Multipath routing in TDM NoCs

***Abstract*—The exponential increase in transistor count due to technological progress has resulted in an increase in complexity and processing power of on-chip elements. Recently a stage has been reached where it is not practical anymore to increase the core size, in either the traditional CPU or embedded domain and as a consequence the number of cores, processing elements or peripherals is being increased instead. Providing an efficient interconnect for the increasing number of elements is an important area of research. In this study we focus on improving the efficiency of a network using alternative routing strategies. We focus on a multi-path slot allocation method in networks with static resource reservations, in particular TDM NoCs. The simplicity of these networks makes it possible to implement this routing scheme without significant hardware overhead. Our proposed method, although displaying large variations between test cases, provides significant overall gains in terms of allocated bandwidth, with an average gain across all tests of 29% against an exhaustive search of single-path routes, and even larger gains of 47% when compared to other single-path routing algorithms.**

I. Introduction

In systems of increasing complexity, modularization is an essential procedure for allowing timely development. Modularization however relies on the ability of interconnecting the modules into complete systems. Networks on chip [1], [2] are an emerging paradigm for on-chip communication which simplifies the design of large systems and provides a scalable solution for connecting the on chip processing elements, memories and peripherals. As the interconnect has an increasing influence on the overall performance of the design, optimizing the networks on chip is an important direction of research.

The amount of allocated resources obviously has a direct influence on the achieved performance, however it is also of major importance how efficiently the available resource are used, as the simple increase in path width or router count also has the negative effect of increasing the area. From large-scale packet-switched networks we have learned that a major limit on efficiency is represented by congestion, while in circuit-switched networks the limitation consists of blocking during allocation and underutilizing the allocated channels.

We perform our study using the NoCs that employ a time-division-multiplexing scheme for allocating bandwidth for connections, in particular the Æthereal. To limit the employed hardware resources, minimal buffering is provided, circuit-switched network flits having to depart from a router in the immediate next slot after arrival. This results in low end-to-end latencies, but requires an efficient slot allocation scheme for allocating time slots over the TDM shared links.

In this paper we propose a multi-path routing technique that performs slot allocation, at the same time ensuring in-order delivery of packets. The method is used to improve utilization by taking advantage of the additional routing freedom provided by the multi-path approach, and is based on established algorithms known to be optimal in several respects. We show that, in some cases, a multi-path allocation scheme may offer significantly increased bandwidth over classical single-path approach when used on top of the same underlying network. In order to minimize the cost and reduce the impact of our method on the already existing hardware implementation in Æthereal we use the static time slot allocation scheme to ensure in-order packet delivery.

The rest of the paper is organized as follows. In the following section we present related work, and usage of multi-path routing in other domains than networks-on-chip. In section III we give a description of our own proposed algorithms, followed by an analysis of required hardware resources in section IV. Experimental results are presented in section V while the last section presents our conclusions.

II. Related Work

A sizable amount of research has been invested in optimizing routing algorithms in both large scale and on-chip networks, of which a significant portion was dedicated to deadlock-free routing [3], [4], [5], [6].

In large scale networks, multi-path routing has already been in used for a long time, for example in Internet traffic engineering [7] The problem presents other challenges though than the small-scale networks found on-chip. Buffering is in general plentiful by comparison and in-order delivery is not a requirement, because the protocol stack implements reordering in another layer, before the data is delivered to the user.

The problem of multi-path routing in networks with resource reservation ie. asynchronous transfer mode or ATM was studied by Cidon et al. [8], [9], and was shown to provide a benefit in terms of connection establishing time, while having mixed results from the bandwidth point of view.

Multi-path routing in NoCs has been previously proposed in [10], because of the routing mechanism employed however, a more complex approach is required in order to ensure in-order delivery, consisting of arbitration at the point where different paths converge and additional sequence ids for packets. In contrast our solution uses pre-computation at design time, thus minimizing the additional hardware requirements.

