Off-Chip Congestion Management for GPU-based Non-Uniform Processing-in-Memory Networks

Kishore Punniyamurthy
The University of Texas at Austin
kishore.punniyamurthy@utexas.edu

Andreas Gerstlauer
The University of Texas at Austin
gerstl@ece.utexas.edu

***Abstract*—Advancements in packaging and 3D-stacking technology have enabled novel non-uniform processing-in-memory (NUPIM) architectures integrating a network of hybrid modules each with varying sizes and proportions of compute and memory. In such systems, network congestion can have a significant impact on performance and energy-efficiency scaling. At the same time, NUPIM systems often have strict power and thermal constraints, where traditional routing/redundancy or buffering based approaches to reduce congestion cannot be applied.**

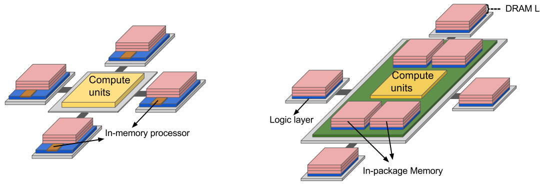
In this paper, we propose a low-overhead yet scalable scheme for congestion management in off-chip networks of emerging GPU-based NUPIM systems. We note that congestion in NUPIM networks exhibits a form of tree saturation in which individually congested links form congested paths, where congestion effects due to back pressure vary across paths. Additionally, memory responses can have different performance impact due to memory divergence. Our approach exploits differences in performance impact of NUPIM traffic to mitigate congestion effects. We dynamically track congested paths and divergent data to manage priorities and accelerate performance-critical traffic. We further use critical paths knowledge to dynamically manage link widths and save I/O energy.

Results show that our congestion management outperforms a configuration with double the number of virtual channels and buffers indicating its effectiveness in managing congestion under stringent constraints. It improves performance by 15% (and up to 31%) over baseline and 10% (and up to 29%) against a memory distance based approach for congestion-affected benchmarks. Our scheme reduces 78% of I/O energy on average and up to 29% of system energy across all benchmarks.

I. Introduction

Recent technological trends have aided the design and development of large-scale heterogeneous systems: 1) 3D-stacking has enabled opportunities to place memory over compute units [1] resulting in architectures as shown in Fig. 1a, and 2) advancements in packaging technology now allow integrating high-bandwidth memory in the same package as compute using silicon interposers [2] as shown in Fig. 1b. These trends have opened up a new class of non-uniform processing-in-memory (NUPIM) system architectures. NUPIM systems consist of multiple modules each integrating memory and compute (2.5D or 3D stacked) together in the same package and interconnected via an off-chip network [3], where GPUs have become the most popular compute units for in- or near-memory processing [2], [3].

Unlike existing systems that have a centralized compute architecture (Fig. 2), the inherently distributed nature of NUPIM systems (Fig. 3) enables scaling of memory and compute capabilities with minimum fabrication and engineering overhead.



(a) In-memory processing.

(b) In-package memory.

Fig. 1: Technology trends.

At the same time, this scalability comes at the cost of large off-chip data-movement. Associated network congestion can be a major factor that adversely affects performance in such systems. As such, alleviating the adverse impact of off-chip congestion is vital for scaling of performance with system size.

Traditional solutions to reduce congestion involve adaptive routing over redundant links in the network [4], [5]. However, NUPIM systems often have more stringent constraints. Both individual stacked modules as well as complete systems deploying NUPIM architectures have strict power and thermal budgets [6], [7], where I/O links and buffers already contribute to high energy overhead constituting 73% of network [8] and 31-35% of router energy [9], respectively. As such, adding redundant links between modules or additional monitoring/overlay networks is typically not feasible. Alternative approaches that rather aim to mitigate the adverse effects of congestion have been proposed for memory networks [10], but they don't address any-to-any traffic patterns seen in NUPIM systems with distributed compute/memory modules.

In this paper, we study the data movement and congestion within off-chip networks of GPU-based NUPIM systems, and we propose a scalable way to manage congestion and alleviate its adverse effects with minimum additional resources. NUPIM systems exhibit any-to-any inter-module traffic patterns in their off-chip networks, where individual links can become oversubscribed and saturated. They are therefore susceptible to a form of tree saturation [5], [11] in which the branches of multiple overlayed saturated trees are comprised of individually congested links, where the effect of congestion increases along some paths more than others due to back pressure. Our proposed scheme dynamically predicts congested paths within the network. Once a critically congested path has been identified, flits traversing along it are prioritized over other flits to improve performance. We also use critical path information to reduce interconnect energy by aggressively and adaptively varying width of non-critical links without significantly impacting performance.

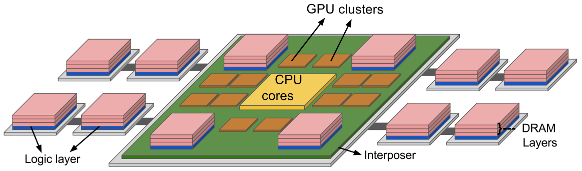


Fig. 2: Traditional centralized system architecture.

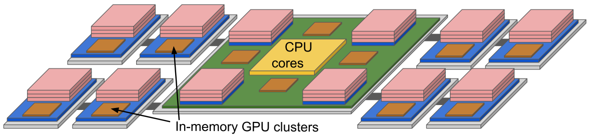


Fig. 3: NUPIM system architecture.

Furthermore, we exploit that not all memory response traffic equally contributes to performance. In GPUs, memory divergent data responses are less likely to contribute to warp progress [12]. We mark flits to indicate if they originate from divergent memory instructions, where flits from non-divergent instructions are prioritized to maximize performance benefits.

