A DRAM Centric NoC Architecture and Topology Design Approach

Ciprian Seiculescu*, Srinivasan Murali§ *, Luca Benini‡, Giovanni De Micheli*
* LSI, EPFL, Lausanne, Switzerland,{ciprian.seiculescu, giovanni.demicheli}@epfl.ch
§ iNoCs, Lausanne, Switzerland, murali@inocs.com
‡ DEIS, University of Bologna, Bologna, Italy, luca.benini@unibo.it

ABSTRACT

Most communication traffic in today's System on Chips (SoC) is DRAM centric. The NoC should be designed to efficiently handle the many-to-one communication pattern, funneling to and from the DRAM controller. In this paper, we motivate the use of a separate network for the DRAM traffic and justify the power overhead and performance improvement obtained, when compared to traditional solutions. We also show how the topology of this DRAM network can be designed and optimized to account for the funnel-shaped pattern. Our experiments on a realistic SoC multimedia benchmark shows a large reduction in power consumption and improvement in performance when compared to existing solutions.

Keywords

Network-on-Chip (NoC), DRAM

1. INTRODUCTION

Many of today's *Systems on Chips (SoCs)* are DRAM centric as the applications running on them have large data storage requirements. In such systems many cores access the external DRAM memory through a single on-chip DRAM controller. Since there are many cores that have high communication demands to a single DRAM controller, the controller can become the bottleneck for the system performance. It can even affect the performance of cores that do not communicate with it directly. Several works have presented methods to improve the efficiency for accessing DRAM through transaction scheduling and reordering [12]. However reordering transaction makes the latency for accessing DRAM unpredictable and highly dependent on the address traces generated by the applications accessing it. The unpredictable nature of DRAM access can lead to transitory bottlenecks at the DRAM controller ports. If a *Network-on-Chip (NoC)* is used as the system interconnect, packets backlog at the DRAM ports would queue up in the network and interfere with non DRAM traffic. It is highly undesirable for non DRAM traffic to be blocked by DRAM traffic as it can take tens to hundreds of cycles to clear out the DRAM backlog.

To prevent DRAM traffic from blocking non DRAM traffic end-to-end flow control can be used. However end-to-end flow control requires the use of a specialized mechanism to notify the cores of the state of the DRAM. For example a separate network or high priority virtual channel [20] can be used. This not only increases the design complexity, but also the buffering requirements needed to drain the messages that are already in the network. Another solution is to separate the traffic going to DRAM (DRAM traffic) from the traffic going between the other cores (non DRAM traffic) using virtual channels or physical channels. However for most SoC the traffic patterns are known at design time and the interconnect topology can be optimized for that specific application. In such cases physical channels can be used when the NoC is designed to split the traffic. Since the bandwidth requirements to the DRAM controller are large, the ports of the DRAM can have wider data paths than the rest o the cores. In this work we show how designing a separate network to DRAM is better, as it can account for the funnel-shaped pattern communication pattern and reduce the number of data size converters.

The contributions are three fold as follows: It is unclear what is the power and performance trade-off for splitting DRAM and non DRAM traffic using dedicated physical channels in the NoC. Therefore we first analyze this and also compare it to a solution using end-to-end flow control. We present results for a realistic multimedia and wireless communication SoC benchmark on application specific topologies. We use a cycle accurate NoC simulator and DRAM controller simulator [17] to compare the performance of the shared and split NoCs for the generated topologies. Second we show how the network to DRAM can be optimized due to the funnel-shaped communication pattern when the cores and the DRAM controller have different data bus sizes. Third we present an architecture for data size converter, capable of transferring data full bandwidth on the wider side.

Our experiments show several interesting results: i) As expected *split* network and *end-to-end flow control* perform significantly better for non DRAM traffic (2.2× lower write latency and 1.7× lower read latency); ii) *Split* network and *end-to-end flow control* have similar power overhead; iii) All solutions perform similarly for DRAM traffic; iv) Designing a separate network to DRAM can be optimized in the presence of heterogeneous cores leading to power saving of 23%.

2. RELATED WORK

NoC as a new interconnect paradigm and the benefits of NoCs were presented in [1] and [2]. Many methods for designing application specific NoC topologies are available in literature [3]-[11]. In this work we analyze the impact the a shared DRAM controller has on the traffic in the network. We use the method from [11] to generate the topologies.

