *k-mer* counting e.g. KMC2 [1], Gerbil [2], do not consider hardware limitations, resulting in sub-optimal performance. The counting process generates irregular memory accesses and sparse computations that results in more frequent memory read/writes. Conventional memory architectures provide limited bandwidth with higher read/write latencies. This presents a performance bottleneck for *k-mer* counting, which relies on repeated memory access. As a result, computing units often remain idle, as a large portion of the execution time is spent moving data to/from memory. Moreover, *k-mer* counting generates significant communication between the Processing Elements (PEs). For example, in Gerbil [2], *k-mer*s are repeatedly distributed among the threads responsible for counting, introducing significant amount of on-chip traffic. Without an efficient communication backbone, this can lead to longer execution times as PEs would remain idle for a greater number of cycles waiting for data. To overcome these inefficiencies in conventional architectures, we posit that a carefully designed software/hardware co-design framework can be better equipped to derive the best out of both worlds. More specifically, in this work, we argue that the emerging paradigm of Processing-in-Memory (PIM), enabled by a Network-on-Chip (NoC), presents a promising solution to these applications.

PIM takes advantage of emerging 3D-stacked memory + logic devices (such as Micron's Hybrid Memory Cube or HMC) to enable high-bandwidth, low latency and low energy memory access [3]. However, conventional PIM architectures are restricted by thermal constraints as temperature impacts both memory retention and overall performance [4, 5]. Conventional 2.5D PIM and 3D PIM architectures often have limited power budget and computational capability before reaching the allowable temperature threshold [5, 6]. The role of NoC in PIM architectures is also understudied. Due to a single layer of logic in existing PIM architectures, planar NoC is used for efficient communication among the PEs [7]. However, 2D NoCs such as mesh, are not suited for long range communication that is inherent in *k-mer* counting. Moreover, *k-mer* counting generates unbalanced traffic that imposes an additional layer of design complexity.

To overcome the above-mentioned challenges, in this paper, we propose an NoC enabled manycore architecture that exploits the benefits of emerging Monolithic 3D (M3D) integration to integrate *multiple* logic layers in PIM architectures with 3D-stacked memory, for high-performance *k-mer* counting. Experimentally, we show that even with multiple logic layers and higher power budget, the proposed architecture does not violate thermal constraints. The main contributions of this work are:

NOCS '19, October 17–18, 2019, New York, NY, USA

ACM ISBN 978-1-4503-6700-4/19/10…$15.00
https://doi.org/10.1145/3313231.3352367

- We profile *k-mer* counting to extract features relevant to the design of a high-performance NoC and the overall PIM architecture for *k-mer* counting.
- We present a software/hardware co-design framework: the hardware consists of a PIM with multiple logic layers enabled by M3D integration while the software component enables high-performance *k-mer* counting by utilizing the benefits of PIM and aids in NoC design.
- We perform a thorough experimental evaluation of the proposed co-design framework and show significant improvements over an appropriate baseline.

2 RELATED WORKS

2.1 *k-mer* counting

The task of *k-mer* counting is memory-intensive and involves creating a histogram of all k-length substrings in a DNA sequence. KMC2 [1], DSK [8] and Gerbil [2] are some of the popular tools for this purpose, with Gerbil representing the state-of-the-art in software as it outperforms most of the other tools. However, it requires the repetitive use of off-chip secondary memory and therefore will fail to fully exploit the high-bandwidth, low latency memory access facilitated by PIM. Manycore CPU- and GPU-based platforms are the preferred choices for implementing *k-mer* counting [1, 2]. However, these works do not address the memory bottleneck issues. In [9], the authors designed a custom FPGA-based architecture connected to an HMC for approximate *k-mer* counting. However, the memory (HMC) is connected to the PEs using serial links in a 2.5D architecture, which is not as efficient as completely on-chip solutions i.e. 3D PIM [3, 10].

2.2 Processing-in-memory

Processing-in-Memory (PIM) involves moving the computational units closer to memory. This allows efficient data transfer from memory enabled by 3D integration [3]. Prior works has mostly focused on Through Silicon Via (TSV)-based PIM architectures, which are prone to high temperatures [5, 11]. Heat from processing elements can significantly affect the retention time of DRAMs [4]. Beyond 85°C, the overheads to counter lower DRAM retention can significantly offset the benefits of PIM [5].

To reduce the effect of temperature, 2.5D architecture is a popular choice to implement PIM where PEs are placed *near* the memory and connected via interposers. However, lateral heat flow from PEs can significantly affect memory temperature [6]. In 3D-PIM architectures, memory is stacked directly on top of the PEs, which enables better throughput and latency [3]. However, they are more prone to high temperature as PEs are placed in the same vertical stack. In [12], the authors propose to use a mix of 2.5D + 3D PIMs. They map the application on PEs based on its memory and compute requirements for best performance. However, their methodology assumes prior knowledge of an application, which is often not feasible. Memory-centric NoC, that connects multiple HMCs to facilitate efficient data transfer has been studied [7]. The role of NoC connecting different vaults of an HMC is discussed in [13]. However, these implementations are limited to 2D NoC which is inefficient in addressing the challenges posed by *k-mer* counting i.e. unbalanced traffic and long-range communication. To overcome these limitations, we propose an NoC-enabled PIM-based architecture that amalgamates: (a) multiple logic layers in conventional PIM, (b) M3D-based vertical integration, and (c) efficient 3D-NoC design for high-performance *k-mer* counting, while remaining within 85°C temperature. To take advantage of PIM's more efficient memory access and aid NoC design, we also propose an alternative software approach to count *k-mer*s that outperforms the Gerbil framework.