The rest of the paper is organized as follows: We analyze congestion and its impact in Section II. Our proposed congestion management scheme is described in Section III. Section IV presents our experimental setup followed by results in Section V. Related work is discussed in Section VI. The paper concludes with a summary in Section VII.

II. Motivation

In the following, we compare the performance and data movement bottlenecks of NUPIM and traditional centralized systems to evaluate the penalty for distributing compute. We further study network congestion and memory divergence behavior to identify opportunities for mitigating congestion effects. We evaluate systems with a total of 64 SMs and 16 memory stacks, where 8 memory stacks are placed in a large central module while the rest are externally connected in a tree topology. For a traditional centralized configuration, all compute is placed in the central module. By contrast, NUPIM configuration consists of modules with equal compute to memory ratios with the central module having 32 SMs and other modules having 4 each. Details about our experimental setup are provided in Section IV.

Fig. 4 shows the performance and off-chip data movement for both configurations normalized to that of centralized configuration. Benchmarks are categorized as high or low sharing based on their data-thread locality. Applications in which threadblocks share pages and therefore cannot be cleanly partitioned are categorized as high-sharing. By contrast, applications with mostly private pages are labelled as low-sharing (see Section IV-B). We use round-robin page mapping for the former and locality-based data mapping for the latter. We notice that, in addition to their inherent scalability, NUPIM systems can, in some cases significantly, reduce inter-module data movement and improve performance for low-sharing benchmarks. By contrast, a NUPIM system loses up to 20% performance and endures up to 88% higher off-chip traffic for high-sharing benchmarks. This performance degradation is due to the high off-chip traffic and resulting network congestion.

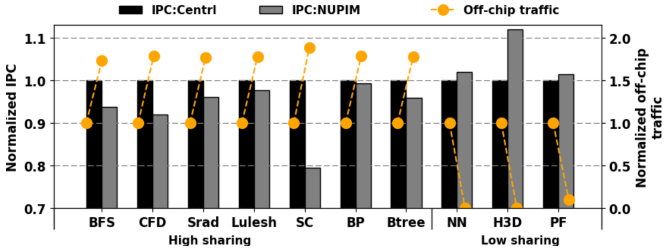
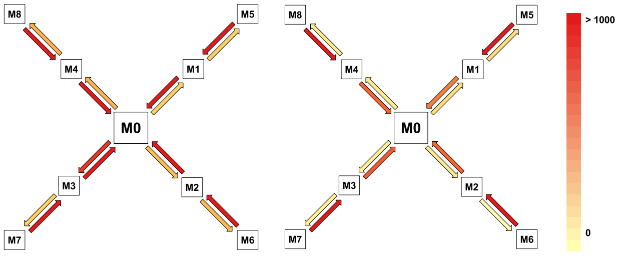


Fig. 4: IPC vs off-chip traffic.



(a) *BFS* (b) *CFD*

Fig. 5: Interconnect congestion stalls.

In order to scale performance and energy-efficiency with system size, the penalty for distributing compute has to be reduced. NUPIM configurations have inherent advantages for low-sharing benchmarks. However, data cannot be cleanly partitioned for high-sharing applications. To design a general-purpose architecture that can support all classes of applications with highest performance, there is a need to mitigate the adverse effects of off-chip data traffic and congestion under given resource/energy constraints in NUPIM systems. We observe that data traversing the links of NUPIM networks can have varying impact on performance. We aim to exploit such behavior to mitigate the performance impact of off-chip data traffic and congestion.

We first study congestion behavior to demonstrate the varying importance of different traffic paths within NUPIM networks. Congestion heatmaps for *BFS* and *CFD* benchmarks (representative high-sharing benchmarks, see Section IV) executed on a NUPIM system with central module *M0* and external modules *M1-M8* are shown in Fig. 5. The colors of the links indicate the number of stalls experienced by flits traversing along the specific link due to a buffer being full. They are normalized against the average congestion stalls experienced by a single connection to an in-package stack. We can make the following observations: 1) Individual congested off-chip links can form a large critically congested path. E.g., in *BFS* (Fig.5a), $M6 \rightarrow M2 \rightarrow M0 \rightarrow M3$ is a critical path; 2) Due to back-pressure, a small number of stalls at the end of a critical path snowballs into a large number of stalls at the source of the path as evident from the increasing intensity of congestion from *M0* to *M5* along the critical path in *CFD* (Fig. 5b); and 3) The critical path can vary across applications as seen from the differences between Fig. 5a and 5b.

These observations show that distributed NUPIM systems are vulnerable to a form of tree saturation [5], [11]. While tree saturation can occur from both end-point and network congestion [13], we find network-congestion due to over-

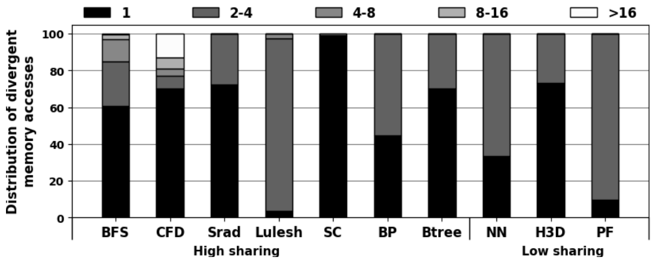


Fig. 6: Distribution of divergent memory accesses.