Furthermore, the multi-path routing is found in the various forms of adaptive routing or other forms of non-deterministic routing [11], largely addressed by studies of multiprocessors and now also applied to networks-on-chip [12], [13], [14], [15], [16]. The target of these studies is mainly guaranteeing deadlock freedom while maximizing utilization, but without explicitly addressing the costs of in-order delivery.

Our study targets NoCs that support resource reservation using Time-Division-Multiplexing, in particular the Æthereal network [17]. Our algorithm performs both routing and slot allocation.

The same technique can also be applied to other TDM networks described in the literature, like the Nostrum network [18], aSoC [19] and the TDM test delivery in [20] and perhaps also to networks using space division circuit switching like [21].

The authors of [22] propose a graph coloring algorithm to solve a slot allocation problem in a similar network implementation but with more relaxed constraints regarding timing.

A solution for performing mapping, routing, slot allocation in the Æthereal networks, currently in used by the tool flow was presented in [23].

III. Proposed Algorithms

We propose and demonstrate a multi-path routing technique within the framework of Æthereal [17]. The Æthereal network offers a combination of packet and circuit switching mechanisms in the form of best effort and guaranteed throughput services. As multi-path routing for packet switched NoCs has already been addressed in the literature [10] our focus is on the circuit-switching like guaranteed services.

In guaranteed services mode, the user requests dedicated channels between two IPs connected to the network. A number of TDM time slots are allocated within a global, periodically repeating slot table, based on the requested bandwidth. The connections are generally long-lived, possibly for the entire lifetime of the application. For cost reasons, consecutive time slots have to be allocated for consecutive links on the path, as buffering is provided in the routers for one flit cycle only, although extensions of this basic scheme are possible. This particular requirement makes the slot allocation task more challenging and is specifically addressed by our study.

A. Maximal set of slots

We show that connections of larger bandwidth can be established over a network with preexisting traffic using the multi-path approach when compared to the standard single-path method. For GS traffic, the allocated bandwidth is directly related to the number of time slots that have been allocated to each connections, thus conferring the problem a discrete nature.

The problem of finding a set of paths of maximum capacity between two nodes in a graph has an established solution in the Maximal Flow Algorithm [24]. We use a variation of the flow algorithm, which ensure that the solution found is of minimal cost with respect to any set of positive cost values attached to the edges. By defining a constant cost of 1 for each edge, we minimize the number of hops taken by packets in order to traverse the network.

As mentioned before there are restrictions in the way the slots can be allocated on a given path. Each GS flit arriving at a network router will have to depart in the immediately following time slot, as no buffering is provided to store more than one flit. To adapt the algorithm to this additional requirement, we represent the network as a directed graph when each network node (router or network interface) is split into s nodes, where s is the number of time slots. Note that the graph is split only from the logical point of view, and it is not necessary to store the connections of a graph that is s times larger as they can be generated dynamically using simple rules. The generated nodes are connected in a way which reflects the delay of traversing a network link and crossbar, as presented in Figure 1. From the link from router A to router B, a flit departing from slot A_i will arrive to B in slot $B_{(i+1) \text{mod}_s}$.

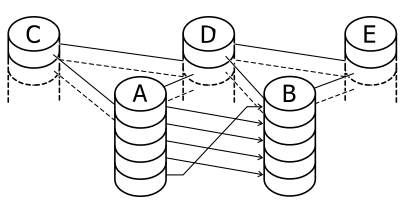

Fig. 1. Splitting nodes in a graph, $s = 5$

In the flow algorithm, the edges of a graph have associated capacities and can receive a flow, which denotes the movement of some commodity along that edge, which has to be bounded by that capacity. The incoming and outgoing flow at each node must balance out at each node, in a way similar to Kirchhoff's Current Law, except for nodes marked as source and destination.

Commonly, flow algorithm start with a graph with zero flow, and then iteratively increase the flow between source and destination, along augmenting paths which are determined based on the edge capacity which is still not utilized. If the augmenting paths are always chosen so that they have minimal cost, then the total flow is also guaranteed to have minimal cost [25]. As the already allocated flow increases and links become saturated, augmenting paths steadily increase in cost, until no further increase in flow is possible.

