Delegated Replies: Alleviating Network Clogging in Heterogeneous Architectures

Xia Zhao
Artificial Intelligence Research Center
Academy of Military Science
Beijing, China
zhaoxiahust@gmail.com

Lieven Eeckhout
ELIS Department
Ghent University
Ghent, Belgium
lieven.eeckhout@ugent.be

Magnus Jahre
Department of Computer Science
Norwegian University of Science and Technology
Trondheim, Norway
magnus.jahre@ntnu.no

***Abstract*—Heterogeneous architectures with latency-sensitive CPU cores and bandwidth-intensive accelerators are attractive as they deliver high performance at favorable cost. These architectures typically have significantly more compute cores than memory nodes. The many bandwidth-intensive accelerators hence overwhelm the few memory nodes, resulting in suboptimal accelerator performance — as their bandwidth needs are not met — and poor CPU performance — because memory node blocking creates high latencies. We call this phenomenon network clogging. Since network clogging is a widespread issue in heterogeneous architectures, we first investigate if existing state-of-the-art approaches can address it. We find that the most effective prior approach, called Realistic Probing (RP), is suboptimal because it searches the local caches of other cores for missing data.**

We propose Delegated Replies which lets memory nodes speculatively delegate the responsibility of replying to last-level cache hits to the private cache that last accessed the requested cache block, hence avoiding the search that fundamentally limits RP. Moreover, Delegated Replies uses the (typically) under-utilized request network for delegation; it is the reply network links of the memory nodes that commonly clog because replies include complete cache blocks in addition to metadata. We evaluate Delegated Replies in the context of heterogeneous architectures with latency-sensitive CPU cores and bandwidth-intensive GPU cores and find that it improves GPU (CPU) performance by 14.2% (5.2%) and 25.7% (8.8%) on average compared to RP and our baseline, respectively.

***Keywords*-Network on Chip (NoC); heterogeneous architecture; multi-core processor; Graphics Processing Unit (GPU).**

I. Introduction

Heterogeneous architectures that integrate CPUs and accelerators on a single chip are attractive because they provide high performance while limiting cost, power consumption, and physical size [48], [57]. Moreover, heterogeneous architectures provide low-overhead accelerator invocation as they avoid explicitly or implicitly copying input/output data at accelerator invocation and completion [59]. For these reasons, heterogeneous architectures have become popular: see Apple's M1 [7], Intel's Skylake series [21], AMD's Accelerated Processing Units (APUs) [55], NVIDIA's Tegra-family [48], and Xilinx's Adaptive Compute Acceleration Platform (ACAP) [23].

Compute cores typically outnumber memory nodes in single-chip architectures (due to pin restrictions [9]) and multi-chip modules (due to high integration costs [39]). While many CPU applications are latency-sensitive, accelerators typically hide memory latencies by combining streaming programming models with highly parallel compute architectures — making them both bandwidth-intensive and bandwidth-sensitive [19], [34]. Single-chip integration of bandwidth-intensive and latency-sensitive cores exposes a difficult trade-off to the memory system, i.e., it must maximize bandwidth for accelerator requests and simultaneously minimize latency for CPU requests. Furthermore, accelerators and CPUs exhibit completely different Network-on-Chip (NoC) injection characteristics. In particular, accelerator cores inject massive bursts of requests into the NoC while CPU cores inject relatively few requests. This leads to a fundamental problem, which we call *network clogging*, where requests from the many compute cores clog the links leaving the few memory nodes [3], [33], [58]. Clogging typically occurs in the reply links of the memory nodes because (i) accelerators typically load more data than they store, and (ii) load replies contain both metadata and a complete cache line (while load requests only contain metadata).

NoC clogging is undesirable for bandwidth-sensitive accelerators as they are severely performance-limited by the bandwidth of the bottleneck links. Moreover, clogging is independent of network topology since each memory node has a single reply network link in contemporary topologies [17], [41], [42]. Clogging also has a second-order effect on CPU performance as the flood of accelerator requests delay CPU requests. To minimize the latency of CPU requests, our baseline architecture (i) gives priority to CPU traffic over accelerator traffic, and (ii) carefully places memory controllers to minimize interference (see Section V for a detailed analysis). Unfortunately, clogging frequently causes the reply network interface of the memory nodes to block due to full injection buffers. Giving CPU traffic high priority is insufficient because blocking denies it from entering the buffer. The scheduler is hence unaware of the CPU requests and cannot prioritize them.

Since NoC clogging is such an ubiquitous issue in heterogeneous architectures, it is natural to expect that state-of-the-art approaches would address it effectively. We hence

take great care in designing an efficient baseline and then implement and evaluate state-of-the-art approaches that we expect could potentially alleviate network clogging, including various NoC topologies [41], [42] as well as the DC-L1 [30] and DynEB [29] L1 cache sharing schemes. (We describe this endeavor in detail in Section III.) Perhaps surprisingly, we find that existing approaches are either too costly or ineffective because they attempt to overcome NoC clogging within the cache subsystem *or* the NoC. Cache-centric approaches (e.g., [5], [13], [28], [29], [30], [31]) rely solely on inter-core locality, but are fundamentally limited as they either (i) incur the complexity and overhead of precisely tracking sharers for correctness (either at the level of a cache block [5], [28] or across memory regions as in coarse-grain coherence [13]); (ii) deliver suboptimal effective bandwidth to applications with (large) shared data sets (as in shared L1 caches [29], [30]); or (iii) must search for potential sharers (as in Realistic Probing (RP) [31]). In contrast, NoC-centric approaches (e.g., [22], [33], [37], [45]) attempt to address clogging through (selective) overprovisioning or advanced network policies. Within the NoC, performance is fundamentally constrained by the clogged links and can only be improved by providing more bandwidth. This is unfortunately too costly; doubling NoC bandwidth alleviates clogging but increases NoC area by 2.5×.

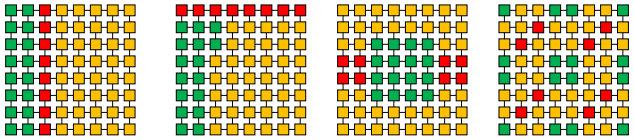
Our extensive analysis leads to our key insight: Effectively addressing NoC clogging requires carefully co-designing the cache subsystem *and* the NoC — which we proceed to do with *Delegated Replies*. In the cache subsystem, Delegated Replies leverages inter-core locality by enabling memory nodes to speculatively delegate the responsibility of providing data to an accelerator core that is likely to have the requested data in its local cache — thereby transforming the few-to-many access pattern into a more favorable many-to-many access pattern at the cost of increased latency. The streaming compute model commonly used in accelerators means that the effective bandwidth improvement significantly outweighs its latency overhead. To identify a core that is likely to have the requested data, we store a pointer in the Last-Level Cache (LLC) to the core that last accessed it; a simple yet accurate heuristic (74.5% average hit rate). By imprecisely tracking sharers, we significantly reduce complexity compared to coarse-grain coherence [13] (which has to track shared-memory regions precisely for correctness), retain the low-complexity private L1 cache organization (unlike shared L1 caches [29], [30]), and avoid searching for (potential) sharers (unlike RP [31]).

Delegated Replies' NoC component (i) only invokes speculative delegation when memory nodes cannot inject reply traffic into the NoC, and (ii) uses the (typically) under-utilized request network to delegate replies. Since the delegated reply message only contains metadata while a regular reply contains metadata and a complete cache line, Delegated Replies effectively deflects traffic away from the clogged links. More specifically, a request occupies a single flow control unit (flit) while a reply occupies 9 flits in our setup. Each delegated reply hence reduces the bandwidth demand on the bottleneck link by 9×. Delegated Replies is applicable to any NoC topology that has core-to-core links between accelerator nodes. Although we use the mesh topology in our baseline, Delegated Replies performs equally well with other topologies such as the crossbar, flattened butterfly [41] and Dragonfly [42] (see Section VII).

Our evaluation focuses on heterogeneous architectures with CPU and GPU cores, but our observations are relevant to any heterogeneous architecture that combines latency-sensitive cores (CPUs) with bandwidth-intensive cores (GPUs and other accelerators). We find that RP [31], which (selectively) searches the L1 caches of other cores for the missing data, is the only state-of-the-art approach that reduces NoC clogging. Unfortunately, RP is suboptimal. On multi-programmed CPU-GPU workloads, Delegated Replies improves GPU performance by 14.2% (up to 30.6%) and 25.7% (up to 65.9%) compared to RP and our baseline, respectively, because RP searches for sharers whereas Delegated Replies accesses a single likely sharer. Delegated Replies effectively moves traffic from the clogged memory-node links to less-utilized inter-GPU links and thereby improves the effective data bandwidth delivered to the GPU cores by 26.5% on average compared to the baseline. By effectively draining the injection buffers of the memory-node routers, Delegated Replies allows CPU traffic to enter the buffer and hence be prioritized — thereby improving average CPU performance by 5.2% (up to 12.8%) and 8.8% (up to 19.8%) compared to RP and the baseline, respectively, across the workloads where the GPU cores clog the memory nodes. We find that the only effective NoC-centric approach is to double NoC bandwidth, but Delegated Replies has much lower overhead. More specifically, Delegated Replies only incurs 5% of the area overhead of the double-bandwidth NoC.