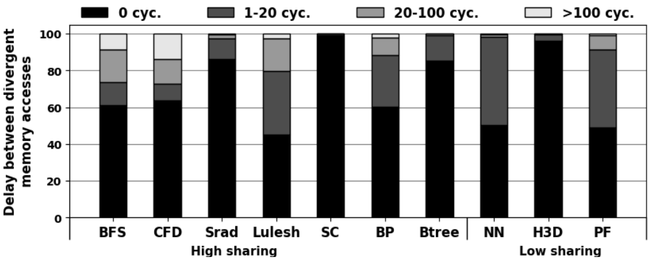


Fig. 7: Delay between first and last divergent response.

subscription of links around the central module to be the dominant cause. In our applications, traffic was roughly equal across all modules (the average coefficient of variation in the number of flits across different nodes is $< 7\%$) and no end-point hot-spots were observed. This is expected because of interleaved data-mapping [14] and similarity in thread memory accesses. Even if a different mapping results in hot-spots, network congestion is likely to remain the main cause for saturation due to limitations on the number of off-chip links in NUPIM systems. All in all, links comprising the critical paths have larger impact on performance than other links. As such, prioritizing traffic on critical over non-critical paths allow reducing congestion impact and improving performance.

Similar to path-based differences in importance, the performance impact of data packets traversing a link can also depend on the nature of the request. In GPUs, a warp load instruction can result in up to 32 memory accesses due to memory divergence [12], where all resulting memory requests must be serviced for the warp instruction to complete. Fig. 6 shows the distribution of divergent memory accesses resulting from individual warp instructions for different benchmarks. On average, 46% of warp load instructions result in more than one memory access. Memory divergence is more prevalent in high-sharing applications due to their irregular access patterns. For example, *BFS* and *CFD* have 15% and 23% of their memory instructions resulting in 4 or more accesses, respectively.

In case of multiple accesses, there can be a significant delay between divergent responses. Fig. 7 shows the distribution of delay between the first and last response (excluding L1 hits) received per memory instruction for different benchmarks. A delay of 0 indicates that only 1 request was serviced at L2 or memory. It can be seen that up to 27% of memory responses have more than 20 cycles delay between first and last response. This indicates that even if 1 divergent response returns, there can be a significant delay before the instruction completes. Such delays are only expected to exacerbate as NUPIM system size scales. As such, servicing requests with lower memory divergence has higher probability to contribute to warp progress. Prioritizing responses for load instructions with lower divergence reduces more memory stalls and increases the chance of making forward progress.

III. Off-chip Congestion Management

We propose a low-overhead and scalable way to detect critically congested paths and divergent data responses to intelligently manage congestion within a NUPIM network while minimizing cost. We detect critical paths using a novel lightweight technique to link congested ports across switches. On detecting the critical paths, we prioritize the packets traversing along the critical path during the virtual channel (VC) and switch allocation stages of the router pipeline to improve performance. Additionally, the flits originating from non-divergent instructions are identified and prioritized over other flits traversing along the same link. Finally, knowledge about congested paths is further used to save energy by aggressively varying link width [15] for non-critical links while not affecting performance significantly.

A. Critical Path Detection

As discussed earlier, a critical path is a combination of individual congested links. Detecting a critical path requires identifying: 1) the congested output ports in every switch, and 2) the input ports that are linked to congested ports of adjacent switches. This information should be gathered in a scalable manner to ensure feasibility across networks of varying sizes. Congestion in output ports can be tracked by having stall counters for every output port [16]. Every switch has counters to track the number of stalls due to congestion (i.e., output buffer full). These counters get reset at regular intervals of time (epoch) and the counter values are used to determine the congestion for each port in a specific epoch.

Traditional solutions to reduce congestion in on-chip networks approaches employ separate monitoring networks to communicate stall counters to adjacent switches and thus achieve regional congestion awareness [16]. We propose a novel approach to track globally congested paths without the need for additional communication overhead. Our approach uses tracker bits attached to regular flits to identify the input ports that are linked to congested ports of adjacent switches and thus establish congested paths.

Our approach requires 1 additional bit for bookkeeping in the header of every flit. Since the packet size is not usually a multiple of the flit size, 1 extra bit does not usually result in additional flits per packet. Tracker flits are regular flits whose tracker flag is set. Any regular flit can act as a tracker flit when its tracker flag is set. At the beginning of each epoch, a tracker flit is sent along the most congested downstream port in every switch. This is done by setting the tracker flag of a flit that is about to traverse the selected link. The flag is cleared once it reaches the subsequent router. When a tracker flit is sent along an output port, the port is marked as critical by setting the bit corresponding to the port in a critical port bitmask that is introduced in every switch. Every switch also

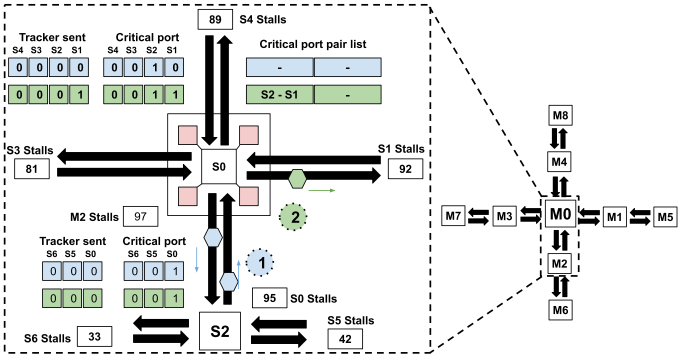


Fig. 8: Critical path detection.

sends a tracker flit along its most congested downstream port on receiving a tracker flit. Hence, a chain of tracker flits are sent along the sequence of congested links tracking the critical path. Every time a tracker flit is triggered due to an incoming tracker flit, the input port from where the incoming tracker was received and the output port along which the outgoing tracker was sent are recorded in a list of critical input-output port pairs per switch. The critical port pair list is a list of input-output port tuples that are part of a critical path, where the information about the entire critical path is not available in any single switch but is distributed among the routers allowing the technique to scale with network size.

