PowerMax: An Automated Methodology for Generating Peak-Power Traffic in Networks-on-Chip

Ioannis Seitanidis*, Chrysostomos Nicopoulos† and Giorgos Dimitrakopoulos*
*Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece
†Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus

***Abstract*—Early estimation of the peak power consumption of a system under development is crucial in assessing the design's reliability and thermal profile, and for benchmarking various architectural options and chip-level power management features. In this paper, we present a versatile power-virus generation technique for Networks-on-Chip (NoC), which allows the designer to quantify the realistically attainable peak power consumption, in order to efficiently guide the design process. The proposed *PowerMax* methodology generates appropriate network traffic patterns that cause peak power consumption within the NoC. More importantly, PowerMax is a fully automated high-level methodology that can be applied to any network topology and any routing algorithm. The proposed technique maximizes both the network utilization and the data switching activity, thereby causing, on average, 5.5× higher power consumption than synthetic traffic patterns with random behavior. PowerMax can be used as a stand-alone tool to test the power characteristics of the NoC, or it can be embedded in other system-level power-virus applications.**

I. Introduction

Technology scaling has enabled digital system designs with billions of transistors integrated on a single chip. Besides the abundance of resources, which has been the driving force behind the multicore archetype, key (micro-)architectural decisions are dictated by *power* constraints. Excessive power dissipation increases packaging/cooling costs, reduces battery life in mobile devices, and adversely affects hardware reliability, primarily due to elevated temperatures [1]. The increasingly stringent requirement to adhere to a given power budget has rendered power consumption a first-class design constraint [2]. Hence, it is imperative for system architects to understand and accurately quantify their design's power usage from the early stages of the design process [3].

One particular salient attribute is of paramount importance: the *peak* (i.e., worst-case) power consumption [4], [5]. Assuming a fake scenario whereby all circuit nodes switch simultaneously will inevitably result in an overly pessimistic estimate. Instead, it is crucial to *accurately* quantify the *realistically* attainable peak power consumption, in order to efficiently guide the design process. Note that both the system's maximum performance and implementation costs (power delivery, packaging, and cooling) are directly impacted by this worst-case power consumption. A pessimistic peak power estimate will unnecessarily curtail performance, while an optimistic estimate could potentially lead to reliability problems.

The accurate identification of worst-case power consumption is extremely challenging, especially as chips become more complex. To further compound the problem, the worst-case power usage of a system is not simply the sum of the maximum power of each component, due to under-utilization and contention for shared resources. Hence, the peak power consumption must be estimated using a stimulus that is realistic and resides within the functionally feasible workload space of the system under evaluation. Typically, designers rely on hand-crafted, custom-made *power viruses* (also referred to as "stressmarks," or "powermarks") to estimate a system's peak power consumption [6], [7], [8]. However, the task of manually constructing a program for a specific architecture is very cumbersome and error-prone. Most importantly, the generated virus is not guaranteed to yield the maximum possible power usage. Thus, *automatic* approaches to the generation of effective power viruses are highly desirable.

Power viruses have been explored within the realm of CPUs, main memory, and off-chip I/O. One notable absentee is the *on-chip* network; there is currently no methodology to identify the peak power consumption in the system's Network-on-Chip (NoC) backbone. The NoC has become the de facto communication medium in multi-core setups, due to its modularity and scalability traits. Being an integral part of the system, the NoC provides connectivity across the chip, and it synergistically contributes to overall system performance [9]. Given the NoC's functional/performance criticality, it is obvious that peak-power analysis cannot ignore such an elemental actor.

This paper presents, for the first time (to the best of our knowledge), a fully automated high-level methodology to generate appropriate traffic patterns that cause peak power consumption within the NoC, by leveraging the intertwined and complementary roles of high network utilization and data switching activity. The proposed framework, aptly called *PowerMax*, can generate a peak-power "traffic virus" for *any network topology* and *any routing algorithm*. The generated traffic patterns are realistic – i.e., feasible for the particular topology and routing algorithm – and ensure that the resulting peak power consumption accurately reflects the potential of the underlying NoC configuration.

The proposed methodology is two-pronged; it combines the contributions of high network utilization – through the generation of appropriate traffic patterns – and data switching activity, in a user-controllable manner. The interaction of these two facets is non-trivial, and it is a key feature of the problem formulation. For example, very high network utilization alone is not enough to generate peak power, because, if the data payloads happen to be "favorable" (i.e., yielding low switching activity), the NoC power consumption may be quite low. Moreover, high network utilization is often confused with low

I.Seitanidis is supported by the PhD scholarship of Alexander S. Onassis foundation.

978-1-5090-0172-9/15/$31.00

saturation throughput, i.e., highly congested networks. However, high congestion may severely affect certain NoC regions, but may leave other regions under-utilized. Hence, formulating the generic problem of peak power consumption within the NoC involves the intricate interaction of all aforementioned nuances, which is one of the key contributions of PowerMax.

PowerMax can be used within multiple contexts: (1) as a stand-alone tool to test the peak power characteristics of a NoC under development; (2) as a benchmark to assess low-power NoC architectures [10]; and (3) in conjunction with other power viruses that stress other components of the system, when evaluating system-level peak power consumption.