Several works have presented method to improve DRAM access efficiency by scheduling and reordering transactions in the DRAM controller [12]-[17]. In [13], [15] are presented optimizations for multicore systems, and in [14] a predictable DRAM controller is presented. Memory 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 [17] in order to include these memory access optimizations in our simulation framework.

In [18] the authors present a way to reorder DRAM transactions

978-0-7695-4447-2/11 $26.00 © 2011 IEEE
DOI 10.1109/ISVLSI.2011.60

IEEE computer society

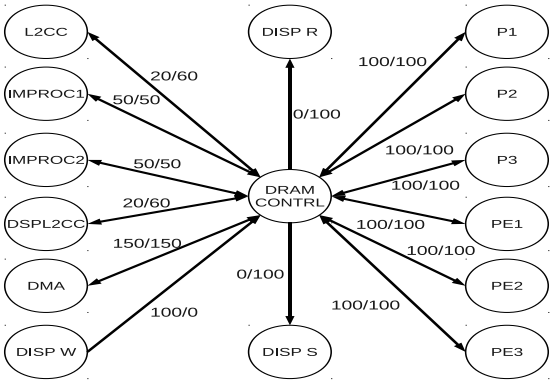

Figure 1: Communication specifications to DRAM

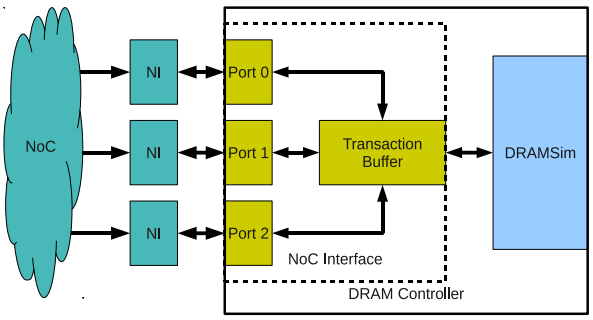

Figure 2: Example of DRAM controller connection

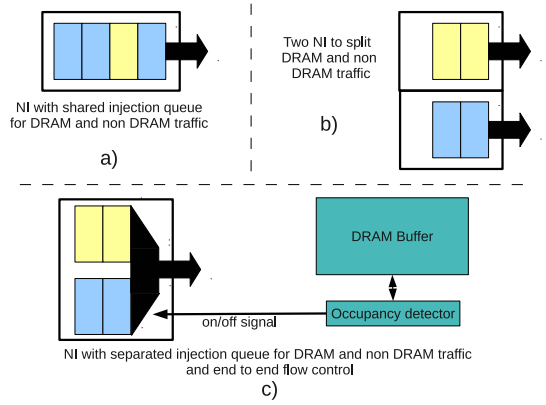

Figure 3: NI injection queues: a) *shared* **b)** *split* **and c)** *end-to-end flow control*

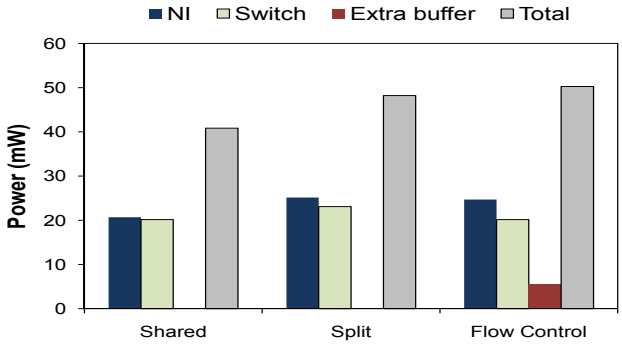

Figure 4: Power consumption for shared and separated NoCs

while in the network and simplify the DRAM controller. However this does not solve the problem of interference with the normal traffic in the NoC. To reduce the traffic to DRAM the authors propose in [19] 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 [20] to prevent the DRAM traffic from waiting into the network and interfering with non DRAM traffic. In this work we propose the use of a separated DRAM network to prevent non DRAM traffic to be blocked by DRAM traffic. In [21] 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. However in custom topologies the response path is usually not congested. Memory centric NoC architectures with real chip implementations are provided in [22] and [23]. However in this systems there are several memories which are on-chip and the communication between cores is done through these memories.