3 High-performance *k-mer* counting

Problem statement: Given a DNA sequence s of length l, a "*k-mer*" is defined as a substring of length k in s (k being an integer, $k \leq l$). All *k-mer*s in s can be generated by simply sliding a window of length k over s. Given a set S of n such input sequences (aka. "reads"), the problem of *k-mer counting* is one of determining the total number of occurrences for each *distinct k-mer* that is present in the *reads* of S. In this work, we mainly focus on Gerbil [2] as our software baseline for *k-mer* counting. Gerbil is a recently proposed methodology that outperforms several well-known counters e.g. KMC2 [1] and DSK [8] for higher values of k, e.g. k=32, making it an appropriate software baseline to consider.

3.1 Gerbil: Overview and Analysis

Gerbil is a two-phase algorithmic implementation for *k-mer* counting: (a) 'Distribution' phase where the input *reads* are partitioned into multiple intermediate files on the disk, and (b) 'Counting' phase, which reads each of these intermediate files (one by one) for counting. The entire procedure is designed to make optimal use of conventional manycore architectures. Each step in Gerbil operates in a pipelined fashion for high throughput counting. Popular techniques like load balancing and use of failure buffers to handle hash conflicts, etc., have also been used for better performance. Fig. 1 illustrates the overall workflow of Gerbil.

Next, we thoroughly analyze Gerbil using detailed full-system simulations on Gem5 [14] to study relevant features that are crucial in designing an efficient manycore architecture. Fig. 2(a) shows the different categories of instructions (operations) involved in Gerbil with real-world inputs. We observe that nearly two-thirds of Gerbil consists of integer operations. Memory operations (including I/O) constitute the second largest category (32.5%). The remaining is made up of *NoOps* while floating-point instructions are negligible.

Traditional off-chip main-memory and secondary memory accesses are slower [10], which can cause significant CPU stalls. In Gerbil, memory and I/O operations contribute significantly to

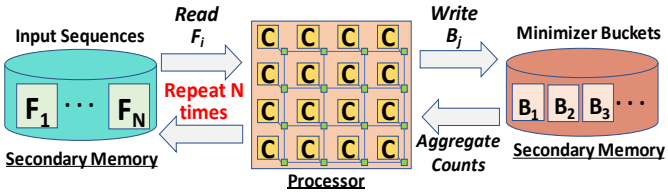
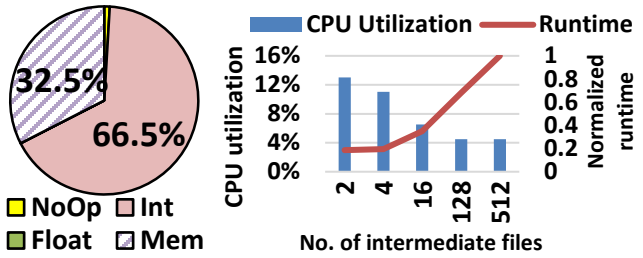


Fig. 1: Illustration of workflow in Gerbil (F: Input files, B: Buckets, C: CPU; 'Buckets' is synonymous to 'intermediate files' in [2])



(a) (b)

Fig. 2: Gerbil: (a) Instruction types, and (b) CPU utilization and runtime (normalized) for varying number of intermediate files

the runtime, the effect of which is captured in Fig. 2(b). Full-system simulations on Gem5 with 64 Intel x86 cores executing Gerbil shows that CPUs are utilized less than 15% of the time (Fig. 2(b)) for any number of intermediate files. The intermediate files are generated after *Distribution* phase of Gerbil which are stored and then eventually read back from the slow off-chip secondary memory for further processing (as shown in Fig. 1). Moreover, the counting process involves irregular memory accesses, which makes caching ineffective. As a result, even though integer instructions constitute the majority of Gerbil operations, Fig. 2(b) clearly highlights that most of the execution time is spent in fetching/storing data rather than actual computation. The CPU utilization gets worse if more intermediate files are generated (4.5% CPU utilization in the case of 512 files) as it involves more off-chip memory access. This translates to a significant increase in runtime as observed in Fig. 2(b). Overall, it is clear that Gerbil does not efficiently utilize the computing resources resulting in sub-optimal performance. Fig. 2 also proves that *slow memory access presents a more serious bottleneck to performance than computation*, for k-mer counting, making it an ideal case for PIM. However, Gerbil's dependence on secondary memory (Fig. 1) makes it inappropriate for PIM architectures as it'll fail to fully exploit the high-bandwidth, low latency memory access facilitated by PIM. Therefore, A PIM-friendly *k-mer* counting software solution that complements the hardware is necessary.

Fig. 3 shows the communication between every CPU (C_i) pair for three different input datasets in the form of a heat map. Here, we define amount of communication (traffic) as the number of flits exchanged between a pair of cores during *k-mer* counting as obtained using full-system Gem5 simulations considering a manycore system with 64 cores. As shown in Fig. 3, *k-mer* counting in Gerbil exhibits significant amount of data exchanges between cores. Darker patches (a few have been highlighted in red in Fig. 3(a) as examples) indicate heavier communication between a pair of cores. Planar logic in conventional PIM offers only a limited number of design and floor-planning choices. Hence, frequently communicating cores may get potentially placed far from each other, leading to long range communication. Also, we observe several lighter patches indicating lower communication in Fig. 3. This shows that the communication in Gerbil is highly unbalanced. Few of the cores have heavy data traffic while the rest have relatively negligible traffic. These heavily communicating cores e.g. C1 in Fig. 3(a), can become traffic hotspot during execution, which affects performance. Without a suitable NoC backbone, this can result in higher latency that in turn will increase execution time. It is well known that 2D NoCs (due to single layer of logic in conventional PIM) are not scalable and not suited to handle long range communication. Therefore, an efficient NoC is crucial for high-performance *k-mer* counting.