Extensive and detailed experimental evaluations using various NoC topologies and routing algorithms validate the effectiveness of the proposed methodology. PowerMax is demonstrated to cause an average of 5.5× higher power consumption than randomly selected traffic patterns, or patterns that result in high saturation throughput.

II. Problem Formulation

The dynamic power consumption of a NoC that operates at a pre-determined clock frequency and voltage is directly proportional to (a) the capacitance of every gate-level node of the circuit, and (b) the switching activity of these nodes [11]. NoC components also consume static/leakage power, but this becomes relevant only in *idle* components. The goal of any power virus is to cause full utilization of all components, thereby rendering static/leakage power irrelevant, since no components are left idle to consume static/leakage power. Consequently, leakage power is ignored when trying to discover a realistic peak-power consumption scenario.

A. The interplay of contention and data switching activity

The peak power consumption of a NoC jointly depends on (1) the network component utilization, and (2) the data switching activity caused by the traffic flowing inside the NoC every cycle. A network component – e.g., the links, the buffers, the crossbar, etc. – is considered utilized, as long as it performs a useful operation in each cycle. The amount of power consumed is directly proportional to the switching activity caused by the bits of the traversing flits. For example, let us assume a NoC link that transfers useful flits in every clock cycle, but those flits happen to have almost the same bit-level profile. In such a case, even though the experienced utilization is maximized, the actual power consumption remains very low, due to minimal switching activity. The same holds for most NoC components like the router's buffers and the multiplexers of the crossbar.

The power consumption of every NoC component – considering both utilization and data switching activity – is directly related to the effect of *contention* and *multiplexing*. Whenever at least two flows compete for the same resource, e.g., a NoC link through a router's output port, they will possibly gain access to the shared resource in a time-multiplexed manner, depending on the employed arbitration policy. In this case, there is no way to predict the data switching profile seen at the output of the shared resource, since the output data stream is the result of multiplexing-in-time of two (or more) flows that are unrelated in terms of their data properties.

An example of the unpredictability of the output data stream is shown in Fig. 1, assuming two 4-bit-wide data flows

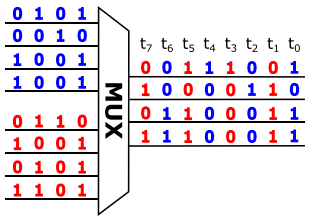

Fig. 1. The process of multiplexing different data flows "corrupts" the data switching profile of each individual incoming data stream, and can possibly lead to very low dynamic power consumption.

traveling through a multiplexer. Although the two incoming streams exhibit considerable switching activity when viewed independently of each other, the arbitrated traffic that passes to the output of the multiplexer exhibits very few bits switching in every clock cycle.

Additionally, in NoCs that consist of routers with equal numbers of input and output ports (i.e., the most common case in the majority of designs), contention for the same output port inevitably leaves at least another output port unutilized. This implies that at least one NoC output link, together with the input buffers at the end of said link (in the downstream router), are not used for a certain number of cycles, which directly translates to zero dynamic power.

Therefore, in order to guarantee that (a) all NoC components will be fully utilized in each cycle, and (b) the data switching activity caused within each component is directly controllable by the sources of the NoC (not affected by intermediate contention/multiplexing points), we need to derive *contention-free traffic patterns, which utilize all network links*. Full-link utilization effectively causes full buffer and crossbar utilization in each NoC router, thus approaching, as much as possible, the realistically attainable peak power of the NoC (when driven by appropriate data). The desired traffic patterns would allow the network endpoints (sources/destinations) to transmit and receive a new flit every cycle (100% throughput), while controlling the data switching profile of the injected traffic and avoiding the unpredictability caused by multiplexing of unrelated data streams.

B. Permutation traffic, contention, and network utilization

To identify contention-free traffic patterns, we start by removing contention at the endpoints of the network, i.e., the traffic injected by each source is directed to a different destination. Therefore, out of all possible traffic patterns, we need to identify those *permutation traffic patterns* where (a) traffic is exchanged between unique source-destination pairs, (b) no flow contention is caused inside the network, and (c) all NoC links are simultaneously fully utilized.

An example of such a permutation traffic pattern is shown in Fig. 2(a) for a 3×3 2D mesh NoC, where the paths of the injected flows are determined by the XY routing algorithm. The terminal nodes are depicted as circles in the figure. The numbers beneath each terminal node indicate the source/destination pair of the flow originating from that terminal node, e.g., the "1→8" designation below node 1 indicates that the traffic generated at this node goes to node 8. Each source/destination pair is unique. The selected traffic pattern allows for full NoC component utilization (injection and ejection throughput can be 100%), while, at the same