In summary, we make the following key contributions:

- We analyze the network clogging problem in heterogeneous architectures consisting of bandwidth-intensive and latency-sensitive cores. Clogging occurs because cost constraints result in compute cores outnumbering memory nodes which is fundamentally conflicted with the high bandwidth demands imposed by accelerators.
- We carefully analyze core layout in heterogeneous architectures and find that interference between CPU and accelerator traffic can be reduced by (i) placing memory nodes in a column/row between the CPUs and accelerators, (ii) giving CPU requests priority over accelerator requests, and (iii) routing traffic along minimally interfering paths.
- We qualitatively and quantitatively investigate a wide range of architectural optimizations that presumably should address NoC clogging. We find that while cache-



(a) Baseline (b) Layout B (c) Layout C (d) Layout D

Figure 1: The high-level integrated CPU-GPU architectures considered in this work. The CPU cores are colored green, the GPU cores are yellow, and the memory nodes are red.

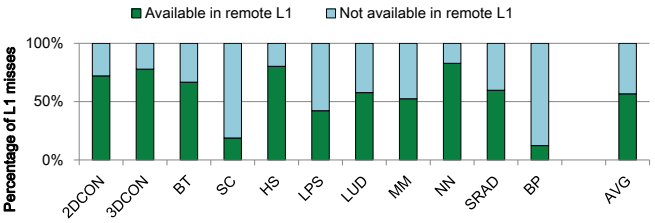


Figure 2: Inter-core locality. *More than 57% of the L1 cache misses are duplicated in the L1 caches of remote GPU cores.*

centric RP [31] improves GPU (CPU) performance relative to our baseline by 10.1% (3.5%) on average, it is suboptimal. Doubling NoC bandwidth alleviates clogging but increases NoC area by 2.5× and is hence too costly.

- We propose *Delegated Replies* which, in contrast to state-of-the-art RP, effectively alleviates NoC clogging by carefully co-designing the cache subsystem and the NoC. More specifically, Delegated Replies improves GPU (CPU) performance by 14.2% (5.2%) and 25.7% (8.8%) on average compared to RP and our baseline, respectively, while only incurring 5% of the area overhead of the double-bandwidth NoC.

II. Explaining Delegated Replies

Understanding network clogging. Our baseline heterogeneous architecture balances CPU, GPU, and memory capabilities as well as minimizes interference between CPU and GPU traffic. We focus on a 64-node system with 40 GPU cores, 16 CPU cores, and 8 memory nodes. The GPU cores consist of a Streaming Multiprocessor (SM) and a local cache, the CPU nodes consist of a processor and a local cache, and the memory nodes include a slice of the shared Last Level Cache (LLC) and a memory controller. Our baseline features a mesh NoC with a 358 GB/s bisection bandwidth and a 236 GB/s DRAM system. We consider a mesh topology because of its scalability and wide deployment in SoC systems, but Delegated Replies is equally applicable to other topologies such as the flattened butterfly [41], the Dragonfly [42], and the crossbar with inter-core links. Details about our experimental setup are provided in Section VI, and Section VII presents various sensitivity analyses with respect to network topology and bandwidth, chip layout, and routing policies.

Our baseline minimizes interference between CPU and GPU traffic by (i) placing the memory controllers in a column of the mesh between the CPUs and GPUs, (ii) prioritizing CPU requests over GPU requests (since GPU applications are typically latency-tolerant), and (iii) employing state-of-the-art Class-based Deterministic Routing (CDR) [3] (see Figure 1a and Section V). CDR is a Dimension-Order Routing (DOR) scheme that chooses a different dimension-order for requests and replies to better balance traffic across links. The baseline uses a light-weight software-based coherence protocol for the GPU caches which flushes the private cache when necessary (e.g., on a kernel boundary) as is common in GPUs [8], [44]. (We discuss coherence in detail in Section III.)

Even if we have taken great to minimize network clogging in our baseline, all of the memory node's GPU-side NoC-links are heavily loaded (i.e., have over 60% utilization on average); note that 100% single-link utilization is unattainable in practice because different traffic streams interleave in downstream routers. Clogging exerts significant back-pressure on the memory nodes which results in blocking rates between 72.3% and 78.8% on average — effectively reducing NoC bandwidth to the capacity of the clogged links. In other words, the GPU cores overwhelm the reply capability of the memory nodes which results in suboptimal performance for the GPU workloads as they receive insufficient bandwidth. At the same time, clogging creates sufficient back-pressure to deny CPU requests access to the buffers of the memory nodes, causing high latencies which in turn reduces CPU performance.

Addressing network clogging hence requires reducing the bandwidth demand on the GPU-side links of the memory nodes. Fortunately, Figure 2 shows that GPU-compute applications have significant inter-core locality. On average, over 57% of the cache lines missing in the local L1 cache are available in at least one remote L1 cache (see Section VI for more details regarding our benchmarks). The key challenge is hence to retrieve data from remote L1 caches rather than the LLC for the subset of cache misses for which remote L1 copies exist without incurring excessive overhead — thereby deflecting traffic away from the clogged links of the memory nodes.

Network clogging mechanisms. We now show how Delegated Replies addresses NoC clogging by speculatively delegating the responsibility of responding to a GPU request to an L1 cache that is likely to have the requested data. Since clogging occurs in the GPU-side links of the memory nodes, Figure 3 focuses on the injection buffer of the memory node and its reply and request routers. Although we assume physically separate request and reply networks in our baseline, Delegated Replies is equally applicable to systems that use Virtual Channels (VCs) to implement logically separate networks (see Section VII).

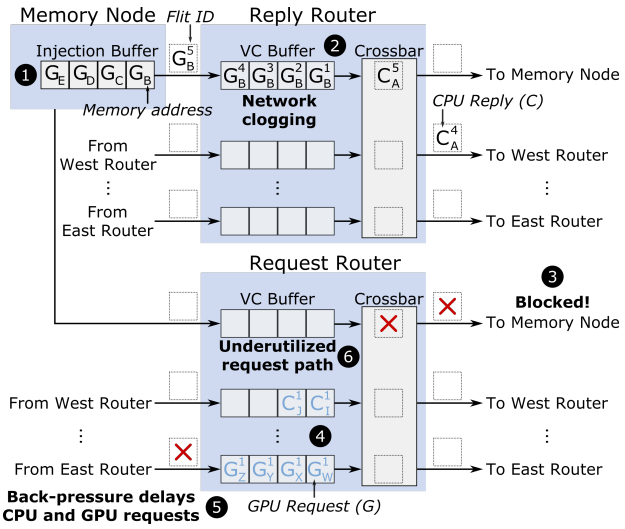


Figure 3: Network clogging example.

When a request has been satisfied by the LLC (or memory), the memory node places the reply data and metadata in its injection buffer. Figure 3 shows the state of the injection buffer and memory node routers in a given cycle. Here, GPU replies G_B, G_C, G_D, and G_E are available in the injection buffer and ready to be inserted into the reply network (see ❶). Since replies are larger than the NoC link width, they are divided into flow control units (flits), and a reply leaves the injection buffer when all of its flits have been injected into the network; our NoC does not drop flits. In our setup, a response contains 9 (5) flits because each GPU (CPU) response consists of a header flit and 8 (4) data flits (our link width is 16 bytes and the L1 cache lines are 128 (64) bytes).

We employ wormhole flow control [35] which means that the flits of a single reply can be distributed across multiple routers. For the purpose of this example, we assume that each router can store four flits internally and contains a single VC; our simulated baseline NoC supports 2 VCs that each can buffer 4 flits. When a flit arrives at the router, it is placed in the VC buffer. Then, the switch arbitrates access to the router's crossbar. The crossbar is fully connected and can hence transfer any flit that does not (i) come from the same input port, or (ii) requires the same output port in parallel. To minimize latency for CPU traffic, we give it higher priority than GPU traffic throughout the memory system (including the switch allocator). Hence, the flits of CPU reply C_A are given priority over the flits of GPU reply G_B (see ❷). In this example, we assume single-cycle link traversal, VC allocation, and crossbar traversal (our experiments faithfully model a 4-cycle router).

CPU traffic uses the west-ward links of the routers and GPU traffic uses the east-ward links in our baseline core layout (see Figure 1a). Unfortunately, the many-to-few access pattern leads to the memory node's injection buffer

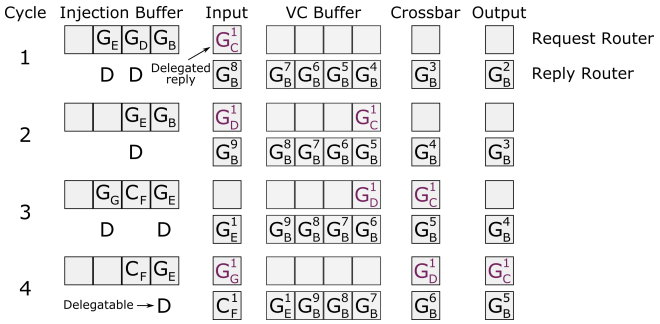


Figure 4: Delegated Replies example.

quickly filling up with GPU replies. The first-order effect of this behavior is that the flits of request G_B occupy all of the buffer space in the reply router and hence delay the other ready replies (i.e., G_C, G_D, and G_E) at ❷. We denote requests/replies from CPUs and GPUs as C and G, respectively; the subscript denotes the memory address, and the superscript is the flit identifier.

The second-order effect of clogging is that the memory node blocks as it has no place to store replies (see ❸). Hence, CPU request C_I^1 cannot proceed beyond the VC buffer of the request router as the memory node cannot accept requests when it is blocked (see ❹). Giving CPU requests high priority is hence insufficient in isolation. Memory node blocking also creates back-pressure that propagates through the NoC and eventually results in CPU and GPU stalls because (i) they cannot proceed without the requested data, or (ii) they run out of space in their injection buffers or MSHRs (see ❺).