In general, any path finding algorithm can be used for determining an augmenting path, with one observation: the flow algorithm must be able to displace previously allocated flows to make room for new ones. For this, in addition to normal edges that are not yet saturated the path-finding algorithm is allowed to traverse links used by previously found paths in the reverse direction, thus canceling out the flow on that segment. The cost of such traversal is negative, as the segment will be removed both from the previous and the current path.

An example is presented in figure 2. Assume that in the first step of the algorithm, a flow was allocated on the $A-B-E-F$ path. Clearly such a choice is unfortunate because it blocks all other paths from A to F. A new path will then be found in the form $A-D-(E-B)-C-F$. The common part (the $B-E$ segment) will be erased from both solutions and the first two segments of the second augmenting path will be combined with the last one of the first augmenting path and vice versa.

In the same example it can be seen that the first path contained three segments while the augmenting path contained five, but because one of these segments is has a negative cost, the total cost after the addition of the new path is $3 + (4 - 1) = 6$. The negative edge cost resulting from erasing edges makes the path finding algorithm more difficult, but in [25] a solution is presented solving the problem without an increase in complexity.

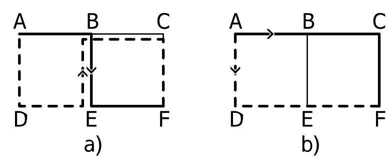

Fig. 2. Flow displacement, a) during search for augmenting path b) after the addition of the new path

There is one characteristic of the Æthereal networks which further complicates the problem. Flits belonging to the same connection, traveling in consecutive time slots along the same path do not need to repeat the routing header, and as a result the useful payload to packet size ratio is improved. In order to deal with this additional optimization criterion we employ a heuristic: once an augmenting path is found by the flow algorithm, we try to allocate more slots along the same route, if of course the new slots are available. This modification does not interfere with the proper functioning of the flow algorithm, because there is no restriction on the path finding algorithm we employ, and the allocated paths are of the same cost as the last path that was produced.

B. In-order delivery

A common problem of routing algorithms which offer several alternative paths to the destination is that it creates a possibility for packets to arrive in another order than they were transmitted. This represents a significant problem in NoCs, because the cost of buffering necessary for reordering the packets might be prohibitive on-chip. The guaranteed services offered by the Æthereal networks present an opportunity here because TDM slots are statically reserved and the sequence of arrivals can be controlled at allocation time.

The result produced by the flow algorithm is not guaranteed to contain only routes delivering the packets in the right order. The different delays of the shorter and longer routes may result in packet reordering, as illustrated in figure 3. Our solution is to discard some of the routes, so that all remaining ones provide in-order delivery. It should be noted though that a difference in path length of at least two hops is necessary for this phenomenon to take place, which implies that as long as the routing is minimal or near-minimal no flows will be discarded, and the solution will be optimal with respect to the allocated number of slots. We present here an algorithm for the selection of routes to be discarded.

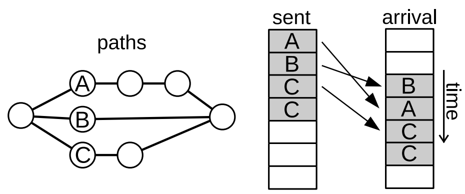

Fig. 3. Out-of-order delivery

The problem can be formulated as a Monotonic Subsequence Problem [26], for which optimal solutions exist with polynomial time complexity. The paths are ordered by slot departure time and the solution must comprise of a subsequence with only increasing arrival times. Several modifications are again needed in order to take into account the particularities of our problem. Because consecutive slots have a different payload efficiency, the items in the sequence need to be weighted, and, the algorithm needs to take into account the wrap-around that occurs at the end of the slot table. In some respects the wrap-around is similar to the problem described in [27]. The first requirement does not introduce significant changes into the algorithm, but for the second, the algorithm will have to be applied repeatedly in a window which slides over the set of paths.

