A Distributed Interleaving Scheme for Efficient Access to WideIO DRAM Memory

Ciprian Seiculescu
LSI, EPFL
Lausanne, Switzerland
ciprian.seiculescu@epfl.ch

Luca Benini
DEIS, University of Bologna
Bologna, Italy
luca.benini@unibo.it

Giovanni De Micheli
LSI, EPFL
Lausanne, Switzerland
giovanni.demicheli@epfl.ch

ABSTRACT

Achieving the main memory (DRAM) required bandwidth at acceptable power levels for current and future applications is a major challenge for *System-on-Chip* designers for mobile platforms. *Three dimensional* (3D) integration and 3D stacked DRAM memories promise to provide a significant boost in bandwidth at low power levels by exploiting multiple channels and wide data interfaces. In this paper, we address the problem of efficiently exploiting the multiple channels provided by standard (JEDEC's WIDE-IO) 3D-stacked memories, to extract maximal effective bandwidth and minimize latency for main memory access. We propose a new distributed interleaved access method that leverages the on-chip interconnect to simplify the design and implementation of the DRAM controller, without impacting performance compared to traditional centralized implementations. We perform experiments on realistic workload for a mobile communication and multimedia platform and show that our proposed distributed interleaving memory access method improves the overall throughput while minimally impacting the performance of latency sensitive communication flows.

Categories and Subject Descriptors

B.4.3 [**INPUT/OUTPUT AND DATA COMMUNICATIONS**]: Interconnections (Subsystems)—*topology*

General Terms

Design

Keywords

NoC, WideIO, DRAM controller, interleaving

1. INTRODUCTION

Todays applications that drive the *System-on-Chip* (SoC) design require more and more storage capacity as well as high bandwidth to main memory. To provide the required storage capacity, high-density commodity DRAM is the only cost-effective main memory option, and external DRAM modules are needed. To deliver the required bandwidth when accessing external DRAM is a major challenge and main memory access is the main bottleneck for scaling computing performance, also known in literature as the *Memory Wall* [1]. In mobile SoCs platforms that have strict power consumption constraints, providing the required bandwidth to main memory at manageable power levels is an even more challenging problem. Mobile SoCs rely on parallelization and specialization of the functionality in order to obtain the required performance for the application at low power level. A natural solution for providing increased memory bandwidth with low power would be to parallelize the access to the memory. However, as the main storage is external, the limited number of available off-chip IO ports prevents the design of a parallel memory system.

Three dimensional integration is a promising technology for increasing the number of transistors on-chip [2]. A major advantage of 3D integration is the ability to integrate efficiently heterogeneous manufacturing technologies in a single chip stack, by putting together dies that have been processed separately. 3D-stacked memories that leverage the benefits of heterogeneous integration have been proposed as a solution to overcome the *Memory Wall* [3]. As each layer in a 3D stack is processed separately, high density DRAM memory can be stacked on top of logic to satisfy the large storage needs of applications (at low cost per MB). *Through Silicon Vias* (TSVs) are used to provide vertical connectivity and as 3D integration technology matures the density of TSVs increases. Stacked memories can benefit from the large number of TSVs to provide wide interfaces and multiple channels, such that the large bandwidth requirements can be met at low power levels. Apart from the wider data-paths that allow the memory to operate at lower frequency, while achieving the same bandwidth, power is saved in 3D stacked memories by removing the need to go off-chip through power-hungry IO ports. On the other hand, having access to multiple ports, requires the SoC to be able to efficiently use them to achieve the overall bandwidth requirement.

Given the promise of three-dimensional DRAM stacking, industry is actively pushing standardization of TSV-based interfaces. WideIO 3D-stacked DRAM [4] is an emerging JEDEC standard. WideIO memories use TSVs to provide interfaces to 4 channels each with a 128bit wide data interface. By running at 200MHz with single data rate the WideIO memory can provide a peak bandwidth of 12.8GB/s at lower power levels than current *Low-Power Double Data Rate* (LPDDR) DRAM. According to the JEDEC standard WideIO will provide 50% more bandwidth with 20% less power than an existing dual-channel LPDDR2 off-chip solution. In this work, we focus on WideIO DRAM memory integration in the SoC design and we address the problem of efficiently using the 4 channels to achieve the required application performance.

CODES+ISSS'12, October 7-12, 2012, Tampere, Finland.

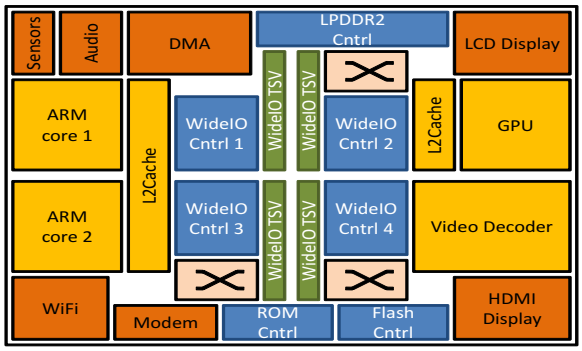

Figure 1: Example of SoC floorplan with WideIO TSV interfaces

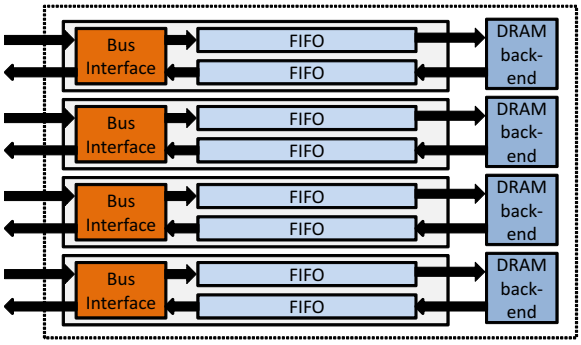

Figure 2: Distributed DRAM controller with 4 separate channels

Networks-on-Chip (NoCs) have been proposed as a scalable interconnect for on-chip communication [5]. NoCs have been already adopted in many high-end mobile SoC products and can be found on most of next-generation SoCs [6]. Therefore in this work, we focus on SoC design that use a Noc for the global interconnect. We analyze two architectures for the memory sub-system that access the 4 channels of the WideIO DRAM: i) a simple to implement architecture that uses 4 DRAM controllers to independently access the four channels that rely on software to balance the traffic between the channels and ii) a complex centralized controller with multiple ports and interleaved address space to balance the traffic among the four channels. Since the TSV size is large, when compared to transistor size, the wide data interfaces that require many TSVs will be far apart on the floorplan. An example of such a heterogeneous SoC floorplan with WideIO connectivity is presented in Figure 1. The position of the TSV interfaces in the example was chosen to accommodate the WideIO interface from [4]. This makes the centralized controller difficult and expensive to implement. Therefore, we also propose a new distributed interleaving method to access the memory controller that leverages the advantages of both of the previously mentioned designs, while avoiding their most serious shortcomings. On one hand it uses a simple memory interface design, as the first method, but provides an interleaved address space such that balancing the traffic to the four channels is transparent to the software. Our proposed approach requires the integrated design of both the NoC and the DRAM controller, as the interleaving support is now distributed within the NoC, which also result in balancing the traffic in the NoC itself.