To prevent the same output port from being repeatedly selected in the same epoch, every switch maintains a bit mask of output ports along which tracker flits have been previously sent. When a new incoming tracker is received, an outgoing tracker flit is sent along the most congested downstream port that has not yet been tracked. To limit the number of ports being tracked, ports should have at least $k\%$ of the stalls of the port with highest congestion to be considered as a candidate, where k is a tunable parameter.

Fig.8 shows our critical path detection scheme in action. Consider a NUPIM system in a tree topology, specifically focusing on S0 and S2 switches. For the purpose of this explanation, flits from S5 and S6 to S2 are not considered. Initially, the critical port and tracker sent bitmasks for every port are cleared. Furthermore, the critical port pair list is empty and the congestion stalls incurred by every output port in the previous epoch are assumed to be available. The tracking threshold is assumed to be $k = 50\%$. At the beginning of an epoch (step 1 in the figure), modules M2 and M0 dispatch tracker flits along their most congested output ports, which are assumed to be along S0 and S2. The corresponding critical bits for the ports are set in each switch. As these tracker flits reach the destination switches, they in turn trigger additional tracker flits in those switches (step 2 in Fig. 8). In switch S0, a tracker flit is sent along S1 since it is the next most congested downstream port that has not been tracked and has congestion above the threshold. The critical bit for the corresponding port is set and the input-output port tuple is inserted into the critical port pair list. In switch S2, no additional tracker flit is sent since the congestion along all downstream ports is lower than 50% of the highest congested port.

At the end of each epoch, the tracker sent bitmask and stall counters are cleared. The entries in the critical port pair list and critical port bitmask need to be cleared to prevent saturation, but such that the entries don't fluctuate in and out of the list across epochs. This is achieved by adding a h-bit hysteresis counter. List and bitmask entries are cleared only if they are not selected for 2^h consecutive epochs. A 6-bit hysteresis counter was determined suitable for our design empirically and is used in all our evaluations.

B. *Path-based Prioritization*

At the end of each epoch, every switch knows the the input-output port tuples that are part of the critically congested path(s) and the relative extent of congestion along each. The switch then uses this information to prioritize the flits traversing along the identified critical input-output port pairs during the VC and switch allocation stages. This in turn reduces congestion along the critical path at the cost of data moved along other paths. The manner in which the most congested ports in each switch are identified ensures that the input-output port tuples are inserted into the critical port pair list in decreasing order of congestion. In the proposed scheme, packets are prioritized according to the order of their input-output port pairs in the list. Therefore, packets traveling along highly congested paths are selected over other congested paths. Since a strictly priority-based scheme is used, fairness and starvation can be potential issues. However, we did not observe any deadlocks in the applications evaluated. We experimented with fairness-enforcing schemes that select non-critical packets with some probability, but they did not perform better.

C. *Divergence-based Prioritization*

As mentioned earlier, not all types of data request traffic contribute equally to performance. In our scheme, we prioritize requests/responses with no divergence (i.e. instructions resulting in 1 coalesced memory access) over others traversing along the same link as they have higher chance of contributing to forward warp progress. Other prioritization heuristics are possible. For example, prioritizing the slowest among the divergent requests. However, identifying such requests is not simple and individual memory instruction latency does not significantly impact GPU performance as long as some threads can make progress and hide it. Identifying non-divergent memory requests is simpler and prioritizing them ensures that their corresponding warp will make progress. Note that our prioritization does not reduce memory divergence.

The coalescer within each SM generates memory requests for every memory instruction. In our scheme, the coalescer also sets a bit to indicate if the request is divergent or not as part of its header information. A 4-entry priority buffer is added per input port, which is exclusively used by flits of non-divergent requests/responses. The priority buffer acts as small virtual channel with higher priority that allows non-divergent responses to bypass other traffic traversing along the same link. We reduce the size of normal VCs by the size of the priority buffer to keep the total buffer overhead constant. During VC allocation, non-divergent flits are allocated entries within the priority buffer if space is available. Otherwise, normal VCs are used similar to other traffic. When a port is selected during VC or switch allocation, the flits within the priority buffer are selected over those from normal VCs. Since the priority-buffer is only used by non-divergent requests/responses, it is vulnerable to under-utilization. However, since the applications evaluated have significant proportion of non-divergent traffic (54%), under-utilization of the priority buffer is not a significant problem. For a general case, it is possible to have designs where the priority buffer acts as normal VC when not in use, as previously done in [17].

D. Link Width Management

Knowledge about critical paths can be further exploited for dynamic and adaptive optimization of energy consumption. In order to save energy, the width of interconnect links can be varied dynamically [15], where their bandwidth and energy changes proportionally. We adopt this approach to reduce links down to 1/2, 1/4 or as low as 1/8 of their original width every epoch. The penalty for varying the width is set as 1μs [15], epoch length is set to trade off responsiveness versus per-epoch link management overhead. We selected a length of 40μs (40x larger than the link adjustment time) to balance and amortize the overhead. Link width is determined differently for critical and non-critical links. For critical links, the width is adjusted in accordance with the intensity of data traffic. By contrast, for non-critical links, the width is adjusted aggressively to save energy and less importance is given to traffic intensity.