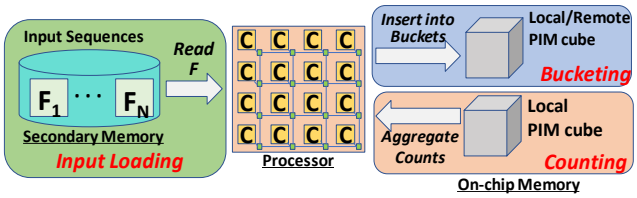


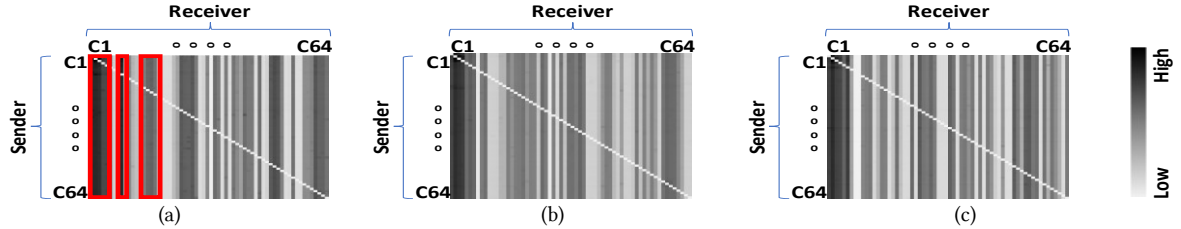
Fig. 4: Illustration of workflow in proposed *k-mer* counting methodology: PIM-Counter (F: Input files, B: Buckets, C: CPU)

3.2 PIM-Counter: PIM Friendly *k-mer* Counter

In this section we present *PIM-Counter*, a PIM-friendly multi-threaded algorithm designed to overcome the I/O bottleneck of Gerbil, exploit the PIM-based architecture and aid in the NoC design. Fig. 4 shows the workflow of PIM-Counter. As discussed earlier, Gerbil relies on secondary memory usage, which results in inefficient CPU utilization (Fig. 2) and is not suited for PIM-based architectures. In contrast, the proposed PIM-counter (Fig. 4), uses an on-chip memory-friendly approach to utilize the benefits of PIM. As illustrated in Fig. 4, PIM-Counter has three main steps:

Step-1: Input loading: Instead of reading the input files in batches using multiple I/O passes as in Gerbil (Fig. 1), PIM-Counter performs a single I/O pass. The inputs are then loaded uniformly across the PIM cubes. Here, a *cube* (Fig. 5, which shows the overall PIM architecture) is analogous to an HMC vault [13] that consists of both logic and memory. However unlike conventional HMC vaults, we also consider PEs e.g. CPUs, as part of the logic layer (i.e. 3D PIM). We discuss the hardware architecture in more details in next section.

Step-2: Bucketing: Once the strings are loaded onto the memory, uniformly across the partitions ('cubes' in Fig. 5), the local thread(s) in the corresponding cubes generate all *k-mer*s from each string by sliding a window of length *k*. To overcome the



(a) (b) (c)

Fig 3: CPU-to-CPU communication profile for Gerbil in the form of heat map for input datasets (a) *E. Coli*, (b) *Prochlococcus sp.*, and (c) *Vibrio cholerae* (C_i: CPU*i*; Red boxes highlight *few* of the patches of heavy communication)

challenge imposed by the use of a large value of *k*, we use the concept of *minimizers*, which was originally introduced in the context of building de Bruijn graphs [15]. The idea is to hash each *k-mer* using its least (or equivalently, most) frequent *m*-mer, where *m*<*k* (e.g., *m*=7; *k*=32) and migrate that *k-mer* to the minimizing *m-mer's* bucket. Here, the term *bucket* refers to the collection of all *k-mer*s that share the same minimizer and is analogous to the 'intermediate files' used in Gerbil. However, unlike Gerbil, these buckets are present in the on-chip memory. Each cube is responsible for a different, non-overlapping set of buckets. The mapping of bucket to *cube id* is achieved using a hash function in linear congruential form (*e.g.* ((Ax+B) mod P), A, B and P are constants), which distributes all possible buckets across the different cubes. As a result, the responsible bucket for a *k-mer* could either reside on the local cube (same cube as the computing PE) or on a remote cube (any other cube except the *local cube*). For example, in Fig. 5, Cube-16 is a local cube to CPU-16, while Cube-1 is remote cube to CPU-16. Memory in local cube can be accessed by the cores using vertical interconnects *only*. A remote cube, however, must be reached via the use of one or more planar links. Accessing remote cubes is costlier as data must traverse longer physical distance that can result in higher execution time. The NoC should support this data movement.

In PIM-Counter, the data movement (traffic pattern) between PEs depends on the hash function which defines the mapping of *k-mer* buckets to cube-ids. Therefore, it is important to choose suitable values for A, B and P (and hence the hash function) such that the resulting traffic is balanced. Overall, our aim here is to choose a suitable mapping that distributes the traffic among the PEs evenly to avoid hotspots in the NoC during execution. We use full-system Gem5 simulations to determine the hash function that yield better traffic distribution (shown in experimental results).

Step 3: Counting: In the final step, the thread(s) local to each cube aggregate the counts for each distinct *k-mer* represented in its local buckets; this is achieved using a parallel reduction. Here, PIM-Counter fully exploits the locality benefits of PIM as data is already available on each thread's corresponding local cube (due to the previous *bucketing* phase) and can be accessed using just the vertical links. Due to the physical proximity of memory in PIM, CPU stalls are greatly reduced as data can be fetched relatively faster than in conventional architectures (where data is fetched from physically distant/off-chip memory).

Overall, PIM-Counter presents a PIM-friendly *k-mer* counting alternative that can outperform other counting tools as it benefits from high-bandwidth, low-latency and low-energy memory access facilitated by PIM. It also enables efficient communication between PEs by reducing traffic hotspots.

