PREPRINT: "H. Kashif, H. Patel, and S. Fischmeister, "Using Link-level Latency Analysis for Path Selection for Real-time Communication on No Cs," in proceedings of IEEE Asia and South Pacific Design Automation Conference (ASP-DAC), 2012, pp. 499-504."

Using Link-level Latency Analysis for Path Selection for Real-time Communication on NoCs

Hany Kashif, Hiren D. Patel and Sebastian Fischmeister

Electrical and Computer Engineering
University of Waterloo, Waterloo, Canada
e-mail: {hkashif, hdpatel, sfischme}@uwaterloo.ca

Abstract— We present a path selection algorithm that is used when deploying hard real-time traffic flows onto a chip-multiprocessor system. This chip-multiprocessor system uses a priority-based real-time network-on-chip interconnect between the multiple processors. The problem we address is the following: given a mapping of the tasks onto a chip-multiprocessor system, we need to determine the paths that the traffic flows take such that the flows meet there deadlines. Furthermore, we must ensure that the deadline is met even in the presence of direct and indirect interference from other flows sharing network links on the path. To achieve this, our algorithm utilizes a link-level analysis to determine the impact of a link being used by a flow, and its affect on other flows sharing the link. Our experimental results show that we can improve schedulability by about 8% and 15% over Minimum Interference Routing and Widest Shortest Path algorithms, respectively.

I. Introduction

To deploy hard real-time embedded applications, it is important to accurately predict its worst-case execution times (WCETs). The tasks of the application are schedulable if the WCETs are less than or equal to the temporal deadlines of the application tasks. A WCET analysis typically uses static program analysis techniques, and combines it with a platform model with details of its pipeline micro-architecture, cache organization, and exception mechanisms to produce an upper-bound on the execution time of the task. However, with chip-multiprocessing architectures becoming common nowadays, we must include the execution delays experienced by the communication between multiple processing elements in the WCET analysis. This is known as on-chip real-time communication.

The common on-chip interconnects for real-time either use resource reservation or run-time arbitration. An example of resource reservation is time-division multiplexing (TDM) as proposed by AEthereal NoCs [1]. TDM NoCs allocate resources statically prior to the execution of the application, and mandates a static schedule identifying when packets are transferred on specific channels. It ensures that there is no contention for a resource between any two packets. Runtime arbitration NoCs, on the other hand, use priority-aware routers (PAR) to arbitrate channels at runtime. This introduces contention, but expects the analysis to accommodate it. An example is proposed by Shi et al. [2, 3]. The advantage of PAR NoCs over TDM NoCs is that resources are better shared and reused. In addition, low latency flows in TDM are tightly coupled with the bandwidth resulting in the over-allocation of bandwidth. Therefore, we focus on CMPs with a PAR on-chip interconnection network.

There are two approaches to the WCET analysis of PAR NoCs: Shi et al. [2, 3] provide a flow-based analysis (FLA), and the other approach is a link-level analysis (LLA) for WCET estimates. At the expense of a detailed analysis, LLA results in significantly tighter bounds than FLA. These analyses model communication as periodic flows on the PAR NoC, and they assume that the mapping of flows, and the paths the flows take are given. The set of flows of an application are schedulable if the WCET of every flow is less than or equal to its deadline. However, we notice that for an application with the same set of flows and deadlines, the choice of paths can greatly influence the schedulability result of the entire application. Assuming a given mapping of the flows onto the NoC, we contend that by judiciously selecting the paths the flows take, we can increase the number of schedulable flows; in turn, allowing more tasks to be schedulable.

This brings us to the main contribution of our work: a path selection algorithm (PSA) assisted by the LLA that aims to improve the number of schedulable flows by selecting appropriate paths in the NoC. We use LLA because it considers the pipelining effect of worm-hole switched NoCs, and it provides tight WCET bounds. This is unlike FLA, which treats the flow as an indivisible unit across multiple network links. In particular, we propose a PSA that utilizes observations from LLA to efficiently select paths in the PAR NoC. To avoid the high complexity of an optimal algorithm, PSA uses heuristics to find least interference paths and to consider lower priority flows while selecting paths for the higher priority ones. Based on our experimentation results, we can improve schedulability by about 8% and 15% over Minimum Interference Routing and Widest Shortest Path algorithms, respectively.

