A Unified Approach to Constrained Mapping and Routing on Network-on-Chip Architectures

Andreas Hansson
Dept. of Information Technology
Lund University, Box 118, 221 00
Lund, Sweden
hansson@natlab.research.philips.com

Kees Goossens
Philips Research Laboratories
Prof. Holstlaan 4, 5656 AA
Eindhoven, The Netherlands
kees.goossens@philips.com

Andrei Rădulescu
Philips Research Laboratories
Prof. Holstlaan 4, 5656 AA
Eindhoven, The Netherlands
andrei.radulescu@philips.com

ABSTRACT

One of the key steps in *Network-on-Chip* (NoC) based design is spatial mapping of cores and routing of the communication between those cores. Known solutions to the mapping and routing problem first map cores onto a topology and then route communication, using separated and possibly conflicting objective functions. In this paper we present a unified single-objective algorithm, called Unified MApping, Routing and Slot allocation (UMARS). As the main contribution we show how to couple path selection, mapping of cores and TDMA time-slot allocation such that the network required to meet the constraints of the application is minimized. The time-complexity of UMARS is low and experimental results indicate a run-time only 20% higher than that of path selection alone. We apply the algorithm to an MPEG decoder *System-on-Chip* (SoC), reducing area by 33%, power by 35% and worst-case latency by a factor four over a traditional multi-step approach.

Categories and Subject Descriptors: B.4.3 [Input/Output and Data Communications]: Interconnections – *Topology*

General Terms: Design, Algorithms, Performance

Keywords: System-on-Chip, Network-on-Chip, Quality-of-Service, Mapping, Routing

1. INTRODUCTION

Systems-on-Chip (SoC) grow in size with the advance of semiconductor technology enabling integration of dozens of cores on a chip. The continuously increasing number of cores calls for a new communication architecture as traditional architectures are inherently non-scalable, making communication a bottleneck [1, 21].

System architectures are shifting towards a more communication-centric methodology [21]. Growing SoC complexity makes *communication* subsystem design as important as *computation* subsystem design [2]. The communication infrastructure must efficiently accommodate the communication needs of the integrated computation and storage elements. In application domains such as multi-media processing, the bandwidth requirements are already in the range of several hundred Mbps and are continuously growing [17].

Networks-on-Chip (NoC) have emerged as the design paradigm for design of scalable on-chip communication architectures, providing better structure and modularity [1, 3, 7, 21]. Although NoCs solve the interconnect scalability issues, SoC integration is still a problem.

To enable cores to be designed and validated independently, computation and communication must be decoupled [20]. Decoupling requires well defined communication services [13]. Service guarantees are essential in many SoCs as numerous application domains require real-time performance [20]. Quality-of-Service (QoS) guarantees enable independent design and validation of every part of the SoC by ensuring that real-time application requirements are met under all circumstances [7].

Creating a NoC-based system with guaranteed services requires efficient mapping of cores and distribution of NoC resources. Design choices include core port to network port binding, routing of communication between cores and allotment of network channel capacity over time. These choices have significant impact on energy, area and performance metrics of the system.

Existing solutions rely on a multi-step approach where mapping is carried out before routing [7, 12, 19]. Routing and mapping objectives do hereby not necessarily coincide. The routing phase must adhere to decisions taken in the mapping phase which invariably limits the routing solution space. Mapping therefore significantly impacts energy and performance metrics of the system [12].

We propose a unified algorithm, called Unified MApping, Routing and Slot allocation (UMARS), that couples mapping, path selection and time-slot allocation, using a single consistent objective. The time-complexity of UMARS is low and experimental results indicate a run-time only 20% higher than that of path selection alone. We apply the algorithm to an MPEG decoder SoC, reducing area by 33%, power by 35% and worst-case latency by a factor four over a traditional multi-step approach.

The problem domain is described in Section 3 and formalized in Section 4. The UMARS algorithm, which solves the unified allocation problem under application constraints, is described in Section 5. Experimental results are shown in Section 6. Finally, conclusions are drawn in Section 7.

2. RELATED WORK

QoS routing objectives are discussed in [9, 22] and implications with common-practice load-balancing solutions are addressed in [16]. In addition to spatial, temporal characteristics are included in path selection in [8, 10].

The problem of mapping cores onto NoC architectures is addressed in [7, 11, 12, 17, 18, 19].

In [11] a branch-and-bound algorithm is used to map cores onto a tile-based architecture, aiming to minimize energy while bandwidth constraints are satisfied. Static xy routing is used in this work. In [12] the algorithm is extended to route with the objective of balancing network load.