The link width for critical links is determined based on the flit rate and downstream traffic intensity. We obtain the flit rate by counting the number of credits received at each output port in an epoch. In addition, the congestion stalls for each output port reflect the traffic intensity downstream along the port. The traffic intensity is compared against the flit rate for each output port and the link width (and in turn the bandwidth and energy) is set to its maximum if the former is higher than the latter. Otherwise, the width is set such that the resulting bandwidth is a tier higher than the minimum required to support the data rate downstream. More specifically, let FR_p and T_p be the flit rate and traffic intensity (number of congestion stalls) measured along port p, and let W and BW be default/maximum link width and the full bandwidth possible along any link. The link width W_p along a critical port p is then determined as:

$$W_p = \begin{cases} W, & T_p > FR_p \\ W, & FR_p > BW/4 \\ W/2, & FR_p > BW/8 \\ W/4, & \text{otherwise} \end{cases}$$

For non-critical links, the width is only determined by the downstream traffic intensity. If the traffic intensity along a port is more than a preset threshold w times that of the port with the highest stalls, then the width is raised to the next higher tier. Otherwise, the link width is reduced to the next lower tier until it is set to the lowest width. The threshold w is set to 0.9 for our evaluations. Let the width of W_p along a non-critical port p be initially set to its default/maximum W. Furthermore, let $T_{max} = \max_p T_p$ be the highest stalls incurred along any output port in a specific epoch. The link width is adjusted as:

$$W_p = \begin{cases} \min(2W_p, W), & T_p > 0.9T_{max} \\ \max(W_p/2, W/8), & \text{otherwise} \end{cases}$$

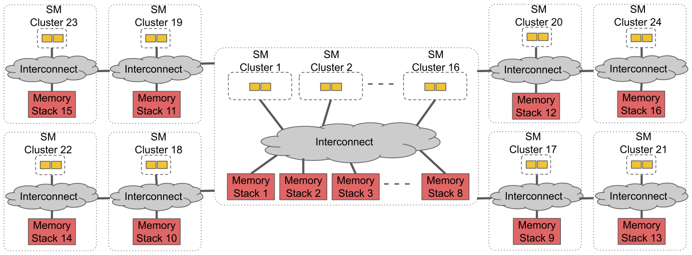


Fig. 9: Simulation setup.

IV. Experimental Setup

We evaluate the impact of our optimizations on performance and energy for a distributed NUPIM system architecture over a range of benchmarks. We first provide details of system configurations and our experimental setup followed by a discussion of the benchmarks evaluated.

A. System configurations

We use GPGPUSim v3.2.2 [18] coupled with GPUWattch [19] for modeling of SM power consumption and DRAMPower [20] to model the memory and I/O power. The simulator has been modified to replicate our system configurations and congestion management scheme. We configure GPGPUSim to model 3D-stacked memory according to Hybrid Memory Cube (HMC) specifications [21] since HMC allows integration of units into a network. The capacity of each stack is the same and total memory is set to be equal to the memory footprint of each application evaluated. We assume that up to 4 GTX480 SMs fit within the logic layer of external memory stacks. Considering the area available in HMC logic layers [22] and the size of vault controllers [23] and the area for GTX480 SMs [24]. 4 SMs can comfortably fit within the logic layer [25] without violating thermal and power considerations [7]. While many NUPIM configurations are possible, we analyze a NUPIM system in which multiple modules having homogeneous memory-to-compute ratios are interconnected together. This configuration can easily be scaled with growing compute and memory requirements. Our simulation setup is shown in Fig. 9, and the exact simulator parameters for the configurations evaluated are summarized in Table I.

B. Benchmarks

We use a range of benchmarks from Rodinia [28] and Lulesh [29] benchmark suites to evaluate our scheme. The complete list of benchmarks is shown in Table II. Benchmarks are simulated with specified inputs to completion or for 1 billion instructions, whichever occurred first.

Since congestion in a NUPIM network is greatly dependent on application behavior and partitioning of data and compute, it is necessary to understand application behavior and data mapping used for evaluating such systems. As mentioned

TABLE I: Simulation parameters.

Architectures		
System parameters	Total SMs Total DRAM stacks Centrl pkg SMs SMs/Ext. stack	64 Centrl:8, Ext.:8 32 4
SM Configuration		
Core configuration	1.4Ghz, GTO warp scheduler [18]	
Private L1 cache	32kB, 4-way, write through [1]	
Shared L2 cache	1MB, 16-way, write through [1]	
Interconnect		
Frequency	2.5Ghz	
Topology	Ext: Tree, Centrl pkg: Fully connected	
Switch	islip allocation, credit-based flow control [18], Min routing [26]	
Virtual channels(VC)	2 VC per port	
Bandwidth		
In-pkg bandwidth	160 GB/s per stack [1]	
Off-chip bandwidth	80 GB/s [1]	
Memory Stack		
Memory stack configuration	16 memory stacks, 16 vaults/stack, 16 banks/vault, 64 TSVs/vault [1]	
Scheduling policy	FR-FCFS	
DRAM Timing	DDR3 [27]	

TABLE II: Benchmarks.

Benchmark	Input
BFS [28]	1MW
CFD [28]	0.2M
Srad [28]	100, 0.5, 502, 458
Lulesh [29]	Default
Streamcluster (SC) [28]	Default
Backprop (BP) [28]	256k
Btree [28]	mil.tx
NN [28]	list5120k_512.txt
Pathfinder (PF) [28]	100000,100,20
Hotspot3D (H3D) [28]	512, 8, 100, power512x8, temp512x8

earlier, applications can vary from having negligible to significant inter-thread sharing. To understand and differentiate application memory access patterns, we define the average page sharing of an application as $PS_{avg} = \frac{1}{N}\sum_{k=1}^{N} Tb_k$, where Tb_k is the number of unique threadblocks accessing page k and N is the total pages in the application's working set. Thus, PS_{avg} indicates the average number of unique threadblocks sharing a page. A PS_{avg} value of 1 indicates that no two threadblocks access same page. For applications with multiple kernels, the PS_{avg} of each kernel is weighted and averaged based on the dynamic memory instruction count.