Delegated Replies. Figure 3 also shows that the request path from the memory node to the GPUs (i.e., the east router) is underutilized (see ❻), and we now illustrate how Delegated Replies exploits this resource to re-balance network load. As aforementioned, Delegated Replies requires the LLC to store a pointer to the GPU core that most recently accessed each cache line (say core A). If a different core than A, say core B, sent the current cache-hit request, we define the reply as *delegatable*. When the memory node cannot inject traffic into the reply network, it delegates the responsibility of replying to B's delegatable reply to core A as it most likely still has the data in its local L1 cache. Repeated LLC accesses to the same cache block from a single GPU core can occur when the core has recently evicted this cache line from its local cache. Note that all decision-making in Delegated Replies is done at the end-points (i.e., memory nodes or GPU cores) since they have the necessary information available, whereas the NoC routers treat delegated replies as any other request.

Figure 4 shows the memory node injection buffer and the relevant state of the request and reply routers. Replies G_C, G_D, and G_E are delegatable (and hence marked D). Since G_C is delegatable and the reply router is full, the memory node issues a delegated reply for G_C in cycle 1. In cycle 2, the reply router is still clogged and the memory node

delegates G_D. In cycle 3, the last flit of G_B (i.e., G_B^9) has entered the reply router's VC buffer and the reply network can accept additional traffic. The memory node hence decides to not delegate G_E even if it is delegatable. The reason is that delegated replies incur a latency overhead. Although GPUs are inherently latency-tolerant, we do not want to unnecessarily expose the cores to overhead.

Delegating G_C and G_D has made space available in the injection buffer which immediately unblocks the memory node. As a result, the replies C_F and G_G enter the injection buffer in cycle 4. Since C_F is a CPU reply, the scheduler gives it priority over G_E in cycle 5; Delegated Replies hence enhances the effectiveness of priority-based mechanisms. Since the reply network is now servicing the flits of C_F (and subsequently the remaining flits of G_E), the scheduler delegates G_G.

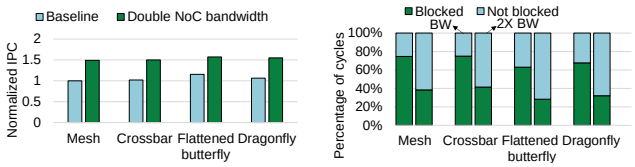
III. Evaluating Prior Approaches

While we have now explained how Delegated Replies effectively alleviates network clogging, we have yet to explain why existing approaches fall short. Hence, we now delve into the details of prior cache-centric and NoC-centric approaches and show that they either have high complexity, high area overhead, or are ineffective against network clogging.

A. Cache-Centric Approaches

Directory coherence. A well-known approach to exploit inter-core locality is to add forwarding states to directory-based cache coherence protocols (e.g., MESIF and MOESI). In MESIF [28], a single private cache is given responsibility for servicing requests to a shared (clean) cache block (the F state), while the MOESI [5] protocol provides similar functionality for dirty cache blocks (the O state). These approaches are fundamentally different from Delegated Replies as they have to track *all* sharers *precisely* (for correctness in case a cache line needs to be invalidated) while Delegated Replies tracks a *single* sharer *imprecisely*, i.e., erroneously delegating a reply can hurt performance but the application still executes correctly. Approaches such as coarse-grain coherence [13] reduce coherence traffic by tracking memory regions rather than cache lines, but they are still precise at the region level for correctness.

Precisely tracking sharers incurs a storage cost, and, more importantly, increases design and verification complexity [53] — even to the point that protocols of commercial chips may only be partially verified [2]. Optimizations such as cache-to-cache transfers [4] or coarse-grain coherence [13] do not address this inherent source of complexity. Contemporary GPUs avoid these overheads by implementing cache coherence in software since their programming model enables straightforwardly identifying synchronization boundaries [8], [44]. In conclusion, addressing NoC clogging with coherence is simply too costly.



(a) GPU performance. (b) Memory node blocking rate.

Figure 5: The impact of changing NoC topology and doubling reply network bandwidth across our GPU benchmarks. *Changing NoC topology does not address network clogging — because all topologies suffer from memory node blocking. Doubling NoC bandwidth helps, but is not cost-effective.*

Shared GPU L1 caches. Another option is to share L1 caches among GPU cores [29], [30]. Fundamentally, L1 cache sharing improves performance when the benefit of improved effective L1 cache capacity (i.e., data accessed by multiple SMs is stored only once) outweighs the cost of lower effective L1 bandwidth (i.e., multiple SMs accessing the shared data *around the same time* results in serialization). L1 cache sharing is orthogonal to Delegated Replies (see Section VII).

Realistic probing (RP). RP [31] provides the benefit of redirecting requests to the private caches of other cores without incurring overheads of coherence protocols and shared L1 caches. Unfortunately, RP is inefficient as it first needs to correctly predict if a cache block is in the private cache of another core and then probe all other private caches to be guaranteed to find the requested data. Hence, RP is stuck between a rock and a hard place: searching too many private caches wastes bandwidth and energy, while searching too few makes finding a cached copy unlikely. Delegated Replies is hence (i) much less complex than RP, and (ii) provides significantly better performance than RP (see Section VII).

B. NoC-Centric Approaches

Overprovisioned NoC. The most straightforward way to address NoC clogging is to provide enough bandwidth such that even the most heavily loaded links perform well. To evaluate this option, we change the NoC topology and increase NoC bandwidth, as shown in Figure 5a which reports GPU performance for the crossbar, flattened butterfly [41] and Dragonfly [42] topologies with a nominal and increased ($2\times$) bandwidth, all normalized to our mesh baseline with nominal bandwidth. We conclude that (i) changing the NoC topology hardly impacts GPU performance, while (ii) doubling NoC bandwidth significantly improves performance. The reason is that clogging in the reply links of the memory nodes results in the memory nodes being blocked most of the time, see Figure 5b. Changing NoC topology does not reduce clogging, as each memory node has a single reply link regardless of topology, whereas bandwidth overprovisioning helps because it doubles the bandwidth of the bottleneck links, regardless

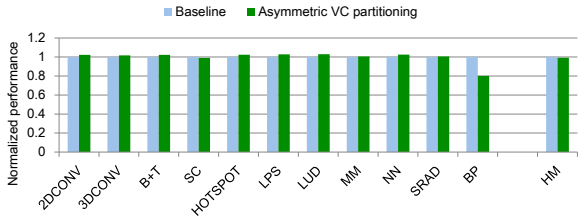


Figure 6: GPU performance with Asymmetric VC partitioning (AVCP) [33] versus our baseline. *AVCP is ineffective.*

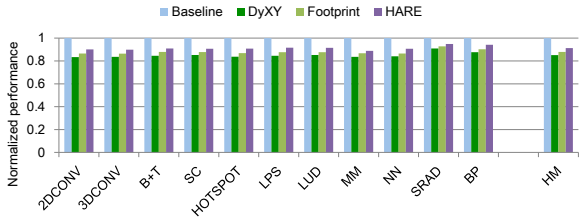


Figure 7: GPU performance with adaptive routing schemes versus our baseline. *Adaptive routing is ineffective.*

of the specific network topology. NoC overprovisioning can be non-uniform [12], but this increases design complexity.

NoC overprovisioning is unfortunately not cost-effective in terms of chip area and power consumption [20], [27]. Using DSENT [54], we find that the area cost of the double-bandwidth mesh is 2.5× higher than our baseline (5.76 mm^2 versus 2.27 mm^2). The reason is primarily that increasing bandwidth increases the area overhead of the routers. More specifically, the area overhead of input buffers and virtual channels increases linearly with channel width, while the area overhead of the router-internal crossbar is quadratic in both port count and channel width. In contrast, Delegated Replies increases NoC area by only 4% (0.092 mm^2); the total area overhead of Delegated Replies is 0.172 mm^2 (see Section IV).

Asymmetric VC partitioning (AVCP). Since NoC overprovisioning is not cost-effective, AVCP [33] attempts to better distribute NoC load by allocating more VCs to reply traffic than to request traffic. While this approach has both low implementation complexity and low area overhead, it is ineffective. More specifically, Figure 6 shows that asymmetrically allocating VCs even in the best case only improves performance by 3% compared to our baseline; Harmonic Mean (HM) performance is practically unaffected. We adopt physically shared request and response networks in this experiment, as this is required by AVCP, but we retain the same aggregate bandwidth as the baseline. AVCP is ineffective because flits get serialized in the output links. For BP, asymmetric partitioning hurts performance because it is write-heavy which stresses the (virtual) request network rather than the (virtual) response network. In summary, AVCP is ineffective because the limiting factor is the bandwidth of the clogged links and using more or less VCs for each traffic class does not affect link bandwidth.

Adaptive routing. To assess the efficacy of adaptive routing against NoC clogging, we implement and evaluate the DyXY [45], Footprint [22], and HARE [37] adaptive routing schemes. Figure 7 compares GPU performance with the adaptive policies to our baseline which uses CDR routing [3]. Perhaps counter-intuitively, adaptive routing actually reduces performance. In the request network, there is no unbalanced congestion which means that we incur the overhead of adaptive routing (i.e., increased congestion in the router-internal crossbar switch [18]) without getting the benefit.[1] In the reply network, all paths from the memory nodes to the GPU cores are clogged. Hence, it is not possible to route traffic around the clogged links — reiterating that the fundamental limitation is link bandwidth.