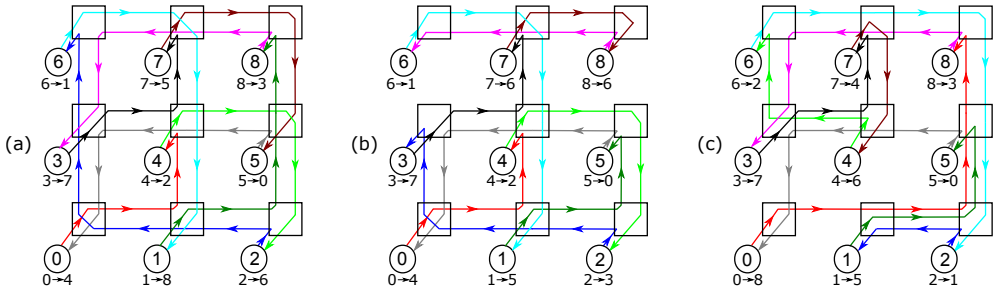

Fig. 2. Three different permutation traffic patterns in a 3×3 2D mesh: (a) A pattern that causes maximum NoC component utilization, which yields peak power consumption (i.e., the desired pattern); (b) A pattern that leaves certain links idle, even though it achieves full injection/ejection throughput and avoids contention; (c) A pattern that merely congests certain network channels, while leaving parts of the NoC unutilized.

time, the sources can control, from outside the network, the data switching activity inside the NoC.

On the contrary, the permutation traffic shown in Fig. 2(b) leaves 4 links idle (links 6→3, 3→6, 5→8, and 8→5), even though it achieves full injection/ejection throughput, without contention, on the remaining links.

Peak power consumption should not be confused with network congestion and worst-case throughput. The permutation traffic pattern shown in Fig. 2(c) triggers the worst-case throughput in this NoC. The traffic that is produced using the method in [12], maximizes the load on certain channels, in order to stress the network and highlight its worst-case throughput. However, even though the NoC is congested, many portions of it remain under-/un-utilized, and exhibit lower power consumption. For example, the 1→0 and 4→5 links are not used at all. Moreover, some of the utilized links are not *fully* utilized, i.e., they do not receive a flit every cycle. For instance, the 0→1 link is only used with 50% throughput, because the red flow going from node 0 to node 8 encounters contention in the downstream node 1, and is forced to share the 1→2 link with the dark green flow going from node 1 to node 5. Since the red flow only uses 50% of the bandwidth of the 1→2 link, the throughput of the 0→1 link also falls to 50%, i.e., it remains idle for 50% of the time. Hence, the approach of simply triggering worst-case NoC throughput results in both idle links, and links that are not fully utilized, due to downstream contention. Once a link is un-/under-utilized, the router components connected to the link's endpoints (multiplexers, pipeline registers, flow control logic, and input buffers) also remain idle, or under-utilized.

By computing the average link utilization in the network, one can get a feeling of how much the corresponding traffic patterns stress the network, and how close they can drive it to its peak power consumption. For example, the traffic patterns in Figs. 2(b) and (c) produce an average link utilization of 83% and 68%, respectively, counting also the utilization of the links connecting the terminal nodes with the network.

C. Number of appropriate permutation traffic patterns

The permutation traffic patterns that yield extremely high link utilization constitutes an extremely small subset of the entire set of possible traffic patterns. To gain insight as to how small this subset is, we performed two experiments.

In the first case, we produced all $\approx 135 \times 10^3$ permutation traffic patterns for a 9-node 3×3 2D mesh (self traffic is not allowed), we computed the utilization of each link (or, equivalently, the channel load [12]), and then the average link utilization in the network, when traffic is routed under XY routing. The distribution of all permutation traffic patterns, in terms of achieved average network link utilization, is depicted in Fig. 3(a). For the 3×3 2D mesh, only 648 permutation traffic patterns (0.5% of the total) can achieve full link utilization.

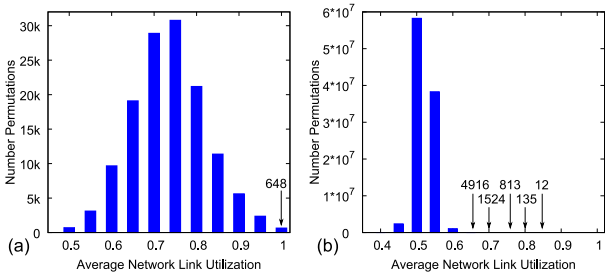

Fig. 3. The distribution of permutation traffic patterns, in terms of achieved average network link utilization. (a) All permutations for a 3×3 2D mesh, and (b) 10^8 randomly generated permutation traffic patterns for an 8×8 2D mesh network. In both cases, XY routing is assumed.

In the second case, we assumed a larger 64-node network, organized as an 8×8 2D mesh. In this case, we produced 10 million randomly-generated traffic patterns out of $\approx 1.98 \times 10^{87}$ possible ones. As shown in Fig. 3(b), the majority of the permutation traffic patterns cause an average link utilization of about 50%, while the number of traffic patterns that yield link utilizations above 80% (desired when trying to cause peak power consumption) is quite low (only 147 out of 10 million were found). Obviously, the random generation of traffic patterns is neither efficient, nor effective; out of the 10 million randomly generated traffic patterns, *not a single one was found to yield 100% (or even 90%) link utilization.*

III. The PowerMax Methodology

PowerMax identifies permutation traffic patterns that route packets from any source to a different destination. The paths selected by PowerMax are legal with respect to the employed routing algorithm, and they utilize all network links of the network. At the same time, the injected packet flows are always conflict-free, i.e., they never compete with each other for any network resource.