3. SYSTEM ARCHITECTURE

We use NoC switches and *Network Interfaces (NIs)* similar to those from [24]. We use input buffering, on/off flow control and source routing for the switches with no virtual channels. We consider a high performance DRAM controller capable of scheduling and reordering transactions. We synthesized power efficient application specific topologies using the tool from [11]. All the topologies have 4 switches for the request network and 4 switches on the response network. We chose these topologies as they were the most power efficient generated by the tool. The power models for the NoC components are base on values provided by commercial tools after place and route of the library components from [24] in 65nm technology. We analyze three designs. In the first design we synthesized a topology where the traffic that goes to DRAM and the non DRAM traffic can share channels. In the rest of the paper we call this design as *shared* NoC. In the second case we synthetized a topology where the DRAM traffic is physically separated from the non DRAM traffic, which we call *split* NoC. In the third design we use the same topology as in the case of the shared NoC, but we apply *end-to-end flow control* on top, to prevent DRAM messages from queuing in the NoC when the DRAM is backlogged. We use on/off flow control and the occupancy of the DRAM controller buffers determines when core can inject packets to DRAM. This requires a specialized interconnect to give the off signal to the cores.

3.1 Benchmark

We used a realistic benchmark for a multimedia and wireless communication System-on-Chip. The benchmark has 28 cores, among which an processor, a DSP both with L2 caches, a DMA, two image accelerators, a display controller, 6 clusters of peripherals, five scratch pad memories, a flash memory controller and a DRAM controller to connect to an external DRAM memory. The communication graph to and from DRAM is shown in Figure 1. The vertices represent the cores and the edges the communication flows between the cores. The weights on the edges represent the required communication bandwidth for write/read in MB/s. The total