II. Related Work

Worst-case latency computation on inter-process communication in real-time systems is well-established for connection-based packet networks [4, 5]. However, these methods are inefficient due to the overhead of the establishment and tear down of channels between source and destination pairs which contributes to the communication latency, as well as under utilization of the system's resources. Storing of packets at intermediate nodes requires expensive buffer capacity for storing early arriving packets and queueing packets in order of arrival [4].

An alternative option is worm-hole switching, which increases throughput, and decreases the required buffer capacity in the network. The authors in [2, 3] present an efficient method for WCET analysis of flows (recall FLA) with worm-hole switching in a PAR NoC. They view a flow and its path as one indivisible entity. This flow-level view of the communication in the PAR NoC leads to two issues: (1) Overestimation of the worst-case latencies when compared to what could be achieved using LLA, and (2) High cost of using FLA for path selection because the smallest unit at which we view a flow is its whole path. Thus, requiring a path selection algorithm using FLA to enumerate all paths.

Several algorithms exist for path selection in a network. Among these are Shortest Path, Widest Shortest Path, and Shortest Widest Path, which are greedy approaches. Another class of algorithms consider other flows while selecting a path, but are more computationally expensive. Examples are Minimum Interference Routing [6], Light Minimum Interference Routing [7], and Profile-based Routing [8]. Distributed routing algorithms also exist [9], which are online algorithms that either require a global state which leads to high communication overhead and performance degradation or do not share a global state and compensate with large number of control messages and subsequently do not scale.

There is also research on path selection for worm-hole switched networks [10, 11, 12, 13]. Some of these methods attempt to find contention free paths or minimize total cost, sometimes leading to higher ratios of unschedulability of flows [13]. Others attempt to minimize the maximum contention value, which is similar to the techniques used in MIRA [6, 7]. However, all of these approaches consider the path of a flow as an indivisible unit, thus cannot use actual latency costs that would require a computationally extensive enumeration of paths. Contrary to that, we construct a path by using link latencies as costs to find the most suitable path for a flow.

III. Background: Network Model

For an application with parallel tasks, we assume a given mapping of these tasks on the PAR NoC. We only consider PAR NoCs with mesh topologies. These tasks are marked as source and destination pairs based on the communication flow between them. All nodes of the PAR NoC contain both a processing element that executes tasks, and a router. Recall that in the PAR NoC, the routers are priority-aware arbiters that implement worm-hole switching with flit-level preemption, and flow control. The router architecture we employ was originally proposed in [2, 3], but for clarity we briefly describe its architecture. The router has a VC for every distinct communication flow with a unique priority that passes through the router. Consequently, there exists a VC for each priority level. The VCs are designed as FIFO buffers at the input ports of the router. These FIFOs store the flits to be routed. The router selects the output port for a flit in the VCs based on its desired destination. When there are multiple flits waiting to be routed, the router selects and forwards the flit to the output port with the highest priority amongst all the waiting flits. Flow control guarantees that the router only sends data to the neighbouring router if the neighbour has enough buffer space to store the data. If the highest priority flit is blocked in the network, the next highest priority flit can access the output link. Nodes are connected using bidirectional links with uniform bandwidth.

Since there is a VC for every distinct communication flow, this guarantees that deadlocks due to cyclic dependencies never occur. The reason is that each flow has its own buffers and thus never blocks another flow for buffer space. If we, however, extend our model to allow sharing of VCs by multiple flows, we must guarantee that our deterministic path selection algorithm is still deadlock-free. We can achieve this by ensuring that as the algorithm proceeds, we have an acyclic channel dependency graph [14].

We model the network as a graph $G = \langle V, E \rangle$, and a set of periodic real-time communication flows $\Gamma = \{\tau_1, \ldots, \tau_k\}$. A flow τ_i is a tuple $\langle v_s, v_d, L_i, P_i, T_i, D_i, J_i^R \rangle$ where v_s and v_d are the source and destination nodes, L_i is the basic link latency, P_i is the priority, T_i is the period, D_i is the deadline, and J_i^R is the release jitter. The basic link latency L_i equals $\frac{flit\_size * num\_flits}{bandwidth}$, which is the total packet size (number of flits multiplied by the flit size) divided by the link bandwidth.

IV. Link-Level Analysis

LLA computes the worst-case latencies by including the direct, and indirect interferences caused by flows sharing links on a path. We present an overview of this link-level analysis in this section.