A. Enhanced channel dependency graph

The proposed algorithm is applied to the network's Channel Dependency Graph (CDG). Every vertex of the CDG corresponds to a link in the network, and every edge of the CDG

corresponds to an allowed network turn (moving from one link to another through a NoC router). Cycles in the CDG are a sign for routing deadlocks, where in-flight packets are holding onto a set of network resources in a cyclic manner, thereby inhibiting routing progress indefinitely [13]. A routing algorithm is responsible to break cyclic dependencies, while still preserving the connectivity among all network nodes. To achieve this dual objective, the routing algorithm restricts the turns that a packet can make in the network (edge removals from the CDG) which in effect transforms the CDG to a cycle-free graph [14], [15].

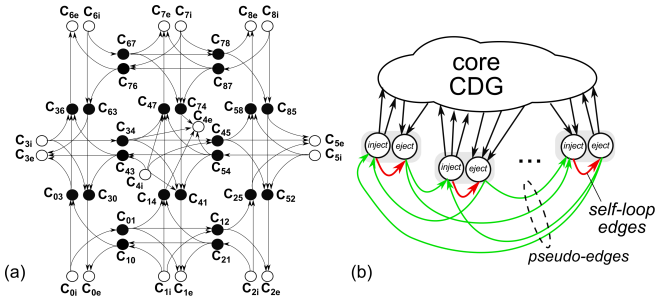

Fig. 4. (a) The channel dependency graph (CDG) of a 3×3 2D mesh, with certain turns across network links removed, following the XY routing algorithm. (b) An abstract form of the enhanced CDG, which includes additional pseudo-edges across sink and source nodes, and self-loop edges connecting the source and the sink node of the same terminal.

An example CDG for a 3×3 2D mesh, derived after applying the turn restrictions of the XY routing algorithm, is shown in Fig. 4(a). The CDG in said figure includes a vertex for each link in the network (black vertices), including the links that connect network routers to the injection and ejection ports of the terminal nodes (white vertices). Another routing algorithm would have used a different set of edges in the CDG, depending on the algorithm's properties [16].

In PowerMax, the cycle-free CDG is enhanced with additional "pseudo-edges," which connect all the sink nodes to all the source nodes of the network. Self-loop edges are also added (local 180^o turns), which connect the source and the sink node of the same terminal. An abstract enhanced CDG that includes pseudo-edges and self-loops is depicted in Fig. 4(b). The enhanced CDG is no longer cycle free, and any possible cycle should pass – by construction – through the added pseudo-edges. Our goal is *to discover a Hamiltonian path in the enhanced CDG*, which begins from any source node and visits each and every vertex of the enhanced CDG exactly once, before arriving at any sink node.

B. Discovering a Hamiltonian path in the enhanced CDG

A Hamiltonian path in the enhanced CDG starts from a source vertex, moves through an arbitrary number of vertices of the "core CDG" (see Fig. 4(b)), and at some point it reaches a sink node. A path cannot evolve indefinitely in the core CDG without visiting a sink node, since the core CDG is cycle-free by construction (the turn restrictions of the routing algorithm impose this feature). Once at a sink node, the Hamiltonian path inevitably moves back into the core CDG via a source node, after using one of the available pseudo-edges (which connect all sink nodes to all source nodes).

By splitting the Hamiltonian path every time a pseudo-edge is encountered, PowerMax derives multiple sub-paths that connect distinct source and destination pairs. Due to the properties of a Hamiltonian path, each vertex of the CDG appears in only one of those sub-paths. Equivalently, each link of the network will be used to transfer the flits of only a single flow across the network, thereby allowing for conflict-free data transfer across a source and a sink node. Furthermore, since the Hamiltonian path covers all vertices of the CDG, it is guaranteed that all links of the network will receive actual traffic, thus yielding full utilization of the NoC components.

The discovery of a Hamiltonian path is an NP-complete problem, solved via path enumeration and backtracking [17]. At each recursive step, the algorithm examines all outgoing edges of an already-visited vertex. When an edge does not lead to a Hamiltonian path, the algorithm backtracks and tries another outgoing edge. In order to avoid redundant backtracks (arising from examining outgoing edges in a random order), we select to follow the implicit order imposed by the routing algorithm. The connection between link ordering and routing algorithms is described in detail in [13], [18]. For example, in a 2D mesh with XY routing, *X+* turns are visited first, followed by *X-*, *Y+*, and *Y-* turns. In other topologies, a different order fits best. For example, in a tree or a random topology that follows up/down routing, *up* edges are examined before *down* edges. In this way, the recursive algorithm starts augmenting the Hamiltonian paths by including vertices that "match" the path selection process of the routing algorithm.

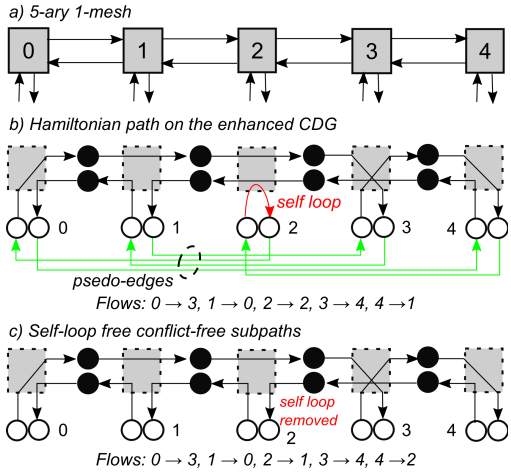

Fig. 5. The Hamiltonian path for a 5-ary 1-mesh, as derived by PowerMax via pseudo-edges and self-loops. The path is then transformed into non-conflicting flows, through the removal of self-loops and pseudo-edges.

