Guaranteed Service Virtual Channel Allocation in NoCs for Run-Time Task Scheduling

Markus Winter and Gerhard P. Fettweis
Technische Universität Dresden, Vodafone Chair for Mobile Communications Systems
Email: winter@ifn.et.tu-dresden.de

***Abstract*—Quality-of-Service becomes a vital requirement in MPSoCs with NoCs. In order to serve them NoCs provide guarantees for latency, jitter and bandwidth by virtual channels. But the allocation of these guaranteed service channels is still an important question. In this paper we present and evaluate different realizations of a central hardware unit which allocates at run-time guaranteed service virtual channels providing QoS in packet-switched NoCs. We evaluate their performance in terms of allocation success, compare it to distributed channel setup techniques for different NoC sizes and traffic scenarios and analyze the required hardware area consumption. We find centralized channel allocation to be very suitable for our run-time task scheduling programming model.**

***Index Terms*—Network-on-Chip, virtual channel, guaranteed service, channel allocation, Quality-of-Service.**

I. Introduction

Packet-switched Networks-on-Chip (NoCs) are considered as promising solution to the interconnection problem in Multi-Processor Systems-on-Chip (MPSoC) [1], [3]. Examples like QNoC [2], AEthereal [4] or Nostrum [13] even support Quality-of-Service (QoS) mechanisms, i.e. assertions about throughput, latency and jitter of data transfers. They are an upcoming necessity for the real-time requirements imposed by processing deadlines of e.g. multimedia and wireless communication applications [4]. The major challenge today is how the application can utilize the provided QoS techniques. How can we find free QoS resources in the NoC and how can we efficiently allocate guaranteed service (GS) channels? The distribution of application entities (jobs, threads or tasks) onto processing elements (PE) must be completed by the establishment of communication channels between the PEs, memories and peripherals in order to achieve system-wide guarantees for real-time applications.

Approaches [6] and [7] try to solve this combined MPSoC programming problem at NoC design and application compile time. However, the data dependent control flow of some applications like H.264 and the possibility of several applications running in parallel on an MPSoC do not allow for efficient static schemes associated with task mapping and channel allocation at compile time. Run-time algorithms in opposite decide about task scheduling and communication channel establishment depending on the current situation in the MPSoC and can exploit the MPSoC inherent parallelism and data locality much better. A central authority schedules jobs and allocates communication resources [5], [9], [11], [14], [15]. The algorithms for solving the scheduling and graph problem typically require a significant amount of computation resources and time. That is why these allocation models work on thread or job level and reassignment of the workload to computational resources is done in the range of milliseconds and seconds. A different and much faster task based approach for scheduling is described in [10]. In this approach, the utilization of a dedicated hardware unit allows for scheduling of tasks onto PEs within about 100 clock cycles while taking real-time constraints of tasks into account, too.

Since real-time constraints have to be considered not only at task scheduling but also at inter-task communication and data transfers, we must extend the approach of [10] to NoCs. Based on this idea, we proposed the concept of centralized NoC management and introduced a dedicated hardware unit called 'NoCManager' which searches and allocates GS routes in the NoC between two arbitrary modules on request [16]. The path search and allocation time depends linearly on the length of the found path introducing uncertainties concerning the allocation time. In this paper we present an architecture which overcomes this limitation. In addition to this we compare the area and the allocation performance of the two NoCManager architectures with GS route allocation via distributed methods for different 2D-mesh NoC sizes and traffic scenarios. This will provide us for the very first time a complete overview on the advantages and disadvantages of the different GS channel allocation approaches.

Section 2 briefly explains the task scheduling concept. Section 3 describes the NoC architecture and section 4 the different GS channel allocation units. Section 5 presents simulation results and section 6 hardware area numbers. Finally, section 7 concludes the paper.