In [17, 18, 19] a heuristic improvement method is used. An initial mapping is derived with objectives such as minimizing communication delay, area or power dissipation. This is succeeded by routing according to a predefined routing function. Routing and evaluation is repeated for pair-wise swaps of nodes in the topology, thereby exploring the design space in search for an efficient mapping. In [19] the algorithm integrates physical planning and QoS guarantees. Design space exploration is improved with a robust tabu search.

In all these approaches [11, 12, 17, 18, 19], multiple mapping and routing solutions are evaluated iteratively to mitigate the negative effects mapping decisions may have on routing.

A greedy non-iterative algorithm is presented in [7]. Mapping is done based on core clustering whereafter communication is routed using static xy routing.

Known mapping and routing algorithms that incorporate QoS guarantees [10, 19] assume static communication flows, where traffic does not vary with input data.

In this work, our methodology unifies the three resource allocation phases: spatial mapping of cores, spatial routing of communication, and the restricted form of temporal mapping that assigns time-slots to these routes. We consider the communication real-time requirements, and guarantee that application constraints on bandwidth and latency are met. The proposed solution is fundamentally different from [7, 11, 12, 17, 18, 19] in that mapping is no longer done *prior to* routing but instead *during* it. However, we compare UMARS only to [7], and a more extensive comparison with traditional algorithms [11, 12, 17, 18, 19] is of value.

3. PROBLEM DESCRIPTION

We assume that the application is mapped onto cores. The bandwidth and latency constraints of the application flows are determined beforehand by means of static analysis or simulation.

Our problem is to: 1) map those cores onto any given NoC topology, 2) statically route the communication and 3) allocate TDMA time-slots on network channels so that application constraints are met. Services are provided on the level of *flows* where a flow is a sequence of packets being sent from a source to a destination. Regular, as well as irregular topologies are supported to enable dedicated solutions.

Two important requirements can be identified and the onus is, in both cases, on the mapping and routing phases. Firstly, the constraints of individual flows must be satisfied. These constraints must hence be reflected in the selection of mapping, path and time slots such that proper resources are reserved. Secondly, all flows must fit within the available network resources. Failure in allocating a flow is attributable to non-optimal previous allocations or insufficient amounts of network resources. This calls for conservation of the finite pool of resources, namely the channels and their time-slots.

This paper shows how path selection can be extended to span also mapping and time-slot allocation. This enables the aforementioned requirements to be formulated as path selection constraints and optimization goals.

4. PROBLEM FORMULATION

The application is characterized by an application graph.

Definition 1. An *application graph* is a directed multigraph, $A(P, F)$, where the vertices P represent the set of *cores*, and the arcs F represent the set of *flows* between cores. More than a single flow is allowed to connect a given pair of cores and no core is isolated. Each flow $f \in F$ is associated with a minimum bandwidth constraint measured in number of slots, $b(f)$, and a maximum latency constraint, $l(f)$. Let $s(f)$ denote the source node of f and $d(f)$ destination node.

To be able to constrain mapping according to physical layout requirements, we group the cores in P and map groups instead of individual cores. UMARS is thereby forced to map certain cores to the same spatial location. The mapping groups correspond to a partition P_M of P, where the elements of P_M are *jointly exhaustive* and *mutually exclusive*. The equivalence relation this partition corresponds to, considers two elements in P to be equal if they must be mapped to the same spatial location. The equivalence class of a core p is hereafter denoted by $[p]$.

NoCs are represented by interconnection network graphs.

Definition 2. An *interconnection network graph* I is a strongly connected directed multigraph, $I(N, C)$. The set of vertices N is composed of three mutually exclusive subsets, N_R, N_{NI} and N_P containing *routers*, *network interfaces* (NI) and *core mapping nodes* as shown in Figure 1. The latter are dummy nodes to allow unmapped cores to be integrated in the interconnection graph. The number of core mapping nodes is equal to the number of core subsets to be mapped, $|N_P| = |P_M|$.

The set of arcs C is composed of two mutually exclusive subsets, C_R and C_P containing physical network channels and virtual mapping channels. Channels in C_R interconnect nodes in N_R and N_{NI} according to the physical router network architecture. Channels in C_P interconnect *every* node in N_P to *all* nodes in N_{NI}.

More than a single physical channel is allowed to connect a given pair of routers. However, an NI n_{NI} is always connected to a single router through one egress channel $c_E(n_{NI}) \in C_R$ and one ingress channel $c_I(n_{NI}) \in C_R$, as depicted in Figure 1.

The time division of network channel capacity is governed by slot tables. These tables are used to set up *pipelined virtual circuits* and divide bandwidth between flows [20]. A slot table is a sequence of elements in $T = F \cup \{\varnothing\}$. Slots are either occupied by a flow $f \in F$ or empty, represented by $\varnothing$. The number of residual slots in a slot table t is denoted $\sigma(t)$. The same slot table size S_T is used throughout the entire network.