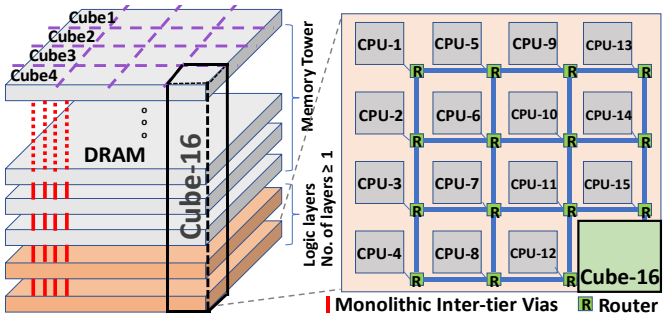


Fig. 5: Proposed PIM-based architecture with multiple logic and memory layers enabled via M3D integration

4 NoC-ENABLED 3D-PIM DESIGN

In this section, we introduce the features of the proposed 3D-PIM enabled by M3D integration followed by the NoC design that supports the communication generated by *k-mer* counting.

4.1 PIM-architecture for *k-mer* Counting

PIM allows high bandwidth, low-latency and low-energy memory access by moving computation closer to memory [10]. The faster memory access enabled by PIM is crucial for *k-mer* counting as a large fraction of time (>85%) is spent in fetching/storing data to/from memory in Gerbil (Fig. 2). However, temperature presents an important limitation in conventional PIM architectures. DRAM retention capability is lowered beyond 85°C. After temperature exceeds this threshold, refresh rate must be doubled for every ~10°C increase in memory temperature. Higher refresh rates consume more power and, results in lower memory performance [5]. Also, traditional power management techniques are often not tailored for memory. Therefore, placing memory directly on top of (or nearby) the PEs in PIM, without addressing thermal issues, can be detrimental to performance.

In [6], the authors found that 2.5D PIM architectures are prone to lateral heat flow from PEs even when placed 10mm farther from the HMC. Placing memory farther away to reduce temperature, also defeats the main purpose of PIM, which is to bring computation closer to memory. 3D PIM architectures where PEs are in same vertical stack as memory, are even more sensitive. Therefore, conventional PIM architectures (both 2.5D and 3D) typically use either (a) PEs with simpler architectures (as complex cores e.g. Out-of-Order (OoO) CPUs tend to consume more power [5]), (b) fewer number of cores e.g. [12], or (c) minimal computing power [6], (or all of the above) to remain within the temperature threshold. Due to these restrictions, conventional PIM architectures have lower computation capability that affects performance and are not scalable with increasing system size.

Moreover, PIM architectures are restricted to *single logic layer* and multiple memory layers, as logic (PEs) dissipates more heat than memory [5]. It is well known that 2D logic provides limited floor-planning choices and require more die area than an equivalent 3D counterpart. However, multiple logic layers stacked vertically in 3D ICs are prone to higher temperatures as PEs farther away from the sink cannot dissipate heat easily, resulting in worse temperature [16]. As PEs consume more power than memory, use of multiple layers of logic in PIMs is typically avoided. As a result, only a few cores can be integrated given a fixed area constraint. Overall, our objective for a "suitable PIM architecture" is one that should: (a) allow larger volume of computation (logic) to be integrated without incurring extra area and thermal overheads; and (b) enable efficient data exchange between cores and memory. Taking advantage of the benefits of 3D ICs in this work, we propose a PIM architecture that incorporates *multiple logic layers* in conventional PIM for high-performance. Fig. 5 shows the proposed architecture with multiple logic layers (similar to 3D ICs

[16]) and multiple memory layers. Each logic layer consists of multiple PEs, while the memory layers consists of conventional DRAM. The cores are connected using a Network-on-Chip (NoC) to support efficient on-chip communication between cores. We discuss NoC design in the next sub-section. The use of multiple logic layers enables a greater number of cores to be integrated compared to traditional PIM (single logic layer) under an "*iso-area*" setting. All the layers are *virtually* (not physically) divided into several equal cubes. Each cube consists of equal amount of resources *i.e.* one core per logic layer (placed vertically on top of each other) and the portion of memory directly above it. For example, in Fig. 5 (assuming 2-logic layers and following similar numbering convention of CPUs), Cube-16 consists of CPU-16, CPU-32 and part of memory directly above it.

Conventional TSV-based 3D architectures are susceptible to higher temperatures and hence cannot be used to design the proposed architecture (Fig. 5) [11]. Consecutive layers in TSV-based designs are physically attached using a bonding material e.g. Benzocyclobutene (BCB), that exhibits poor thermal conductivity. This impedes the seamless flow of heat across the layers resulting in considerable increase in temperature in the layers away from the heat sink. Moreover, the relatively thicker silicon substrate (several micrometers) in TSV-based designs causes the heat to spread laterally within the substrate instead of vertically towards the sink. This results in higher on-chip temperatures, which is undesirable in PIM architectures.

On the other hand, emerging M3D integration allows faster dissipation of heat than its TSV-based counterparts [16]. Absence of a bonding material and relatively smaller dimensions (nanometers as opposed to micrometers) leads to superior thermal characteristics than TSV-based designs. Therefore, we argue that we should design high-performance yet thermally viable PIM architectures with multiple logic (and memory) layers as shown in Fig. 5 *using M3D integration*. Experimentally, we show that M3D-based PIM designs are superior in terms of both performance and temperature, enabling higher power budgets compared to their TSV-based solutions. Moreover, M3D enables design of area- and power-efficient multi-tier logic blocks [17]. The possibility of multi-tier logic blocks e.g. NoC routers, enable design of high-performance and energy-efficient NoCs, which is essential to support efficient *k-mer* counting.