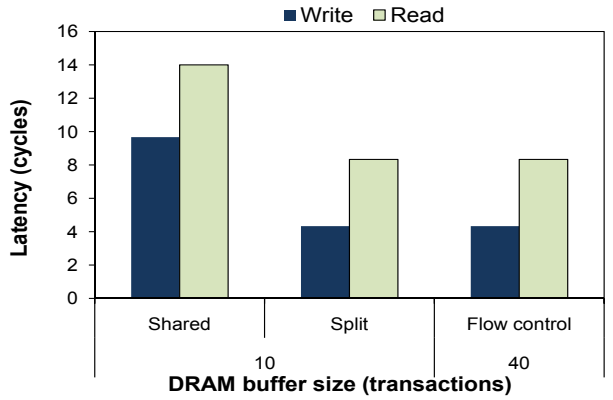

Figure 5: Average latency for non DRAM flows

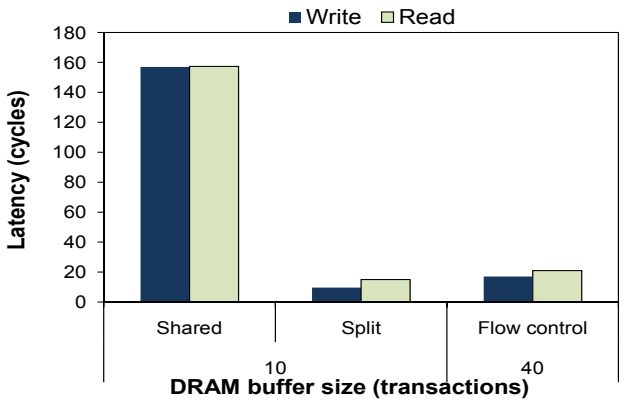

Figure 6: Maximum latency for non DRAM flows

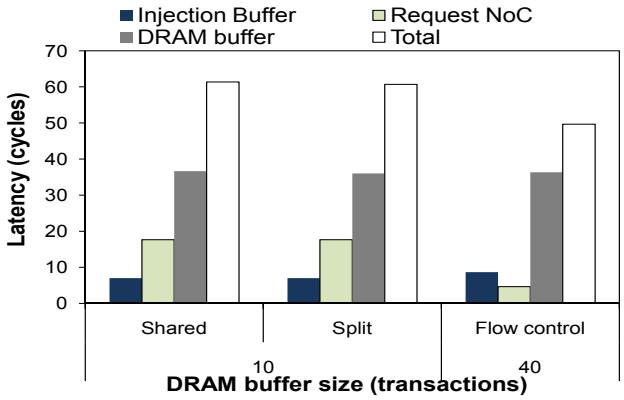

Figure 7: Average latency split for write DRAM flows

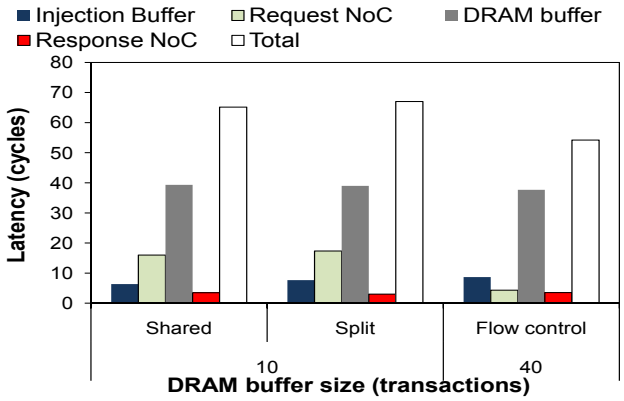

Figure 8: Average latency split for read DRAM flows

desired bandwidth to the DRAM controller is of 2160 MB/s. For reasons of space we only show the communication specifications to the DRAM controller.

3.2 Simulation infrastructure

To run experiments, we use a cycle accurate NoC simulator capable to simulate custom topologies. In the NoC simulator we plugged-in DRAMsim2 [17], a cycle accurate DRAM simulator. We configured DRAMsim2 with open row scheduling policy and bank round robin. Transactions are interleaved in the banks based on the least significant bits of the address. We connect the network simulator to the DRAM simulator using a custom interface to provide support for multiple ports and to assemble the NoC messages into DRAM transactions of fixed size, as showed in Figure 2. We assume the DRAM to be a DDR2 device running at 333 MHz, and a 64 bit data bus from the DRAM controller to the external memory. We use the timing parameters of the Micron DDR2 of 32MB with 4 banks and 4 bit interface, 4 word burst and -3E speed grade, that is provided with the DRAMSim2 package. We design the NoC synchronous running at the DRAM controller frequency of 333 MHz.

The traffic generators have infinite injection queues. This is to capture in the measured latency the effect of the core being delayed, when they are unable to inject due to congestion the network. Traffic generators are connected to the initiator *Network Interfaces (NIs)* and inject traffic according to the bandwidth specified in the communication specification. The injected packets have a size of 32 byte of actual data for the write requests and read responses. The actual bandwidth that is injected in the network accounts for the packetization overhead that the NoC architecture creates. In Figure 3 we present how the NIs are modeled. In case of the *shared* NoC the DRAM and non DRAM packets can mix (Figure 3 a). In the *split* NoC, two NIs are used to separate the packets (Figure 3 b). For *end-to-end flow control* separate queues in the same NI are used (Figure 3 c). More over based on the occupancy of the dram buffer the queue holding the DRAM packets can be prevented from injecting. In our work we use a separate signal to notify the NIs when they can inject based on the DRAM buffer occupancy. The target traffic generators respond to the read requests immediately for the flows that do not go to the DRAM controller. For the DRAM traffic the read and write requests are buffer in the DRAM controller (however the DRAM controller buffer is finite), and the response is sent back through the network once the DRAM controller services the transaction.

4. SEPARATED NOC VS. SHARED NOC

4.1 Power analysis

The power consumption of the NoCs is estimated by the synthesis tool that uses the power models of the components from the library in [24]. We show the breakdown of the power consumption in Figure 4. The *split* network has higher power as expected, as more NI are needed for cores that have both DRAM and non DRAM traffic. Also switches are larger to connect the extra NIs. Interestingly the overhead of the switch to switch connectivity is