In some applications, a small set of pages is shared by a large number of threadblocks while the vast majority of pages are shared among a few. This results in a small PS_{avg} value despite having page sharing. Therefore, we also define the maximum page sharing $PS_{max} = \max_k Tb_k$ as the maximum number of threadblocks any page is shared with.

PS_{avg} and PS_{max} for different benchmarks are plotted in Fig.10. We distinguish between low-sharing applications that can be cleanly partitioned and high-sharing applications that share pages among threadblocks throughout this paper. In reality, applications usually don't fall into a single category completely. We therefore categorize applications based on their PS_{avg} and PS_{max} values. *NN*, *Hotspot3D*, and *Pathfinder* have small PS_{avg} and PS_{max} and are categorized as low-sharing while others are considered as high-sharing.

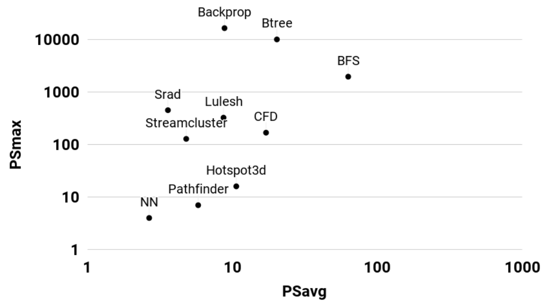
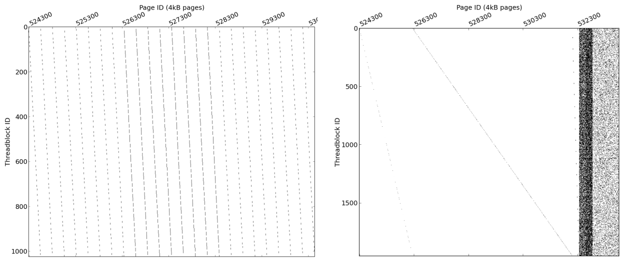


Fig. 10: Average (PS_{avg}) vs. maximum (PS_{max}) page sharing of benchmark applications.



(a) Hotspot3D (b) BFS (Z6kernel)

Fig. 11: Benchmark memory access patterns.

Previous work [30] suggested that a locality-based data mapping suits regular benchmarks with low page sharing. Hence, while evaluating low-sharing benchmarks, we map data based on the locality policy presented in [30]. We use pre-collected memory traces to statically place the pages into appropriate memory stacks. For irregular benchmarks with high page sharing, other work has demonstrated that interleaving data provides best performance [14]. For such high-sharing benchmarks, we thus interleave data across modules in chunks of 256 Bytes [31] for our evaluations.

V. Results

We apply our congestion management scheme on a homogeneous NUPIM configuration as described in Section IV to evaluate its effectiveness in alleviating the adverse effects of off-chip congestion. We compare a baseline configuration without improvements (*Base*) to a configuration that performs only critical path prioritization (*Crit*), a configuration that performs both critical path and non-divergent traffic prioritization (*Crit+Tr*) and a setup that additionally performs dynamic link width management (*Crit+Tr+W*). We compare our schemes against distance-based (*Dist*) arbitration from [10] and a baseline NUPIM configuration with double the number of virtual channels and buffers (*Double*). Note that doubling the virtual channels introduces high area and power overheads (up to 74%) [32] that would violate NUPIM constraints and thus reduce the available budget for in-memory compute.

A. Performance Results

Fig. 12 shows the IPC of different benchmarks across different configurations normalized to *Base*. For high-sharing benchmarks, we can notice that *Dist* only achieves marginal performance improvement (up to 3%). This is because it does not account for variations in congestion between paths of the

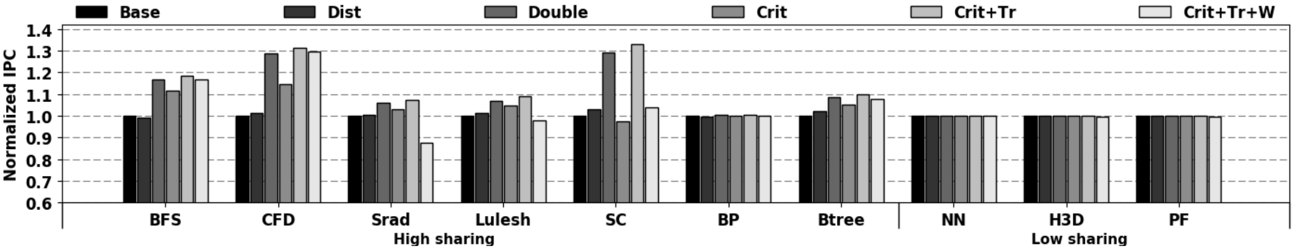


Fig. 12: Performance across different NUPIM congestion management configurations.