When the vertices of the core CDG are already included in a large path, only some of the source and sink nodes remain unconnected. Due to the proposed addition of pseudo-edges and self-loop edges to the CDG, the algorithm augments the path by using said edges, instead of back-tracking. As a result, the algorithm swiftly includes the remaining unconnected source and sink nodes, thereby significantly reducing the complexity of forming a Hamiltonian path.

Once a Hamiltonian path is formed, it connects network links (the visited vertices of the CDG) and source and sink nodes. Of course, the connections across the vertices of the

TABLE I
POWERMAX RUN-TIMES: TIME NEEDED TO DERIVE PEAK-POWER PERMUTATION TRAFFIC PATTERNS.

#Nodes	Ring	Hierachical Ring	2D Mesh	3D Mesh	2D Torus
16	0m 1s	0m 12s	0m 2s	0m 4s	0m 14s
64	0m 2s	0m 25s	0m 9s	0m 10s	1m 8s
256	0m 6s	0m 48s	0m 25s	0m 46s	4m 16s
1024	0m 12s	2m 52s	1m 3s	2m 12s	6m 10s

CDG include pseudo-edges and, possibly, self-loops. To illustrate this scenario, let us assume a simple 5-node line network (5-ary 1-mesh), as illustrated in Fig. 5(a). The Hamiltonian path discovered by PowerMax in the network's enhanced CDG is shown in Fig. 5(b). A Hamiltonian path that uses the local self-loop connections can be transformed to a self-loop-free Hamiltonian path by using a simple post-processing step, which is depicted in Fig. 5(c). The self-loop is removed, and then one edge of the CDG is selected and "broken" into two segments. The incoming segment is connected to the local sink node, while the outgoing segment is connected to the local source. The Hamiltonian path remains connected via the pseudo-edges. This simple post-processing step merely implies that the Hamiltonian path has included the source and sink nodes in a different order. Once the pseudo-edges are removed, the source-to-sink flows are correctly identified (Fig. 5(c)).

The proposed algorithm can solve the Hamiltonian path problem for well-known NoC topologies of ***hundreds of nodes*** **in just a** ***few minutes***. The reason for this efficiency is two-fold: (1) The use of self-loops across the source/sink nodes of the same terminal node enables PowerMax to avoid repeated backtracking when encountering already-visited CDG vertices. (2) The visiting order of the edges that resembles the implicit ordering of channels imposed by the routing algorithm. Without these two properties, solving the problem for medium-scale topologies would require days, thus rendering the tool impractical for modern systems. The run-times required to derive peak-power permutation traffic patterns for various topologies and various network sizes are shown in Table I, using a Linux computer with a 2.3 GHz Intel Core i7-4712HQ Processor and 16 GBs of RAM.

C. Applying PowerMax to various NoC topologies

Fig. 6(a) depicts the resulting sub-paths identified by PowerMax, when applied to the CDG of the 3×3 2D mesh network of Fig. 4(a), and after removing the pseudo-edges of the identified Hamiltonian path. The generated sub-paths include every vertex of the original CDG and only a subset of the edges of the CDG. In this way, it is ensured that all network links are utilized, by using nine distinct and non-conflicting traffic flows, as depicted for clarity in Fig. 6(b).

PowerMax can be applied to any topology and any routing algorithm. Fig. 7(a) depicts the PowerMax-derived non-conflicting permutation traffic that causes 100% link utilization in an asymmetric 2D mesh network (the asymmetry is the result of faulty router 3, which is decommissioned). In this case, the turning restrictions of the routing algorithm [19] (depicted as small arrows at certain turn-points within the network) guarantee connectivity and deadlock freedom. Equivalently, the peak power traffic for a tree that applies the up/down routing algorithm is highlighted in Fig. 7(b).

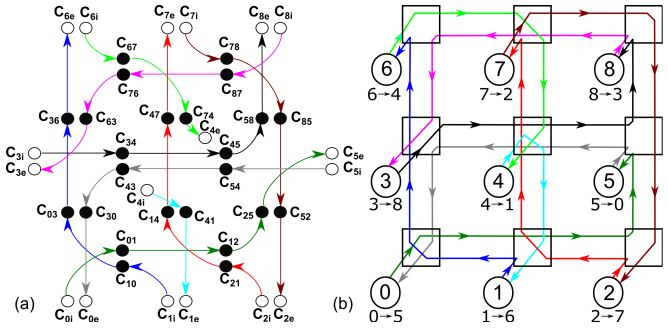

Fig. 6. (a) The sub-paths that lead to unique source/sink pairs, as identified by PowerMax for the CDG of Fig. 4(a). (b) The corresponding flows depicted on the 3×3 2D mesh.