We perform experiments on a realistic benchmark for a heterogeneous mobile communication and multimedia SoC platform. An important contribution is given by our advanced traffic modeling environment and by modeling the details of the NoC micro-architecture (i.e., packetization, bandwidth inflation). From experiments, we show that our proposed distributed interleaving memory access method improves the overall throughput while minimally impacting the performance of latency sensitive communication flows. For example, our design improves the frame rate of the video decoding and display subsystem by 2 frames/second when compared to simple separated controller with the best memory allocation (42 frames/second with the worst memory allocation) and by 7 frames/second when compared to the complex controller with interleaved address space.

2. RELATED WORK

Several works have presented methods to improve DRAM access efficiency by scheduling and reordering transactions in the DRAM controller [7], [8], [9], [10], [11], [12]. In [8], [10] optimizations for multicore systems are presented, and in [9] a predictable DRAM controller is presented. Memory transaction scheduling is important in increasing the efficiency of DRAM access, but it is orthogonal to the scope of this paper. We use the simulator model from [12] to include these memory access optimizations in our simulation framework.

Many research groups have focused on the technological aspects to manufacture 3D-integrated memory systems [13], [14], [15], [16]. Others have investigated system architectures using 3D-integrated on-chip DRAM [17], [18], [19], [20]. In [3], Weis et al. present a design space exploration of SoCs with 3D-stacked DRAM and in [21] the authors show an overview of 3D-integrated DRAMs and the challenges associated with it. In this work we assume a specific 3D-stacked DRAM architecture, i.e. the WideIO JEDEC standard[4]. A study showing the advantages of using multi-channel memory systems for video encoding SoCs is presented in [22]. The study shows the impact of multiple channels on the system performance and our work is different from that as we are concerned with how to efficiently access multiple channels.

Several researches have looked at the co-design of the NoC with the DRAM controller. In [23] the authors present a way to reorder DRAM transactions while in the network and simplify the DRAM controller. However this work addresses the problem of scheduling and not of accessing multiple channels. To reduce the traffic to DRAM the authors propose in [24] to have a processor base DRAM controller capable to process complex requests and return only the results. A credit based flow control method is used in [25] to prevent the DRAM traffic from waiting into the network and interfering with non DRAM traffic. In [26] the authors propose to give higher priority to transactions waiting in the DRAM queue that have to be sent back to cores in areas of the NoC which are less congested. These works however do not address the problem of accessing multiple channels. Memory centric NoC architectures with real chip implementations are provided in [27] and [28]. However in these systems there are several memories which are on-chip and the communication between cores is done through these memories.

Arteris [29] and Sonics [30] offer memory interleaving support that is integrated with their interconnect solutions. However they do not provide any comparison to traditional solutions. Moreover in the white paper from Sonics [30], experiments are shown only for two channels using existing off-chip DDR3 memories. In this work we target 3D-stacked WideIO with 4 channels and provide an extensive comparison with traditional architectures like centralized controllers and software managed independent channels.

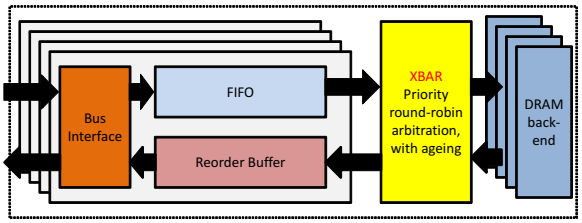

Figure 3: Centralized DRAM controller with interleaving at the controller side

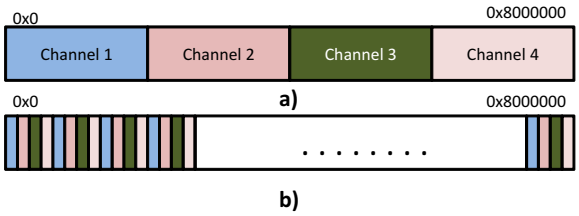

Figure 4: Example of a) partitioned- and b) interleaved address space

3. NOC AND CONTROLLER ARCHITECTURES FOR MULTI-CHANNEL DRAM

WideIO DRAM provides high-peak bandwidth at low power levels by using multiple channels with independent wide data-interfaces operated at low frequency. Efficiently exploiting the 4 channels of the WideIO memories, in order to satisfy the bandwidth demands of the application, is up to the SoC designer. Having multiple channels provides new opportunities for designing the memory controller to efficiently access the memory subsystem. In this section, we introduce three architectures: two traditional ones and we also propose a new architecture that moves part of the DRAM controller complexity into the NoC fabric to provide distributed parallel access to the four channels of a WideIO memory.

In this section we describe the three analyze architecture and discuss their advantages and disadvantages:

- *Distributed controller with separate independent channels.* This is a simple solution where an independent DRAM controller is assigned to each channel.
- *Centralized controller with interleaved address space.* This is a single multi-ported controller that can access all the four channels of the WideIO memory interface. The requests from the four ports are interleaved based on the address to the four channels to balance the traffic.
- *Distributed controller with interleaving in the NI.* This is our proposed method that uses 4 simple DRAM controllers for the 4 channels, but the transactions are split and interleaved based on address within the interconnect at the *Network Interfaces* (NIs) of the NoC.

3.1 Distributed controller

The simplest solution to access the four channels is to have four independent DRAM controllers and to split the address space in four large contiguous blocks. A block diagram of such an architecture is presented in Figure 2. By ensuring at software level that a core does not access more than one channel at the same time, this architecture does not require any changes to existing DRAM controller or NoC architectures. To each channel a DRAM controller is assigned that only needs a bus interface and address generator, with FIFO buffers to drain the requests from the NoC and an out-of-order back-end. Designing the memory subsystem of the SoC in such a distributed fashion, with separate independent DRAM channels, is easy. The main drawback of this simple architecture is that the performance of the memory subsystem is highly dependent on the memory mapping. In the worst case, if the memory regions of all cores are mapped to the same channel (which could happen for certain periods of time in systems where the memory is dynamically managed), then the peak-bandwidth of the WideIO DRAM memory is effectively reduced four times. To tackle this problem, such a system would have to expose hardware details to the software and make sure that at software level, the memory is allocated as to balance the accesses to the 4 channels. However as such an allocation, reliant on the software, would be coarse grained it may not be possible to efficiently balance the accesses to the channels.

To tackle this drawback of separated independent channels a popular technique is to interleave the address space between the channels [31]. By splitting the address space in fine-grained blocks and assigning them interleaved to the channels as shown in Figure 4, we can make sure in hardware that the data is distributed uniformly among the channels. Accessing the memory is now balanced between the channels and it is transparent to the software. The decision to which channel a transaction (or a piece of a transaction) belongs depends on the address of that transaction. Depending on where that decision is taken we look at two solutions: i) centralized in the DRAM controller and ii) distributed at the initiator *Network Interfaces* (NIs) of the NoC.