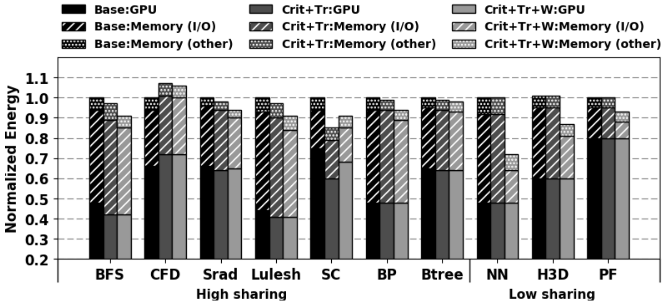


Fig. 13: Energy breakdown.

same length and treats all data as equally important. *Crit* achieves 5% average (and up to 15%) improvement, while *Crit+Tr* achieves 16% average (and up to 33%) improvement over baseline. *Crit+Tr* outperforms *Dist* by 10% on average (and up to 29%) and exceeds the performance obtained by *Double*, indicating that the proposed congestion management scheme is effective in eliminating the adverse impact of off-chip traffic congestion while adding minimum overhead. Our link width management scheme (*Crit+Tr+W*) achieves lower benefits (6% on average and up to 30%) due to reduced link widths, but saves I/O and system energy as will be shown later. The extent of improvement observed across benchmarks varies depending on the impact of congestion on their performance. For example, congestion is more severe for *CFD* compared to *Backprop* and *CFD* thus benefits more from our proposed optimizations. The *Srad* benchmark experiences a drop in performance since it has multiple kernels that are smaller than the epoch length, where our scheme responds slowly.

Low-sharing benchmarks are not affected by our proposed scheme as they have low congestion. This is because data can be partitioned cleanly among modules for these benchmarks, thereby avoiding the congestion problem in the first place.

B. Energy Results

Fig. 13 shows the normalized system energy consumed by baseline, *Crit+Tr* and *Crit+Tr+W* configurations. We break system energy down into GPU, memory I/O, and and other memory components. It can be noted that memory energy is dominated by its I/O component. Similar observations have been made in [8]. Our approach for link width management (*Crit+Tr+W*) reduces I/O energy by 78% on average and system energy by 8% (and up to 29%). For *Streamcluster*, *Crit+Tr* consumes less energy because of its higher performance and smaller execution time. Energy consumed increases for *CFD* because of increased utilization of GPU resources.

VI. Related Work

While memory networks [8], [10] and processing-in-memory [1], [33] have been studied and evaluated extensively in isolation, to the best of our knowledge no prior work has evaluated combined networks of non-uniform processing-in-memory (NUPIM) modules. For tree saturation due to network congestion, adaptive routing has been proposed and widely studied [13]. Alternatively, to mitigate congestion impact, traffic prioritization and acceleration have been proposed. In the following, we discuss related work in these areas.

Numerous works on adaptive routing [5], [16], [34] measure interconnect congestion and dynamically configure the routing algorithm accordingly. But such approaches assume topologies that provide multiple, redundant routes between nodes. This may not be feasible in a NUPIM system due to higher I/O energy required for any additional/redundant off-chip links [8]. Indirect adaptive routing techniques [35] have been proposed for large interconnect networks without the need for additional overlay networks. We use a similar Piggyback (PB) approach to communicate 1-bit information across routers over packets, but our approach carries information needed for detecting congested paths and not identifying alternate routes. As such, we also send information (tracker bits) only along congested links and do not broadcast it across all routers. Furthermore, our approach only stores state about input-output port pairs and does not need to track the state of other routers.

Recently, researchers have analyzed memory networks and have shown the importance of interconnect networks on performance and energy. The work in [10] proposed a distance-based arbitration scheme to improve memory network performance. However, as we have seen in Fig. 5, equidistant modules within NUPIM systems can have varying congestion along the route and hence traffic across them should be prioritized accordingly. [8] explores different power management techniques for memory networks such as DVFS and variable width links (VWL) under network-unaware and -aware scenarios. Their approach relies on the assumption that upstream links have higher traffic than immediate downstream links. This does not apply for distributed NUPIM architectures as each module could have both memory and compute.

Some previous approaches have proposed accelerating packets but mostly rely on network/interconnect characteristics or data packet type [36] to select packets to be prioritized. [37] allows packets traversing a certain number of hops in a particular direction to skip the internal processing pipeline of and thus creating an effect of bypassing intermediate routers. Others have suggested skipping switch allocation stages based on temporal locality of crossbar connections [38]. Our approach considers the behavior of both the network and the core issuing requests while prioritizing traffic.

Memory divergence has been widely studied and many approaches have been proposed using warp scheduling [39], cache management [12], [40] and memory scheduling [41] to solve divergence issues. In contrast to these approaches, we do not target reducing memory divergence itself, but exploit memory divergence information to accelerate packets with the highest performance impact.

VII. Summary and Conclusions

Current technology trends have opened up a new class of NUPIM system architectures. Although many NUPIM system configurations are possible, configurations with distributed memory and compute are best suited for ensuring system scalability. At the same time, congestion can have adverse impact on performance scaling in such systems. Any-to-any inter-module traffic (and congestion) patterns and stronger power and thermal constraints make congestion in such systems different from traditional on-chip and memory networks. Therefore, existing redundancy/routing based or up/down stream specific solutions cannot be applied.

We show that NUPIM systems are susceptible to tree saturation due to network congestion resulting in critical paths comprising of several individual links. We exploit the varying congestion across paths as well as the differing impact of data packets on performance due to memory divergence in NUPIM networks. We propose a novel, lightweight scheme that exploit such variations to mitigate congestion effects while reducing energy without incurring the overhead of additional links. Our scheme achieves on average 16% (and up to 33%) improvement over baseline and 10% (and up to 29%) improvement over other congestion mitigation schemes for high-sharing benchmarks. Our scheme with link width management can save 78% of I/O energy on average and up to 29% system energy while achieving up to 30% improvement.

VIII. Acknowledgements

The authors would like to thank the reviewers for their comments. This work was partially supported by the National Science Foundation (NSF) under grant CCF-1725743.

References

[1] K. Hsieh *et al.*, "Transparent offloading and mapping (TOM): Enabling programmer-transparent near-data processing in gpu systems," in *ISCA*, 2016.