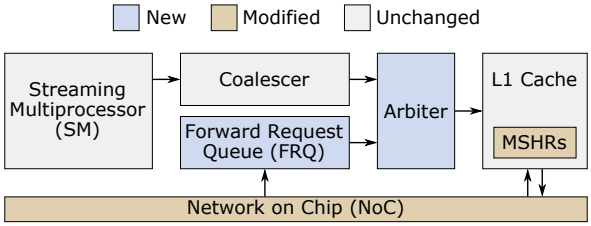


Figure 8: GPU core with support for Delegated Replies.

IV. Implementing Delegated Replies

Architectural support. Figure 8 illustrates the minor architectural modifications required to implement Delegated Replies in a GPU core. The main change is the Forwarded Request Queue (FRQ) which stores the delegated replies sent to the GPU core. The L1 cache treats remote memory requests from other cores in the FRQ similarly to local requests. We do not merge requests in the FRQ as only 4.8% of FRQ-entries access the same cache block and fully exploiting merging requires supporting a multicast mechanism within the NoC.

To avoid deadlock, we need to give remote requests in the FRQ higher priority than local requests. Consider two GPU cores that both experience a local cache miss causing them to stall on a resource that is temporarily unavailable (e.g., MSHR entry or cache line). A deadlock may occur if both cache misses cause remote accesses to the other core. For example, memory requests from core *A* and core *B* first access the LLC and are forwarded to each other, i.e., the core IDs of the requested cache lines point to *B* and *A*, respectively. In core *A*, the local requests may be stalled if the core runs out of L1 cache resources, i.e., all MSHR entries or all cache

[1]We also evaluated adaptive routing with 4 and 8 VCs per router. This partially alleviates router congestion and hence reduces the performance gap between the adaptive schemes and the baseline. However, our baseline is the top performer across all configurations.

lines in a set are allocated for outstanding requests that are sent to the LLC and then to *B*. Core *B* may be in the same situation, in which case both cores are stalled.

The culprit for the deadlock is the fact that priority is given to local requests. One solution is to switch priority between local and remote requests when the local requests cannot proceed due to an L1 stall. This requires monitoring the status of the L1 cache and switching priority back to local memory requests when the L1 is no longer stalled. The alternative solution is to always give priority to remote requests over local requests. Both solutions remove the resource dependence and avoid the deadlock. We experimentally evaluated both solutions and found that they perform similarly. Hence, we adopt the latter because of its simplicity.

Processing delegated replies. There are three possible outcomes when the remote L1 cache serves a delegated reply: (i) a cache hit and the data is available; (ii) a cache hit but the data is not yet available — a hit to an outstanding cache line; or (iii) a cache miss — we call this a *remote miss*. In the first case (cache hit), the core sends a reply to the requesting core through the reply network, i.e., the local cache sends the requested cache line to the remote core. In the second case (delayed hit), the memory request will be added to the list in the MSHR and the core will send the cache line to the requesting core as soon as the miss returns (we assume an allocate-on-miss policy in the LLC). In the third case (remote miss), the memory request will be re-sent to the LLC without allocating an MSHR entry in the GPU core, upon which the LLC will send the data and update the core pointer to the requesting core. Although this third case incurs extra overhead, the negative impact is minimal because the extra request network transfer latency is small and hidden by other memory requests. In addition, keeping the LLC in the loop provides it with improved knowledge of which cores are likely to have a copy of the cache block.

NoC modifications. Delegated Replies also introduces extra requests and replies to enable core-to-core communication. However, we require no changes to the encoding of reply packets and only minimal changes to the encoding of request packets. More specifically, the delegated replies are encoded as normal requests except that they use the identifier of the requesting core as the sender ID (even if they are sent from a memory node). This is necessary for the recipient core to know which core to supply the data to. In addition, we add one additional bit to the request packet, called the *Do-Not-Forward (DNF)* bit, to support remote misses. This DNF bit tells the LLC slice to process the request and not forward it again. The DNF bit does not incur any overhead because a read request is 8 bytes which is smaller than the typical channel width (16 bytes in our setup).

Hardware overhead. The hardware cost of Delegated Replies is (very) small. Assuming 40 GPU cores, each core pointer requires only 6 bits; we add core IDs to the LLC and the MSHRs. Assuming a 48-bit address space [52] and 128 B cache lines, the total hardware cost for the core pointers amounts to $0.08\,\text{mm}^2$ according to CACTI 6.5 [47] assuming a 22 nm technology node. We further set the FRQ size to 8 entries. Using DSENT [54], we estimate the total area cost for the FRQs across all 40 GPU cores to be $0.092\,\text{mm}^2$. The overall hardware overhead of Delegated Replies thus amounts to $0.172\,\text{mm}^2$, which is small (less than 0.1%) compared to a die size of several hundred mm^2 for modern-day chips.

Coherence implications. Software and hardware-based approaches have been proposed by academics [24], [50], [51] and industry consortia (e.g., CXL [16] and CCIX [14]) to maintain coherence between CPUs and accelerators. For software-managed full-system coherence, programmers explicitly specify and copy data between CPU and accelerator address spaces [1], [8], [44]. For hardware-based full-system coherence, a directory protocol is employed at block or region-based granularity [50], [51] to give permission to access a block or region. Delegated Replies operates within the GPU coherence domain and does not cross the CPU-GPU coherence boundary; we model MESI directory coherence in the CPU-domain.

Within the GPU coherence domain, L1 caches need to implement a write-through policy and need to be flushed (invalidated) through compiler-inserted cache control operations, e.g., at the end of kernel execution [50]. Delegated Replies can only operate within a region after having obtained permission, i.e., the GPU cores can only forward shared cache lines between GPU cores in regions for which the GPU holds permission. Upon a write to L1, an invalidation request is sent to the LLC to invalidate the core pointer of the respective cache to make sure that other cores requesting the same cache line from the LLC receive the most recent copy, as in the conventional organization. This means that Delegated Replies will only benefit shared read-only data which is fortunately much more common than shared read-write data in GPU workloads [61]. In addition, when the L1 cache is flushed (invalidated) in response to coherence instructions, all corresponding core pointers in the LLC are also invalidated. We faithfully model the overhead of coherence operations.

V. Chip Layout and Routing

We now return to our baseline architecture to argue how it leverages the placement of memory nodes and choice for dimension-order routing to minimize interference between CPU and GPU cores. More specifically, we discuss the four high-level architectures in Figures 1 (our baseline is Figure 1a). We keep the number of CPU and GPU cores as well as memory nodes constant across architectures to maintain the same theoretical performance and hence ensure fair cross-architecture comparisons. The architectures have a 2.5:1 ratio of GPU cores versus CPU cores, since a CPU core such as Intel Nehalem is nearly 2–3× larger than a GPU core that resembles Nvidia's Fermi [56] scaled to 22 nm

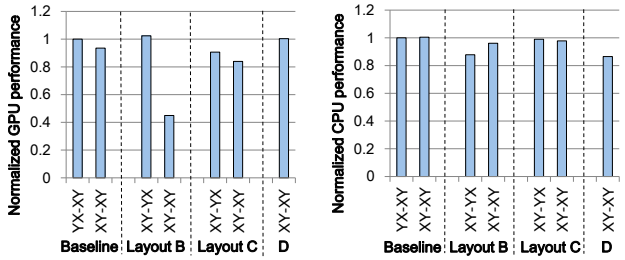
technology, and 8 memory nodes to enable allocating a complete column/row (similar to prior work [32], [40], [46]). We explore architectures with different ratios of core types in Section VI.

Baseline: Traffic isolation. Our baseline layout (Figure 1a) has the desirable property that CPU traffic and GPU traffic are isolated from each other except from within memory node routers where they multiplex onto the links to/from the memory nodes (see Figure 3). Further, the architecture maximizes NoC bandwidth between the memory nodes and both the CPU and GPU cores since the memory nodes have 8 CPU-facing links and 8 GPU-facing links. To avoid congestion in the links of the memory column, we employ CDR routing [3], i.e., YX-order for requests and XY-order for replies. (We will quantify the impact of this choice.)

Layout B: Memory nodes at die edge. Figure 1b is inspired by die photos of recent commercial heterogeneous architectures (e.g., [6]) which place the memory controllers at the edge of the chip, presumably to simplify package integration. Hence, we let the memory nodes occupy the top row of the architecture and allocate columns of CPU and GPU cores (except for one column which contains two CPU cores and five GPU cores). Layout B is similar to Baseline, but prioritizes simplifying integration at the expense of more interference between CPU and GPU traffic. Since we use XY-ordering for requests and YX-ordering for replies to avoid congestion in the memory row, the GPU requests for the three left-most memory controllers must traverse the CPU/mixed columns, and vice versa, the CPU requests for the five (or six) right-most memory controllers must traverse the GPU columns. The architecture also only provides 8 NoC links between the memory nodes and CPUs/GPUs.

Layout C: Clustered CPU cores. Figure 1c prioritizes inter-CPU communication by placing the CPU cores within as few hops of each other as possible. Hence, it prioritizes communication-latency-sensitive CPU workloads at the expense bandwidth-sensitive GPU workloads. In this architecture, we are able to separate a fair amount of the GPU and CPU traffic by adopting an XY-order for requests and YX-order for replies. Layout C provides less bandwidth for GPU cores than Baseline and Layout B since vertical traffic to/from memory nodes is multiplexed onto 4 NoC links.