II. Task Based Programming Model

Demand for concurrent execution of several applications on the MPSoC calls for run-time instead of compile-time resource scheduling and allocation. We employ the concept of [10] who proposes an MPSoC programming model with dynamic resource allocation via run-time task scheduling. The application represents a system thread while a task in the sense of this programming model is an atomic computational kernel which can be scheduled and executed on a processing element (DSP, ASIC, ASIP) as soon as its input data is available. A task typically takes some hundreds to few thousand clock

978-3-9810801-7-9/DATE11/©2011 EDAA

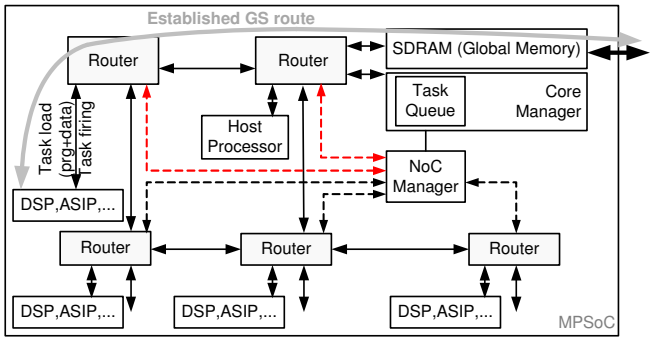

Fig. 1. Schematic Block Diagram of the MPSoC platform. The dotted lines indicate the direct connections between the NoCManager and the routers (red: examplary allocation of these routers by the NoCManager at the moment).

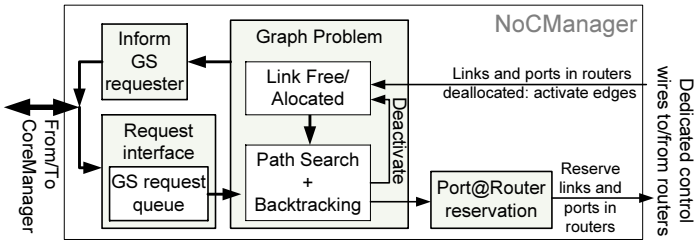

Fig. 2. Block diagram of the NoCManager.

cycles for computation. A RISC host processor executes the operating system and the control code of one or more applications. Within the operating system tasks are instantiated sequentially as parts forming an application. This instantiation is translated into a task description which fills a queue in the CoreManager. The CoreManager is a dedicated hardware unit which schedules tasks onto the available processing elements at run-time while considering the data dependencies between different tasks. It also issues the required data transfers to and from the PE for task input and output data. [10] supports real-time applications by considering deadlines for task execution and allowing cancellation of lower priority tasks. Thus, in an overloaded system tasks will have to be aborted before they even started in order to fulfill the assertions for other tasks and complete applications.

However, considering only the scheduling of tasks is not sufficient for real-time applications since not only the application execution must be in time but also the data transferred across the NoC. Thereby, QoS assertions by GS channels must be allocated within few tens of clock cycles or less. The centralized allocation of communication resources at run-time described in [5], [9], [11], [14], [15] takes too long for this. In order to overcome this limitation we propose the central channel allocation unit, called NoCManager, which is able to allocate the GS channels in the required timing range. As soon as the CoreManager has scheduled a task onto a PE, it instructs the NoCManager to find a GS route in the NoC. If the route search was successful, the NoCManager allocates the found route via dedicated control wires to the routers where the appropriate registers are set, Fig. 1. If no free route could be found the NoCManager informs the CoreManager about this failure and the CoreManager has to abort the real-time task because of a heavily loaded system.

III. Network-on-Chip Architecture

Our NoC is formed of routers and network interfaces (NI) connected to each other by bidirectional links. Every port of a router and an NI consists of an output and an input part. A flit (flow control digit) is transmitted in one clock cycle across a link and consists of a header and a payload. The header contains the target NI address, a burst mode identifier and a flit type identifier. The type identifier differentiates between the two traffic classes (explained in the following), setup, Ack, NAck and tear-down flits.

Similar to [4] there are two kinds of flits: Best Effort (BE) and Guaranteed Service (GS) which form the different traffic types. BE uses deterministic xy-routing and input queuing in the routers. A stall/go protocol realizes flow control at BE traffic between the routers. Round robin arbitration between all input ports is used in case two BE flits want to get access to the same output port in the same clock cycle. A BE flit requires 2 cycles per hop (1 cycle for link traversal into FIFO, 1 cycle for routing and putting out onto link).