3.2 Centralized Controller with Interleaving

One way to design an interleaved memory subsystem is to have a centralized DRAM controller with multiple ports toward the NoC and multiple channels to connect to the WideIO DRAM as shown in Figure 3 similar with that from [32] (with different arbitration scheme though as we do not require that level of predictability). The advantage of this solution is that the memory controller and the NoC can be designed separately and regardless of which port you use you have access to the entire memory space. Such a centralized solution would require a crossbar to allow for parallel access between ports and the back-end controllers connected to the channels. In our architecture the bus interfaces, corresponding to each port, take the bus request and generate transactions for the DRAM back-end with the same size as the interleaving blocks. These transactions are stored in the input FIFO of the port. The arbiter takes these transactions from the ports and using the crossbar sends them to the appropriate channel according to the address. A popular schemes for arbitration (which we also assume) is priority with aging to prevent starvation. Ports can have different priorities (in our case we use two) and the transactions in the low priority ports keep an age counter. After a certain age given as parameter their priority becomes the highest and get serviced next. The reordering of the transaction to be memory friendly is done in the back-end and that is beyond the scope of this work. However since transactions from the same port are sent to different back-ends they can be serviced and returned out of order, so the response buffers in the ports have to be able to reorder the transactions.

While this solution fixes the problem of application memory mapping, there are two important drawbacks of this solution. The first drawback relates to the physical implementation of the centralized controller. The WideIO uses TSVs for connectivity, which are considerably larger than other features, and has wide interfaces so re-

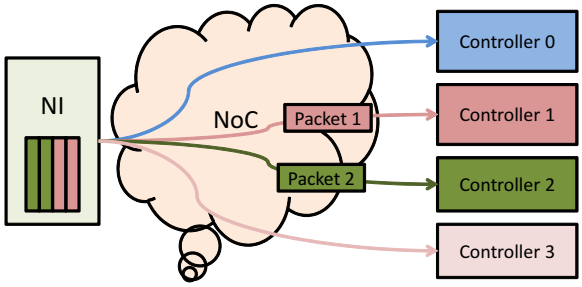

Figure 5: Example of distributed interleaving where the NI sends out a 4 beat burst transaction in 2 packets on different routes

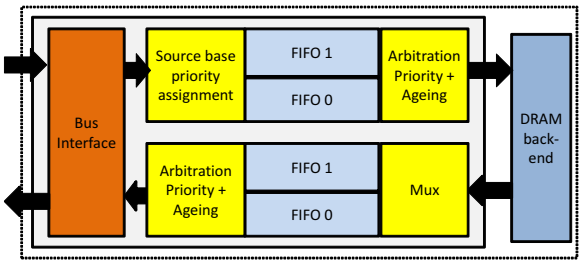

Figure 6: DRAM controller port with priority

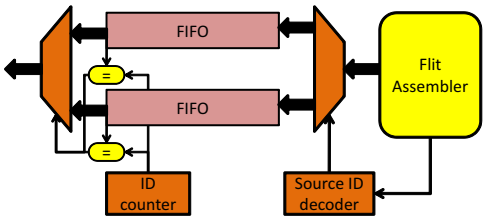

Figure 7: Simple reorder buffer schematic

quiring many such TSVs. According to the JEDEC standard the size of the 4 channels interfaces is around 0.54mm by 5.27mm. So the interfaces are far apart on the floorplan of the resulting chip as shown in Figure 1. Therefore designing the crossbar to connect the ports to the DRAM back-ends may be costly and slow. The second drawback is that the as there are more IP-cores that used the DRAM than ports, it is up to the designer to decide the assignment of the IP-cores to ports. As different IP-cores may be active for different application there could be cases where even a good assignment could still result in over-congesting a port and parts of the NoC. This would result in a degradation of performance.

3.3 Distributed Controller with Interleaving

A compromise between the two solutions discussed before is to distribute the interleaving function from the DRAM controller in the NoC. This solution would require the simpler controllers of the first solution, which could be easily implemented next to the interfaces to the WideIO memory. In this case the source NIs are responsible to implement the interleaving function (performed by the crossbar in the centralized controller). A schematic representation of this architecture which we will call as distributed controller with interleaving in the NIs is presented in Figure 5. The NI needs to have a route to each port of the independent memory controllers. Based on the address the NI chooses the path to the corresponding port. In case of transactions that are larger than the size of the interleaving block, the NI is responsible to split these transactions into multiple packets and to send those packets to the corresponding controllers. In this case the responses or partial responses (as the transaction can receive the responses in multiple packets) have to be reordered and reassembled in the NI. The details describing the NI are presented in Section 4.1. To provide two level of priority as in the centralized controller case, we use two FIFO buffers per port as shown in Figure 6. The assignment of the transactions to the priority queue is done based on the ID of the source.

The disadvantage of this method is that it generates more bandwidth in the NoC, due to the packetization overhead in the case when transactions are split. An important thing to note is that even in the previous case long transactions may need to be split to prevent blocking the interfering traffic for too long from accessing the memory controller as well. In this architecture the NoC and the memory controller need to be designed together. The main advantage of this architecture is that it balances the traffic in both the NoC and at the DRAM controller channels, regardless of the application memory mapping.

3.4 Simple Reorder-buffer Implementation

If the NI has support for interleaving and splitting of packets then it needs to be able to reorder the transactions. However in cases where there are only few routes handled by the NI the reorder buffer can be implemented in a simpler way. By leveraging the property that packets on the same route will arrive in order, then the reorder buffer can be implemented only with FIFOs and few comparators, instead of expensive CAM memories. In Figure 7, we show an example of such a reorder buffer for an NI that uses two routes to interleave the packets.

To track the order, the transactions are assigned an ID (consecutive values of a counter) when they are sent out to the target. The responses from the target will contain the same ID that was assigned to the request. We need as many FIFO buffers as there are routes. As the transaction responses coming from the same path arrive in-order, we can use simple FIFOs to buffer them. Based on the ID of the source (the ID of the target that responds) the transaction are stored to the corresponding FIFO. A counter at the receiver NI also indicates the ID of the transaction that has to be delivered next and it is incremented whenever a transaction is delivered from the FIFO to the bus interface. One comparator per FIFO buffer compares the ID of the transaction that is in the front of the buffer to the ID counter. If the ID of the transaction matches the ID of the counter it indicates that as the next transaction to be transfered.

4. EXPLORATION ENVIRONMENT

Our experimental setup is based on a cycle-accurate NoC simulator. The simulator models accurately the ×pipes NoC library [33] which includes: accurate NI models that account for packetization effects, FLIT size converter models and switch models. On top of that we extended the initiator NI models to add support for multiple outstanding transactions and for reordering responses and for transaction splitting and interleaving. We implemented three types of traffic generators: i) the initiator traffic generator, ii) the target traffic generator and iii) the DRAM target traffic generator. The initiator traffic generator supports multiple outstanding transactions and out-of-order execution and it is described in detail in the next sub-section. The target traffic generator receives the requests from the initiator traffic generators and generates the response traffic. The target traffic generator can also be configured as synchroniza-

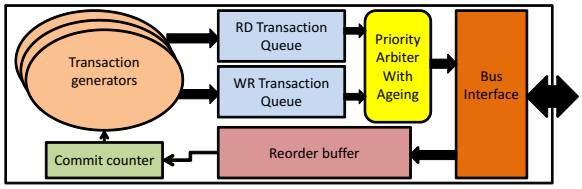

Figure 8: Initiator traffic generator

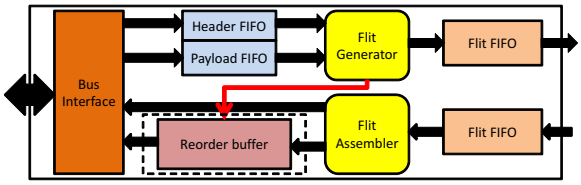

Figure 9: Initiator network interface

tion block. The DRAM target traffic generator uses DRAMSim 2 [12] to accurately emulate the functionality of the back-end of the DRAM controller. The DRAM traffic generator interface the NI to the DRAMSim instances and can be configured with multiple ports as well as channels in order to simulate all the architectures proposed in Section 3. Our benchmarking is oriented to real-life traffic in a state-of-the-art mobile SoC. We model different types of traffic and addressing modes to emulate different IP-cores like: video and graphic accelerators, display drivers, DMAs, peripherals and not just CPUs. Hence our environment is a significant step forward with respect to homogeneous architecture modeling environments with traffic generated only by cache misses.