4.2 NoC design for *k-mer* Counting

For achieving high performance, the choice of overall NoC connectivity should be governed by the traffic pattern generated by the application under consideration. As shown in Fig. 3, *k-mer* counting introduces significant long-range and unbalanced traffic pattern that should be handled by the NoC. The unbalanced traffic in *k-mer* counting is addressed by choosing a suitable mapping to cubes (hash-function) in PIM-Counter as discussed in Section 3. PIM-Counter makes the traffic more *uniform* compared to Gerbil, reducing chances of traffic hotspots (shown later in experimental results section). To efficiently handle the long-range traffic pattern, 3D small-world (SW) NoC architecture is a suitable choice. The vertical links in 3D NoCs bring cores physically closer

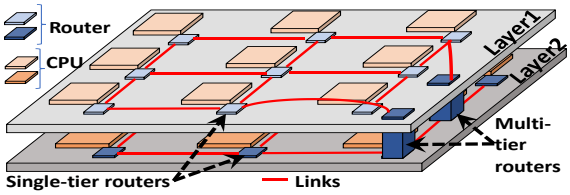


Fig. 6: Illustration of proposed M3D-enabled SW-NoC with multi-tier routers [17] and small-world properties [19] (The color contrast between *Layer1* and *Layer2* is for differentiation only)

and enable long-range communication shortcuts necessary for designing high-performance SW NoC [19]. Moreover, the vertical connectivity in M3D is facilitated by monolithic inter-tier vias (MIVs), which are 100x smaller and more energy efficient than conventional TSVs [18]. Overall, we utilize the benefits of M3D to design high-performance, yet energy efficient SW NoCs.

To design a suitable 3D SW NoC, the placement of links and routers need to be optimized based on the application (*k-mer* counting in this case). By optimizing the placement of the routers/links, it is possible to address the communication challenges inherent in *k-mer* counting (Fig. 3) effectively. We demonstrate that the designed NoC (executing PIM-Counter) outperforms Gerbil running on an equivalent platform, in later section. Next, we discuss the details of the NoC optimization.

Optimization Objective: For the NoC performance evaluation, we consider two objectives: latency and energy. We estimate network latency and energy using analytical models proposed in [16] for optimization purpose. For an N core system, the average network latency is modeled as:

$$Lat = \frac{1}{\sum f_{ij}} \sum_{i=1}^{N} \sum_{j=1}^{N} \left(r \cdot h_{ij} + d_{ij}\right) \cdot f_{ij} \quad (1)$$

Here, f_{ij} represents the number of flits exchanged between core i and core j (Fig. 3) obtained from full-system *k-mer* counting simulations on Gem5. The parameter r represents the number of router stages, h_{ij} denotes the number of hops between the two cores while d_{ij} incorporates the effect of physical distance that messages must traverse based on the routing protocol.

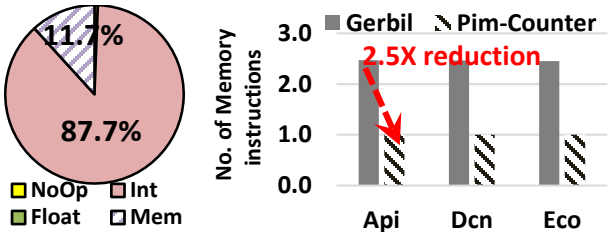
The network energy is modeled using the following equations:

$$E_{router} = \sum_{i=1}^{N} \sum_{j=1}^{N} f_{ij} \cdot \sum_{k=1}^{R} r_{ijk} \cdot (E_r \cdot P_k) \quad (2)$$

$$E_{link} = \sum_{i=1}^{N} \sum_{j=1}^{N} f_{ij} \cdot \left(\sum_{k=1}^{L} p_{ijk} \cdot d_k \cdot \mathrm{E}_{planar} + \sum_{k=1}^{V} q_{ijk} \cdot E_{vertical} \right) \quad (3)$$

$$E = E_{router} + E_{link} \quad (4)$$

Here E_r denotes the average router logic energy per port and P_k denotes the number of ports available at router k. The total link energy can be divided into two parts due to the different physical characteristics of planar and vertical links. d_k represents the physical link length of link k. Here, q_{ijk} and r_{ijk} indicate if a vertical link or router k is utilized to communicate between core i and core j respectively. E_{planar} and $\mathrm{E}_{vertical}$ denote the energy consumed per flit by planar metal wires and vertical links (TSV or MIV) respectively. All the required power numbers were obtained



(a) (b)

Fig 7: PIM-Counter: (a) Instruction types, and (b) Number of memory operations compared to Gerbil (normalized)

using Synopsys Prime Power for 28nm nodes. The total network energy E is the sum of router logic and link energy.

We optimize the two objectives, latency and energy, using a machine learning-enabled Multi-Objective Optimization (MOO) algorithm: MOO-STAGE [16]. By *learning* the search space, MOO-STAGE can find better solutions than several conventionally used MOO algorithms in much less time. Hence, it is a suitable choice of MOO solver for optimizing the NoC. Overall, MOO-STAGE finds the best placement of links and routers in the SW NoC that achieves good trade-off between both objectives: latency and energy, to enable high-performance *k-mer* counting To ensure that the optimized architectures are realistic, we make sure that there is always at-least one path for communication between any pair of cores. M3D specific design aspects, e.g. possibility of multi-tier routers has also been incorporated in the optimization. Following prior designs, e.g. [17], we restrict routers to span across at-most two layers only. Fig, 6 shows an illustration of the NoC for the proposed M3D-enabled PIM

5 EXPERIMENTAL RESULTS

In this section, we evaluate the performance of the proposed NoC-enabled, software/hardware co-design framework for *k-mer* counting. First, we analyze PIM-Counter (Sec. 3) in detail. Next, we evaluate the performance and thermal profile of the proposed PIM architecture and the designed NoC.

5.1 Experimental Setup

We use a detailed full system simulator, Gem5 [14] to characterize the performance of the PIM-based manycore architecture proposed in this work. We modify the memory and Garnet network models within Gem5 to implement the various memory and NoC architectures considered in this work. All experiments are performed on a 64-core system where each core is based on Intel x86 architecture. The memory system comprises of private 32KB L1 instruction and data caches, shared 16MB L2 cache (256KB distributed L2 per core). The simulated system operates with a clock frequency of 2.5GHz. The CPU power profiles are extracted using McPAT [20] while the on-chip temperatures are obtained using Hotspot [21] simulations. The TSV and M3D layers are modeled in Hotspot based on parameters e.g. layer thickness, thermal conductivity, etc. as listed in [11]. For all experiments, we have considered DRAM-based memory. Following prior works, the memory is modeled as a multi-layer stack in the proposed PIM design. All layers in the same vertical stack have equal area i.e. the memory die area is assumed to be same as the logic die area in the proposed 3D-PIM architecture.