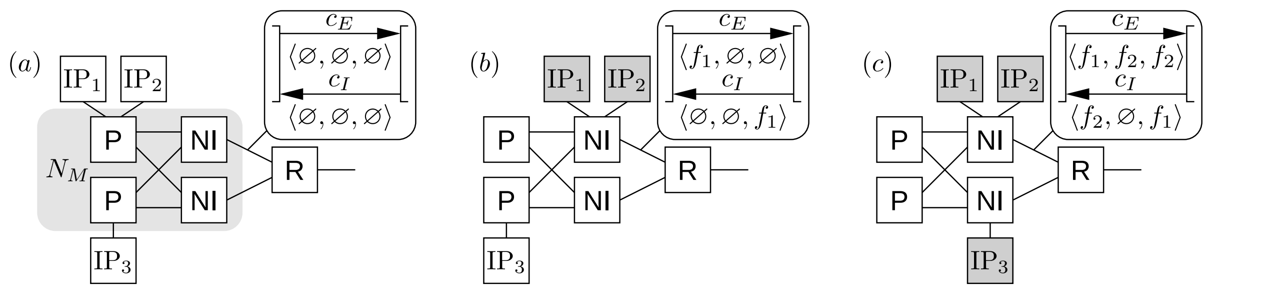
Each channel $c \in C$ is associated with the bandwidth not yet reserved (residual bandwidth) measured in number of slots, $\beta(c)$, and a slot table, $t(c)$. Let $s(c)$ denote the source node of c and $d(c)$ destination node.

As residual bandwidth and slot tables change over iterations, I is subscripted with an index. I_0 denotes the initial network where $\beta(c) = S_T$ and every slot in $t(c)$ is empty for every channel $c \in C$.

Definition 3. A path $\pi \in \operatorname{seq} C$ from source $n_s \in N$ to destination $n_d \in N$ is a non-empty sequence of channels $\langle c_1, \ldots, c_k \rangle$ such that:

1. $d(c_i) = s(c_{i+1})$ for $k = 1 \,.\,. k - 1$
2. $s(\mathit{head}\ \pi) = n_s$ and $d(\mathit{last}\ \pi) = n_d$.

A path $\pi = \langle c_1, \ldots, c_k \rangle$ is associated with an aggregated slot table $t(\pi)$. Every channel slot table $t(c_i), i = 1 \,.\,. k$, is shifted cyclically $i - 1$ steps left and a slot in $t(\pi)$ is empty iff it is empty in *all* shifted slot tables [20].



$$\begin{array}{ccccccccc} I_0 & \rightarrow & I_1 & \rightarrow & \dots & \rightarrow & I_k \\ map_0 & \rightarrow & map_1 & \rightarrow & \dots & \rightarrow & map_k \\ \varnothing = P'_0 & \subseteq & P'_1 & \subseteq & \dots & \subseteq & P'_k = \{\mathrm{IP}_1, \mathrm{IP}_2, \mathrm{IP}_3\} = P \end{array}$$

Figure 1: Iteration and successive refinement of mapping and interconnection network

Definition 4. For a source and destination node $n_s, n_d \in N$, $\Pi(n_s, n_d)$ is the set of all possible paths from n_s to n_d.

The NIs and core mapping nodes together form the set of *mappable nodes* $N_M = N_{NI} \cup N_P$ as shown in Figure 1(a). N_M contains all nodes to which the elements of P_M can be mapped. We define a mapping function, $map_i : P_M \rightarrow N_M$, that maps sets of cores (the elements in P_M) to mappable nodes. Like I, this function is iterated over, hence the index. Our starting point is an initial mapping, map_0, where every $[p] \in P_M$ is mapped to a unique $n_P \in N_P$.

As seen in Figure 1(a), the range of map_0 initially covers only N_P. As the algorithm progresses (b), the range of map_i covers both N_P and N_{NI} partially. Successive iterations of map_i progressively replace elements of N_P with elements of N_{NI} until a final mapping is derived (c), where the range of map_k contains elements of N_{NI} exclusively.

Let the set of mapped cores P'_i denote those elements of P where $map_i([p]) \in N_{NI}$. From our definition of map_0 it follows that $P'_0 = \varnothing$.

4.1 UMARS contribution

We now introduce a major change from previous work and formulate mapping and path selection problem as a pure path selection problem.

Given an interconnection network I_0 and an application graph A, we must select a path π for every flow $f \in F$ such that bandwidth (1) and latency (2) requirements of the flow are met without overallocating the network channels (3).

$$\text{bandwidth of } t(\pi) \geq b(f) \qquad (1)$$

$$\text{latency of } t(\pi) \leq l(f) \qquad (2)$$

$$\beta(c) \geq 0, \forall c \in C \qquad (3)$$

The theory required to derive worst-case bandwidth and latency from a slot table is covered in [5].