The algorithm is based on dynamic programming, and consists of generating partial solutions that are optimal with respect to our optimization criteria, and sufficient for ensuring the optimality of larger solutions. A formal description of the algorithm is given in Algorithm 1. The selection method is optimal in that it delivers the largest bandwidth possible for the given set of paths, however that should not be interpreted as the combination of flow and discarding algorithms being optimal together.

C. Algorithm complexity

In asymptotic notation (”big O” notation) [28], the multi-path approach has a higher complexity than the single path one. Intuitively, since multi-path implies that several paths have to be found, the solution is expected to be several times slower. In practice, the performance depends on which single path algorithm our method is compared to. As the single-path version may use different heuristics, their complexity may vary.

Let’s consider a simple implementation of the Dijkstra algorithm [29] for the single path solution. A reasonable implementation using priority queues for maintaining a sorted list of nodes as a complexity of $O(VlogV + ElogE)$ where V is the number of vertices (network nodes) and E is the number of edges (network links). This does not take into account the fact that the heuristic has to check S available time slots for each traversed edge, so a fair estimate would be $O(VlogV + ElogE + ES)$.

For the flow algorithm, a reasonable implementation would be the Edmonds-Karp algorithm [25], which is further simplified by the fact that all edges have a capacity of 1, and all edge costs have a value of 1. With this restriction, finding a single augmenting path can be done in complexity $O(E)$. On the other hand, the graph size is increased by a factor S, and at most S augmenting paths have to be found, which leads to a complexity of $O(ES^2)$. The same-path heuristic may be implemented in different ways, but the most effective solution is to integrate it with the normal path-finding used by the flow algorithm, by ensuring that the order of visiting links is the same that was used for the previous path. This approach would not increase the algorithm complexity.

The in-order path selection algorithm has a running time of $O(k^3)$ where k is the number of paths, which is bounded by the number of slots S, hence the worst-case complexity is $O(S^3)$.

Data: set of paths with given departure time slot, count of allocated slots and length
Result: reduced set of paths with in-order delivery

```
s ← number of time slots;
Duplicate paths p_1...p_n as p_{n+1}...p_{2n} with delay s;
solution ← ∅;
for ∀i ∈ {1, 2, ...n} do
    consider set Q={p_i...p_{i+n−1}};
    Q ← Q \ {p_j ∈ Q |p_j arrives later than p_{i+n−1}};
    Q is the working window;
    initialize t_1...t_{2n}, t_∅ = 0;
    for all flows p_j ∈ Q do
        best ← ∅;
        for all flows p_q ∈ Q, q < j do
            if p_q arrives before p_j and t_q > t_best then
                best ← i;
                predecessor_j ← q;
            end
        end
        t_j ← t_best + bandwidth of p_j;
        if t_j is best solution so far then
            solution ←solution reconstructed by following the
            chain of predecessors of j;
        end
    end
end
```

Algorithm 1: In-order path selection

IV. Hardware Implementation

As mentioned in section III we are addressing the GS mode of operation of the Æthereal networks. For guaranteed services, the network supports two possible implementations, distributed routing and source routing. In this section we will present implementation considerations regarding both cases.

A. Distributed routing

In the distributed routing architecture [17], all routers possess a slot table are synchronized at flit level, which allows them to operate in lock-step. During each time-slot, a flit is routed to a pre-defined destination, based on the contents of the slot table. Routers are unaware of the final destination of flits, but only of one crossbar traversal. The final destination of a packet thus depends only on the time it was inserted into the network, as shown in Figure 4.

One of the advantages of this technique is that a header does not have to be sent along each flit in order to determine its destination

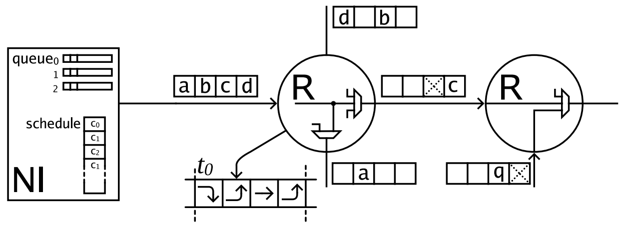

Fig. 4. Distributed routing architecture. The schedule ensures that no two packets will be delivered to the same link at the same time

thus increasing the size of the useful payload. The disadvantage is that slot tables need to be distributed across all routers, and a configuration mechanism has to be provided. The routing is completely oblivious to the number of routes that are used, and our technique can be applied without any change to the hardware.