GS on the other hand employs the reservation of a virtual channel between two communicating modules. In the routers along the channel the corresponding input and output ports are reserved for GS flits of this channel. As soon as a GS flit arrives at an allocated input port the corresponding output port forwards it. The BE routing is bypassed and forwarding of BE flits to this output port from any input port is stalled. If at an outport a channel is reserved but in a cycle no GS flit wants to be forwarded, the free port will be used for BE flits. Due to the fact that every router in the NoC works the same way, a GS route is constituted between two modules. Note, in opposite to [4] we currently do not support several GS channels per link and port by time multiplexing. We know about wasting communication resources and will extend our principles in future to handle more GS channels along a physical route.

IV. Central Channel Allocation Architectures

There are two types of NoCManagers we want to discuss and compare in this paper. Both have to search and allocate free GS routes between two modules. So, they have to solve a shortest path problem on the NoC graph making path search and backtracking the heart of the channel allocation unit. Each of the two NoCManager types realizes them in different ways, explained in the subsections a) and b). The rest of the NoCManager block diagram is the same for both, Fig. 2.

Since the topology and size of the NoC is known at design time the graph containing the links as edges and the routers as nodes can be realized in hardware in conjunction with a hardware based search algorithm. For this, the graph search requires knowledge about the actual allocation state of the links in the NoC (edges in the graph). This state is stored in the 'link free flipflop' for every link existing in the NoC. An AND gate connects the flipflop with the edge in the NoC

graph representation (see also Fig. 3c) and d)). When an NoC link is allocated, the flipflop of the corresponding edge is set to '0' excluding it from next route search.

Allocation of the routers is done during or directly after path backtracking. Direct connections between the NoCManager and the routers (Fig. 1) allow allocation in few clock cycles without the risk of setup flits being stalled in the NoC speeding up channel allocation significantly. Two wires per port in a router are necessary. One wire indicates that the port is allocated, the other wire goes the way back and indicates to the NoCManager when it is freed again. When data transmission via GS channel is complete one module transmits a tear-down GS flit. As it travels along the GS route it frees the allocated ports in the routers. This deallocation is forwarded by the direct wire connection to the NoCManager which resets the according 'link free flipflop' of that link to '1'. This link can now be used during next route search again.

The NoCManager is completed by a small queue where incoming requests can be buffered and a unit organizing the response to the GS route requester.

A. Sequential HAGAR NoCManager

The NoCManager was initially presented in [16] with a detailed discussion of the architecture but without analysis on the performance of allocation success. The graph realization of the NoC, path search and backtracking are realized by a HArdware Graph ARray (HAGAR) proposed by [8], [12]. Nodes (routers) are represented by simple, unconnected, horizontal and vertical wires. If an edge (a link) between two nodes (A) and (B) exists, an AND-OR-gate will connect the horizontal wire of node (A) with the vertical wire of node (B).

When the NoCManager wants to find a route between two modules it activates the horizontal wire which represents the router connected to one of the two GS requested modules (the GS master). Via the AND-OR-gates all nodes reachable in one hop are activated and detected on the vertical wires. The reached nodes remember the node which activated them for the first time. In the next clock cycle all nodes which were activated on the vertical wires are set to logic '1' on their horizontal wires. They activate other nodes on the vertical wires again. This goes on until the intended end node is reached. At the end, the path is backtracked stepwise by sorting out the predecessors starting at the end node.

Finding a route in this sequential HAGAR takes as many cycles as the found route has hops. Path backtracking requires as many cycles, too. In an 8x8 mesh a typical route from the upper left to the lower right corner is about 14 hops, thus, route search and backtracking require 28 clock cycles. Taking into account a few more cycles for setup request receive, router allocation and send of answer we end up at about 35 cycles for allocation of a route from the upper left to the lower right corner in an 8x8 mesh NoC. Though this is quite short, the indeterminism in the allocation duration is undesirable. Additionally, at large NoCs with high and very dynamic load (i.e. many GS routes with short life time) sequential HAGAR NoCManager shows performance degradation. The reason for