Assume that the latency of a flow τ_i suffering interference from a higher priority flow τ_j on a particular link e in the NoC is given by M_{i_e}. The interval during which τ_j preempts τ_i is $M_{i_e} + J_j^R + J_j^I$ where J_j^I is the interference jitter of τ_j which represents any interference τ_j suffers from higher priority flows. Hence, the number of times that τ_j preempts τ_i equals $\lceil \frac{M_{i_e} + J_j^R + J_j^I}{T_j} \rceil$. The total contribution of flow τ_j in the latency of τ_i is then given by $\lceil \frac{M_{i_e} + J_j^R + J_j^I}{T_j} \rceil * L_j$. Therefore, the worst-case latency of τ_i is equal to $M_{i_e} = \lceil \frac{M_{i_e} + J_j^R + J_j^I}{T_j} \rceil * L_j + L_i$. This form can be generalized to $M_{i_e} = \sum_{\forall \tau_j \in S_{i_e}^D} \lceil \frac{M_{i_e} + J_j^R + J_j^I}{T_j} \rceil * L_j + L_i$ where $S_{i_e}^D$ is the set of higher priority flows directly interfering with τ_i on e.

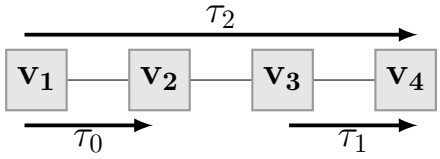


Fig. 1: Illustrative example

LLA provides tighter bounds than FLA as presented by Shi and Burns [2, 3]. We use the example from Figure 1 to describe the difference between both types of analyses, and further explain the details of LLA. Consider the data given in Table I for flows τ_0, τ_1 and τ_2 that are ordered in decreasing priorities. The FLA in [2, 3] views the path of τ_2 as one entity suffering simultaneous interference from both τ_0 and τ_1. Basically, FLA computes the number of times higher priority flows interrupt a packet of a particular flow. It then multiplies the number of interrupts by the latency of the higher priority pack-

ets to get the total interference latency, and adds the interference latency to the basic flow latency to get the worst-case latency. The total worst-case latency for τ_2 in this case is given by $M_2 = \lceil \frac{M_2}{4} \rceil * 2 + \lceil \frac{M_2}{4} \rceil * 2 + 5$ (for illustration purposes of the FLA, we neglect the routing delay in this particular computation). Notice, that this equation has no solution which follows directly from the fact that each of the flows τ_0 and τ_1 utilizes 2 time units with a period of 4 units thus needing 50% of the capacity and deeming τ_2 unschedulable.

TABLE I: Illustrative example data

Flow	L	T	D	J
τ_0	2	4	4	0
τ_1	2	4	4	0
τ_2	5	30	30	0

LLA, on the other hand, analyzes each link separately. The latency on link (v_1, v_2) equals $M_2 = \lceil \frac{M_2}{4} * 2 \rceil + 5 = 11$. τ_2 suffers no interference on link (v_2, v_3) and, hence, in the worst-case, the packet from the previous link (v_1, v_2) continues with the same latency of 11 time units. The interference that occurs on (v_1, v_2) is the intermission of τ_2 flits by flits of τ_0 resulting in a sequence of flits from both flows. This interference discontinues on the next link but still leaves gaps between the flits of τ_2 and, hence, leading to the conclusion that the latency stays the same in the worst-case. On link (v_3, v_4), τ_1 interferes with τ_2, but in this case the time interval between the first and last flits of the τ_2 packet is different from the first case on link (v_1, v_2) (due to the gaps caused by τ_0). Therefore, to calculate the latency on link (v_3, v_4), instead of using L_2 in the latency equation, we use the latency from the previous link. The latency then becomes $M_2 = \lceil \frac{M_2}{4} * 2 \rceil + 11 = 23$. The total latency using LLA is $M_2 = 23 + 3 = 26$ (assuming a one time unit routing delay per hop).

A conclusion that can directly be drawn from the comparison of LLA and FLA is that LLA provides tighter worst-case latency upper bounds. It can also easily be shown that in the worst-case of interferences (all interfering flows share all links with the flow of interest) the LLA provides an equivalent result to that of the FLA. Tighter latency upper bounds improve the schedulability of time-constrained flows in a NoC as it has been shown in the previous example.