4.1 Traffic Generators and NIs

The *initiator traffic generator* (ITGen) along with the initiator NI are some of the most important components of the simulator. The ITGen has to be able to emulate the traffic patterns generated by processors, accelerators, custom controllers, communication IP-cores. To be able to emulate all these behaviors, we added described in the simulator the ITGen to have the structure as presented in Figure 8. The basic blocks of the ITGen are the transaction generators, the read and write queues, the arbiter between the read and the write queues, the bus interface and the reorder buffer.

The transaction generators are responsible to generate the high-level read/write transactions (e.g. cache line refill or write-backs in a cache controller). The transaction generator can be programmed to generate different patterns of transactions, with different sizes and different time intervals between them. The types of transactions we use are: i) reads, ii) non posted writes and iii) barriers which are used for synchronization. We only considered non posted writes in this work as we assume that acknowledgments are needed to implement memory consistency protocols since data is shared among some cores. However posted writes can also have be used without impacting the results, especially since no acknowledgment traffic is necessary in that case. The transaction in the pattern are dependent as the transaction generator blocks if it tries to inject a transaction in a full queue. Consequently the following transaction are delayed by the same amount of time that the transaction generator is blocked. A ITGen can have multiple transaction generators which are independent from one another. For example if a transaction generator blocks on a full read queue another transaction generator could still generate write transactions if the write queue is not full. The use of multiple transaction generator can emulate multiple threads in a multi-threaded CPU or multiple independent units in an accelerator. The arbiter selects between the read and the write queue the transaction that has the highest priority, and which will be sent next to the bus interface. The bus interface has to convert the current selected transaction to bus requests (i.e. similar to AXI bus requests), which could comprise of a burst of requests in case of writes. Once a transaction has been processed by the bus interface, it is placed in the reorder buffer to await the response from the target traffic generator. The ITGen can be configured to expect responses in-order or out-of-order. In the first case when a response it received the ITGen checks that the response corresponds to the first transaction in the reorder buffer. In the latter case the response is matched to the corresponding request transaction in the reorder buffer. The serviced transactions in the reorder buffer are committed in-order, and the latency is tracked at commit time.

One important feature of the transaction generators is that they support multiple addressing modes, which can be configured when they are instantiated. The supported addressing modes are: i) *incremental*, ii) *incremental with synchronization*, iii) *block addressing*, iv) *random block addressing* and v) *trace*. *Incremental* addressing as the name suggests generates addresses incrementally with the possibility of wrapping around after a certain number of transactions. This mode can be used to emulate IP-cores that work with arrays or that move large amounts of data (e.g. DMA). In case of the *incremental with synchronization* addressing mode at the end of the accessed address range when it wraps around or moves to another address range it requires to synchronize with other IP-cores. New transactions are not generated until the synchronization completes. This mode can be used for IP-cores like the LCD or HDMI which read a frame with incremental transaction, but require synchronization before moving to another frame. *Block addressing* is designed to emulate the addressing mode of the video decoder. The frame is viewed as an M by N matrix which is further divided in blocks of size m by n ($m < M$ and $n < N$). The blocks are chosen in row order and for each block the addresses are generated in row order as well. This mode also requires synchronization at the end of the frame. In case of *random block addressing*, the next block to be accessed is chosen in a random manner, to emulate the fact that video decoding is data dependent. For *trace* addressing, we read a trace of addresses from file. These traces can be generated beforehand with a functional simulator and can be used to have a realistic behavior of IP-core for which the behavior is harder to emulate.

The initiator NI is described in Figure 9. The initiator NI contains the following blocks: the bus interface, the request queues (as header information and payload data queues), the FLIT generator block, the FLIT assembler block, the FLIT FIFOs and an optional reorder buffer. The bus interface is required to connect the NI to the traffic generators. The FLIT generator and FLIT assembler blocks convert the bus transactions into NoC packets and vice versa. FLIT buffers are used toward the switch as well to improve performance. In the case when the ITGen supports multiple outstanding transaction, but it requires the responses in order, a reorder buffer is necessary. To prevent deadlocks, the FLIT generator block will reserve space in the reorder buffer when generating a packet. If there is not enough space in the reorder buffer, the FLIT generator will block until space becomes available. In case the transaction requires more space than the size of the buffer the FLIT generator will wait until the buffer is completely free before starting to generate the packet. If transaction splitting and interleaving is supported by the NI then the reorder buffer is also necessary as the different pieces of a single transaction sent as different packets have to be reassembled in-order into the response transaction, which will be returned to the ITGen.

The target traffic generator is simpler. It only has the bus interface and a queue in which to store the incoming requests. The bus interface takes the stored requests and generates the appropriate response. The target traffic generator can be configured as a synchronization block as well. In that case it will only generate the responses when it received requests from all the ITGens that need to be synchronized. The DRAM controller simulation is done as described in Section 3. The target NI is similar to the initiator version, but it is simpler as it does not need support for reordering or splitting and interleaving.

4.2 Benchmark and Use-cases

To perform experiments we use a realistic benchmark that describes a mobile multimedia platform capable of running applications on a CPU cluster with 3D acceleration, perform video decoding, support multiple high definition displays and wireless communication. The communication requirements are represented by the graph from Figure 10. The vertices in the graph represent the task/IP-cores. In some case several tasks are performed by the same IP-core (e.g. Display LCD and CPU cluster). The edges in the graph represent a communication flow between two IP-cores. The weights on the edges represent the read/write bandwidth demands in MB/s. As can be seen from the plot this benchmark is DRAM centric with most communication going to the WideIO SDRAM.

The communication graph describes all the possible communication flows in the SoC, however different operating modes may require only a subset of the IP-cores to be active. To emulate this behavior and to see from experiments the trends on bandwidth and latency as more flow become active, we define four use-cases. Three of the use-case activate only a subset of the flows, while the last one will use all. A description of the use-case is provided in Table 1.

Use-case 1 (UC1) uses only the Video decoder IP-core and the HDMI display IP-core to provide basic video playback. Use-case 2 assumes that along with the video playback, applications are also running on the CPU that require 3D graphic acceleration. In use-case 3, two displays are used. In case of the LCD display the resolution is considered to be less than 1080p so downscaling and rotation operations are needed. All the IP-cores that are active in UC3 are high bandwidth. In use-case 4 we consider the influence of the remaining IP-cores most of which are low bandwidth, but may require low latency as is the case of the Modem IP-core.