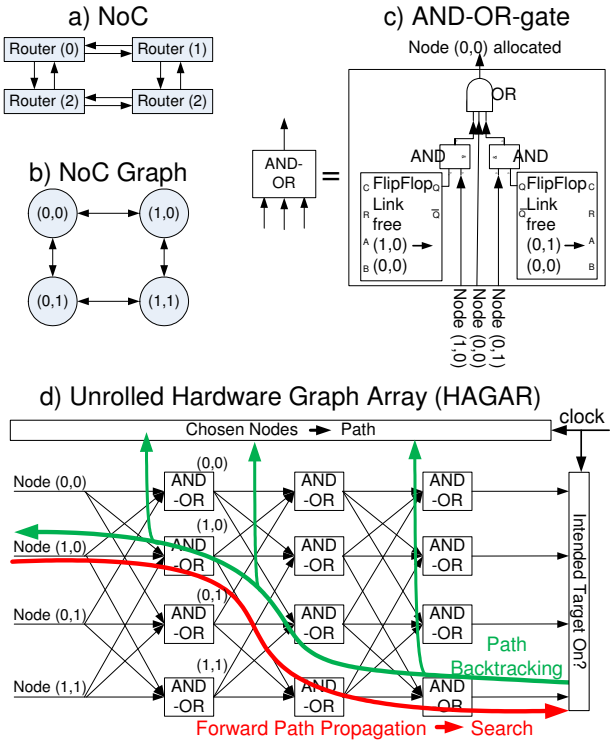

Fig. 3. a) the 2x2 2D-mesh example NoC; b) the resulting NoC graph; c) detailed structure of an example AND-OR-gate incorporating the 'link free' flipflop storing the actual allocation state of a specific link in the system; d) schematic structure of the serialized/combinatorial graph matrix for the example NoC

this is the direct rejection of incoming requests with a NAck (route could not be setup) although there might be free routes because it has already requests to handle and the request queue is full. Shortening the search time can improve the allocation success significantly what is one scope of this paper.

B. Combinatorial HAGAR NoCManager

The sequential HAGAR architecture achieves clock frequencies of more than 1 GHz in UMC 130nm technology since there are only very few AND-OR-gates connected in a row forming the critical path. We expect MPSoC systems for signal processing in 130nm to run only at about 200-300MHz. This reveals much laxity and optimization potential. We can lengthen the critical path and in return save clock cycles by loop unrolling. Instead of reusing the hardware representation of the NoC graph in every cycle for every path step we can connect several of these graphs in series as combinatorial logic in a trellis structure, Fig. 3d). Similar to the sequential HAGAR we have the two different phases of path search and path backtracking except that each of them consumes now 1 clock cycle independent of the path length.

We begin at the start node of the intended GS route by setting it to logic '1'. The signal travels along the connected wires to the first stage of AND-OR-gates. The AND-OR-gates in Fig. 3c) incorporate every link arriving at this node by AND-connecting the 'link free' flipflop to the wire coming

from the neighbor (predecessor) node. So, already allocated links with a 'link free' flipflop set to logic '0' are not considered and not selected by the path search. It passes the first AND-OR-gate stage and activates its reachable neighbor nodes. They, in turn activate their neighbors in the next stage of AND-OR-gates. Each node at each stage stores by which of the activation wires at its AND-OR-gate input it was activated. If the node was already active in the previous stage, it will store itself as the node it was activated by. So, we can make sure to select exactly that path in the NoC which is the shortest between the GS source and target module.

This goes on until the end of the trellis where a register stores which of the nodes could be reached through the NoC. If the intended GS target was activated a GS route through the NoC will exist and backtracking is started in the next cycle. By reading the stored preceding input wire starting at the intended GS target, the path is backtracked and selected. Each selected node at each stage updates that it was selected into the 'chosen node' register where eventually the complete path from source to target can be found.

There must be as many stages of AND-OR-gates as the longest minimal path has hops. At 8x8 mesh there are at least 14 hops. So, the trellis requires at least 14 stages of AND-OR-gates. There can be realized more than 14 stages allowing non-minimal paths for the longest distance in the NoC, too (for shorter distances between GS source and target non-minimal paths are possible anyway). But our simulations proved, for 2D-mesh topologies additional stages do not improve the performance. There is enough diversity in the mesh.