V. Path Selection

For a hard real-time system, the path selection problem is the following: find the possible paths that flows can take given their source and destination (S, D) pairs, and flow requirements such that the flows meet their respective deadlines. That is, given a graph $G = \langle V, E \rangle$, and a set of flows $\Gamma = \{\tau_1, \ldots, \tau_k\}$, is it possible to select a path δ_i for each flow τ_i such that its worst-case latency is less than or equal to its deadline D_i?

A. Optimal Path Selection Algorithm

Objective. *Satisfy the deadline requirements of the flows by searching all possible paths from source to the destination of each flow.*

Assuming that a path visits a node only once, each flow will have $4 * 3^{v-1}$ possible paths (assuming a NoC with v nodes) in a mesh topology because from each node descends three possible nodes to traverse (four for the source node). An optimal algorithm selects a path for the first flow, then selects one for the second and so on. If at any point the deadline constraints are not met, the algorithm backtracks one step and selects an alternative path. Hence, the decision tree has k levels corresponding to the number of flows, and from each node descends $4 * 3^{v-1}$ choices that correspond to all possible paths. The total complexity of the algorithm, therefore, becomes $O((3^v)^k)$.

B. Heuristic-based Path Selection Algorithm

The optimal path selection algorithm has a high exponential complexity which makes it impractical. Therefore, we present a heuristic-based path selection algorithm (PSA) that uses the link-level analysis (LLA) to guide the path selection. We pointed out in Section IV that the worst-case latency of a flow on a link depends on its latency and the interfering flows on the preceding link. Since we are concerned with path selection, we require a heuristic so that the algorithm does not have to backtrack, and in the worst-case enumerate all paths. We also need a heuristic that while the algorithm routes higher priority flows, considers lower priority ones.

Objectives

Our overall goal with path selection is to select paths for flows that improve the schedulability of the flows while incorporating the following:

Objective 1. *Account for lower priority flows.*

While selecting paths for higher priority flows, expected paths for lower priority ones are taken into account to maximize schedulability. Otherwise, the PSA might overload some links making lower priority flows unschedulable. By accounting for lower priority flows, we prevent the lower priority flows from being starved.

Objective 2. *Consider the availability of shortest paths.*

The criticality of links for lower priority flows depends on their utilization as part of all available shortest paths. The intuition behind the criticality is that the more the number of available shortest paths, the more likely it is for a lower priority flow to be schedulable. Similarly, the lower the number of available shortest paths, the less likely it is for the lower priority flow to be schedulable. This concept is similar to that of critical links in [6, 7].

Objective 3. *Minimize the heterogeneity of interfering flows.*

We promote sharing of links between flows that share prior links. Consider a flow τ_5 that has a path with 3 links. Using the data in Table I, if τ_5 has interference with τ_0 on the 1^{st} link, with τ_1 on the 2^{nd}, and none on the 3^{rd}, the latency on the 1^{st} link equals $R_5 = \lceil \frac{R_5}{8} \rceil * 2 + 9 = 13$, on the 2^{nd} equals $R_5 = \lceil \frac{R_5}{8} \rceil * 2 + 13 = 19$, and on the 3^{rd} equals 19. The total worst-case latency is 22. However, if τ_5 has interference with τ_0 on all 3 links, then the latency on each link equals 13, and the total worst-case latency is 16. Hence, we identify from LLA

that a flow τ_i sharing multiple links with a flow τ_j results in a lower worst-case latency than sharing fewer links with different flows. Therefore, PSA favours sharing links with the same flow over a variety of flows.

Algorithm

We use the interference that a flow suffers on a link as the cost of that link. We construct the network graph G to capture the topology, and as the algorithm proceeds, it adds edges that represent sharing more than one successive link with the same flow, but do not actually exist as links in the NoC. These edges hold the cost of interference over multiple links, and the intermediate nodes that represent actual NoC nodes. So for example, if the algorithm selects the path $[v_1, v_2, v_3]$ for flow τ_1, then when selecting the path for τ_2, the algorithm will set the interference for links (v_1, v_2) and (v_2, v_3) and creates a new edge (v_1, v_3) that has a cost of interference with τ_1 on both links and saves v_2 as an intermediate node. Although, this is not an optimal solution for representing all possible cases of multiple link interferences, this heuristic allows us to achieve objective 3 efficiently and maximize schedulability.