4.3 Topology

For the experiments we use a 6 switch NoC topology. To prevent message level deadlocks, we use two separate networks so 3 of the switches are used for the request network and 3 for the response network. The request network topology is presented in Figure 11 and the response network is symmetric, the only difference being that the data flows in the opposite direction. In the figure the round nodes represent the IP-core (in the simulator these require an instance of the traffic generator as well as an instance of an NI) and the number below the name represents the size of the data interface for that IP-core. The low bandwidth IP-cores have a 32 bit interface, while the high bandwidth IP-cores use 64 bits. Since the WideIO SDRAM has 4 channels with a data size of 128 bits, to be able to transfer full bandwidth we assume the DRAM controller interface toward the NoC is also 128 bit wide. Similarly some if the IP-core that generate large regular transactions like the display drivers also use 128 bit data interfaces.

As can be seen from the figure the IP-cores that have the same data width are clustered on the same switch. To reduce the power consumption the switches have a FLIT size similar to the data size of the cores connected to it. The only exception is the Video IP-core which is connected directly to the 128 bit switch. When transactions are converted into packets the bandwidth requirements increase due to the extra information that is added in each packet. Since the Video IP-core has high bandwidth demands, a link in the NoC with a 64bit FLIT width could not support the bandwidth resulting after packetization and therefore we connected it directly to the wider switch. Even though the Video IP-core generates significant traffic, it does so in small transaction and therefore we assumed that it has a 64bit data-interface. Also this setup allows us to capture the packetization effects from different data-sizes.

Since the switches have different FLIT sizes, converters need to be used on each link to change the width of the FLITS in each packet. It is important to accurately simulate the size converters as they can also increase the bandwidth when going from narrow to wide. Since we use wormhole switching, once the head of a packet has reserved a channel, that channel remains reserved until the tail of the packet passes through it. Since the narrow network cannot provide sufficient data to generate a FLIT every cycle after the size converter. Since the channels remain reserved until the tail passes, the resulting packet appears to have more FLITs in the wide network. This effect manifests itself as having a higher bandwidth flow. However in case of downstream contention these empty FLITS can be dropped. To have accurate results we model all these effects.

Apart from the IP-cores described in the communication graph of the benchmark, we attached another target IP-core on the narrow 32 bit switch. This core called *Semaphore* is used to synchronize the operation of the Video, LCD and HDMI IP-cores when they change from one frame to another.

5. EXPERIMENTAL RESULTS

In the experiments, using the benchmark described in Section 4.2, we analyze how the bandwidth and latency are affected by the presence of interfering flows at the WideIO DRAM memory. We analyze 5 setups based on the three architectures described in Section 3. The simulations setups with their corresponding names are the following:

- *Separate 1CH*: uses a distributed DRAM controller architecture with separate independent channels. This simulates the worst-case when all the active memory regions addressed by the ITGens are mapped to the same channel;
- *Separate 4CH*: uses the same distributed architecture, but in this case the active memory regions were carefully mapped to different channels. For example for the Video IP-core which generate high bandwidth traffic, the address memory regions have been distributed in all channels.
- *Controller I 3 Port*: uses a centralized DRAM architecture with interleaving at the controller. As there are more ITGens than ports in this setup several ITGens have been assigned to use the same port (only three of the 4 ports are used).
- *Controller I 4 Port*: uses the same centralized architecture, but the ITGens have been assigned to all four ports so that there is less contention at the DRAM controller ports.
- *NI I*: uses a distributed DRAM controller architecture, however the address space it is interleaved between the four channels of the WideIO memory. In this case the initiator NIs are responsible to split transactions into packets and to send the packets to the appropriate memory channel according to the address.

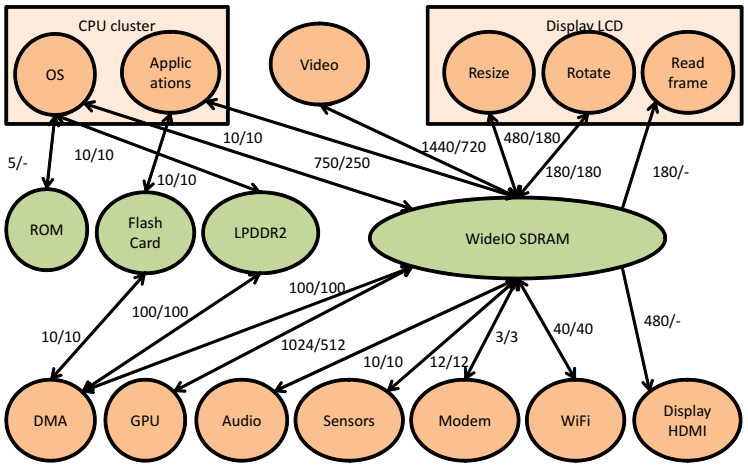

Figure 10: Benchmark communication graph

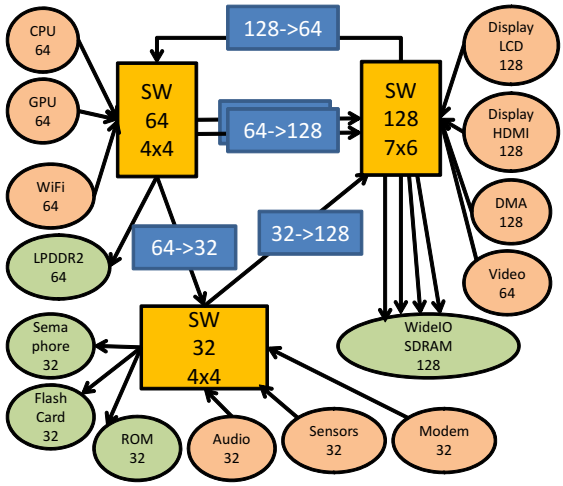

Figure 11: Topology

	CPU	GPU	Video	LCD	HDMI	DMA	Audio	Sensors	WiFi	Modem
UC1			x		x					
UC2	x	x	x		x					
UC3	x	x	x	x	x					
UC4	x	x	x	x	x	x	x	x	x	x

Table 1: Description of the use-cases

The simulation parameters used for the initiator traffic generators are presented in Table 2. For the initiator NIs, we used 2-deep bus transaction buffers and 5-deep FLIT FIFOs. For the setups using the distributed DRAM controller with independent channels and for the centralized DRAM controller, reorder buffers are not needed as there is a single route to reach the memory controller for the ITGens that require responses to be return in-order. In case of the distributed controller with splitting and interleaving at the NI, reorder buffers are not needed for the CPU, GPU and Video IP-cores. That is because the largest transaction is the same size as the interleaving size so no splitting is required. Also these traffic generators accept responses out-of-order. In case of the LCD and HDMI display controllers as well as the DMA, the transactions have to be split and therefore we need a reorder buffer. We used 24-deep reorder buffer (each entry in the reorder buffer is a bus transaction). In case of the Audio, WiFi, Sensor and Modem IP-cores, no splitting is necessary, but the responses have to be delivered in order so, there is a reorder buffer. Since these cores are low bandwidth, we use 16 deep reorder buffers.