5. UNIFIED MAPPING AND ROUTING

The outmost level of UMARS is outlined in Algorithm 5.1 and briefly introduced here, whereafter further explanations follow in Sections 5.1 and 5.2.

UMARS iterates over the monotonically decreasing set of unallocated flows F'_i and never back-tracks to reevaluate an already allocated flow, as seen in Step 2a. This results in a low time-complexity at the expense of optimality. The flow f is selected based on the current mapping map_i and network I_i. When a path π is selected for f in Step 2b, the first and last channel implicitly determine what NI $s(f)$ and $d(f)$ should be mapped to respectively. Time-slots are allocated to f on π whereafter map_i and I_i are refined to reflect the new state. The procedure is repeated until all flows are allocated.

Algorithm 5.1 Allocation of all flows F

```
1. Let the set of unallocated flows $F'_0 = F$
2. While $F'_i \neq \varnothing$:
    (a) Get flow $\arg\max_{f \in F''} b(f)$
    (b) Select a path $\pi \in \Pi(s(f), d(f))$
    (c) $F'_{i+1} = F'_i \setminus \{f\}$
```

5.1 Flow traversal order

We order flows by bandwidth requirements as it: 1) helps in reducing bandwidth fragmentation [16], 2) is important from an energy consumption and resource conservation perspective since the benefits of a shorter path grow with communication demands [12], 3) gives precedence to flows with a more limited set of possible paths [12].

Ordering by $b(f)$ alone may affect resource consumption negatively as clusters of communicating cores are disregarded. Consideration is taken by limiting the selection to flows having $s(f)$ or $d(f)$ mapped to a node in N_{NI}. As a result, every cluster of communicating cores have their flows allocated in sequence. A similar approach is used in [17, 18] where the next core is selected based on communication to already mapped cores.

Due to the nature of the least-cost path selection algorithm, explained in Section 5.2.2, we restrain the domain even more and only consider flows where $s(f) \in P'_i$. This restriction can be removed if path selection is done also in the reverse direction, from destination to source.

The next flow is chosen according to Equation (4), where $f \in F''_i$ iff $f \in F'_i \wedge s(f) \in P'_i$. When the latter condition is not fulfilled by any flow, the entire F'_i is used as domain.

$$\arg\max_{f \in F''} b(f) \qquad (4)$$

5.2 Path selection

When a flow f is chosen, we proceed to Step 2b of Algorithm 5.1 and select a path for f. This is done according to Algorithm 5.2, briefly presented here, followed by in-depth discussions in Sections 5.2.1 through 5.2.5.

Path selection for f is composed of three major tasks: 1) Speculative bandwidth reservations for f are restored in Steps 1 and 2 to have I_i reflect what resources are available to f prior to its allocation. Speculative reservations are required as interdependent flows are not allocated simultaneously and are further discussed in Section 5.2.1. 2) A path from $s(f)$

Algorithm 5.2 Path selection for a given f

```
1. If $s(f) \in P'_i$, restore bandwidth reservation on egress channel by adding $\lceil b(f) \rceil$ to $\beta(c_E(map_i([s(f)])))$
2. If $d(f) \in P'_i$, restore bandwidth reservation on ingress channel by adding $\lceil b(f) \rceil$ to $\beta(c_I(map_i([d(f)])))$
3. Select a constrained least-cost path $\pi_s$ from $map_i([s(f)])$ to the router $n_R \in N_R$ with lowest cost. Arity is used to distinguish between routers with equal cost.
4. If $s(f) \notin P'_i$, then
   (a) Refine $map_{i+1} = map_i \oplus \{[s(f)] \mapsto d(head\ \pi_s)\}$
   (b) Reserve egress bandwidth for all flows emanating from $[s(f)]$ by subtracting $\sum_{f_E \in F_E} \lceil b(f_E) \rceil$ from $\beta(c_E(d(head\ \pi_s)))$ where $f_E \in F_E$ iff $s(f_E) \in [s(f)]$ and $f_E \neq f$
   (c) Reserve ingress bandwidth for all flows incident to $[s(f)]$ by subtracting $\sum_{f_I \in F_I} \lceil b(f_I) \rceil$ from $\beta(c_I(d(head\ \pi_s)))$ where $f_I \in F_I$ iff $d(f_I) \in [s(f)]$
5. Select a constrained least-cost path $\pi_d$ from $d(last\ \pi_s)$ to $map_i([d(f)])$
6. If $d(f) \notin P'_i$, then
   (a) Refine $map_{i+1} = map_i \oplus \{[d(f)] \mapsto s(last\ \pi_d)\}$
   (b) Reserve egress bandwidth for all flows emanating from $[d(f)]$ by subtracting $\sum_{f_E \in F_E} \lceil b(f_E) \rceil$ from $\beta(c_E(s(last\ \pi_d)))$ where $f_E \in F_E$ iff $s(f_E) \in [d(f)]$
   (c) Reserve ingress bandwidth for all flows incident to $[d(f)]$ by subtracting $\sum_{f_I \in F_I} \lceil b(f_I) \rceil$ from $\beta(c_I(s(last\ \pi_d)))$ where $f_I \in F_I$ iff $d(f_I) \in [d(f)]$ and $f_I \neq f$
7. Select a constrained set of slots $T_S$ in $t(\pi)$ for the complete path $\pi = \pi_s \frown \pi_d$ and update $t(c), \forall c \in \pi$. Do a final bandwidth reservation by subtracting $|T_S|$ from $\beta(c), \forall c \in \pi$.
```