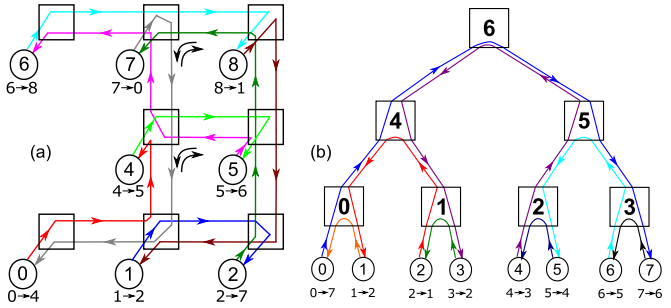

Fig. 7. (a) PowerMax applied to irregular networks, such as an asymmetric 2D mesh. (b) Mapped flows (as generated by PowerMax) on a tree topology.

Ring and tori topologies employ Virtual Channels (VCs) to ensure freedom from possible routing deadlocks. The CDG of VC-based networks is more complex: each vertex (i.e., link) appears multiple times within the CDG; as many times as the number of supported VCs per network link. In these cases, cycles within the CDG are prohibited at the VC level by appropriate turn restrictions (edge removals on the CDG) [20], [21]. Consequently, the resulting routing algorithm is deadlock-free, through the appropriate use of VCs.

PowerMax requires only one additional feature to handle the case of VC-enhanced CDGs. The equivalent vertices that correspond to the VCs of the same physical link are treated as a hyper-vertex, which is used only once in the Hamiltonian path. In other words, only one VC is used per physical channel of the network to carry the generated peak-power traffic. The derived traffic flows are allowed to change VC in-flight, as long as this is dictated by the routing algorithm; in any other case, each traffic flow remains within the same VC. In this way, the algorithm exercises all possible turns at the VC level, but, ultimately, it selects only one VC-to-VC connection. PowerMax does not impose any specific rule for acquiring a VC, other than the ones imposed by the routing algorithm.

Figure 8 illustrates a peak-power traffic scenario for a hierarchical ring. It should be noted that the example of Fig. 8, and the run-times of Table I for the ring, hierarchical ring, and 2D torus, correspond to the application of PowerMax on the enhanced CDG with VCs.

IV. MAXIMIZING THE DATA SWITCHING ACTIVITY

The data patterns that cause the exact maximum power consumption can only be derived using specific gate-level techniques [22], [23], which can be applied only in certain

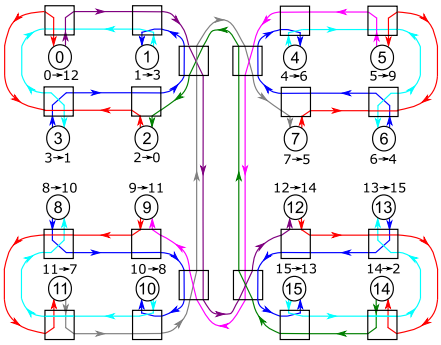

Fig. 8. Peak-power traffic patterns derived by PowerMax for a hierarchical ring, which employs virtual channels for deadlock freedom.

sub-modules of the NoC and cannot be extended to the entire-network level. Therefore, to tackle the problem of identifying the worst-case data patterns, we need to rely on the most common micro-architectural features of the majority of NoC designs. Subsequently, and in conjunction with the guaranteed full-network utilization achieved by the non-conflicting permutation traffic patterns, we can approach (as much as possible) the peak power consumption of the NoC as a whole.

For the links, a repetitive data pattern that switches between $0101\ldots01 \rightarrow 1010\ldots10$ is enough to trigger worst-case power consumption. Each bit experiences a change in every cycle, either 0→1 or 1→0, which switches the corresponding capacitance of the wire to ground. Further, this data pattern ensures that neighboring wires always switch in the opposite direction, thereby causing the worst-case power consumption, due to the link's coupling capacitance.

However, for the VC buffers and the internal logic of the router, we cannot be sure of the exact switching activity caused by this 2-data vector pattern. Assume, for example, the case of VC buffers that are built using register-based (i.e., flip-flop-based) FIFO queues, or using SRAM blocks. In either case, power is consumed every time a new flit is written to, or read from, the VC buffers. PowerMax satisfies the goal of always keeping the VC buffers active (reading/writing), since it allows all NoC links to be fully utilized every cycle. Link utilization translates to a new flit being read (dequeued) from a VC buffer, and moving to an output link that is connected to another VC buffer (in the downstream router), which accepts (enqueues) the new flit.

On each write, a new flit is written to only one VC. Inside the queue of each VC, the flit is written into the register that corresponds to the address pointed to by the tail pointer of the enabled VC queue. Therefore, on each write, only the bits of one register can change value. The rest are not enabled, or remain clock-gated, to save power in the clock pins of the flip-flops. Therefore, to maximize power, we need to guarantee that (a) the new value written to the register is different from the one already stored, and (b) the two values (the old and the new one) differ by as many bits as possible. This can only occur if we know beforehand the specific slot of the VC queue into which the incoming flit will be stored.