[2] T. Vijayaraghavan *et al.*, "Design and analysis of an apu for exascale computing," in *HPCA*, 2017.

[3] (2014, Feb.) The amd fastforward project. [Online]. Available: https://asc.llnl.gov/fastforward/AMD-FF.pdf

[4] M. Farrens, B. Wetmore, and A. Woodruff, "Alleviation of tree saturation in multistage interconnection networks," in *SC*, 1991.

[5] T. Lang and L. Kurisaki, "Nonuniform traffic spots (nuts) in multistage interconnection networks," *Journal of Parallel and Distributed Computing*, vol. 10, no. 1, Sep. 1990.

[6] A. Subcommittee, "The opportunities and challenges of exascale computing," U.S. Dept. of Energy, Tech. Rep., 2010.

[7] D. Zhang *et al.*, "Top-pim: throughput-oriented programmable processing in memory," in *HPDC*, 2014.

[8] X. Jian, P. K. Hanumolu, and R. Kumar, "Understanding and optimizing power consumption in memory networks," in *HPCA*, 2017.

[9] H. Wang, L. Peh, and S. Malik, "Power-driven design of router microarchitectures in on-chip networks," in *MICRO*, 2003.

[10] M. Poremba *et al.*, "There and back again: Optimizing the interconnect in networks of memory cubes," in *ISCA*, 2017.

[11] G. F. Pfister and V. A. Norton, "Hot spot contention and combining in multistage interconnection networks," *IEEE Transaction on Computers*, vol. C-34, Oct. 1985.

[12] R. Ausavarungnirun *et al.*, "Exploiting inter-warp heterogeneity to improve gpgpu performance," in *PACT*, 2015.

[13] G. Kim *et al.*, "Contention-based congestion management in large-scale networks," in *MICRO*, 2016.

[14] A. Mariano *et al.*, "Analyzing and improving memory access patterns of large irregular applications on numa machines," in *PDP*, 2016.

[15] D. Abts *et al.*, "Energy proportional datacenter networks," in *ISCA*, 2010.

[16] P. Gratz, B. Grot, and S. W. Keckler., "Regional congestion awareness for load balance in networks-on-chip," in *HPCA*, 2008.

[17] W. Kwon *et al.*, "In-network reorder buffer to improve overall noc performance while resolving the in-order requirement problem," in *DATE*, 2009.

[18] A. Bakhoda *et al.*, "Analyzing cuda workloads using a detailed gpu simulator," in *ISPASS*, 2009.

[19] J. Leng *et al.*, "Gpuwattch: enabling energy optimizations in gpgpus," in *ISCA*, 2013.

[20] K. Chandrasekar *et al.*, "System and circuit level power modeling of energy-efficient 3d-stacked wide i/o drams," in *DATE*, 2013.

[21] H. M. C. Consortium, "Hybrid memory cube specification 2.1," Tech. Rep., 2015.

[22] J. Jeddeloh and B. Keeth, "Hybrid memory cube new dram architecture increases density and performance," in *VLSIT*, 2012.

[23] E. Azharkish *et al.*, "High performance axi-4.0 based interconnect for extensible smart memory cubes," in *DATE*, 2015.

[24] (2009, Oct.) Ati and nvidia face off–obliquely. [Online]. Available: https://www.cnet.com/news/ati-and-nvidia-face-off-obliquely

[25] R. Panda *et al.*, "Prefetching techniques for near-memory throughput processors," in *ICS*, 2016.

[26] N. Jiang *et al.*, "A detailed and flexible cycle-accurate network-on-chip simulator," in *ISPASS*, 2013.

[27] *4Gb: x4 x8 x16 DDR3 SDRAM*, 2011. [Online]. Available: Micron Technology

[28] S. Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," in *IISWC*, 2009.

[29] I. Karlin *et al.*, "Lulesh programming model and performance ports overview," Lawrence Livermore National Lab, Tech. Rep., 2012.

[30] M. Diener, E. H. M. Cruz, and P. O. A. Navaux, "Locality vs. balance: Exploring data mapping policies on numa systems," in *PDP*, 2015.

[31] *GPGPUSim v3.2.2. GTX 480 Configuration*, 2009.

[32] Y. J. Yoon *et al.*, "Virtual channels vs. multiple physical networks: A comparative analysis," in *DAC*, 2010.

[33] B. Hong *et al.*, "Accelerating linked-list traversal through near-data processing," in *PACT*, 2016.

[34] Z. Qian *et al.*, "A traffic-aware adaptive routing algorithm on a highly reconfigurable network-on-chip architecture," in *CODES+ISSS*, 2012.

[35] N. Jiang, J. Kim, and W. J. Dally., "Indirect adaptive routing on large scale interconnection networks," in *ISCA*, 2009.

[36] Z. Li *et al.*, "Latency criticality aware on-chip communication," in *DATE*, 2009.

[37] A. Kumar *et al.*, "Express virtual channels: towards the ideal interconnection fabric," in *ISCA*, 2007.

[38] M. Ahn and E. J. Kim., "Pseudo-circuit: Accelerating communication for on-chip interconnection networks," in *MICRO*, 2010.

[39] T. G. Rogers, M. O'Connor, and T. M. Aamodt, "Divergence-aware warp scheduling," in *MICRO*, 2013.

[40] A. Arunkumar, S. Lee, and C. Wu, "Id-cache: instruction and memory divergence based cache management for gpus," in *IISWC*, 2016.

[41] N. Chatterjee *et al.*, "Managing dram latency divergence in irregular gpgpu applications," in *SC*, 2014.