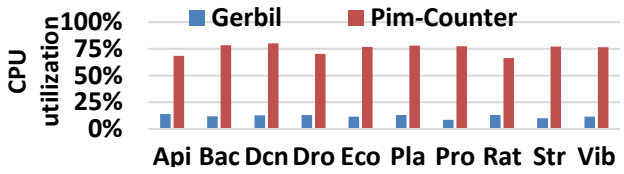


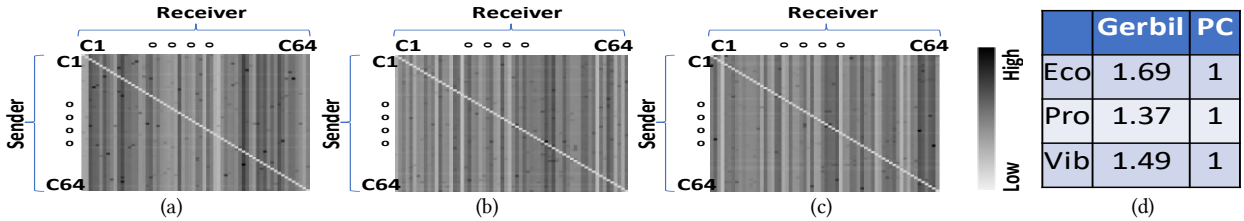
Fig 8: Average CPU utilization in Gerbil and PIM-Counter

For experimental evaluations, we chose ten different genomic sequences from across the species spectrum: six prokaryotic genomes including, *Prochlorococcus sp.* (Pro), *S. pneumoniae* (Str), *V. cholerae* (Vib), *E. coli* (Eco), *B. circulans* (Bac), P. *vivax* (Pla) and four eukaryotic genomes namely, *A. melifera* (Api), *D. melanogaster* (Dro), *D. labrax* (Dcn) and *R. norvegicus* (Rat). This collection represents a wide variety in genome input complexities (including *k-mer* composition and abundance levels), intended to help us test our framework under different input scenarios, as *k-mer* counting is known to be an input-dependent problem [22].

5.2 Performance Evaluation of PIM-Counter

We first evaluate the performance of the proposed PIM-Counter framework based on its traffic pattern, types of instructions and CPU utilization to compare with Gerbil. We profile PIM-Counter using full-system simulations on Gem5 similar to Gerbil (Sec. 3). Fig. 7(a) shows the distribution of instructions for PIM-Counter (similar to Fig. 2). Interestingly, we note that integer operations constitute a far greater proportion i.e. 87.7% in PIM-Counter (as opposed to 66.5% in Gerbil (Fig. 2)). Memory (including I/O) instructions only contribute to 11.7% of overall instructions. Floating point instructions and *NoOps* were negligible in both Gerbil and PIM-Counter. Fig. 7(b) compares the actual number of memory (including I/O) instructions for Gerbil and the PIM-Counter. We note that the PIM-Counter reduces the number of memory operations by ~2.5X compared to Gerbil. This is important as memory operations contribute significantly to execution time. Coupled with a more efficient memory access provided by PIM, this results in a significant improvement in CPU utilization – as shown in Fig. 8. On average, the CPUs are utilized 75% of the time as opposed to <15% in Gerbil. This shows that by facilitating easier memory access, we can improve hardware utilization significantly. This is expected as PIM-Counter reduces the amount of I/O operations (which are slow), while promoting more on-chip memory usage to take advantage of PIM.

Next, we look at the corresponding traffic pattern generated when PIM-Counter is executed on a 64-core architecture. Fig. 9 (a-c) shows the traffic pattern between CPUs for PIM-Counter with three real world datasets, namely: Eco, Pro and Vib, as examples. Fig. 9(d) compares the standard deviation of Gerbil's traffic (normalized) with respect to that of PIM-Counter for these three datasets. The standard deviation of traffic captures the variation among the number of flits associated with each PE. Higher values of standard deviation indicate a more unbalanced traffic which can lead to traffic hotspots in the NoC resulting in higher execution times (discussed in Sec. 3). Contrasting these traffic patterns with those of Gerbil shown in Fig. 3, it is clear that PIM-Counter achieves a more balanced traffic. For instance, in the case of Eco, PIM-Counter generates a 69% (Fig. 9(d)) better balanced



	Gerbil	PC
Eco	1.69	1
Pro	1.37	1
Vib	1.49	1

(a) (b) (c) (d)

Fig 9: CPU-to-CPU communication profile for PIM-Counter as heat map (C_i: Core i;) for input datasets (a) *E. Coli (Eco)*, (b) *Prochlococcus sp. (Pro)*, (c) *Vibrio cholerae (Vib)*, and (d) Standard Deviation of Gerbil's traffic normalized with respect to that of PIM-Counter (PC)

traffic pattern. This can be attributed to the appropriate mapping of *k-mer*s to cube-id in PIM-Counter. Hence, traffic hotspots are less likely, leading to better performance in PIM-Counter.

5.3 Thermal Evaluation

For any new PIM architecture, thermal feasibility is a major concern [5]. In Sec. 4, we argued that it is possible to integrate multiple layers of logic (similar to conventional 3D ICs) in PIM using M3D. Therefore, before performance analysis, in this section, we first investigate and experimentally validate the thermal feasibility of the proposed PIM architecture.