Note, though in this paper we restrict to 2D-mesh topologies at our analysis the HAGAR architecture (sequential and combinatorial) can be realized for any kind of topology including a completely irregular one.

We also researched alternative realizations of combinatorial, trellis and tree based NoCManagers with exactly one starting point. But compared to the combinatorial HAGAR it was much more area hungry without better performance results.

V. Simulation Results

In opposite to [16], in this paper we want to provide performance comparisons between the two NoCManagers and distributed allocation techniques. At distributed GS channel allocation, the intended GS master (in Fig. 1 the DSP/ASIP when it was allocated by the CoreManager) sends a setup flit into the NoC where the flit itself searches its path through the NoC and allocates the routers along this path via a forward-and-allocate-algorithm. We realized xy-routing, flooding and flooding-minimal-path (flooding but forwarding of setup flits is only allowed if this shortens the distance to the target). The request queue of the NoCManager was set to two entries because more entries did not show significant performance improvements in our simulations.

For evaluation we considered the following parameters:

- *GS Master Percentage*: The percentage of modules in the MPSoC which can be GS master. Set to 20%, 35% and 50%. They are distributed uniformly random in the system. All other modules (except CoreManager and NoCManager) can be GS slaves.
- *GS Channel Lifetime*: The number of clock cycles after which a GS route will be torn down by the GS master. The shorter the lifetime the higher is the dynamism of GS route allocation and deallocation at a specific route rate. We simulated 200, 500 and 1000 cycles.
- *GS Setup Success Rate*: The ratio between established and desired GS routes. Indicates how many of the requested GS routes could be established in the NoC.
- *GS Route Rate*: The portion of clock cycles in which a GS master wants an active GS route. It indicates the relation of requested GS active to GS not active cycles at a GS master and the requested GS connection load in the NoC. It is computed per GS master as:
$$GS\ route\ rate = \frac{\#(GS\ routes\ wanted)\ \cdot\ (GS\ lifetime)}{\#(simulation\ cycles)}$$

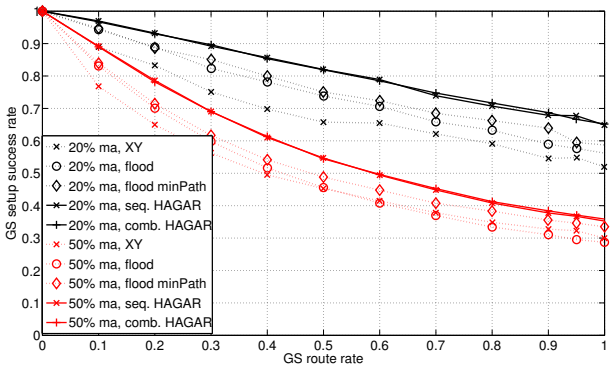

Fig. 4. Comparison of the different GS channel allocation techniques for a route lifetime of 200 clock cycles at 6x6 mesh and for a GS master percentage in the system of 20% (black) and 50% (red).

In our simulations we used 4x4, 6x6, 8x8, 12x12 and 16x16 2D-mesh-grid based NoC. We simulated 50.000.000 clock cycles for analysis of the setup success rate resulting in several ten thousands to millions of GS route setups. The CoreManager issues a GS setup request to the modules (distributed GS setup) or the NoCManager (centralized GS setup) in a POISSON stream meeting the GS route rate requirements of the GS masters. The first and last 100,000 simulation cycles were not considered in order to prevent transient effects. Due to lack of space we will only show the results for a small NoC (6x6 mesh) and those for a large NoC (16x16 mesh). But the principal insights are also true for the NoC sizes between.