to $d(f)$ is selected in Steps 3 and 5, a procedure elaborated on in Section 5.2.2. If $s(f)$ or $d(f)$ are not yet mapped to NIs, these steps include refinement of map_i, which is covered in Section 5.2.4. If map_i is refined, then bandwidth reservations are made on ingress and egress channels for flows other than f now having their source or destination mapped to an NI. 3) Time-slots are selected and reserved on the resulting path π, as discussed in Section 5.2.5.

5.2.1 *Bandwidth reservation*

When $s(f)$ for a flow f is mapped to an NI, the communication burden placed on the ingress and egress channels of the NI is not determined by f only. As every p in $[s(f)]$ is fixed to this NI, the aggregated communication burden of all flows incident to those cores is placed on the ingress channel. The egress channel similarly has to accommodate all flows emanating from those cores. When $d(f)$ is mapped, all flows to or from $[d(f)]$ must be accounted for accordingly.

Failing to acknowledge the above might result in overallocation of network resources. Numerous flows, still not allocated, may be forced to use the ingress and egress channel due to an already fixed mapping. An NI would thereby be associated with an implicit load, not accounted for when evaluating possible paths. We make this load explicit by exploiting knowledge of ingress-egress pairs. Although we have no knowledge of exactly what time slots will be needed by future flows, we can estimate the bandwidth required by $\lceil b(f) \rceil$ and incorporate average load $\beta(c)$ in the cost function, further discussed in Section 5.2.3.

Steps 1 and 2 of Algorithm 5.2 restore the speculative reservations for f on egress and ingress channel to have I_i reflect what resources are available prior to its allocation.

The corresponding bandwidth reservations on egress and ingress channels are carried out in Steps 4b, 4c and Steps 6b, 6c for source and destination NI respectively.

5.2.2 *Selecting constrained least-cost path*

Steps 3 and 5 of Algorithm 5.2 select a constrained least-cost path using Dijkstra's algorithm.

Two minor modifications are done to the standard relaxation procedure, where π_p denotes the partial path from $s(f)$ to the current node: 1) Search space is pruned by discarding emanating channels where $\beta(c) < b(f)$ or $\sigma(t(\pi_p \frown \langle c \rangle)) < b(f)$. Channels that cannot meet bandwidth constraints are thereby omitted. 2) As the final path must contain only physical network resources, channels in C_P may only be the first or last element of a path. Hence, if $d(last\ \pi_p) \in N_P$ then all emanating channels are discarded.

The NI architecture requires a path to incorporate at least one physical channel as packets cannot turn around inside an NI. From a least-cost perspective the best path from an NI to itself would be the empty path and we force the algorithm into leaving the NI by doing path selection in two steps.

The first part of the path π_s is selected in Step 3 of Algorithm 5.2. We start at $s(f)$ and find the router with the lowest cost. If several such routers exist, then arity is used to distinguish between them. Routing flexibility is thereby maximized and the flows with the highest communication volume have their $s(f)$ and $d(f)$ mapped to NIs connected to high arity routers as suggested in [18].

The second part of the path π_d is selected in Step 5, starting where π_s ended. From there we continue to the location where $d(f)$ is currently mapped. The complete path is then just the two parts concatenated, $\pi = \pi_s \frown \pi_d$.

Deriving π as two separate least-cost parts might, without further care, lead to a path which is not the least-cost path in $\Pi(s(f), d(f))$ as minimization is done on the parts in isolation. However, if a flow f has $s(f) \in P'_i$ then there is only one possible least-cost router and hence only one possible π_s. As this π_s is a part of any path in $\Pi(s(f), d(f))$ and π_d is a least-cost path, π must be a least-cost path in $\Pi(s(f), d(f))$. We therefore prefer allocating flows where $s(f) \in P'_i$, as discussed in Section 5.1.