5.1 Throughput oriented communication

In our benchmark, we have IP-cores that are throughput sensitive (Video, HDMI, LCD) and some that are latency sensitive (CPU, Modem). In this section, we will have a closer look at two representative throughput sensitive IP-cores, namely he Video accelerator and the HDMI display controller. Both IP-cores require a significant amount of bandwidth and have to synchronize after finishing a frame. The Video controller generates small transactions, while the HDMI display controller transfers data in long 1kB burst transactions. In Figure 12, we show the trend of the average latency for the Video transactions as more interfering bandwidth is introduced with the more complex use-cases (in UC1 only the Video and the HDMI cores are active, while in UC4 all the IP-cores in the benchmark are active). As can be seen from the plot the distributed interleaving scheme that balances the traffic in the network as well has the lowest latency. Moreover compared to the other schemes, the distributed interleaving is also the least sensitive to the interfering bandwidth, as it has the lowest growth. The figure also shows that the mapping of data in memory has a significant impact on the performance of the distributed controller with independent channels. However with the best memory mapping as well as with the distributed controller with NI interleaving the latency is lower than in the case of the centralized controller. This is because both schemes exploit the parallelism in the NoC as well.

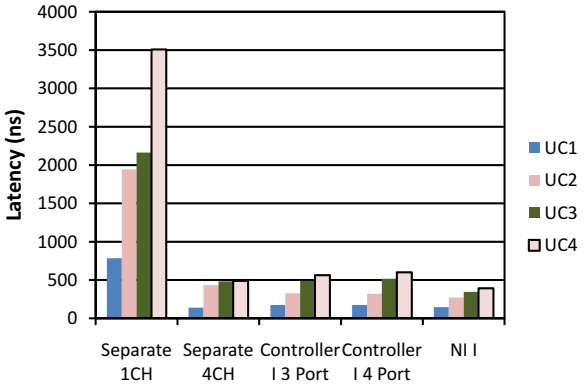

Figure 12: Average latency for the Video IP

Similarly, in Figure 13, we show the average latency of the HDMI transactions. An important thing to remember is that we measure latency of a transaction from the time it is generated by the ITGen and placed in the read/write buffer of the ITGen until the time it received the response and it is committed by the ITGen (in case

	Out-of-order	Maximum outstanding		Number of transaction generators	Address mode	Transaction sizes
		Reads	Writes			
CPU	yes	4	8	6	trace+incremental	64b, 512b
GPU	yes	8	8	1	incremental	512b
Video	yes	30	30	5	block + random block addressing	128b, 256b
LCD	yes	10	10	1	incremental with synchronization	1kB
HDMI	yes	10	10	1	incremental with synchronization	1kB
DMA	yes	10	10	3	incremental	128b, 1kB
WiFi	no	4	4	1	incremental	128b
Sensors	no	2	2	1	incremental	32b
Audio	no	2	2	1	incremental	128b
Modem	no	2	2	1	incremental	256b

Table 2: Simulation parameters for the ITGens

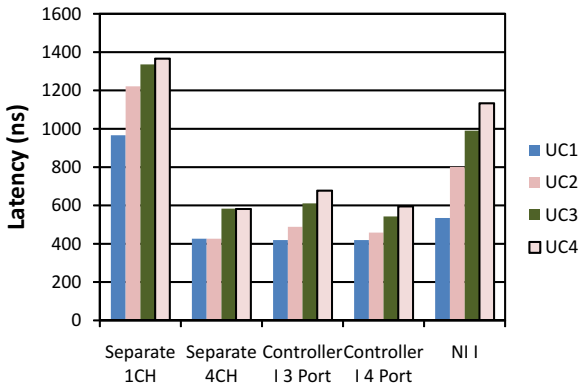

Figure 13: Average latency for the HDMI IP

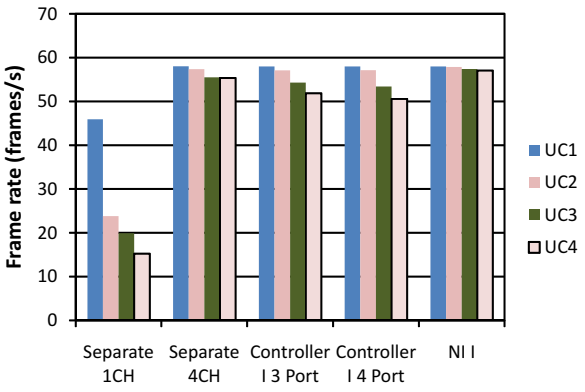

Figure 14: Average frame-rate

of out-of-order ITGens the transactions are committed in-order so any completed transactions cannot be committed until all previous transactions have been completed and committed). In case of the HDMI core, data is transfered in 1kB transactions. Therefore in case of the distributed controller with interleaving in the NI, the transactions have to be split over multiple packets, in order to be sent to the appropriate channel. The different response packets have to be reordered and assembled such that the response from the NI to the ITGen is sent as a single burst of in-order bus transfers. The number of packets of the transactions that can be sent out is limited by the size of the reorder buffer. This results in a higher latency per transaction for the HDMI core as can be seen from the figure. However since a single transactions transfers 1kB of data these transfers are very efficient and the increased latency does not impact the overall performance of the system. If we look at the latency per byte for these 1kB transfers we can see that they are 11 times more efficient than the smaller transfers generated by the Video core (1.1 ns/byte for HDMI and 12.2 ns/byte for the Video in UC4), which explains why the larger latency of the HDMI core does not impact the system performance.

To analyze the overall system performance, in Figure 14, we show the frame rates obtained for the different use-cases in each simulation setup. The distributed DRAM controller with interleaving at the NI obtains the best frame rate and also has the lowest degradation of the frame rate as more interfering communication flows are activated. The frame rate of this setup is better even than that of the distributed controller with separate independent channels even for the best memory allocation scheme, as it balances the traffic from all the cores in the network as well. Surprisingly the centralized controller where only three ports are used has better frame rate than when 4 ports are used. That is because in the case of the three port setup some of the interfering flows that are mapped on the same port interferer among themselves impacting their performances and in this particular case favoring the Video core. This also shows the sensitivity of the centralized controller scheme to the mapping of the communication flows to ports.

5.2 Bandwidth analysis

As the benchmark is representative of real applications, it is DRAM centric with all communication being between the IP-cores and the DRAM controller. Therefore one important metric to assess the performance of the SoC is also the total read/write bandwidth to the DRAM memory. In Figure 15, we show the bandwidth obtained with each simulation setup for all use-cases normalized to the bandwidth required by that use-case. The reported bandwidth refers to the effective data bandwidth measured by the ITgens, the actual bandwidth transfered in the NoC is higher due to packetization and the bandwidth inflation generated by the size converters.