Since we do not enumerate all possible paths, the order of selecting paths to flows affects the latencies and the overall system schedulability. Accommodating multiple flows in the network is known as the multi-commodity flows problem, which is an NP-Complete problem [15, 8]. Once again, this makes the path selection algorithm intractable. We perform the path selection process according to the priority of the flows: higher priorities first. However, we still accommodate lower priority flows while selecting a path for a higher priority flow using three different heuristics.

H 1. *Identify critical links as ones with least residual capacity. Residual capacity is the difference between the capacity of a link and the flow being transmitted over it.*

H 2. *Identify critical links as the links constituting the paths of lower priority flows with only a single shortest path.*

H 3. *Assign costs to all links in all shortest paths of each lower priority flow. This cost depends on the number of available shortest paths to a flow, and how critical the links are depending on how many shortest paths use the same link.*

The input to the Algorithm 1 is a PAR NoC with a mesh topology of size $n \times n$ represented as a graph $G = \langle V, E \rangle$, and Γ with k flows ordered according to their priorities with 1 being the highest. The output is a set of paths for each of the flows in Γ. When selecting a path for a flow, the algorithm updates the cost of that path in the graph G. The cost on each edge accounts for both higher and lower priority flows (using one of the three heuristics).

To account for lower priority flows using H3, the algorithm finds the number of shortest paths available, and the number of times each edge is used amongst all the shortest paths for every lower priority flow. The simplest method to obtain this information finds all the shortest paths for a flow, which for a mesh topology has a complexity of $2^{2*(n-1)}$, i.e. 2^n. However, notice that that the information we require depends only on the relative x and y positions of the source and destination nodes:

Algorithm 1 PATH-SELECTION

Input: $G\langle V, E\rangle, \Gamma = \{\tau_i : \forall i \in [1, k]\}$
Output: $\{\delta_i : \forall\, i \in [1, k]\}$
  Let $SPC[1, n-1][1, n-1]$ be an empty array
  Let $SPE[1, n-1][1, n-1]$ be an empty array

  $SPC[i, j] \leftarrow MAX\ \forall\ (i, j) \in M$
  LOWER-PRIORITY$(SPC, SPE, n-1, n-1)$
  for all $\tau_i \in \Gamma$ **do**
    Let $G'\langle V', E'\rangle$ s.t. $V' = V$ and $E' = E$
    for $\tau_j \in [\tau_{i+1}, \ldots, \tau_k]$ **do**
      $\Delta x \leftarrow |(v_{s_k} \bmod n) - (v_{d_k} \bmod n)|$
      $\Delta y \leftarrow |v_{s_k}/n - v_{d_k}/n|$
      for all $e \in SPE[\Delta x, \Delta y]$ **do**
        $w \leftarrow \frac{L_k}{D_k - C_k} \times \frac{count(SPE[\Delta x, \Delta y], e)}{SPC[\Delta x, \Delta y]}$
        $w(G', e) \leftarrow w(G', e) + w$
        $updateIntermediate(G', e)$
      end for
    end for
    $\delta_i \leftarrow Dijkstra(G', v_{s_i}, v_{d_i})$
    $\delta_i \leftarrow expandIntermediate(G', \delta_i)$
    INTERFERENCE-COSTS(G, δ_i)
  end for
  return $\{\delta_i : \forall\, i \in [1, k]\}$

Δx and Δy. Hence, we use memoization, which is a form of dynamic programming that reduces the complexity to n. The algorithm makes a single call to Function 3 that calculates the number of shortest paths, and the count of each edge on these paths for all possible combinations of Δx and Δy. Array SPC stores the number of shortest paths available for a given Δy and Δx, and array SPE stores, for every Δy and Δx, the number of times each edge appears on these shortest paths. SPC has n^2 entries while SPE has $n^2 * 2n * (n-1)$ entries. The function recursively uses the information from nodes with lower values of Δy and Δx.