5.2.3 *Choice of cost function*

The cost function plays a critical role in meeting the requirements discussed in Section 3. It therefore reflects both resource availability and resource utilization. We select a path with a low contention (high probability of successful allocation) and at the same time try to keep the path length short, not to consume unnecessarily many resources. Similar heuristics are suggested in [14, 15, 22].

Double objective path optimization in general is an intractable problem [9]. Combining objectives in one cost function allows for tractable algorithms at the cost of optimality. We therefore use a linear combination of the two cost measures, where two constants Γ_c and Γ_h control the importance (and normalization) of contention and hop-count respectively.

Contention is traditionally incorporated by making channel cost inversely proportional to residual bandwidth, $\frac{1}{\beta(c)}$, thereby considering only average load. When using pipelined virtual circuits [20], average load is not reflecting what resources are available to the current flow. Not even the slot table $t(c)$ itself provides an accurate view. We exploit knowledge of the partial path π_p traversed so far and determine contention cost for a channel c by how much $t(c)$ reduces the amount of available slots compared to $t(\pi_p)$ if c is traversed. Available bandwidth is incorporated by taking the maximum of the two as contention measure, according to Equation (5).

$$\Gamma_c \max\{S_L - \beta(c), \sigma(t(\pi_p)) - \sigma(t(\pi_p \frown \langle c \rangle))\} + \Gamma_h \qquad (5)$$

Channels in C_P must not contribute to the path cost, as they are not physical interconnect components. We therefore make them *zero-cost* channels.

5.2.4 Refining mapping function

When a path π_s has been selected for a flow f, we check in Step 4 of Algorithm 5.2, whether $s(f)$ is not yet mapped to an NI. If not, π_s decides the NI to which the core is to be mapped. We therefore refine the current mapping function with the newly determined mapping to a node in N_{NI} as seen in Step 6a. This refinement is fixed and every core in $[s(f)]$ is now in P'_i.

Correspondingly we check if $d(f)$ is not yet mapped to an NI in Step 6 and if not, refine the mapping according to π_d in Step 6a.

5.2.5 Resource reservation

When the entire path π is determined in Step 7 of Algorithm 5.2, we deduce the slots available to f by looking at $t(\pi)$. From the empty slots we select a set of slots T_S such that bandwidth and latency requirements of f are met [5]. All channels $c \in \pi$ are then updated with a new $t(c)$ and $\beta(c)$. Slot tables hereafter reflect what slots are reserved to f and $\beta(c)$ is updated with the actual number of slots used.

5.3 Algorithm termination

With each refinement of the map_i, zero, one or two additional sets of cores will be mapped to elements of N_{NI} instead of N_P, hence $P'_{i+1} \supseteq P'_i$, as depicted in Figure 1.

THEOREM 1. *$\exists k$ such that all cores are mapped to NIs, $P'_k = P$.*

PROOF. When a flow is f allocated, map_i will be refined so that $s(f)$ and $d(f)$ are guaranteed to be in P'_i. Hence, for every allocated flow $f \notin F'_i$ we know that $s(f), d(f) \in P'_i$.

When all flows are allocated $F'_k = \varnothing$, $s(f)$ and $d(f), \forall f \in F$ will be in P'_k. As no isolated cores are allowed in A it follows that $P = P'_k$. $\square$

5.4 Algorithm complexity

Due to the greedy nature of UMARS the time-complexity is very low, as seen in Equation (6). The expression is dominated by the first term that is attributable to Dijkstra's algorithm, used for path selection. Experiments indicate that algorithm run-time is only 20% higher than that of load-balancing path selection alone.

$$\mathcal{O}(|F|(|C| + |N|\log|N|)) + \mathcal{O}(|F|(|F| + |P| + S_T)) \qquad (6)$$

6. EXPERIMENTAL RESULTS

A cost function where $\Gamma_c = 1$ and $\Gamma_h = 1$ is used throughout the experiments. Those values favor contention-balancing over hop-count as the slot table size is an order of magnitude larger than network diameter in all use-cases. All results are compared with the traditional multi-step algorithm in [7], referred to as *original*.

For comparison, only mesh topologies are evaluated. For a given slot table size S_T, all unique $n \times m$ router networks with less than 25 routers were generated in increasing size order. For every such router network, up to three NIs were attached to each router until all application flows were allocated, or allocation failed. Slot table size was incremented until allocation was successful.

Each design was simulated during 3×10^6 clock cycles in a flit-accurate SystemC simulator of our NoC, using traffic generators to mimic core behavior.

The *mpeg* use-case is a MPEG codec SoC, further described in Section 6.2. The *uniform* use-case features all-to-all communication with 20 cores and a total aggregated bandwidth of 750 Mbps per core. The remaining use-cases are internal designs, all having a hot-spot around a limited set of cores.