As expected, due to the balancing of traffic through the NoC as well, the distributed DRAM controller with NI interleaving setup achieves the best performance in terms of total average bandwidth as well (6184MB/s out of 6980MB/s demanded by use-case 4). The distributed controller with separate independent channels achieves similar performance for the most demanding use-case when the allocated memory is well distributed in the channels. However, to achieve that performance with independent channels the hardware

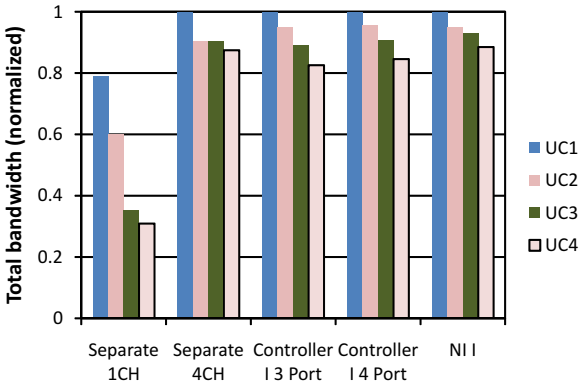

Figure 15: Average total bandwidth

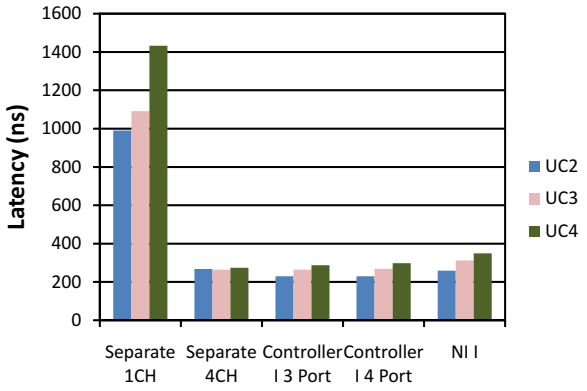

Figure 16: Average latency for the CPU IP

would need to be exposed to the software and the managing operating systems should be smart enough to allocate memory such as to balance the channel usage. In case of the hardware interleaving scheme the memory access is transparent to the software, favoring the NI interleaved scheme which balances traffic in the interconnect as well.

5.3 Latency sensitive communication

So far we have analyzed the bandwidth oriented communication flows. In this section we analyze two of the latency sensitive IP-cores. One of them is the CPU where read are on the critical execution path and therefore the CPU performance is dependent on the latency of reads. On the other hand the CPU generates a significant amount of bandwidth as well. Another latency sensitive core is the Modem which has strict latency requirement imposed by the communication protocol, however the Modem core only requires little bandwidth.

In Figure 16, we present the average latency of the CPU for the three use-cases where the CPU is active. As can be seen from the figure the latency of the CPU is only slightly larger for the distributed controller with interleaving at the NI when compared to the centralized controller (around 17% for UC4). That is because in the NI interleaving scheme the transaction from the CPU are distributed to all ports and they interfere with the traffic of the high-bandwidth cores at all the ports. In the other case the CPU is mapped by itself to one port. One aspect to remember however is that in the case of the distributed controller with separate independent channels the mapping of memory to channels is application dependent and it may not be possible to achieve a good mapping in all cases. As can be seen from the figure for a bad mapping of the memory to channels, the controller with separated channels has very poor performance. Similarly the latency of our solution is slightly larger than that of the centralized controller. Nevertheless, it may be very expensive to physically implement the centralized controller (due to the distance between the channel interfaces), and as such the distributed interleaving method provides a good trade off. Moreover the distributed controller can provide other opportunities for the adaptation of the NoC topology (i.e., connect the different channel controller to different switches).

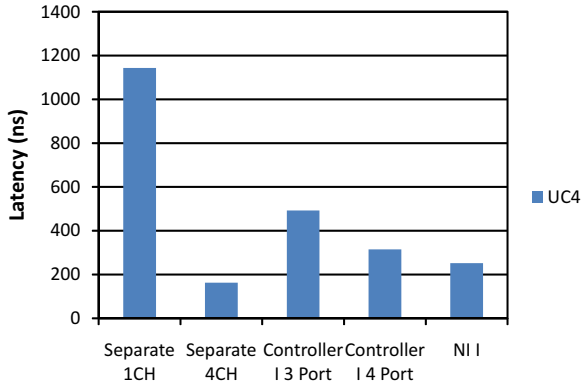

Figure 17: Average latency for the Modem IP

In Figure 17, we show the average latency for the Modem core. This core is latency sensitive, but has low bandwidth requirements. From the setups that perform interleaving the NI interleaved case has the lowest latency. The latency for the controller with separated independent channels, for the best mapping has the lowest latency. That is because in this case the communication of the Modem is to a separated channel where there are no high-bandwidth cores mapped. Interestingly, as the core has low bandwidth its latency is more affected by the port assignment in the case of the centralized controller.

6. CONCLUSIONS

3D-stacked WideIO DRAM memory promises to deliver the required bandwidth to satisfy the demands of current and future application running on mobile SoCs at acceptable power-levels, by providing multiple channels and wide data interfaces. The challenge for SoC designers is to efficiently access the multiple channels provided by the WideIO interface, to achieve the required bandwidth and latency for communication flows.

In this work we analyze three architecture for designing the memory controller that access the WideIO DRAM memory. We propose for one of the architectures a new distributed interleaving method. The new method capitalizes on the advantages of the two analyzed methods, to provide a simple DRAM controller design and to balance the traffic in both the NoC and at the memory channels transparent to the software. From experiments, we show that our proposed distributed interleaving memory access method improves the overall throughput while minimally impacting the performance of latency sensitive communication flows.

Future work will focus on understanding the relationship between network topology and the distributed interleaving scheme,

and to analyze how to implement low overhead *Quality-of-Service* support to aggressively reduce latency for critical flows.

7. ACKNOWLEDGMENTS

We would like to acknowledge the contributions of Eric Flamand from STMicroelectronics and Denis Dutoit from CEA LETI, for their help in the construction of a realistic evaluation environment. This work was supported in part by The European Research Council under project AdG-246810-NANOSYS, by the Pro3D project grant agreement number: 248776 and the ARTIST-DESIGN Network of Excellence.

8. REFERENCES

[1] WULF, W. A., AND MCKEE, S. A. Hitting the memory wall: implications of the obvious. *SIGARCH Comput. Archit. News 23*, 1 (Mar. 1995), 20–24.

[2] BORKAR, S. 3d integration for energy efficient system design. In *Proc. 48th ACM/EDAC/IEEE Design Automation Conf. (DAC)* (2011), pp. 214–219.

[3] WEIS, C., WEHN, N., IGOR, L., AND BENINI, L. Design space exploration for 3d-stacked drams. In *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE)* (2011), pp. 1–6.

[4] KIM, J.-S., OH, C. S., LEE, H., LEE, D., HWANG, H.-R., HWANG, S., NA, B., MOON, J., KIM, J.-G., PARK, H., RYU, J.-W., PARK, K., KANG, S.-K., KIM, S.-Y., KIM, H., BANG, J.-M., CHO, H., JANG, M., HAN, C., LEE, J.-B., KYUNG, K., CHOI, J.-S., AND JUN, Y.-H. A 1.2v 12.8gb/s 2gb mobile wide-i/o dram with 4x128 i/os using tsv-based stacking. In *Proc. IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers (ISSCC)* (2011), pp. 496–498.

[5] DE MICHELI, G., AND BENINI, L. *Networks on Chips: Technology and Tools; electronic version*. Elsevier, Burlington, MA, 2006.