Each selected path will add interferences to the graph according to Function 2. The function call $edgeInterference(G, e)$ calculates the interference on edge e, and function $nodeInterference(G, [v_i, ..., v_j])$ computes interference on a sequence of adjacent nodes $[v_i, \ldots, v_j]$ forming a path. When the algorithm selects a path for a flow, it adds edges between each node on the path and all of its successive nodes. $intermediate$ holds the intermediate nodes in newly created edges. If the edge is an actual link, then the algorithm will add the interference on that link to the weight of the edge. However, if it is not a link, then the algorithm will set the weight of the edge to the interference on the path formed by the intermediate nodes of that edge in one of three cases: (1) the edge does not exist, or (2) the edge exists and has the same intermediate nodes as the one the algorithm is adding, or (3) the interference on the edge being added is less than the existing one.

Algorithm 1 adds costs to the edges based on a speculation of the paths that will be selected for lower priority flows as described above. The function $count(SPE[i, j], e)$ retrieves the count of an edge e for a specific Δy and Δx. The function $expandIntermediate(G, \delta)$ replaces edges in a path that do not belong to the actual topology with the equivalent interme-

diate nodes. The function $updateIntermediate(G, e)$ updates the costs of all edges that do not belong to the topology if they have the edge e as an intermediate edge. The algorithm calculates Δy and Δx for each lower priority flow and adds a cost to the links involved. A weight is used to represent the criticality of the edge which is equal to the edge count in SPE divided by the number of shortest paths. This weight is multiplied by L_k the basic latency of the lower priority flow over the slack that it has to its deadline on the speculated path where D_k is the deadline and C_k is the best case latency. Dijkstra's algorithm is used to find the least cost path for the flow being routed.

Function 2 INTERFERENCE-COSTS

Input: $G, \delta = [v_s, \ldots, v_d]$
  for all $v_i \in \delta$ **do**
    for $v_j \in [v_i, \ldots, v_d]$ **do**
      $e \leftarrow (v_i, v_j)$
      if $j - i = 1$ **then**
        $w(G, e) \leftarrow edgeInterference(G, e)$
      else if $w(G, e) = 0 \vee nodeInterference(G, [v_i, \ldots, v_j]) < w(G, e) \vee intermediate(G, e) = [v_i, \ldots, v_j]$ **then**
        $w(G, e) \leftarrow nodeInterference(G, [v_i, \ldots, v_j])$
        $intermediate(G, e) = [v_i, \ldots, v_j]$
      end if
    end for
  end for

Function 3 LOWER-PRIORITY

Input: SPC, SPE, i, j
  if $SPC[i, j] < MAX$ **then**
    return $SPC[i, j]$
  end if
  if $i = 0$ & $j = 0$ **then**
    $SPC[i, j] \leftarrow 1$
    $SPE[i, j] \leftarrow [\,]$
  else if i = 0 **then**
    $SPC[i, j] \leftarrow$ LOWER-PRIORITY$(SPC, SPE, i, j - 1)$
    $SPE[i, j] \leftarrow SPE[i, j - 1] + edge(i, j, i, j - 1)$
  else if j = 0 **then**
    $SPC[i, j] \leftarrow$LOWER-PRIORITY$(SPC, SPE, i - 1, j)$
    $SPE[i, j] \leftarrow SPE[i - 1, j] + edge(i, j, i - 1, j)$
  else
    $SPC[i, j] \leftarrow$LOWER-PRIORITY$(SPC, SPE, i, j - 1)+$ LOWER-PRIORITY$(SPC, SPE, i - 1, j)$
    $SPE[i, j] \leftarrow SPE[i, j-1] + SPE[i-1, j] + edge(i, j, i, j - 1) + edge(i, j, i - 1, j)$
  end if