Layout D: Traffic distribution. The final architecture we consider was used in prior work [38], [46], [59] (see Figure 1d). The intuition behind this architecture is to spread out the different core types across the chip to better distribute network traffic. Although this property is desirable, the architecture (i) does not separate GPU and CPU traffic, (ii) places CPU cores relatively far from memory nodes, and (iii) places CPU cores relatively far from each other. Hence, it prioritizes GPU performance at the expense of CPU performance. Note that we use XY-ordering for both requests and replies as applying different routing policies will not separate GPU/CPU traffic.



(a) GPU performance (b) CPU performance

Figure 9: Average GPU and CPU performance across our GPU and CPU workloads for different layouts and routing policies. *Overall, our baseline layout with YX-XY routing provides both good GPU and CPU performance.*

Performance Analysis. We now analyze the performance of the heterogeneous architectures in Figure 1 with various routing policies (see Section VI for details regarding our experimental setup). We label the architectures in the form *L RequestOrder-ReplyOrder* where *L* is the layout identifier and *RequestOrder* and *ReplyOrder* are the dimension order used by CDR-routing in the request and reply networks, respectively. We normalize CPU (GPU) performance to the CPU (GPU) performance with Baseline YX-XY. For the layouts where we propose to use CDR (i.e., Baseline, B, and C), we also evaluate a configuration that uses the same dimension order for both requests and replies.

Figure 9 illustrates that Baseline is the only layout that provides both high CPU and GPU performance: CPU performance is insensitive to the routing policy while GPU performance benefits from having its own column of links to the memory nodes. For Layout B, it is critical to adopt XY-YX ordering to avoid congestion in the memory controller row. This hurts GPU performance since Layout B (i) does not isolate GPU and CPU traffic, and (ii) CPU traffic is given priority over GPU traffic. As expected, Layout C prioritizes CPU performance at the expense of GPU performance, but GPU performance is marginally better with XY-YX ordering due to better traffic isolation. Layout D provides good GPU performance and less-than-ideal CPU performance.

VI. Experimental Setup

Simulation Setup. To evaluate Delegated Replies, we integrate GPGPU-sim v3.2.3 [10] with BookSim 2.0 [36] and Netrace [26]. We use GPGPU-sim to simulate the GPU cores and memory nodes; we faithfully model contention between CPU and GPU requests in the NoC, shared caches, and memory controllers. Netrace injects CPU network traffic and models how CPU network latency affects CPU performance. BookSim enables cycle-accurate NoC simulation.

Table I: Simulated CPU-GPU architecture.

Parameters	Value
GPU cores	40 SIMT cores, 1.4 GHz, 2 GTO schedulers/core, 1,536 threads/core, 32 threads/warp, 48 warps/core, 32,768 registers/core, 16 KB shared memory/core, 48 KB L1 cache, 4-way associative, LRU, 128 B line, round-robin CTA scheduling
CPU cores	16 cores, 2.0 GHz, 32 KB L1 data/inst cache 4-way, 64 B lines, MESI coherence protocol
Shared LLC	8 MB total, 1 MB/MC, 16-way, LRU, 128 B line
DRAM	8 MCs, FR-FCFS, 16 banks/MC, 236 GB/s
GDDR5	t_{CL}=12, t_{RP}=12, t_{RC}=40, t_{RAS}=28, t_{RCD}=12, t_{RRD}=6, t_{CCD}=2, t_{WR}=12
NoC	8×8 2D mesh with XY-routing, Islip allocator 128-bit channel width, 2 VCs, 4 flits/VC, 358 GB/s bisection bandwidth

Table II: Heterogeneous CPU-GPU workloads.

GPU benchmark		CPU bmk#1	CPU bmk#2	CPU bmk#3
Name	Grid Dim			
2DCON [25]	(128,512,1)	blackscholes	canneal	dedup
3DCON [25]	(8,32,1)	bodytrack	dedup	fluidanimate
BT [15]	(60000,1,1)	dedup	fluidanimate	vips
SC [15]	(1954,1,1)	bodytrack	ferret	swaptions
HS [15]	(342,342,1)	bodytrack	ferret	x264
LPS [10]	(63,500,1)	fluidanimate	vips	x264
LUD [15]	(127,127,1)	ferret	blackscholes	swaptions
MM [49]	(1000,2000,1)	canneal	fluidanimate	vips
NN [10]	(6,6000,1)	blackscholes	fluidanimate	swaptions
SRAD [15]	(128,128,1)	fluidanimate	ferret	x264
BP [15]	(1,16384,1)	blackscholes	bodytrack	ferret

We use the Baseline layout in Figure 1a as our baseline. Memory is partitioned by memory address range across the 8 memory controllers (MCs) following PAE's randomized address mapping [43]. We consider a traditional two-level non-inclusive cache hierarchy with one LLC slice per MC. Each NoC router features a four-stage pipeline; Islip allocation is used in the switch and virtual channel (VC) allocator, where we assign higher priority to CPU packets [59]. Configuration details of the simulated architecture are provided in Table I. Our simulation infrastructure faithfully models all the extra latencies and actions incurred by Delegated Replies.

We use DSENT v0.91 [54] to evaluate NoC power (22 nm tech node). We import activity factors collected through timing simulation into DSENT to compute power consumption. The NoC links are assumed to measure 4.3 mm, and SM area is estimated to be 18.1 mm^2 (from Fermi die size [56], [59]).

Heterogeneous CPU-GPU Workloads. We evaluate Delegated Replies with multi-program CPU-GPU workloads. The GPU workloads exhibit varying degrees of inter-core locality and are taken from CUDA SDK [49], GPGPU-sim [10], Rodinia [15] and PolyBench [25]; we simulate one billion instructions. The CPU workloads are taken from Parsec [11]; we consider the medium input for all

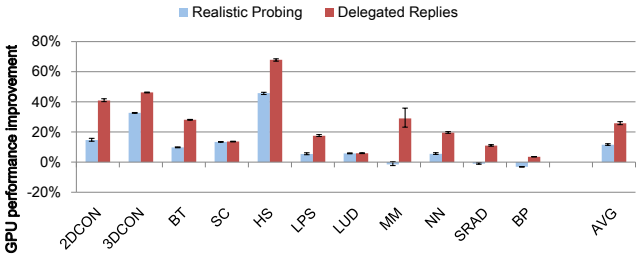


Figure 10: GPU performance improvement. *Delegated Replies improves GPU performance by 25.7% on average and up to 65.9% over the baseline. Compared to RP, Delegated Replies improves performance by 14.2% on average and up to 30.6%.*

benchmarks except two, for which we use the large input (bodytrack and swaptions). The injection rate is (much) higher for the GPU workloads (0.324 to 0.704 flits per cycle) compared to the CPU workloads (0.013 to 0.084 flits per cycle). We construct 33 heterogeneous CPU-GPU workloads by randomly selecting three CPU benchmarks to co-run with each of our 11 GPU benchmarks, see Table II. For each workload, we allocate all CPU cores to the CPU benchmark, and all GPU cores to the GPU benchmark. We use instructions per cycle (IPC) to quantify GPU performance. CPU performance is obtained using Netrace [26] with CPU network latency as input.

VII. Evaluation

We now evaluate Delegated Replies in terms of GPU and CPU performance, effective NoC bandwidth, energy consumption, chip layout, and various sensitivity analyses. We compare to the state-of-the-art Realistic Probing (RP) [31] throughout this section, and we use the best-performing configuration of RP according to the authors.

GPU Performance. We first evaluate the impact of Delegated Replies on GPU performance for our baseline architecture. Delegated Replies improves performance by 25.8% on average, see Figure 10. We note a consistent performance improvement across all workloads which goes up to 67.9% for HS. Further, Delegated Replies improves performance over RP by 14.2% on average and up to 30.6%. It is interesting to observe that GPU performance is rather insensitive to the CPU workload that co-runs with the GPU workload, i.e., the whiskers denote that the minimum to maximum performance variability is limited to on average 1.9% and at most 10.6% (MM). The reason is that although CPU traffic is prioritized in the memory nodes, this does not significantly affect the performance on the GPU side because GPU workloads tend to be latency-tolerant.

The performance improvement strongly correlates with the increase in effective NoC bandwidth. Figure 11 reports the received data rate in flits per cycle for the GPU cores: we

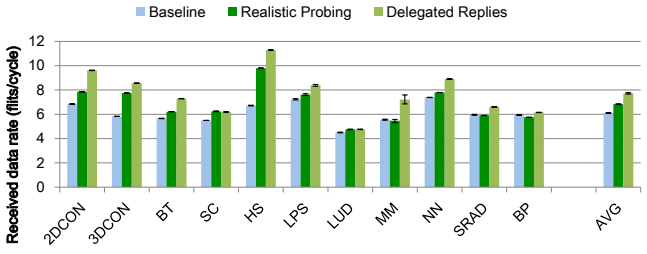


Figure 11: Received data rate for a GPU core in flits per cycle. *Delegated Replies improves the effective NoC bandwidth by 28.5% on average and up to 65.8%.*

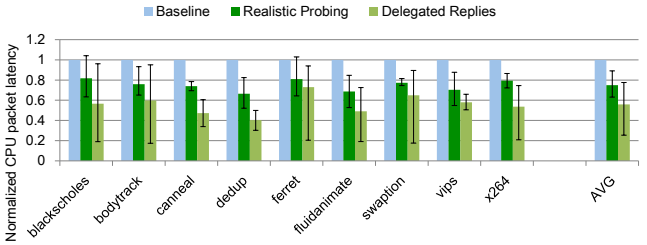


Figure 12: CPU network latency. *Delegated Replies reduces CPU network latency by 44.2% on average and up to 59.7%.*

find that Delegated Replies increases the received data rate by 26.5% on average, and up to 70.9% (HS), versus 11.9% on average for RP. Leveraging core-to-core communication offloads the GPU-side links of the memory nodes in the reply network, which improves the effective NoC bandwidth, i.e., more replies can be satisfied per unit of time. As GPUs are mostly bandwidth-sensitive, this improves performance.