B. Source routing

In the source routing architecture, a single word header is added to the first flit of a sequence, which stores the entire route to the destination. Subsequent consecutive flits from GT traffic, following the same route do not need to repeat the header. We have considered in our experiments the overhead of including the header when multiple routes are used.

The source network interface is then responsible not only for determining the path the packets will follow, but also for ensuring that no collisions will occur with other GT packets. On the other hand, the advantage is that the route is not solely determined by flit entry time into the network, but as long as collision avoidance is ensured, several channels may use the same time slots.

An approach that takes advantage of this new feature is to let outgoing channels of an NI share the beginning of the path before they diverge toward their specific destinations. This approach was proposed in [30] under the name of channel trees.

We propose two multi-path routing implementations, one consisting of storing the route that should be used during each time slot, side-by-side into the same memory block and indexed with the same value as the slot-table, and another having separate tables storing the routes and only a route id be stored for each time slot. The two solutions are presented in figure 5.

The first solution is more efficient when routes are relatively short or the time slots are few, while the second is favorable when few different routes are in use. The second solution also has the advantage of being compatible with channel trees. The overhead is limited to the storage space the extra route id and storage space for the routes themselves, which amounts to tens to hundreds of bits per network interface, depending on the number of channels, routes and size of the slot table.

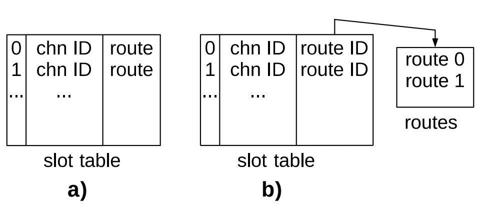

Fig. 5. Hardware implementations: a) directly storing routes in the slot table, b) with separate tables of routes

The changes in figure 5b have been implemented in hardware, and synthesized using Synopsys tools. We have found the overhead to be of 7.5% in terms of area for a network interface kernel with 4 channels and 16-word buffers, when 4 distinct paths are supported for each channel. It should be noted that this is probably an overestimate, as in a real design it may not be necessary to use multi-path routing for all channels, and 4 distinct paths is already in the upper range of expected usage. In terms of speed, we do not expect any negative impact, since the tables are generally static and reading can be pipelined to any extent found necessary. The design has yet to be validated.

V. Experimental Results

For the purpose of evaluating the benefits of our proposal, the Æthereal tool flow [23] has been extended in the following way. The new algorithms were added to the existing tools at source-code level and integration has been performed in order to use the same data structures.

The comparison was made against two single-path allocation algorithms available in the Æthereal tool flow. One, which we will call the ”classic” algorithm consists of a heuristic with polynomial running time, but without guarantees regarding the quality of the solutions while the second algorithm performs an exhaustive search, thus having an exponential running time in the worst-case.

Background traffic was generated, random in both space and time. The source and destination were chosen with equal probability from the set of all nodes, and the bandwidth or correspondingly the number of slots allocated to each channel was chosen following a geometric distribution law capped by the maximum link bandwidth supported by the network. The background traffic was allocated using the classic algorithm from the Æthereal tool flow, with slots allocated in contiguous chunks where possible, subject to restrictions regarding slots that were already in use, starting at a slot position chosen at random.

On top of the background traffic we attempt to allocate channels of the maximum possible bandwidth using each of the algorithms. For the original algorithms in the Æthereal tool flow, we determine the maximum using binary search, while the flow algorithm produces the maximum directly. The channels are then discarded and a new allocation is performed.

The results are presented in Figure 6, in the form of relative improvement of the proposed method, compared to the two existing methods.