When a repetitive data pattern of K words is placed – one word after the other – in a buffer with B slots, then we can guarantee that any incoming word will be written (stored) into a register that already stores a different value, as long as the greatest common divisor of K and B is equal to one. When B is odd, the 2-vector data pattern that also maximizes the power on the links is the proper choice. When B is even, we can select a repetitive data pattern of $B+1$ words. The $B+1$ words can safely include $B/2$ repetitions of the 2-vector data pattern $0101\ldots01 \rightarrow 1010\ldots10 \rightarrow 0101\ldots01 \rightarrow \ldots \rightarrow 1010\ldots10$, plus an all-zero vector. Depending on the NoC configuration, the repetitive set of K data patterns can also extend across different packets, as long as the flits of the packets flow consecutively in the NoC.

For PowerMax, using such data patterns guarantees maximum switching activity, since the traffic is non-conflicting and each VC buffer accepts a new flit every cycle. This also holds for the wires of the links, the input pins of each register of the input buffers, the internal crossbar wires, the crossbar's multiplexers, and the output pipeline registers. Even if those data patterns have the potential of also maximizing the coupling capacitances of the internal logic of the routers, this cannot be ascertained, since the amount of coupling across two circuit nodes depends on the exact layout and placement of the internal wires, which may change across different designs.

In terms of power, the traffic injected can stay within the same VC from source to destination, as long as one flit is written and read per cycle, and the data values written and read have the maximum bit-wise difference. Distributing traffic across VCs for each non-conflicting flow produced by PowerMax is possible, but it needlessly complicates the derivation of the appropriate data switching patterns that cause the maximum switching activity, without any true impact of the triggered power consumption.

Even though the non-conflicting nature of the traffic patterns generated by PowerMax can maximize the switching activity in the datapath of the NoC (links, buffers, crossbar), the arbitration part is kept working on the same requests and grants in each cycle. This causes minimum switching activity in this portion of the NoC. However, this is not a problem in NoCs with wide datapaths of 128 bits or more, where the power of the arbitration logic is low relative to the datapath portion. This argument is also verified by the experimental results that use random traffic; the latter maximizes the switching activity in the arbitration part, due to the random nature of the input requests. Even when compared with this traffic scenario, PowerMax achieves significantly higher power consumption by only appropriately targeting the data switching activity.

V. Experimental Evaluation

The goal of PowerMax is to trigger the peak-power consumption of a NoC by injecting appropriately selected traffic patterns that maximize network component utilization and data switching activity. Therefore, the measured power is *realistic*, in the sense that it can be caused by a real traffic scenario, even if it is rare under normal system operation.

PowerMax is evaluated on 64-node NoCs following 2D mesh and hierarchical ring topologies. Other tested topologies show similar trends. In order to contain the number of possible configurations, we assume a tile-based chip floor-plan similar to the Scorpio chip [24]. Scorpio was built at 45 nm technology (which matches the technology library we used in our implementations), using a tile size of approximately 2×2 mm. Based on the chosen NoC topology, the NoC routers can

have a variable number of input and output ports. For every configuration, we assume that the NoC supports 4 VCs per input port, with 5 buffers/VC, and the NoC routers employ the 3-stage pipelined organization of Scorpio routers [24]. The inter-router NoC links carry 64 bits of data, plus some extra flow control information. The header flit, which also includes network-addressing information, carries fewer actual data bits.

All NoC components used in the evaluation were implemented in SystemVerilog, mapped to a commercial low-power 45 nm 0.8 V standard-cell library, and placed-and-routed using the Cadence EDI flow. Depending on the NoC topology, a different placement-and-routing round was conducted. Power was measured after performing timing-accurate simulations, when operating the NoC at 1 GHz and including all back-annotated layout parasitics.

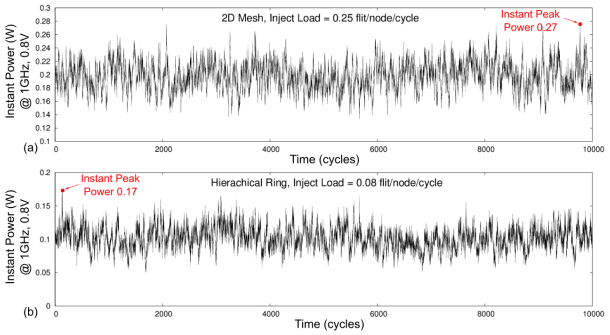

Fig. 9. A 10K-cycle snapshot of the instantaneous power consumption of a 64-node 2D mesh (top) and a hierarchical ring (bottom), after the network reaches steady-state operation, using uniform-random traffic and data.

In the first set of experiments, PowerMax is evaluated against random synthetic traffic patterns, under various data switching and network-injection scenarios. The instantaneous power consumed by a NoC when the incoming traffic causes contention across flows with unrelated data (this occurs in almost all cases under normal operation) can vary significantly over time, depending on the switching activity in various parts of the network in each cycle. This behaviour is highlighted in Fig. 9, for an 8×8 2D mesh and a 64-node two-level hierarchical ring that consists of 8-node local rings connected via an 8-node global ring.

Both networks receive uniform-random traffic at a different rate (close to their saturation throughput), as reported in Fig. 9. The two NoCs have equal link width, i.e., 64 bits plus flow-control bits, and both operate at 1 GHz. Therefore, the bisection bandwidth of the 2D mesh is larger than the bisection bandwidth of the hierarchical ring. The injected packets are 5-flit long and carry random data in their payload portion. In this experiment, the bit of each flit when entering the network has equal probability of being 0 or 1, independent of the rest of the bits of the same flit, or the previous flits.