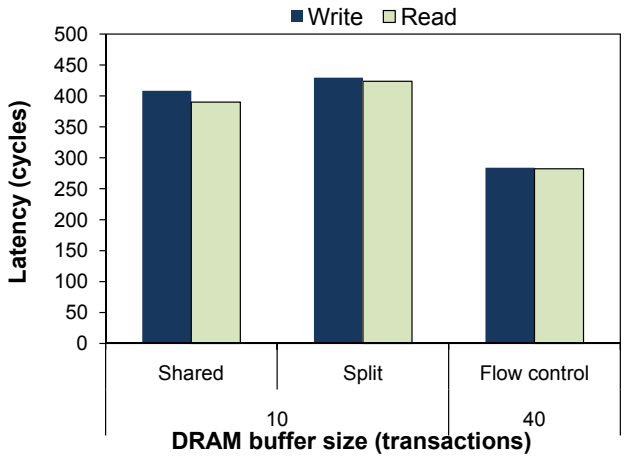

Figure 9: Maximum latency for DRAM flows

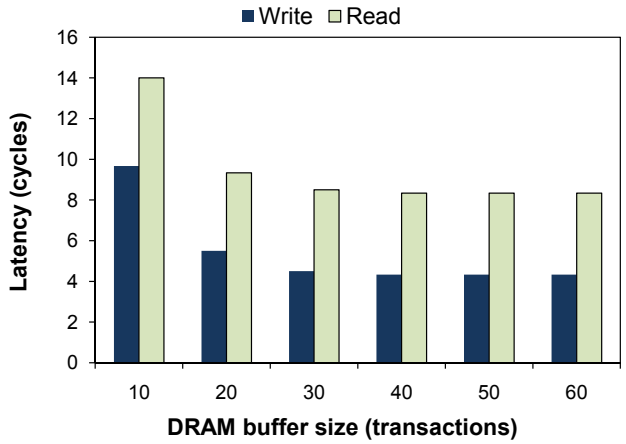

Figure 10: Average latency for non DRAM flows in shared NoC

not significant. For our benchmark the power overhead is 18%.

The *end-to-end flow control* design also has similar power overhead as *split*. The NIs of cores that have both DRAM and non DRAM traffic need separated packet queues. When the occupancy of the DRAM controller buffer reaches a threshold, the off signal is send to the injector NIs, As it takes several cycles to reach all the injectors there can be several inflight packets going to DRAM. Therefore the buffer on the DRAM side has to be large enough to buffer these inflight DRAM packets. We use simulation data to determine the required size of the buffer. As buffering requirements in the DRAM controller are similar to those found in the network to DRAM in the *split* NoC, the power overhead is also similar. The area of the *split* NoC is 30% larger as compared to the area of the *shared* topology. Most of the increase in area comes from the extra NIs that have to be added.

4.2 Performance analysis for non DRAM flows

In Figure 5 we show the average latency that the non DRAM flows have for the thee designs: *shared*, *split* and *end-to-end flow control*. We present both the write and the read latency. The benchmark uses posted writes so no acknowledgement is sent back. The write latency is measured from the time that the write request is generated until the head flit of the packet reaches the destination.

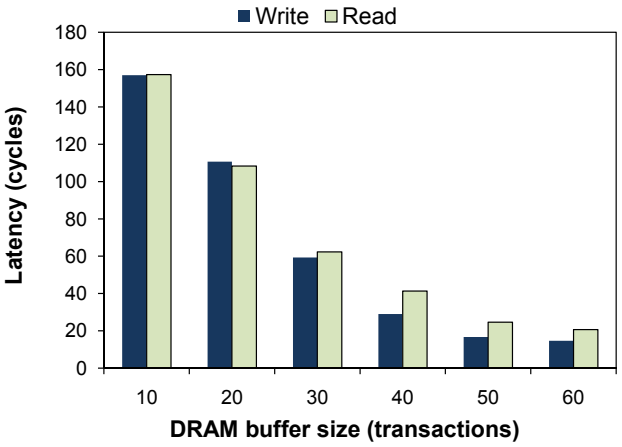

Figure 11: Maximum latency for non DRAM flows in shared NoC

In case of the reads the latency is measured from the time the read request is generate until the head flit of the response packet reaches the requester. From the plot, we can see that the *shared* design has 2.2× larger write latency and a 1.7× larger read latency compared to the *split* or *end-to-end flow control* case. The increase in latency is due to the contention of the non DRAM flows with the DRAM flows when the DRAM backlogged. The traffic to the DRAM is from many initiators to one destination on the request network, and from one source to many destinations on the response network. Therefore the DRAM controller which becomes the bottleneck only creates congestion in the request network. This is the reason why the difference between the write latency is larger than that between the read latencies of the *shared* and *split* designs. The latency for the *split* NoC and *end-to-end flow control* are the same.

In the case of the maximum latency observed for the non DRAM flows we can see a very big difference between the *shared* and *split* network as shown in Figure 6. For applications where the non DRAM flows have real time constraint ensuring that the worst case latency is small is very important for the performance of the system. From the plot we can see that in the case of the shared NoC the DRAM traffic can greatly affect the latency of some packets. In between the *split* topology and the *end-to-end flow control* the results are quite similar, *split* being marginally better.