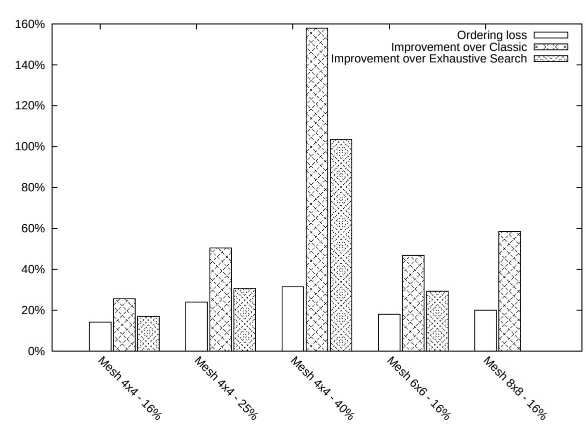

Fig. 6. Reordering overhead and relative improvement in allocated bandwidth

The result already takes into account the bandwidth that was discarded during the ordering stage, but the loss is also represented for comparison. It can be observed that the gain is higher for scenarios with higher network occupation, as well as larger networks which offer higher path diversity. The numerical results are presented in table I, along with another important measure, which is the average number of paths employed by the multi-path algorithm.

TABLE I
ALLOCATED BANDWIDTH OF SINGLE AND MULTI-PATH ALGORITHMS, IN WORDS/SLOT TABLE REVOLUTION AND AVERAGE NUMBER OF PATHS USED

Test	Class.	Exh.	MP bf. discard	MP w/ discard	Gain	paths
4x4-16%	16.73	17.96	24.48	21.01	16.9%	2.91
4x4-25%	8.72	10.06	17.28	13.13	30.5%	3.22
4x4-40%	2.02	2.57	7.63	5.23	103%	2.22
6x6-16%	14.64	16.64	26.24	21.51	29.2%	4.07
8x8-16%	13.35	-	26.45	21.15	58.4%	5.00

Figure 7 presents the results for all loads in the 4-by-4 mesh topology. Each of the points in the graph represents one allocated channel, with the x coordinate being the bandwidth allocated by the flow algorithm (ordering penalty included) and the y coordinate the bandwidth allocated by the single path approach. A small displacement that does not modify the ratio between the coordinates was added in order to differentiate identical results.

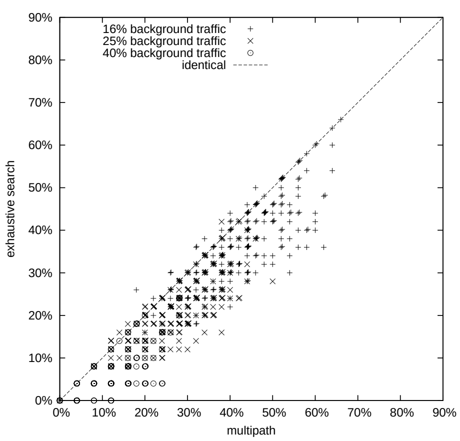

Fig. 7. Allocated bandwidth as a percentage of link capacity, 4-by-4 mesh network under different traffic loads

The results show high variations from one test case to another. For many cases, the single path algorithms and the multi-path algorithm produce the same result. In a few cases, the single-path offers a better solution (the points above the diagonal). This is explained by the fact that the combination of the two algorithms, the flow algorithm and the path selection is not guaranteed to be optimal even though the individual algorithms are. In the vast majority of cases the multi-path algorithm performs better.

It can be observed that in some cases the flow algorithm managed to allocate bandwidth while the exhaustive search allocated nothing at all. This is explained by the fact that the flow algorithm can allocate looping paths passing through one node multiple times, taking care at the same time that the flow does not interfere with itself, while the other algorithm cannot.

As the size of the network increases it can be observed that the advantage of the multi-path approach compared to the single-path solutions also increases. Figure 8 depicts a comparison of our solution with the classic single-path solution from the Æthereal tool flow on a network with 8-by-8 mesh topology. In this setting, the multi-path approach produces a solution with more than twice the bandwidth of the single-path approach in roughly 34% of the test-cases.