6.1 Evaluation experiments

Silicon area requirements are based on the model presented in [6], assuming a 0.13 μm CMOS process. Figure 2 shows that area requirements can be significantly reduced. Up to 33% in total area reduction is observed for the experiment applications. Slot table sizes are reduced why the buffer requirements, analytically derived as described in [7], decrease, and area savings up to 31% are observed for the NIs. The router network is reduced between 30% and 75%, but the impact on total area is much smaller.

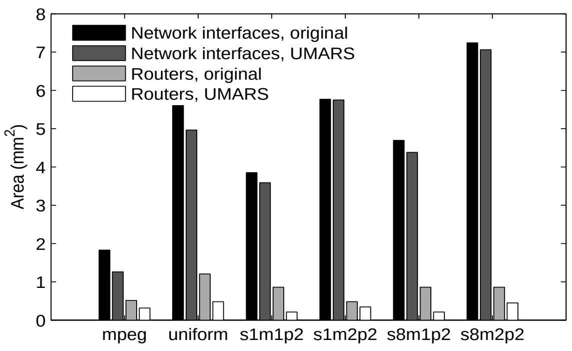


Figure 2: Comparison of area requirements.

The relative energy consumption of the router network, calculated according to the model in [4] is depicted in Figure 3. As the application remains the same and essentially the same bits are being communicated, the savings in energy consumption is attributable to flows being allocated on paths with fewer hops. There is a clear correlation between energy saving ratio and relative reduction in number of routers. However, as the smaller router network is used more extensively, energy is reduced less than the number of routers.

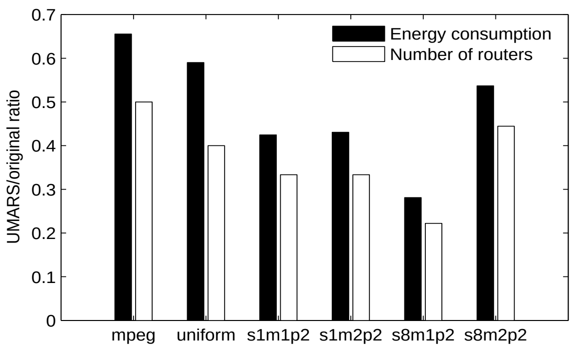


Figure 3: Comparison of energy consumption.

Figure 4 shows the average utilization of channels emanating from NIs and routers respectively. As expected, utilization increase as router network size is reduced and UMARS consequently improves both NI and router utilization. Time-division-multiplexed circuits imply bandwidth discretization, leading to inevitable over-allocation and complicating the task of achieving high utilization. This together with unbalanced hot-spot traffic, leaving some parts of the network lightly loaded and others congested, lead to inherent low utilization in some of the example use-cases. Note that utilization is only to be optimized after all constraints are met.

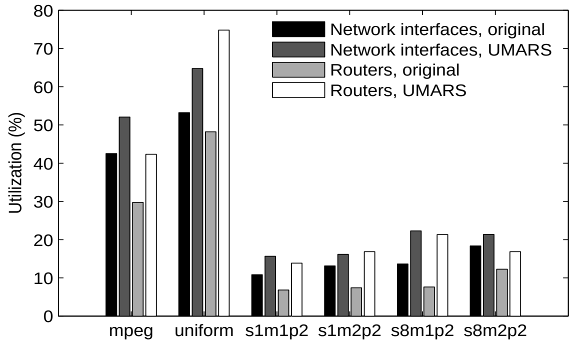


Figure 4: Comparison of NoC resource utilization.

6.2 An MPEG application

An existing MPEG codec SoC with 16 cores constitutes our design example and results are shown in Table 1. The architecture uses a single external SDRAM with three ports to implement all communication between cores. A total of 42 flows tie the cores together. Using the design flow presented in [7] (clustered mapping, *xy* routing and greedy slot allocation) results in a 2×3 mesh, referred to as *clustering* in Table 1, with a total estimated area of 2.35 mm^2. For comparison, a naive mapping with one core partition per NI is almost double in size, whereas the worst-case write latency remains more or less unaffected.

A manually optimized mapping manages to reduce the network area with 21% and an almost four-fold reduction of the average worst-case write latency is observed [7].

UMARS arrives at a mesh of equal size to what is achieved using the manually optimized mapping. Fewer NIs are needed leading to reductions in router area. Smaller buffer requirements, attributable to less bursty time-slot allocation, results in reduced NI area. Total area is reduced by 17% and average worst-case latency by 4% compared to the optimized handcrafted design. The solution is achieved in less than 100 ms on a 500 MHz Solaris UltraSparc IIe. Only a 20% increase in run-time is observed when compared to pure load-balancing path selection, without mapping and slot allocation.