CPU Performance. Delegated Replies reduces pressure on the memory nodes from the GPU side which in turn reduces the network latency on the CPU side. Figure 12 reports normalized CPU packet latency for the CPU workloads. Delegated Replies reduces network latency by 44.2% on average, and up to 59.7% (dedup). The reduction in CPU network latency translates into a CPU performance improvement of 3.8% on average and up to 11.8% (vips), see Figure 13. The whiskers in Figure 12 and 13 denote the min and max packet latency and performance, respectively, per CPU workload across the GPU workloads they co-run with. It is interesting to note that the variability across CPU workloads is typically (much) higher than on the GPU side (compare the whiskers in Figure 13 versus the whiskers in Figure 10). The reason is that some CPU workloads are latency-sensitive (e.g., vips) while others are not (e.g., dedup), and depending on which particular GPU workload co-runs with the CPU workload, this may lead to a larger or smaller impact on CPU performance. Across the workloads where GPU traffic causes significant network clogging, Delegated Replies improves CPU performance by 8.8% on average (and up to 19.8%) compared to the baseline (and by 5.2% on average and up to

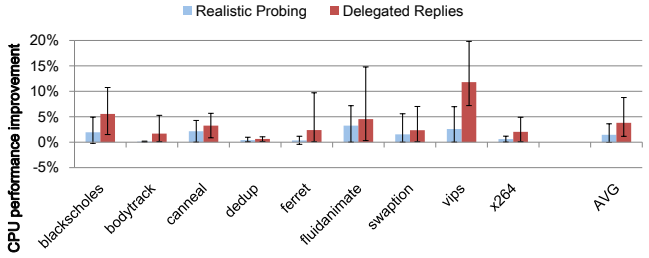


Figure 13: CPU performance improvement. *Across the workloads where GPU traffic causes significant network clogging, Delegated Replies improves CPU performance by 8.8% on average and up to 19.8%.*

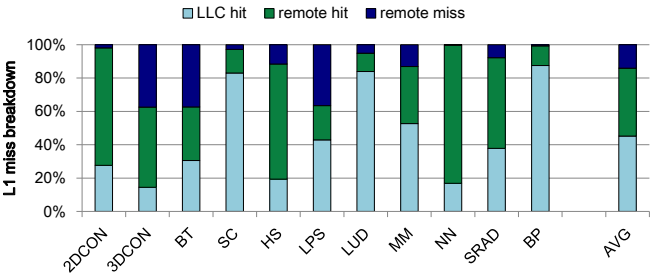
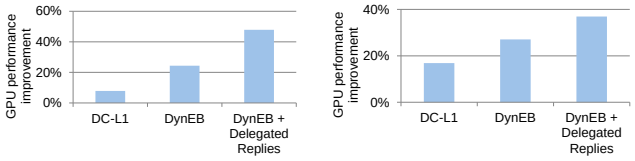


Figure 14: L1 cache miss breakdown. *Delegated Replies on average forwards 54.8% of the L1 misses to remote GPU cores out of which 74.4% are remote hits.*

12.8% compared to RP), see the max line in the whiskers.

L1 Miss Breakdown. Figure 14 breaks down the L1 cache misses into three components: (i) LLC hit, (ii) remote hit, and (iii) remote miss. On average, 54.8% of the memory requests are sent to remote cores out of which 40.8% are satisfied by remote L1s. The applications with the largest remote hit rate are 2DCON, HS and NN — more than 60% of the local L1 misses hit in the remote L1. This translates into significant performance improvements for HS (67.9%), 2DCON (40.9%) and NN (19.5%); the reason for the somewhat lower performance improvement for NN is its relatively low L1 miss rate (4.3%). Benchmarks with a high LLC hit rate and few delegated replies, such as SC, LUD and BP, are the ones for which Delegated Replies yields modest performance improvements (less than 13.7%). For some benchmarks, namely 3DCON, BT and LPS, we note a fair number of remote misses because of frequent replacement of the recently accessed cache lines in the remote L1. Nevertheless, Delegated Replies still improves performance by 46.3%, 28.1% and 17.5%, respectively, thanks to the substantial fraction of remote hits.

Inter-core locality optimizations. Inter-core locality among GPU cores can be improved by L1 cache sharing [29], [30] and distributed CTA scheduling [8], and we now explore the performance of Delegated Replies in such architectures (see Figure 15). We consider the shared L1 schemes DC-L1 [30] and DynEB [29] under both round-robin and distributed



(a) Round-robin scheduling (b) Distributed scheduling

Figure 15: GPU performance improvement due to Delegated Replies on top of inter-core locality optimizations including DC-L1 and DynEB cache sharing and distributed CTA scheduling. *Optimizing inter-core locality does not remove NoC clogging, and Delegated Replies still provides substantial benefits.*

CTA scheduling: DC-L1 statically shares one L1 cache with 4 slices between 8 GPU cores; DynEB dynamically selects a shared or private L1 organization based on the effective bandwidth each configuration can achieve for the current application. Figure 15 shows that DC-L1 and DynEB improve performance for both CTA scheduling policies. Detailed analysis reveals that DynEB consistently improves performance relative to the baseline whereas DC-L1 either improves performance significantly (e.g., SC and LUD) or incurs significant slowdowns (e.g., NN and 2DCON). More specifically, SC and LUD benefit from the higher effective L1 capacity provided by both DC-L1 and DynEB because cache sharing reduces shared data replication. In contrast, DC-L1 slows down NN and 2DCON substantially (by 87.6% and 32.3%, respectively, under round-robin scheduling) because the lack of replication leads to lower effective bandwidth as requests to the same shared data element around the same time creates congestion in front of L1 cache slices (a well-known issue in shared GPU caches [60], [61]). DynEB reverts to the shared L1 cache organization in those cases and hence performs similarly to our baseline.

We now explore the performance impact of Delegated Replies on top of DynEB under both round-robin and distributed CTA scheduling. (We focus on DynEB only since it outperforms DC-L1.) Delegated Replies improves upon DynEB by 23.5% on average (up to 67.9%) for round-robin scheduling versus 9.9% (up to 59.6%) under distributed scheduling. Delegated Replies is synergistic with locality-enhancing optimizations for two reasons. First, DynEB reverts to the baseline private cache organization when it detects that a shared L1 provides insufficient bandwidth. By consequence, DynEB does not eliminate NoC clogging, creating an opportunity for Delegated Replies to further improve performance. Second, distributed CTA scheduling typically improves L1 cache locality but it does not reduce the L1 misses sufficiently to completely remove NoC clogging, again leaving some opportunity for Delegated Replies. The bottom line is that inter-core locality optimizations do not completely eliminate NoC clogging, enabling Delegated Replies to still provide substantial performance benefits.

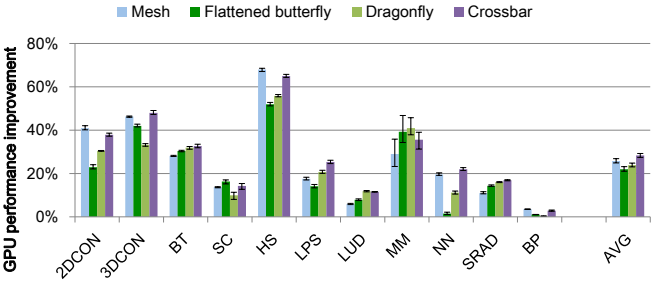


Figure 16: GPU performance improvement across NoC topologies (normalized to the respective topology). *Delegated Replies consistently improves GPU performance.*

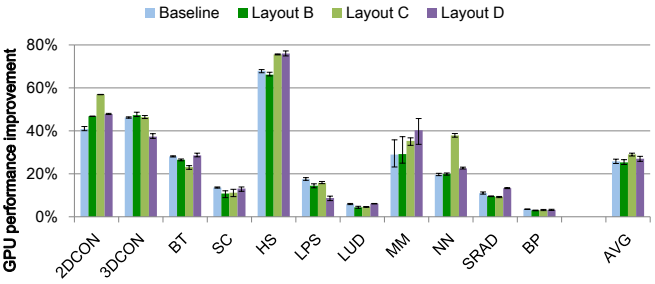


Figure 17: GPU performance improvement across chip layouts (normalized to the respective layout). *Delegated Replies consistently improves GPU performance.*

Energy Consumption. Interestingly, dynamic NoC energy decreases for most benchmarks with Delegated Replies (by 1.1% on average) while it increases with RP (by 9.4% on average). For Delegated Replies, the reason is that the number of hops with data can be lower for remote hits, i.e., even if the single-flit requests first go to the LLC and are then delegated, the multi-flit replies traverse fewer links because there are on average fewer hops between the remote GPU core and the requesting GPU core than there are hops between the memory node and the requesting GPU core on average. In contrast, RP spends significant dynamic energy on unsuccessfully probing other L1 caches. More specifically, RP increases the total number of NoC requests by 5.9× compared to the baseline. Delegated Replies (RP) reduces total system energy by 13.6% (7.4%) on average, primarily due to shorter execution time.