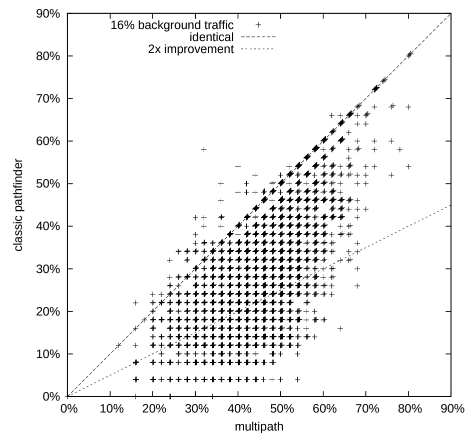

Fig. 8. Multi-path vs Classic allocation algorithms, 8-by-8 mesh 16% background load

Although the common approach is to perform the allocation at design time, other authors have shown in [31] that it is also feasible to compute the allocation at run-time, directly on the embedded device. In order to perform allocation at run-time it is important to have an computationally efficient, low-cost allocation algorithm. The running times of the original Æthereal algorithms as well as the new flow algorithm are presented in Figure 9. Although in asymptotic notation the flow algorithm has a higher complexity, it has performed the fastest in our tests. The explanation is that, the asymptotic notation may hide a constant factor associated with complexity of the operations performed, thus only concerning the way the running time scales with the size of the problem. For the large 8x8 example, we did not find it feasible to run the exhaustive search because of memory and time constraints. In the case of the classic and flow algorithms it can also be observed that the distance between the source and destination nodes has little effect on execution time, while the duration of the exhaustive search grows rapidly.

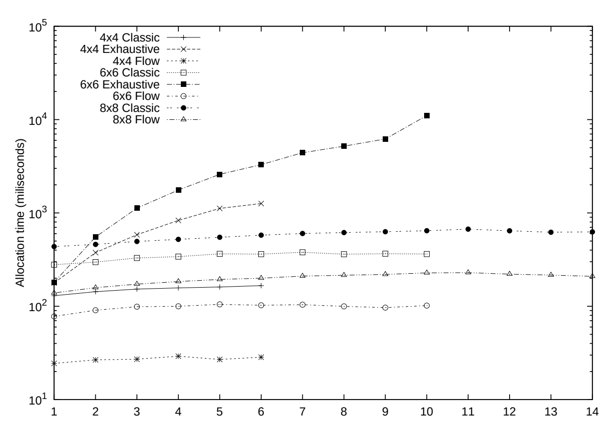

Fig. 9. Algorithm running time

VI. Conclusions and Future Work

In this paper we have studied the use of multi-path routing in networks with fixed resource allocation like Æthereal. Our tests confirmed that the multi-path technique, which was shown to provide benefits in large-scale networks, can provide a significant increase in delivered bandwidth. However it should be noted that the present benchmarks are synthetic, and only compare the performance of path finding algorithms on individual connections. The efficiency of a complete allocation flow based on the multi-path algorithm possibly in combination with other algorithms has yet to be studied.

We have also presented the implications this method on the implementation in hardware. The overhead is shown to be very low: a 7.5% increase in area of the network interface kernels. Regarded in the context of the entire network, the overhead would be even smaller. When distributed slot tables are used the overhead is nonexistent as no modification to the hardware is necessary. Another concern is the complexity of the path-finding algorithm, which is important if slot allocation is to be performed at run-time. We have shown that, despite having a larger complexity, in practice the multi-path algorithm can be faster than single-path solutions.

References

[1] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey, and A. Sangiovanni-vincentelli, "Addressing the system-on-a-chip interconnect woes through communication-based design," in *DAC '01: Proceedings of the 38th Conference on Design Automation*, 2001.

[2] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *Computer*, vol. 35, no. 1, Jan 2002.

[3] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," *SIGARCH Comput. Archit. News*, vol. 20, no. 2, 1992.

[4] G.-M. Chiu, "The odd-even turn model for adaptive routing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, no. 7, 2000.