4.3 Performance analysis for DRAM flows

For the DRAM flows, surprisingly the three solutions perform similarly as the DRAM controller is the bottleneck. In Figure 7 we show a breakup of the average write latency of the dram flows. We show how much time the packets spend in the injection buffer of the NI, how long it takes to traverse the request network, how much time the transactions spend in the DRAM buffer (to be serviced by the DRAM controller before they can be sent back) and the total. Similarly in Figure 8 we present the breakup of the average read latency, where in addition the write we show the time it takes to also traverse the response network. From the experiments we notice that the DRAM controller is the bottleneck, so most time is spent to reach the DRAM controller and for the transaction to be serviced. The latency on the response network is small, as the response network is from one to many and there is no congestion. The *end-to-end flow control* solution performs slightly better that the *split* and *shared*, because it requires a larger DRAM buffer so in turn the DRAM controller itself performs better when services

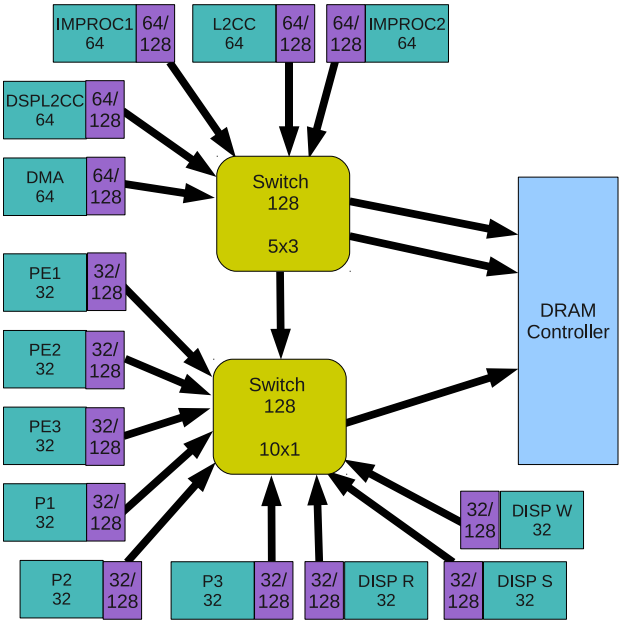

Figure 12: Original topology

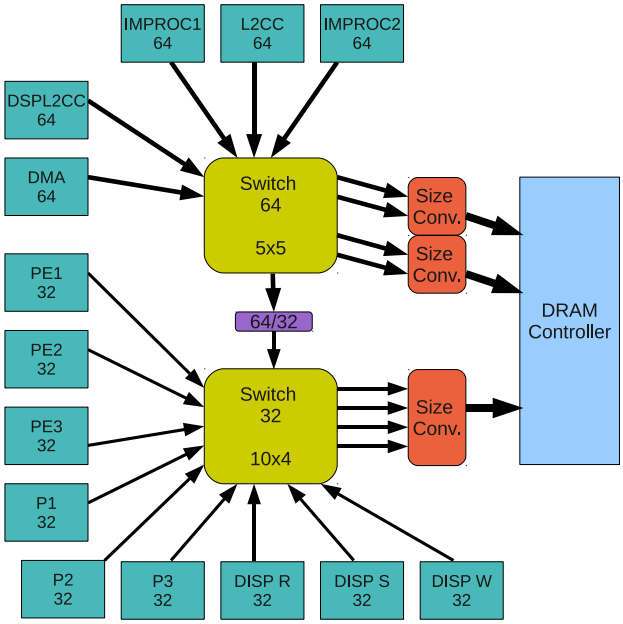

Figure 13: Optimized topology

the transactions. In Figure 9 we also show the maximum latency of the DRAM flows, for both write and read.

4.4 Observations

From the experiments presented before, we can conclude the following: i) the *shared* network preforms poorly for non DRAM flows compared to *split* and *end-to-end flow control*; ii) all designs have comparable performance for the DRAM flows; iii) the *split* and *end-to-end flow control* designs have similar power overhead. Our experiments show that a *split* network can be an efficient solution as it can be implemented using existing NoC components, as opposed to *end-to-end flow control* that would require specialized hardware.

4.5 DRAM buffer size impact on non DRAM flows

We ran experiments with different buffer sizes on the DRAM controller. From the experiments we make the following observations. First having a large buffer does not make a significant difference on the latency of the DRAM flows. This is because packets are waiting in the DRAM buffer instead of the injection queue or the request NoC. Second in for the *shared* NoC we noticed that with large buffering it can achieve similar performance as the *split* network. In Figure 10, we show the average latency of the non DRAM flows for different sizes of the DRAM buffer and in Figure 11, the maximum latency. However with the extra buffering the *shared* design would consume similar or more power than the *split* NoC.