Table 1: Comparison of MPEG NoCs

Generation	Mesh	Slots	NI area	Router area	Total area	Area diff	Avg wc latency
clustering	2x3	128	1.83	0.51	2.35	ref	1570 ns
naive	3x6	128	2.17	2.32	4.49	+91%	1583 ns
optimized	1x3	8	1.51	0.35	1.86	−21%	399 ns
UMARS	1x3	8	1.26	0.32	1.57	−33%	383 ns

7. CONCLUSION AND FUTURE WORK

In this work we have presented the UMARS algorithm which integrates the three resource allocation phases: spatial mapping of cores, spatial routing of communication and TDMA time-slot assignment. The algorithm is decomposed into a hierarchical structure where mapping is no longer done *prior to* routing but instead *during* it. UMARS improves over existing mapping and routing algorithms by using a single consistent objective-function.

The time-complexity of UMARS is low and experimental results indicate a run-time only 20% higher than that of path selection alone.

We apply the algorithm to an MPEG decoder SoC, improving area 33%, power 35% and worst-case latency by a factor four over a traditional multi-step approach.

The importance of the flow traversal order and the objective function are not yet fully evaluated and both play a critical role in improving on the moderate results achieved in some use-cases.

To allow a more extensive design space exploration for both mapping and routing, UMARS can be extended to a k-path algorithm, enabling a trade-off between complexity and optimality.

8. REFERENCES

[1] L. Benini and G. de Micheli. Networks on chips: A new SoC paradigm. *IEEE Comp.*, 35(1), 2002.
[2] D. Bertozzi *et al.* NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. *Trans. on Parallel and Distr. Syst.*, 16(2), 2005.
[3] W. J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Proc. DAC*, 2001.
[4] J. Dielissen *et al.* Power measurements and analysis of a network-on-chip. Technical Report NL-TN-2005-0282, Philips Research Laboratories, Eindhoven, 2005.
[5] O. P. Gangwal *et al. Dynamic and Robust Streaming In And Between Connected Consumer-Electronics Devices*, Building Predictable Systems on Chip: An Analysis of Guaranteed Communication in the Æthereal Network on Chip. Kluwer, 2005.
[6] S. González Pestana *et al.* Cost-performance trade-offs in networks on chip: A simulation-based approach. In *Proc. DATE*, 2004.
[7] K. Goossens *et al.* A design flow for application-specific networks on chip with guaranteed performance to accelerate SOC design and verification. In *Proc. DATE*, 2005.
[8] R. Guérin and A. Orda. Networks with advance reservations: The routing perspective. In *Proc. INFOCOM*, 2000.
[9] R. Guérin *et al.* QoS routing mechanisms and OSPF extensions. In *GLOBECOM*, volume 3, 1997.
[10] W. H. Ho and T. M. Pinkston. A methodology for designing efficient on-chip interconnects on well-behaved communication patterns. In *Proc. HPCA*, 2003.
[11] J. Hu and R. Mărculescu. Energy-aware mapping for tile-based NoC architectures under performance constraints. In *Proc. ASP-DAC*, pages 233–239, 2003.
[12] J. Hu and R. Mărculescu. Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures. In *Proc. DATE*, 2003.
[13] K. Keutzer *et al.* System-level design: Orthogonalization of concerns and platform-based design. *Trans. on CAD of Integrated Circuits and Systems*, 19(12), 2000.
[14] K. Kowalik and M. Collier. Should QoS routing algorithms prefer shortest paths? In *Proc. ICC*, 2003.
[15] Q. Ma and P. Steenkiste. On path selection for traffic with bandwidth guarantees. In *Proc. ICNP*, 1997.
[16] I. Matta and A. Bestavros. A load profiling approach to routing guaranteed bandwidth flows. In *Proc. INFOCOM*, 1998.
[17] S. Murali and G. de Micheli. Bandwidth-constrained mapping of cores onto NoC architectures. In *Proc. DATE*, 2004.
[18] S. Murali and G. de Micheli. SUNMAP: A tool for automatic topology selection and generation for NoCs. In *Proc. DAC*, 2004.
[19] S. Murali *et al.* Mapping and physical planning of networks on chip architectures with quality of service guarantees. In *Proc. ASP-DAC*, 2005. (to appear).
[20] E. Rijpkema *et al.* Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip. *IEEE Proc. Comp. and Dig. Techn.*, 150(5), 2003.
[21] M. Sgroi *et al.* Addressing the system-on-a-chip interconnect woes through communication-based design. In *Proc. DAC*, 2001.
[22] R. Widyono. The design and evaluation of routing algorithms for real-time channels. TR-94-024, Univ. of Calif. at Berkeley & Int'l Comp. Sci. Inst., 1994.