[5] W. J. Dally and H. Aoki, "Deadlock-free adaptive routing in multicomputer networks using virtual channels," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 4, 1993.

[6] J. Duato, "On the design of deadlock-free adaptive routing algorithms for multicomputers: design methodologies," in *PARLE*, 1991.

[7] S. Vutukury, "A simple approximation to minimum-delay routing," in *ACM SIGCOMM*, 1999.

[8] I. Cidon, R. Rom, and Y. Shavitt, "Analysis of multi-path routing," *IEEE/ACM Trans. Netw.*, vol. 7, no. 6, 1999.

[9] ——, "Multi-path routing combined with resource reservation," *INFOCOM*, vol. 1, Apr 1997.

[10] S. Murali, D. Atienza, L. Benini, and G. De Micheli, "A Method for Routing Packets Across Multiple Paths in NoCs with In-Order Delivery and Fault-Tolerance Guarantees," *VLSI-Design Journal*, 2007.

[11] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *Computer*, vol. 26, no. 2, 1993.

[12] J. Hu and R. Marculescu, "DyAD: Smart routing for networks-on-chip," in *DAC*, 2004.

[13] M. Li, Q.-A. Zeng, and W.-B. Jone, "DyXY: A proximity congestion-aware deadlock-free dynamic routing method for network on chip," in *DAC*, 2006.

[14] G. Ascia, V. Catania, M. Palesi, and D. Patti, "A new selection policy for adaptive routing in network on chip," in *EHAC*, 2006.

[15] I.-G. Lee, J. Lee, and S.-C. Park, "Adaptive routing scheme for NoC communication architecture," in *ICACT*, vol. 2, 2005.

[16] M. A. A. Faruque, T. Ebi, and J. Henkel, "Run-time adaptive on-chip communication scheme," in *ICCAD*, 2007.

[17] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal network on chip: Concepts, architectures, and implementations," *IEEE Design & Test of Computers*, vol. 22, no. 5, 2005.

[18] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip," in *DATE*, 2004.

[19] J. Liang, S. Swaminathan, and R. Tessier, "aSOC: A scalable, single-chip communications architecture," in *PACT*, 2000.

[20] J. M. Nolen and R. N. Mahapatra, "Time-division-multiplexed test delivery for NoC systems," *IEEE Des. & Test*, vol. 25, no. 1, 2008.

[21] P. Wolkotte, G. Smit, G. Rauwerda, and L. Smit, "An energy-efficient reconfigurable circuit-switched network-on-chip," in *IPDPS*, April 2005.

[22] J. Liu, L.-R. Zheng, and H. Tenhunen, "Interconnect intellectual property for network-on-chip (NoC)," *J. Syst. Archit.*, vol. 50, no. 2-3, 2004.

[23] A. Hansson, K. Goossens, and A. Rădulescu, "A unified approach to mapping and routing on a network-on-chip for both best-effort and guaranteed service traffic," *VLSI Design*, 2007.

[24] L. Ford and D. Fulkerson, *Flows in Networks*, 1962.

[25] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *J. ACM*, vol. 19, no. 2, 1972.

[26] C. Schensted, "Longest increasing and decreasing subsequences," *Canad. J. Math*, 1961.

[27] M. H. Albert, M. D. Atkinson, D. Nussbaum, J.-R. Sack, and N. Santoro, "On the longest increasing subsequence of a circular list," *Information Processing Letters*, vol. 101, no. 2, 2007.

[28] D. E. Knuth, "Big omicron and big omega and big theta," *SIGACT News*, vol. 8, no. 2, 1976.

[29] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, 1959.

[30] A. Hansson, M. Coenen, and K. Goossens, "Channel trees: reducing latency by sharing time slots in time-multiplexed networks on chip," in *CODES+ISSS*, 2007.

[31] T. Marescaux, B. Bricke, P. Debacker, V. Nollet, and H. Corporaal, "Dynamic time-slot allocation for QoS enabled networks on chip," in *ESTIMedia*, 2005.