4.6 Port count

For the previous experiments we used three ports for the DRAM controller. However having different number of ports can influence the bandwidth that is transferred to the DRAM buffer. To see the impact, we ran experiments for different number of ports. In Table 1 we show how the latency is influenced by the number of ports for the *split* network. Because the DRAM memory is double data rate and the port data size is the same as the size of the bus to the DRAM

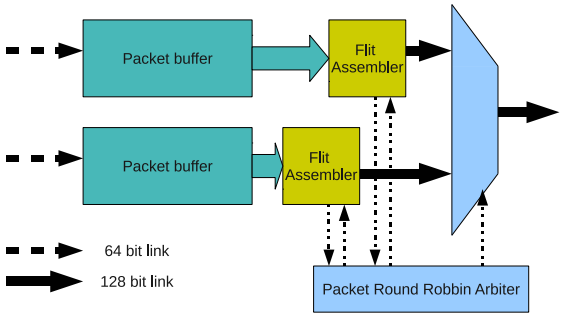

Figure 14: Data size converter architecture

memory, two ports are needed to support the peak bandwidth of the DRAM controller. Therefore there is a larger difference in the latency between one and two ports. When more ports are added the reduction in latency is less significant. Multiple ports are costly as the buffer is implemented as a multiported memory. Therefore an alternative is to increase the data size of the port. In the next section we show how the network to DRAM can be optimized when the cores and the DRAM controller have different data sizes.

5. NETWORK OPTIMIZATION FOR HETEROGENEOUS CORES

In many SoCs, the cores are heterogeneous and they have different data sizes. Also the port of the DRAM controller could be wider than that of the cores. If the network is designed with a single flit size, it needs to be at the size of the widest, core usually the DRAM controller. In Figure 12, we show one topology with uniform flit size for our benchmark. The cores and the switches are annotated with the data size. Some cores are 32 bit, some are 64 bit and the DRAM controller uses 128 bit wide data. In this case, the size conversion is done in the NI that connects the core to the switch. However, since the conversion is done from narrow to wide,

Number of ports	Read latency (cycles)	
	Average	Maximum
1	83	579
2	67.6	448
3	67.3	423
4	61	340

Table 1: Average and maximum read latency from DRAM for different number of ports

to inject at full bandwidth the whole packet needs to be buffered in the NI before it can be sent out. In this case the size converters are at the endpoints. This requires a large number of converters and leads to wider switches. To reduce the number of converters they need to be moved after the switches. In the next section we show a converter architecture that could be placed after the switches and be able to work at full bandwidth.

5.1 Proposed size converter architecture

In order to be able to push at full bandwidth on wide link from a switch with narrow data width we propose a new converter design as showed in Figure 14 for 64 bit to 128 bit conversion. Two links from the switch feed the size converter, and the packets from each link are buffered separately. This enables the converter to buffer a packet from one input link while sending data to the output link from the other buffer. The arbitration between the two input channel is done at packet level in order to prevent flits from different packets to be interleaved, in a round robin fashion. By buffering packets from two input links the converter can push data out on the wider output link at full bandwidth under high traffic load. Similarly the converter between a 32 bit switch and the DRAM can be build, only that in this case the converter would need to buffer packets from four input links in order to be able to send data at full bandwidth to the DRAM controller. The communication flows going to the same port of the DRAM controller could be statically assigned to use one of the input links of the converter. An alternative would be to modify the allocator of the switch to send the packet of any of the flows onto one of the links going to the same converter, but where the buffer is free.

5.2 Optimized NoC

To reduce the number of required packet buffers and size converters we show in Figure 13 an example of how the topology could be designed differently. In the new case cores are clustered and connected to switches based on their data bus size. Therefore the switches can have a flit size closer to that of the cores. This removes the need for size conversion between the core and the switch. Also switches can have different sizes. Existing solutions do not consider the flow data conversion efficiency.

In the case of our new topology only 8 packet buffers are needed, two in each 64 bit to 128 bit converters and four in the 32 bit to 128 bit converter. The size converter between the 64 bit switch and the 32 bit switch is a wide to narrow converter, that in out benchmark is only used by one flow, so it does not need to buffer the packet. In this new design we reduce the number of packet buffers by 42% which leads to an overall NoC power reduction of 23%.