Fig. 4 and Fig. 5 show the allocation success of GS route requests for pretty short GS channels with a lifetime of only 200 clock cycles in the small and the large NoC, respectively. We also varied the number of masters in the SoC. At 6x6 mesh there are 7 (20%) and 17 (50%) of the 36 modules in the SoC master, at 16x16 there are 51 (20%) and 127 (50%) masters, respectively. At the 6x6 NoC the NoCManager based allocation techniques achieve the best allocation performance. They can realize 5% to 10% more allocation success than

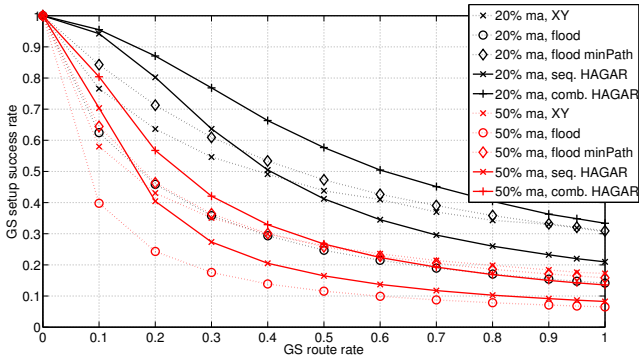

Fig. 5. Comparison of the different GS channel allocation techniques for a route lifetime of 200 clock cycles at 16x16 mesh and for a GS master percentage in the system of 20% (black) and 50% (red).

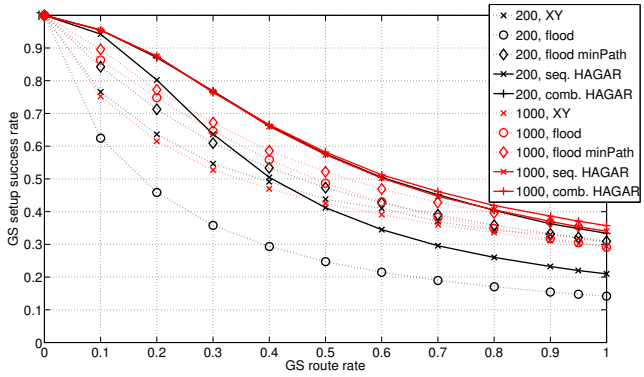

Fig. 6. Comparison of the different GS channel allocation techniques for a route lifetime of 200 (black) and 1000 (red) clock cycles at 16x16 mesh and for a GS master percentage in the system of 20%.

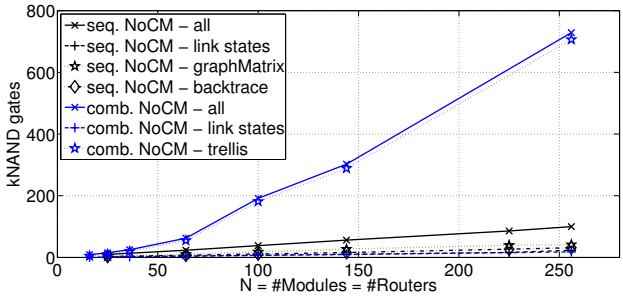

Fig. 7. Area consumption of the two different NoCManagers and there main parts (in kNAND gates).

the distributed methods - at low and high load. We also find, the combinatorial NoCManager does not perform better than the sequential one. It cannot make use of its low allocation latency advantage and cannot turn it into better allocation success. At this small NoC, there are simply to few requests to the NoCManager and the search of a route via sequential NoCManager does not take so many clock cycles due to shorter paths. Therefore, only few requests arriving at the NoCManager must be rejected immediately because of a full request queue although there might be a free path for this route request. The distributed techniques lack of global knowledge of the allocation state in the NoC and cannot utilize possible free routes. Though, the flooding algorithm has this knowledge it obstructs itself. If two different routes shall be setup at the same time the setup flits can block each other making both setup tries to fail although there are free routes. Though, the combinatorial HAGAR achieves no better allocation success than the sequential one, the determinism of the GS route allocation latency can still be a big advantage for the CoreManager.

At the 16x16 NoC, Fig. 5, there is a significant difference between the sequential and combinatorial NoCManager. At such a large NoC with many masters and long possible paths the combinatorial NoCManager can impressively utilize its advantage. It achieves up to 16% more allocation success than the sequential NoCManager which has to reject incoming requests in a high number because of an already full request queue due to long path search and allocation times per request. The shorter search latency of the combinatorial NoCManager enhances the number of requests it can handle significantly providing the required throughput at large NoCs with heavy load. The flooding algorithm performs even worse since its setup flits have to travel through the whole 16x16 NoC. Only at an extremely loaded and dynamic system (50% master, short GS routes of 200 clock cycles and high route rates) xy-based channel allocation and minimal path flooding can realize allocation successes of 2-3% more than the combinatorial NoCManager. But even at such an improbable operation state, the advantage of the distributed methods is minimal.