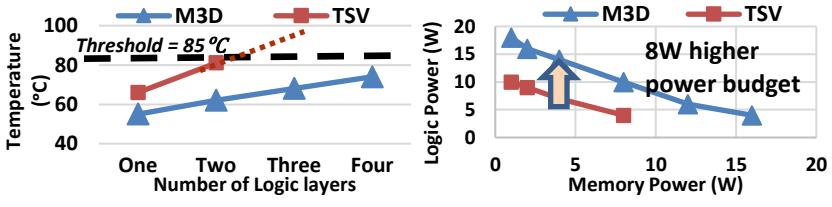
Fig. 10(a) shows the variation of maximum on-chip temperature for *k-mer* counting as more logic layers are added. Each layer has been modeled following [11] in HotSpot. Here, we assume that the number of memory layers to be fixed while varying the number of logic layers beneath it. For all experiments, an ambient temperature of 45°C and an inexpensive low-end cooling (convention resistance = 2°C/W [11]) is used. Fig. 10(a) indicates that even with a simple cooling solution, up to four layers of logic can be easily integrated in M3D-based 3D-PIM without reaching temperature threshold of 85°C. On the other hand, TSV-based PIM only allows a maximum of 2 logic layers for *k-mer* counting. Beyond two layers, TSV-based PIM architectures necessitate higher refresh rates and more expensive cooling solutions to be viable. Note that adding the second layer of logic results in temperature close to the threshold (81°C) which may still necessitate precautions for safe operation. However, even with four logic layers, M3D-based PIM architecture exhibits maximum temperature of 74°C only. Therefore, contrary to conventional PIM architectures, it is possible to have multiple logic layers in an M3D-enabled PIM without violating thermal constraints.

Also, since core and memory power depend on several factors e.g., voltage-frequency settings, technology node etc., it is important to study the power budget available in both architectures (without exceeding 85°C) for a complete analysis. Fig. 10(b) shows the amount of power budget available in both PIM architectures when logic and memory power is varied simultaneously to study the maximum on-chip temperature. From Fig. 10(b) we note that M3D based PIM provides a much higher power budget (up to 8W more) than their TSV-based counterpart under similar settings. The higher power budget is achieved as M3D-based architectures do not have layers with poor thermal conductivity and have relatively smaller dimensions (discussed in Sec. 4) which aide in quick dissipation of heat. As a result, the temperature increase is significantly contained allowing more power budget (and multiple layers of logic) in an M3D-enabled PIM without exceeding 85°C.

5.4 Performance Evaluation

Next, we present the NoC and overall full-system performance evaluation. Fig. 11 shows the performance of the optimized M3D-enabled SW-NoC for both Gerbil and PIM-Counter. The NoCs for both Gerbil and PIM-Counter have been designed following the same optimization methodology discussed in Sec. 4.2. Here, we assume a 64-core architecture arranged in four layers (16 cores per layer) with M3D-integration for both Gerbil and PIM-Counter. From Fig. 11, we note that the optimized NoC for PIM-Counter outperforms its Gerbil counterpart by 14% on average for all the datasets. This happens as PIM-Counter has a more balanced traffic, which reduces hotspots leading to faster communication. On the other hand, Gerbil has a handful of cores contributing significantly higher traffic than the rest (Fig. 3). Routers/Links near them experience more congestion, affecting performance.

Moreover, it is well known that the performance of the *k-mer* counting is input-dependent [22]. Hence, it is possible that an NoC optimized with traffic pattern associated with one input may perform sub-optimally when used for a different input. However, it is not desirable for an NoC design to exhibit significant performance variation across inputs. Fig. 12 shows the NoC performance variation for different datasets. For this experiment, we consider the same NoC as in Fig. 11, which was optimized following the traffic pattern (f_{ij} in Eqns. (1)-(4)) of one input dataset (*seen*) e.g. *Bac.* This NoC is then used for executing *k-mer*



(a) (b)

Fig 10: (a) Maximum on-chip temperatures with varying number of logic layers for *k-mer* counting, and (b) Power budget study, in TSV and M3D-based PIM architectures

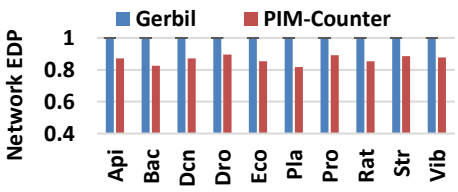


Fig. 11: Performance of optimized M3D-based NoC with PIM-Counter, normalized with respect to Gerbil running on an equivalent platform.

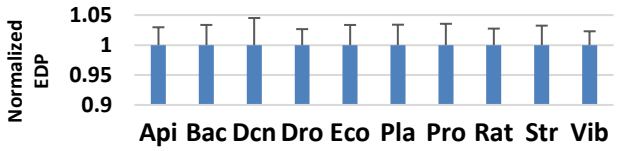


Fig. 12: NoC performance variation for different inputs

counting with other remaining (*unseen*) nine datasets i.e. *excluding Bac*, and so on. We observe from Fig. 12 that the designed M3D-based SW-NoCs for PIM-Counter shows minimal (<= 4.5%) performance variation when other datasets are tested for performance. This happens as PIM-Counter uniformly distributes traffic among the PEs for all datasets (Fig. 9), which reduces the input-dependent behavior. Hence, from Fig. 11 and Fig. 12, it is clear that PIM-Counter enables better NoC design that outperforms Gerbil and deliver high-performance communication support for all inputs tested. However, as Gerbil spends more than 85% of the time for memory (and I/O) accesses (Fig. 2), only improvement in NoC performance does not capture the performance gain of the proposed architecture for a large portion of time Therefore, full system experiments are necessary.

For full-system evaluation, we compare the execution time for the 64-core NoC-enabled 3D-PIM architecture (similar to Fig. 5) running PIM-Counter (PC + 3D-PIM), with Gerbil executing on an equivalent 3D architecture connected to a conventional HMC (GHMC). The cores are equally distributed over four layers and connected by optimized M3D-enabled SW-NoCs (same ones considered in Fig. 11) for both the cases. Overall, GHMC includes both (a) software baseline: Gerbil, and (b) hardware baseline: 2.5D PIM architecture with HMC similar to [9]. Here, we do not use the custom FPGA-based PEs proposed in [9] as they are specifically designed to implement probabilistic approximate counting approaches, which is different than exact counting implemented by both PIM-Counter and Gerbil. However, please note that the proposed PIM architecture is generic and can incorporate any type of PEs (e.g. the exact counting, FPGA equivalent of [9]) instead of the x86 cores used here. Fig. 13 shows the full system runtime comparison between GHMC and PC + 3D-PIM. From Fig. 13, we note that PC + 3D-PIM outperforms GHMC by up to 7.14X in runtime. The improvement is achieved as PIM-Counter avoids external I/O and promotes the use of on-chip memory while multiple layers of logic, optimized NoC and 3D-PIM enable faster computation, communication and easier memory access.