6. CONCLUSIONS & FUTURE WORK

In this paper we presented the power and performance tradeoffs for the NoC design in the presence of a bottleneck core like the DRAM controller. We analyzed three designs and showed that physically separating the DRAM traffic from the non DRAM traffic leads to much lower latency for the non DRAM flows. We also showed that the power overhead of physically splitting the DRAM and non DRAM traffic is similar to other solutions like *end-to-end flow control*, which makes for the use of a *split* network to the DRAM controller an efficient solution. In the presence of heterogeneous cores that have different data sizes we show how the DRAM network can be optimized to reduce the number of data size converter. We also propose a new architecture for the data size converter so that it can transfer data at full bandwidth. The optimized network leads to power savings of 23%. As future work we want to develop a tool for automatic synthesis for DRAM networks in heterogeneous SoCs.

7. ACKNOWLEDGEMENT

We would like to acknowledge the financial contribution of the European Research Council under the project ERC-2009-AdG-246810 and of the ARTIST-DESIGN Network of Excellence. This work has also been supported by the project *NaNoC* (project label 248972) which is [partly] funded by the European Commission within the Research Programme FP7.

8. REFERENCES

[1] L.Benini and G.De Micheli, “Networks on Chips: A New SoC Paradigm”, IEEE Computers, pp. 70-78, Jan. 2002.
[2] G. De Micheli, L. Benini, “Networks on Chips: Technology and Tools”, Morgan Kaufmann, First Edition, July, 2006.
[3] A.Pinto et al., “Efficient Synthesis of Networks on Chip”, ICCD 2003, pp. 146-150, Oct 2003.
[4] W.H.Ho, T.M.Pinkston, “A Methodology for Designing Efficient On-Chip Interconnects on Well-Behaved Communication Patterns”, HPCA, 2003.
[5] T. Ahonen et al. "Topology Optimization for Application Specific Networks on Chip", Proc. SLIP 04.
[6] K. Srinivasan et al., “An Automated Technique for Topology and Route Generation of Application Specific On-Chip Interconnection Networks”, Proc. ICCAD ’05.
[7] A. Hansson et al., “A Unified Approach to Mapping and Routing on a Combined Guaranteed Service and Best-Effort Network-on-Chip Architectures”, Technical Report No: 2005/00340, Philips Research, April 2005.
[8] K. Srinivasan et al., “A low complexity heuristic for design of custom network-on-chip architectures”, Proc. DATE 06, pp. 130-135.
[9] X.Zhu, S.Malik, “A Hierarchical Modeling Framework for On-Chip Communication Architectures”, ICCD 2002, pp. 663-671, Nov 2002.
[10] J. Xu et al., “A design methodology for application-specific networks-on-chip”, ACM TECS, 2006.
[11] S. Murali et al., “Designing Application-Specific Networks on Chips with Floorplan Information”, pp. 355-362, ICCAD 2006.
[12] S. Rixner et al., “Memory Access Scheduling”, ISCA, 2000.
[13] J. Ho Ahn et al., “The Design Space of Data-Parallel Memory Systems”, SC Conference, 2006.
[14] B. Akesson et al., “Predator: A Predictable SDRAM Memory Controller”, CODES+ISSS, 2007.
[15] K. Lee et al., “An efficient quality-aware memory controller for multimedia platform SoC”, Circuits and Systems for Video Technology, 2005.
[16] N. Rafique et. al., “Effective Management of DRAM Bandwidth in Multicore Processors”, PACT, 2007.
[17] D. Wang et al., “DRAMsim: A Memory System Simulator”, ACM SIGARCH Computer Architecture News, 2005.
[18] W. Jang et al., “An SDRAM-Aware Router for Networks-on-Chip”, TCAD, 2010.
[19] J. Yoo et al., “Multiprocessor System-on-Chip Designs with Active Memory Processors for High Memory Efficiency”, DAC, 2009.
[20] I. Walter et al., “Access Regulation to Hot-Modules in Wormhole NoCs”, NOCS, 2007.
[21] D. Kim et al., “A Network Congestion-Aware Memory Controller”, NOCS, 2010.
[22] D. Kim et. al., “Solutions for Real Chip Implementation Issues of NoC and Their Application to Memory-Centric NoC”, NOCS, 2007.
[23] D. Kim et. al, “Implementation of Memory-Centric NoC for 81.6 GOPS Object Recognition Processor”, IEEE Asian Solid-State Circuits Conference, 2007.
[24] S. Stergiou et al., “×pipesLite: a Synthesis Oriented Design Library for Networks on Chips”, pp. 1188-1193, Proc. DATE 2005.