NoC topology. To illustrate that Delegated Replies is insensitive to NoC topology, we evaluate its performance with the flattened butterfly [41], the Dragonfly [42], and a crossbar with core-to-core links while keeping channel width the same as in our baseline (see Figure 16). Delegated Replies increases average GPU performance by 21.9%, 23.9%, and 28.3%, with the flattened butterfly, dragonfly, and crossbar, respectively, compared to 25.8% with the mesh. While these topologies provide higher bandwidth than the mesh, they do not alleviate clogging — illustrating that the benefits of Delegated Replies are independent of NoC topology.

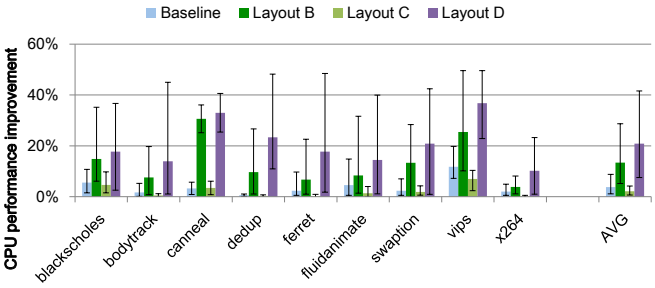


Figure 18: CPU performance improvement across chip layouts (normalized to the respective layout). *Delegated Replies significantly improves CPU performance for layouts B and D due to more severe CPU-GPU interference.*

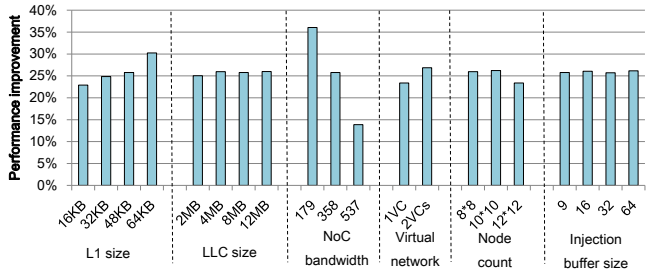


Figure 19: Sensitivity analyses. *Delegated Replies consistently improves GPU performance across the design space.*

Chip Layout Analysis. We find that Delegated Replies consistently improves GPU performance across the four layouts discussed in Section V (and their optimal routing policies), see Figures 17 and 18. The average improvement is 25.8% (Baseline), 25.3% (B), 29.0% (C) and 27.0% (D). This suggests that Delegated Replies consistently improves the effective data rate received by the GPU cores, irrespective of the specific layout. We observe a wider variety in CPU performance improvement across the different layouts. Here, the average improvement is 3.8% (Baseline), 13.4% (B), 2.2% (C) and 20.9% (D). Network interference between CPU and GPU traffic leads to higher CPU performance improvements under Delegated Replies for Layouts B and D compared to Baseline and Layout C because providing higher priority to CPU traffic is (relatively) more important when there is more interference.

Sensitivity analyses. We focus on GPU performance for our sensitivity analyses, see Figure 19.

L1 size. The size of the L1 cache leads to conflicting phenomena. A large L1 decreases the L1 miss rate, reducing the opportunity for Delegated Replies. On the other hand, a large L1 also increases the possibility for a remote L1 hit, increasing the opportunity. Overall, we find that Delegated Replies leads to a net performance increase with increasing L1 cache size from 22.9% (16 KB) to 30.2% (64 KB).

LLC size. Similar to L1, increasing the LLC size leads to both a positive and negative contribution. A larger LLC reduces the opportunity for Delegated Replies because of the increased hit rate. On the other hand, a larger LLC enables storing more remote core information, increasing the opportunity. Overall, we find that Delegated Replies is rather insensitive to LLC size with performance improvements between 25.0% and 26.0%.

NoC bandwidth. We evaluate NoC bandwidth sensitivity by increasing network channel width from 8 byte to 24 byte, thereby increasing NoC bandwidth from 179 GB/s to 537 GB/s. Delegated Replies' performance improvements are more significant for constrained NoC configurations. However, even for 537 GB/s NoC bandwidth, Delegated Replies still improves performance by 13.9% on average.

Virtual networks. Instead of having a separate request and reply network as in our baseline, we now consider two virtual networks sharing the same physical network. We consider one and two VCs per virtual network, and we give high priority to remote L1 requests. On average, Delegated Replies improves performance by 23.4% and 26.9% on average for one VC and two VCs, respectively. Delegated Replies is hence equally beneficial in systems with shared and separate request and reply networks.

Node count. We now increase network size from an 8×8 mesh to a 10×10 and 12×12 mesh while maintaining the same proportion of CPU, GPU, and memory nodes. Delegated Replies features good scalability and achieves similarly high performance improvements across different network sizes.

Injection buffer size. As previously argued, a memory node blocks when its injection buffer fills up. Increasing the injection buffer size does not solve the problem, and we find that the performance improvement through Delegated Replies is largely insensitive to injection buffer size.

Node mix. We now explore the impact of varying the number of CPU cores, GPU cores, and memory nodes on a 64-node architecture (not shown in Figure 19 due to space restrictions). We first fix the number of memory nodes at 8 and vary the number of CPU and GPU cores. The average GPU performance improvement of Delegated Replies equals 30.5%, 25.8% and 22.6% with 8, 16 and 24 CPU cores and 48, 40 and 32 GPU cores, respectively. Second, we keep the CPU cores constant at 8 and vary the number of memory nodes and GPU cores. In this case, the average GPU performance improvement of Delegated Replies equals 38.2%, 30.5% and 10.7% with 4, 8 and 16 memory nodes and 52, 48 and 40 GPU cores, respectively. NoC clogging is more severe with fewer memory nodes and more GPU cores, in which case Delegated Replies provides the highest performance benefits. Overall, Delegated Replies offers substantial improvements across the broad design space, even for configurations with a relatively high ratio of memory nodes versus GPU cores.

VIII. Conclusion

Delegated Replies alleviates NoC clogging in heterogeneous architectures. More specifically, it transforms the clogging-inducing many-to-few request pattern into a more favorable many-to-many pattern by letting memory nodes delegate the responsibility of replying to LLC hit requests to the accelerator core that last accessed the requested cache line and hence typically has the data available in its local cache. On heterogeneous CPU-GPU workloads, Delegated Replies effectively alleviates clogging and thereby improves GPU performance by 25.8% on average, and up 67.9%, compared to our carefully designed baseline. For the workloads with significant network clogging, Delegated Replies improves CPU performance by 8.8% on average, and up to 19.8%.

Acknowledgment

We thank the anonymous reviewers for their valuable comments. This work was supported in part by the European Research Council (ERC) Advanced Grant agreement no. 741097, FWO project G.0144.17N, and NSFC under Grant no. 62102438. Magnus Jahre is supported by the Research Council of Norway (Grant no. 286596).

References

[1] T. M. Aamodt, W. W. L. Fung, and T. G. Rogers, *General-Purpose Graphics Processor Architectures*. Morgan & Claypool Publishers, 2018.

[2] D. Abts, S. Scott, and D. J. Lilja, "So many states, so little time: Verifying memory coherence in the Cray X1," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, 2003.

[3] D. Abts, N. D. Enright Jerger, J. Kim, D. Gibson, and M. H. Lipasti, "Achieving predictable performance through better memory controller placement in many-core CMPs," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2009.

[4] M. E. Acacio, J. Gonzalez, J. M. Garcia, and J. Duato, "Owner Prediction for Accelerating Cache-to-Cache Transfer Misses in a cc-NUMA Architecture," in *Proceedings of the International Conference on Supercomputing (SC)*, November 2002, pp. 49–49.

[5] AMD. (2013) AMD64 Architecture Programmer's Manual Volume 2: System Programming. https://web.archive.org/web/20170619232736/http://developer.amd.com/wordpress/media/2012/10/24593_APM_v21.pdf.

[6] AMD. (2019) AMD Ryzen 3 3200G with Radeon Vega 8 Graphics. https://www.amd.com/en/products/apu/amd-ryzen-3-3200g.

[7] Apple. (2020) Apple M1. https://www.apple.com/mac/m1/.

[8] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, "MCM-GPU: Multi-Chip-Module GPUs for Continued Performance Scalability," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2017, pp. 320–332.

[9] A. Bakhoda, J. Kim, and T. M. Aamodt, "Throughput-Effective On-Chip Networks for Manycore Accelerators," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, December 2010, pp. 421–432.

[10] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *Proceeding of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2009, pp. 163–174.

[11] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *Proceedings of International Conference on Parallel Architectures and Compilation Techniques (PACT)*, October 2008, pp. 72–81.

[12] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip," *Journal of Systems Architecture*, vol. 50, no. 2-3, pp. 105–128, 2004.

[13] J. Cantin, J. Smith, M. Lipasti, A. Moshovos, and B. Falsafi, "Coarse-grain coherence tracking: RegionScout and Region Coherence Arrays," *IEEE Micro*, vol. 26, no. 1, pp. 70–79, 2006.

[14] CCIX Consortium. (2019) An Introduction to CCIX. https://www.ccixconsortium.com/wp-content/uploads/2019/11/CCIX-White-Paper-Rev111219.pdf.

[15] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *Proceedings of the International Symposium on Workload Characterization (IISWC)*, October 2009, pp. 44–54.

[16] CXL Consortium. (2019) Compute Express Link: The Breakthrough CPU-to-Device Interconnect. https://www.computeexpresslink.org/.

[17] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., 2003.