6 CONCLUSION

Counting *k-mers* is a memory-intensive task essential in several bio-informatics applications that process DNA and protein sequences. Existing software frameworks significantly improve the processing of *k-mer* counting. However, without proper architectural support, software gains cannot be fully realized. In this work, we proposed an NoC-enabled software/hardware co-design that enables high-performance *k-mer* counting on a 3D-PIM architecture. We show that using M3D integration, it is possible to design PIM with multiple logic layers and significantly higher power budget without violating temperature constraints. As a result, we upgrade the capabilities of traditional PIM

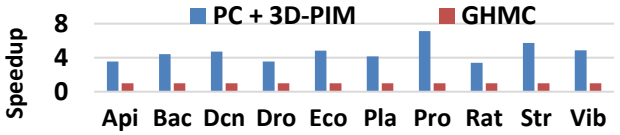


Fig. 13: Speed-up in execution time using proposed co-design framework over Gerbil + conventional HMC

architectures i.e. more computation capability with lesser footprint. Overall, the proposed architecture shows up to 7.14X better execution times compared to a state-of-the-art software framework executed on an equivalent 3D manycore system.

ACKNOWLEDGMENTS

This work was supported in part by the US National Science Foundation (NSF) grants CCF-1815467, CNS-1564014, CCF-1514269 and US Army Research Office grant W911NF-17-1-0485.

REFERENCES

[1] S. Deorowicz, M. Kokot, S. Grabowski, A. Debudaj-Grabysz, 2015. KMC 2: fast and resource-frugal *k-mer* counting. *Bioinformatics*, 31(10),1569–76
[2] M. Erbert et al., 2016. Gerbil: A fast and memory-efficient *k-mer* counter with GPU-support, *Algorithms Mol. Biol.*, 12, 9
[3] E. Azarkhish, D. Rossi, I. Loi, and L. Benini. 2016. Design and Evaluation of a Processing-in-Memory Architecture for the Smart Memory Cube. *ARCS*, Volume 9637
[4] J. Liu et. al., 2013. An experimental study of data retention behavior in modern DRAM devices: implications for retention time profiling mechanisms. *SIGARCH Comput. Archit. News* 41, 3, 60-7
[5] Y. Eckert, N. Jayasena, and G.H. Loh, 2014. Thermal Feasibility of Die-Stacked Processing in Memory. *Workshop on Near-Data Processing*
[6] Y. Zhu, B. Wang, D. Li, and J. Zhao. 2016. Integrated Thermal Analysis for Processing in Die-Stacking Memory. *MEMSYS*. Alexandria, 402-414
[7] G. Kim, J. Kim, J. H. Ahn and J. Kim, 2013. "Memory-centric system interconnect design with Hybrid Memory Cubes," *in PACT*, Edinburgh, pp. 145-155
[8] G. Rizk, D. Lavenier and R. Chikhi, DSK:*k-mer* counting with very low memory usage. *Bioinformatics*. 2013;29(5):652–3
[9] N. Mcvicar, C. Lin and S. Hauck, 2017. "*K-mer* Counting Using Bloom Filters with an FPGA-Attached HMC," in FCCM, Napa, CA, pp. 203-210.
[10] J. Ahn et. al., 2015. A scalable processing-in-memory accelerator for parallel graph processing, *ACM/IEEE ISCA*, Portland, pp. 105-117
[11] S. K. Samal et.al., 2014 Fast and accurate thermal modeling and optimization for monolithic 3D ICs," *ACM/EDAC/IEEE DAC*, San Francisco, pp. 1-6
[12] A. Pattnaik et. al., Scheduling Techniques for GPU Architectures with Processing-in-memory Capabilities. In PACT, 2016.
[13] R. Hadidi *et al.*, 2018. "Performance Implications of NoCs on 3D-Stacked Memories: Insights from the Hybrid Memory Cube," *in IEEE ISPASS*, Belfast, 2018, pp. 99-108
[14] N. Binkert, et al. The GEM5 Simulator, in *ACM SIGARCH Computer Architecture News*, 39(2):1-7
[15] R. Chikhi et. al., 2014. On the representation of de Bruijn graphs. Intl. conf. on Research in computational molecular biology, pp. 35-55. Springer,.
[16] B. K. Joardar, et. al., 2019 "Learning-Based Application-Agnostic 3D NoC Design for Heterogeneous Manycore Systems," in *IEEE TC*, vol. 68, no. 6,
[17] K. Chang, et. al. 2016, "Match-making for Monolithic 3D IC: Finding the right technology node," *in DAC*, Austin, TX, pp. 1-6.
[18] S.K. Samal, D. Nayak, M. Ichihashi, 2016 Monolithic 3D IC vs. TSV-based 3D IC in 14nm FinFET technology, *Proc. SOI-3D-Subthresh. Microel. Tech. Unified Conf.*, Burlingame, 2016, pp. 1-2
[19] S. Das et. al. 2017. Design-Space Exploration and Optimization of an Energy-Efficient and Reliable 3-D Small-World Network-on-Chip. In TCAD. Sys. 36, 5, 719-732.
[20] S. Li et. al., 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures, IEEE/ACM MICRO, New York, pp. 469-480
[21] R. Zhang, M. R. Stan, and K. Skadron, HotSpot 6.0: Validation, Acceleration and Extension, University of Virginia, Tech. Report CS-2015-04
[22] J. Alneberg et.al., 2014. Binning metagenomic contigs by coverage and composition. *Nature methods* 11, no. 11: 1144