Complexity Analysis

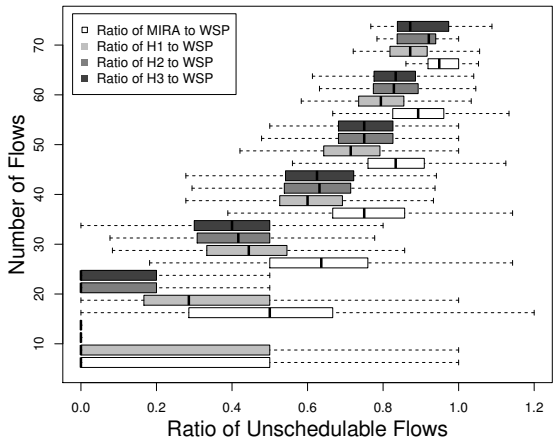
Recall that we have k flows in a graph with v vertices. Function 3 has a complexity v and is called only once. At worst, each flow will have a path with v nodes. The number of edges that the algorithm will create is given by: $k*((v-2)+(v-3)+\ldots+1) = k*\sum_{i=2}^{v-1}(v-i)$. Thus, in the worst case, the algorithm will create $k * v^2$ edges. The functions $Dijkstra$ and $expandIntermediate$ have linear complexity v. The function $updateIntermediate$ uses a structure that saves, for each edge, newly created edges which it is a part of as an intermediate edge. The maximum number of edges that can be involved in all shortest paths between 2 nodes is $2(v - \sqrt{v})$. The overall complexity is therefore given by: $k^2 * 2(v - \sqrt{v}) * k * v^2$ which is $O(k^3 * v^3)$. Although, this is a high complexity compared to minimum interference algorithms for example, PSA never elicits that upper bound computation time which assumes that each edge is part of all newly created edges. It would be beneficial to compute the expected case complexity, but due to space constraints we only present the experimental computation times of the algorithm.

VI. Experimentation

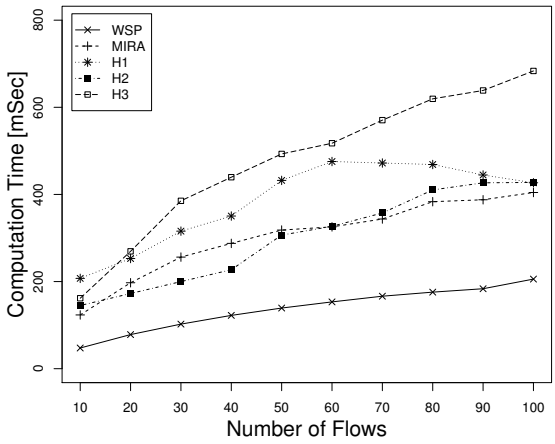
We quantitatively compare PSA with the three different heuristics against the widest shortest path (WSP), and the minimum interference routing algorithm (MIRA) [6, 7]. We vary several parameters during experimentation to assess its effect on the schedulability of flows, and the execution time of the different algorithms. Our experiments uses 4×4, and 8×8 mesh topologies for the NoC. The basic link latency of a flow is randomly chosen from a uniform distribution in the range $[16, 1024]$. The link utilization of a flow τ_i is its basic link latency divided by its period: $U_i = L_i/T_i$. We vary the link utilization between $[0.4, 0.85]$ in step increments of 0.05. The deadline D_i takes values between $[0.7, 1.0]$ in increments of 0.1 as a ratio of period T_i. The number of flows in the network range between $[10, 100]$ in steps of 10. With these parameters, we have 800 different configurations. For each configuration, we generate 1000 test cases with a random mapping of the source and destination nodes of the flows for each test. All tests were run on an AMD Opteron 6174 2.2 GHz processor with 8.0 GB of memory.

Figure 2a shows the ratio of the number of unschedulable flows for MIRA and the three proposed heuristics to WSP against the number of flows for an 8×8 mesh with $U = 0.4$. For a given number of flows, each box in the figure represents 1000 random test cases. The boxes represent the lower quartile, median and upper quartile of the data, and the whiskers show the minimum and maximum observations. It is clear that H2 and H3 always perform better than WSP and MIRA except for the maximum observations. H1 has a comparable performance to MIRA for 10 flows and performs better as the number of flows increase. H1 starts performing better than H2 and H3 as the number of flows increases beyond 40 flows. For lower number of flows, using shortest paths heuristics (H2, H3) can help leave these paths open for lower priority flows thus increasing schedulaiblity over H1. However, as number of flows increase, these paths are occupied and avoiding links with low residual capacity (H1) becomes more important and improves schedulability. The graph also shows that for a small number of cases, WSP has less unschedulable flows compared to the other algorithms. The reason is that, in these cases, increasing the cost of critical links leads to selecting non-shortest paths. This leads in some cases to unschedulable flows due to high interference on the chosen paths.