[6] DE MICHELI, G., SEICULESCU, C., MURALI, S., BENINI, L., ANGIOLINI, F., AND PULLINI, A. Networks on Chips: From research to products. In *Proc. 47th ACM/IEEE Design Automation Conf. (DAC)* (2010), pp. 300–305.

[7] RIXNER, S., DALLY, W. J., KAPASI, U. J., MATTSON, P., AND OWENS, J. D. Memory access scheduling. In *Proc. 27th Int Computer Architecture Symp* (2000), pp. 128–138.

[8] AHN, J. H., EREZ, M., AND DALLY, W. J. The Design Space of Data-Parallel Memory Systems. In *Proc. ACM/IEEE SC 2006 Conf* (2006).

[9] AKESSON, B., GOOSSENS, K., AND RINGHOFER, M. Predator: A predictable SDRAM memory controller. In *Proc. 5th IEEE/ACM/IFIP Int Hardware/Software Codesign and System Synthesis (CODES+ISSS) Conf* (2007), pp. 251–256.

[10] LEE, K.-B., LIN, T.-C., AND JEN, C.-W. An efficient quality-aware memory controller for multimedia platform SoC. *Circuits and Systems for Video Technology, IEEE Trans. on 15*, 5 (2005), 620–633.

[11] RAFIQUE, N., LIM, W.-T., AND THOTTETHODI, M. Effective Management of DRAM Bandwidth in Multicore Processors. In *Proc. 16th Int. Conf. Parallel Architecture and Compilation Techniques PACT 2007* (2007), pp. 245–258.

[12] WANG, D., GANESH, B., TUAYCHAROEN, N., BAYNES, K., JALEEL, A., AND JACOB, B. DRAMsim: a memory system simulator. *SIGARCH Comput. Archit. News 33* (November 2005), 100–107.

[13] DUTOIT, D., AND JERRAYA, A. 3D integration opportunities for memory interconnect in mobile computing architectures. *Future Fab CEA-Leti MINATEC Issue 34* (2010), pp. 38–45.

[14] ANIGUNDI, R., SUN, H., LU, J.-Q., ROSE, K., AND ZHANG, T. Architecture design exploration of three-dimensional (3d) integrated dram. In *Proc. Quality Electronic Design Quality of Electronic Design ISQED 2009* (2009), pp. 86–90.

[15] FACCHINI, M., CARLSON, T., VIGNON, A., PALKOVIC, M., CATTHOOR, F., DEHAENE, W., BENINI, L., AND MARCHAL, P. System-level power/performance evaluation of 3d stacked drams for mobile applications. In *Proc. DATE '09. Design, Automation & Test in Europe Conf. & Exhibition* (2009), pp. 923–928.

[16] WOO, D. H., SEONG, N. H., LEWIS, D. L., AND LEE, H.-H. S. An optimized 3d-stacked memory architecture by exploiting excessive, high-density tsv bandwidth. In *Proc. IEEE 16th Int High Performance Computer Architecture (HPCA) Symp* (2010), pp. 1–12.

[17] LOI, I., AND BENINI, L. An efficient distributed memory interface for many-core platform with 3d stacked dram. In *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE)* (2010), pp. 99–104.

[18] LOH, G. H. 3d-stacked memory architectures for multi-core processors. In *Proc. 35th Int. Symp. Computer Architecture ISCA '08* (2008), pp. 453–464.

[19] LOH, G. H. Extending the effectiveness of 3d-stacked dram caches with an adaptive multi-queue policy. In *Proc. MICRO-42 Microarchitecture 42nd Annual IEEE/ACM Int. Symp* (2009), pp. 201–212.

[20] SUN, H., LIU, J., ANIGUNDI, R. S., ZHENG, N., LU, J.-Q., ROSE, K., AND ZHANG, T. 3d dram design and application to 3d multicore systems. *IEEE Design & Test of Computers 26*, 5 (2009), 36–47.

[21] AKESSON, B., HUANG, P.-C., CLERMIDY, F., DUTOIT, D., GOOSSENS, K., CHANG, Y.-H., KUO, T.-W., VIVET, P., AND WINGARD, D. Memory controllers for high-performance and real-time mpsocs requirements, architectures, and future trends. In *Proc. 9th Int Hardware/Software Codesign and System Synthesis (CODES+ISSS) Conf* (2011), pp. 3–12.

[22] AHO, E., NIKARA, J., TUOMINEN, P. A., AND KUUSILINNA, K. A case for multi-channel memories in video recording. In *Proceedings of the Conference on Design, Automation and Test in Europe* (3001 Leuven, Belgium, Belgium, 2009), DATE '09, European Design and Automation Association, pp. 934–939.

[23] JANG, W., AND PAN, D. Z. An SDRAM-Aware Router for Networks-on-Chip. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 29*, 10 (2010), 1572–1585.

[24] YOO, J., YOO, S., AND CHOI, K. Multiprocessor system-on-chip designs with active memory processors for higher memory efficiency. In *Proc. 46th ACM/IEEE Design Automation Conf. DAC '09* (2009), pp. 806–811.

[25] WALTER, I., CIDON, I., GINOSAR, R., AND KOLODNY, A. Access Regulation to Hot-Modules in Wormhole NoCs. In *Proc. First Int. Symp. Networks-on-Chip NOCS 2007* (2007), pp. 137–148.

[26] KIM, D., YOO, S., AND LEE, S. A Network Congestion-Aware Memory Controller. In *Proc. Fourth ACM/IEEE Int Networks-on-Chip (NOCS) Symp* (2010), pp. 257–264.

[27] KIM, D., KIM, K., KIM, J.-Y., LEE, S.-J., AND YOO, H.-J. Solutions for Real Chip Implementation Issues of NoC and Their Application to Memory-Centric NoC. In *Proc. First Int. Symp. Networks-on-Chip NOCS 2007* (2007), pp. 30–39.

[28] KIM, D., KIM, K., KIM, J.-Y., LEE, S., AND YOO, H.-J. Implementation of Memory-Centric NoC for 81.6 GOPS object recognition processor. In *Proc. IEEE Asian Solid-State Circuits Conf. ASSCC '07* (2007), pp. 47–50.

[29] ARTERIS. http://www.arteris.com/pr_22_oct_08, 2008.

[30] CASINI, P. SoC Architecture to Multichannel Memory Management Using Sonics IMT. Tech. rep., Sonics, Inc., 2008.

[31] WANG, F., AND HAMDI, M. Scalable router memory architecture based on interleaved dram. In *High Performance Switching and Routing, 2006 Workshop on* (0-0 2006), p. 6 pp.

[32] AKESSON, B., AND GOOSSENS, K. Architectures and modeling of predictable memory controllers for improved system integration. In *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE)* (2011), pp. 1–6.

[33] STERGIOU, S., ANGIOLINI, F., CARTA, S., RAFFO, L., BERTOZZI, D., AND MICHELI, G. D. ×pipes Lite: A Synthesis Oriented Design Library For Networks on Chips. In *Proceedings of the conference on Design, Automation and Test in Europe - Volume 2* (Washington, DC, USA, 2005), DATE '05, IEEE Computer Society, pp. 1188–1193.