Fig. 6 shows the results of a highly dynamic system on the one hand and a bit less dynamic on the other hand. At a GS route lifetime of 1000 clock cycles there is less often a new route to be setup than at 200 clock cycles for the same route rate (5x less). There are less requests to the NoCManager and the sequential one is sufficient to handle all these incoming requests even in this large NoC with high route rates.

VI. Area Results

Both NoCManagers as well as the routers are available in synthesizable VerilogHDL and can be generated out of an XML description for different NoC topologies and sizes. Synopsys Design Compiler synthesized them with FARADAY's 130 nm UMC library. While the routers as well as the sequential NoCManager can be realized at 500 MHz (seq. NoCManager even more than 1 GHz) the combinatorial NoCManager allows only 200 MHz until an 8x8 mesh and 50 MHz for a 16x16 2D-mesh NoC topology.

The area requirements of the combinatorial HAGAR are significantly higher than those of the sequential HAGAR. The sequential version has only one representation of the NoC graph, the combinatorial has as many graph representations connected in a chain as the longest path can be. The sequential HAGAR grows with $O(N)$ where the combinatorial HAGAR grows with $O(N \cdot \sqrt[2]{N})$ in a 2D-mesh (N = number of modules), Fig. 7. We also see, the trellis graph of the combinatorial

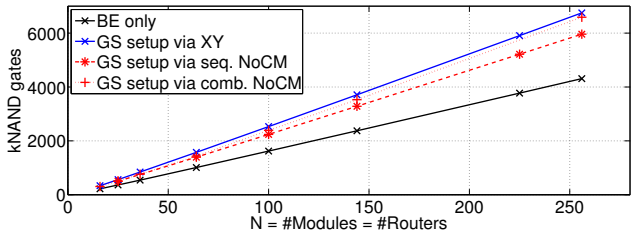

Fig. 8. Area consumption of the whole NoC including FIFOs and the GS route allocation mechanism (in kNAND gates).

HAGAR is responsible for this huge growth where the 'link free' registers require about the same area at the two NoCManager realizations. We find, too, the linear growth of the sequential NoCManager as expected. Compared to a microcontroller which would require hundreds of cycles for route search, the sequential NoCManager area consumption is quite acceptable even at a 16x16 mesh consuming 99 $kNANDs$ and not requiring any SRAM.

In order to compare the centralized NoCManager approach for GS route allocation with the distributed ones, we have to compare the area consumption of the complete NoC. Fig. 8 shows the area a pure Best Effort NoC consumes with its routers including 4 entry input FIFOs for flits of a payload size of 68 bit (32 bit for data, 32 bit for address, 4 bit for read/write/cache/... mode). We find also the NoC area consumption for the simplest distributed channel allocation method - allocation along the xy-routing - and for the NoCManager based allocation where not only the routers but also the NoCManagers are incorporated in the area. We find the xy-routing allocation requires more area than both NoCManager approaches. This might be a surprise but a little overhead in every router and port for xy-allocation multiplied with the large number of routers and ports in the NoC makes up for this difference. So, the NoCManager approaches for GS channel allocation achieve not only much better allocation performance but can save area, too.

VII. Conclusion

In this paper we presented the architecture and evaluated for the very first time the performance and area consumption of different approaches of centralized and distributed GS channel allocation in NoCs at run-time. We compared them in terms of allocation success at different NoC sizes and different traffic scenarios from heavily loaded and highly dynamic to light load with less dynamism and took the area consumption into account, too.

We found three basic reasons for the central NoCManager approach being superior to the distributed techniques:

- The NoCManager fits very well to the central task scheduling CoreManager as an additional unit making real-time data transfers in NoCs for run-time task scheduling possible.
- The GS allocation performance of the NoCManager (sequential or combinatorial) is nearly always superior to the distributed techniques.
- The area consumption of an NoCManager NoC is significantly lower than at a distributed allocation NoC.