[18] W. J. Dally and H. Aoki, "Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 4, pp. 466–475, April 1993.

[19] W. J. Dally, Y. Turakhia, and S. Han, "Domain-specific hardware accelerators," *Communications of the ACM*, vol. 63, no. 7, pp. 48–57, 2020.

[20] B. K. Daya, C. H. O. Chen, S. Subramanian, W. C. Kwon, S. Park, T. Krishna, J. Holt, A. P. Chandrakasan, and L. S. Peh, "SCORPIO: A 36-core research chip demonstrating snoopy coherence on a scalable mesh NoC with in-network ordering," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2014, pp. 25–36.

[21] J. Doweck, W. Kao, A. K. Lu, J. Mandelblat, A. Rahatekar, L. Rappoport, E. Rotem, A. Yasin, and A. Yoaz, "Inside 6th-Generation Intel Core: New Microarchitecture Code-Named Skylake," *IEEE Micro*, vol. 37, no. 2, pp. 52–62, March 2017.

[22] B. Fu and J. Kim, "Footprint: Regulating Routing Adaptiveness in Networks-on-Chip," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2017, pp. 691–702.

[23] B. Gaide, D. Gaitonde, C. Ravishankar, and T. Bauer, "Xilinx adaptive compute acceleration platform: Versal architecture," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2019.

[24] I. Gelado, J. E. Stone, J. Cabezas, S. Patel, N. Navarro, and W.-m. W. Hwu, "An Asymmetric Distributed Shared Memory Model for Heterogeneous Parallel Systems," in *Proceedings of the International Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Month 2010, pp. 347–358.

[25] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-tuning a High-Level Language Targeted to GPU Codes," in *Innovative Parallel Computing (InPar)*, May 2012, pp. 1–10.

[26] J. Hestness, B. Grot, and S. W. Keckler, "Netrace: Dependency-driven Trace-based Network-on-chip Simulation," in *Proceedings of the Third International Workshop on Network on Chip Architectures (NoCArc)*, December 2010, pp. 31–36.

[27] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, 2007.

[28] H. H. Hum and J. R. Goodman, "Forward State for Use in Cache Coherency in a Multiprocessor System," July 2005, US Patent 6922756.

[29] M. A. Ibrahim, O. Kayiran, Y. Eckert, G. H. Loh, and A. Jog, "Analyzing and leveraging shared L1 caches in GPUs," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2020, pp. 161–173.

[30] M. A. Ibrahim, O. Kayiran, Y. Eckert, G. H. Loh, and A. Jog, "Analyzing and leveraging decoupled L1 caches in GPUs," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2021.

[31] M. A. Ibrahim, H. Liu, O. Kayiran, and A. Jog, "Analyzing and Leveraging Remote-Core Bandwidth for Enhanced Performance in GPUs," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, September 2019, pp. 258–271.

[32] H. Jang, J. Kim, P. Gratz, K. Yum, and E. Kim, "A Bandwidth Efficient On-Chip Interconnects Design for GPGPUs," in *Proceedings of the Design Automation Conference (DAC)*, June 2015, pp. 1–6.

[33] H. Jang, J. Kim, P. Gratz, K. H. Yum, and E. J. Kim, "Bandwidth-efficient On-chip Interconnect Designs for GPGPUs," in *Proceedings of the Design Automation Conference (DAC)*, June 2015, pp. 9:1–9:6.

[34] M. K. Jeong, M. Erez, C. Sudanthi, and N. Paver, "A QoS-aware memory controller for dynamically balancing GPU and CPU bandwidth use in an MPSoC," in *Proceedings of the Design Automation Conference (DAC)*, Jun. 2012, pp. 850–855.

[35] N. E. Jerger, T. Krishna, and L. Peh, *On-Chip Networks: Second Edition*. Morgan & Claypool Publishers, 2017.

[36] N. Jiang, J. Balfour, D. U. Becker, B. Towles, W. J. Dally, G. Michelogiannakis, and J. Kim, "A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2013, pp. 86–96.

[37] K. Jin, C. Li, D. Dong, and B. Fu, "HARE: History-Aware Adaptive Routing Algorithm for Endpoint Congestion in Networks-on-Chip," *International Journal of Parallel Programming*, vol. 47, no. 3, pp. 433–450, 2019.

[38] O. Kayiran, N. C. Nachiappan, A. Jog, R. Ausavarungnirun, M. T. Kandemir, G. H. Loh, O. Mutlu, and C. R. Das, "Managing GPU Concurrency in Heterogeneous Architectures," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, December 2014, pp. 114–126.

[39] M. Khairy, V. Nikiforov, D. Nellans, and T. G. Rogers, "Locality-centric data and threadblock management for massive GPUs," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2020.

[40] H. Kim, J. Kim, W. Seo, Y. Cho, and S. Ryu, "Providing Cost-effective On-Chip Network Bandwidth in GPGPUs," in *Proceedings of the International Conference on Computer Design (ICCD)*, September 2012, pp. 407–412.

[41] J. Kim, J. Balfour, and W. Dally, "Flattened Butterfly Topology for On-Chip Networks," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, December 2007, pp. 172–182.

[42] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-Driven, Highly-Scalable Dragonfly Topology," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2008, pp. 77–88.

[43] Y. Liu, X. Zhao, M. Jahre, Z. Wang, X. Wang, Y. Luo, and L. Eeckhout, "Get Out of the Valley: Power-Efficient Address Mapping for GPUs," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2018, pp. 166–179.

[44] U. Milic, O. Villa, E. Bolotin, A. Arunkumar, E. Ebrahimi, A. Jaleel, A. Ramirez, and D. Nellans, "Beyond the Socket: NUMA-aware GPUs," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, October 2017, pp. 123–135.

[45] Ming Li, Qing-An Zeng, and Wen-Ben Jone, "DyXY - A Proximity Congestion-Aware Deadlock-Free Dynamic Routing Method for Network on Chip," in *Proceedings of the 43rd ACM/IEEE Design Automation Conference (DAC)*, 2006, pp. 849–852.

[46] A. Mirhosseini, M. Sadrosadati, B. Soltani, H. Sarbazi-Azad, and T. F. Wenisch, "BiNoCHS: Bimodal Network-on-Chip for CPU-GPU Heterogeneous Systems," in *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, October 2017, pp. 7:1–7:8.

[47] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," *HP laboratories*, vol. 27, p. 28, 2009.

[48] NVIDIA. (2019) Tegra processors. https://www.nvidia.com/object/tegra-x1-processor.html.

[49] NVIDIA CUDA SDK Code Samples. https://developer.nvidia.com/cuda-downloads. NVIDIA Corporation.

[50] J. Power, A. Basu, J. Gu, S. Puthoor, B. M. Beckmann, M. D. Hill, S. K. Reinhardt, and D. A. Wood, "Heterogeneous System Coherence for Integrated CPU-GPU systems," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, December 2013, pp. 457–467.

[51] P. Rogers, "Heterogeneous system architecture overview," in *Proceedings of the Symposium on High Performance Chips (HOTCHIPS)*, August 2013, pp. 1–41.

[52] T. G. Rogers, M. O'Connor, and T. M. Aamodt, "Cache-conscious Wavefront Scheduling," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, December 2012, pp. 72–83.

[53] A. Ros and S. Kaxiras, "Complexity-effective multicore coherence," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2012, pp. 241–252.

[54] C. Sun, C. H. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L. S. Peh, and V. Stojanovic, "DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling," in *Proc. of the Int. Symp. on Networks-on-Chip (NOCS)*, May 2012, pp. 201–210.

[55] T. Vijayaraghavan, Y. Eckert, G. H. Loh, M. J. Schulte, M. Ignatowski, B. M. Beckmann, W. C. Brantley, J. L. Greathouse, W. Huang, A. Karunanithi, O. Kayiran, M. Meswani, I. Paul, M. Poremba, S. Raasch, S. K. Reinhardt, G. Sadowski, and V. Sridharan, "Design and Analysis of an APU for Exascale Computing," in *Proc. of the Int. Symp. on High Performance Computer Architecture (HPCA)*, 2017, pp. 85–96.

[56] C. M. Wittenbrink, E. Kilgariff, and A. Prabhu, "Fermi GF100 GPU Architecture," *IEEE Micro*, vol. 31, no. 2, pp. 50–59, March 2011.

[57] J. Yin, Z. Lin, O. Kayiran, M. Poremba, M. Shoaib Bin Altaf, N. Enright Jerger, and G. H. Loh, "Modular Routing Design for Chiplet-Based Systems," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2018, pp. 726–738.

[58] L. Yunfan and C. Lizhong, "EquiNox: Equivalent NoC Injection Routers for Silicon Interposer-based Throughput Processors," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, February 2020, pp. 203–214.

[59] J. Zhan, O. Kayıran, G. H. Loh, C. R. Das, and Y. Xie, "OSCAR: Orchestrating STT-RAM cache traffic for heterogeneous CPU-GPU architectures," in *Proc. of the Int. Symp. on Microarchitecture (MICRO)*, October 2016, pp. 1–13.

[60] X. Zhao, A. Adileh, Z. Yu, Z. Wang, A. Jaleel, and L. Eeckhout, "Adaptive Memory-Side Last-Level GPU Caching," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2019, pp. 307–319.

[61] X. Zhao, M. Jahre, and L. Eeckhout, "Selective replication in memory-side GPU caches," in *Proc. of the Int. Symp. on Microarchitecture (MICRO)*, 2020, pp. 967–980.