Figure 2b shows the average execution times of the algorithms against the number of flows. Each point represents an average of $80,000$ test cases with 1000 for each of the 80 different configurations. WSP takes the least amount of execution time, and H2 closely follows MIRA. H1 and H3, however, have higher execution times. The reason is that H2 only adds costs



(a) Ratio of unschedulable flows compared to WSP against the number of flows for an 8×8 mesh with $U = 0.4$ and $D = 1.0$



(b) Average computation times for the different algorithms against the number of flows

Fig. 2: Experimental results

to lower priority flows with one shortest path, thus doing less computations than H1 and H3. H3 has the highest computation time because it updates the cost of all edges in all shortest paths of lower priority flows.

In summary, for all $800,000$ tests, we observe an average improvement in schedulability of 12.3%, 14.5% and 15.1% over WSP for H1, H2 and H3, respectively. The average improvement over MIRA is 3.5%, 7.2% and 8.0% for H1, H2 and H3, respectively. The computation times of the proposed algorithms are acceptable and close to that of MIRA.

VII. Conclusion

This paper presents a new path selection algorithm for routing real-time flows in a priority-based interconnect based on a link-level analysis view of the NoC. Our algorithm introduces an edge creation heuristic that accommodates the dependency of latencies on traversed links, enables PSA to minimize the variation of interfering flows, and consequently reduces the worst-case latency. We use three heuristics to account for expected paths of low priority flows. H1 performs worse than MIRA at lower number of flows and has an overall improvement in schedulability of 3.5%. H2 improves over MIRA in schedulability by 7.2%. H3 improves in schedulability over MIRA by 8.0%. The three heuristics have a schedulability improvement over WSP by 12.3%, 14.5% and 15.1%, respectively. The computation times of the three heuristics are comparable to MIRA's and much less than that of the optimal algorithm. Our plan is to investigate additional gains in path selection by introducing a joint mapping and path selection technique.

Acknowledgement

This research was supported in part by NSERC DG 357121-2008, NSERC DG 386714-2010, ORF RE03-045, ORE RE04-036, ORF-RE04-039, ISOP IS09-06-037, APCPJ 386797-09, and CFI 20314 with CMC.

References

[1] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal Network on Chip: Concepts, Architectures, and Implementations," *IEEE Design and Test*, vol. 22(5), 2005.

[2] Z. Shi and A. Burns, "Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching," in *Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip*, Washington, USA, 2008.

[3] Z. Shi, "Real-Time Communication Services for Networks on Chip," Ph.D. dissertation, The University of York, UK, 2009.

[4] D. D. Kandlur, K. G. Shin, and D. Ferrari, "Real-Time Communication in Multihop Networks," *IEEE Trans. Parallel Distrib. Syst.*, 1994.

[5] A. K. Parekh and R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case," *IEEE/ACM Trans. Netw.*, 1994.

[6] K. Kar, M. Kodialam, and T. V. Lakshman, "Minimum Interference Routing of Bandwidth Guaranteed Tunnels with MPLS Traffic Engineering Application," *IEEE Journal on Selected Areas in Communications*, 2000.

[7] G. B. Figueiredo, N. L. S. da Fonseca, and J. A. S. Monteiro, "A Minimum Interference Routing Algorithm with Reduced Computational Complexity," *Comput. Netw.*, vol. 50, 2006.

[8] S. Suri, M. Waldvogel, and P. R. Warkhede, "Profile-Based Routing: A New Framework for MPLS Traffic Engineering," in *Proceedings of International Workshop on Quality of Future Internet Services*. London, UK: Springer-Verlag, 2001.

[9] K. G. Shin, C.-C. Chou, and S.-K. Kweon, "Distributed Route Selection for Establishing Real-Time Channels," *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, 2000.

[10] S. Lee and J. Kim, "Path Selection for Message Passing in a Circuit-Switched Multicomputer," *J. Parallel Distrib. Comput.*, vol. 35, 1996.

[11] S. B. Shukla and D. P. Agrawal, "Scheduling Pipelined Communication in Distributed Memory Multiprocessors for Real-Time Applications," *SIGARCH Comput. Archit. News*, vol. 19, 1991.

[12] D. D. Kandlur and K. G. Shin, "Traffic Routing for Multicomputer Networks with Virtual Cut-Through Capability," *IEEE Trans. Comput.*, vol. 41, 1992.

[13] K. Nam, S. Lee, and J. Kim, "Path Selection for Real-Time Communication in Wormhole Networks," *International Journal of High Speed Computing*, 1999.

[14] W. J. Dally and C. L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Comput.*, vol. 36, 1987.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.