We examined the two extremes of the sequential and combinatorial HAGAR NoCManager. We found, area of the sequential NoCManager scales linearly with the NoC size but its allocation performance scales poorly with NoC size. On the other hand, the combinatorial NoCManager scales very well with growing NoC sizes even at highly dynamic traffic but this is bought with limited scaling possibilities concerning the area consumption. The combination of the sequential and the combinatorial NoCManager can provide a suitable trade-off between area and performance scaling at growing NoC sizes. E.g., the combinatorial HAGAR can be broken down to 7 AND-OR stages instead of 14 (at 8x8 mesh) by running at most two times through the graph representation. Area would be reduced by half and the number of cycles for path search would be set from 2 to only 4 clock cycles. Additionally, we expect clustering of large NoCs which can be exploited at the (completely topology independent) NoCManagers, too.

References

[1] L. Benini and G. Micheli. Networks on Chips: A New SoC Paradigm. *Computer*, 45(1):70–78, January 2002.

[2] E. Bolotin and et al. QNoC: QoS Architecture and Design Process for Network-on-Chip. *Systems Architecture*, 50:105–128, 2004.

[3] W. Dally and B. Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. In *Proc. of DAC*, pages 84–89, June 2001.

[4] K. Goossens, J. Dielissen, and A. Radulescu. AEthereal Network on Chip: Concepts, Architectures and Implementations. *IEEE Design and Test of Computers*, 22(5):21–31, September-October 2005.

[5] A. Hansson and K. Goossens. Trade-offs in the Configuration of a Network-on-Chip for Multiple Use-Cases. In *Proc. of NOCS*, pages 233–242, May 2007.

[6] A. Hansson, K. Goossens, and A. Radulescu. A Unified Approach to Constrained Mapping and Routing on Network-on-Chip Architectures. In *Proc. of 3rd Int. Conf. on HW/SW Codesign and System Synthesis*, pages 75–80, 2005.

[7] J. Hu and R. Marculescu. Energy-Aware Communication and Task Scheduling for Network-on-Chip Architectures under Real-Time Constraints. In *Proc. of DATE*, pages 234–239, February 2004.

[8] L. Huelsbergen. A Representation for Dynamic Graphs in Reconfigurable Hardware and its Application to Fundamental Graph Algorithms. In *Proc. ACM/SIGDA 8th Int. Symp. on FPGAs*, pages 105 – 115, 2000.

[9] N. Kavaldijev and et al. Providing QoS Guarantees in a NoC by Virtual Channel Reservation. In *Proc. of Int. Workshop on Applied and Reconfigurable Computing (ARC)*, March 2006.

[10] T. Limberg, B. Ristau, and G. Fettweis. Real-Time Programming Model for Heterogeneous MPSoCs. In *Proc of SAMOS*, pages 75–84, July 2008.

[11] T. Marescaux and et al. Dynamic Time-Slot Allocation for QoS Enabled Networks on Chip. In *Proc. of Workshop on Embedded Systems for Real-Time Multimedia*, pages 47–52, September 2005.

[12] O. Mencer, Z. Huang, and L. Huelsbergen. HAGAR: Efficient Multi-Context Graph Processors. In *Proc. 12th Int. Conf. on Field-Programmable Logic and Applications*, pages 915 – 924, 2002.

[13] M. Millberg and et al. Guaranteed Bandwidth using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip. In *Proc. of DATE*, pages 890–895, February 2004.

[14] O. Moreira, J.-D. Mol, and M. Bekooij. Online Resource Management in a Multiprocessor with a Network-on-Chip. In *Proc. of ACM Symposium on Applied Computing*, pages 1557–1564, 2007.

[15] V. Nollet and et al. Centralized Run-Time Resource Managment in a Network-on-Chip Containing Reconfigurable Hardware Tiles. In *Proc. of DATE*, pages 234–239, 2005.

[16] M. Winter and G. Fettweis. A Network-on-Chip Channel Allocator for Run-Time Task Scheduling in Multi-Processor System-on-Chips. In *Proc. of 11th Euromicro Conference on Digital System Design (DSD)*, pages 133–140, September 2008.