The peak power consumption achieved by random traffic is merely the peak instantaneous power observed during the simulation's time frame. There is no guarantee that a large power value can be triggered during simulation, due to the unpredictability in switching activity, and the lower NoC utilization caused by contention among different flows. Additionally, the observed peak power consumption simply represents an *instantaneous* peak. This cannot be sustained

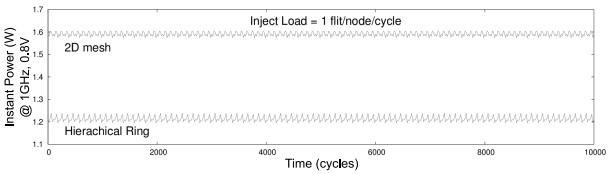

Fig. 10. A 10K-cycle snapshot of the instantaneous power consumption of a 64-node 2D mesh and a hierarchical ring (under steady-state network operation), using PowerMax-derived traffic/data with full injection throughput.

over a longer period of time, which would be required to observe possible temperature increases and identify thermal hot-spots in the system.

On the contrary, PowerMax does not have such limitations. In Fig. 10, we report the instantaneous power consumed by PowerMax under 100% injection load for each case (2D mesh and hierarchical ring). The results are measured by injecting 5-flit packets in the NoC, following the PowerMax-derived permutation traffic patterns and carrying data payloads with the 2-vector data patterns described in Section IV. PowerMax keeps power consumption constantly and consistently very high. The minimal variance in the power consumption is due to the switching profile of the header and flow-control bits, which are not controlled by PowerMax.

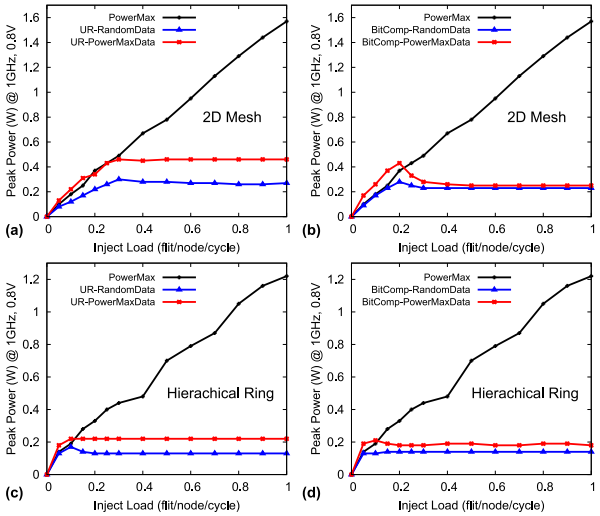

Fig. 11. Peak power consumption vs. injection load. The power consumption triggered by PowerMax-derived traffic compared against the power consumed when using uniform-random and bit-complement traffic patterns.

Next, we compare the peak power consumption of PowerMax and random traffic scenarios (*uniform-random* and *bit-complement* traffic), under the same injection load. For each injection load, the maximum instantaneous power consumption value observed (over 500,000 cycles of simulation) was recorded. The results are depicted in Fig. 11, for the same 2D mesh and hierarchical ring topologies. The peak power consumption of random traffic (blue curves) follows the throughput behavior of the network itself, and, after saturation (when the utilization of NoC components reaches its limit), the peak power consumption observed is rather constant. On the contrary, PowerMax can increase the power consumption to its true maximum value, due to its non-conflicting traffic. The data switching activity is directly controllable by the input sources, and it covers all the intermediate router ports and

network links that are utilized by the injected flow. When compared against uniform-random traffic (Figs. 11(a) and (c)), PowerMax triggers maximum power consumption, which can be more than 6× higher than the one achieved under uniform-random traffic with random data (i.e., blue curves).

This is also true when the NoC is driven by uniform-random traffic that allows contention in the network, but the injected *data patterns* are the same as the ones used with PowerMax (by following the guidelines described in Section IV). This scenario is also depicted in Figs. 11(a) and (c) with the red curves. PowerMax still consumes significantly more power (4× higher), since it simultaneously takes into account both the network utilization and the data switching activity.

Similar conclusions are derived when the power consumption of the NoC is triggered using other permutation traffic patterns, such as bit-complement traffic. In this case (Figs. 11(b) and (d)), the peak power consumption of PowerMax is 4× larger than the *largest* power observed under the bit-complement traffic patterns (red curves).

TABLE II
PEAK POWER OF POWERMAX VS. THE POWER OF A FAKE SCENARIO, WHICH ASSUMES THAT EVERY CIRCUIT NODE SWITCHES IN EVERY CYCLE.

@ 1 GHz, 0.8 V	Fake (pessimistic)	PoweMax
2D Mesh	4.87 W	1.6 W
Hier. Ring	3.76 W	1.23 W

Finally, we compare the power triggered by PowerMax, versus the peak-power consumption that corresponds to the fake scenario of every circuit node switching in every cycle. In both cases depicted in Table II, the peak power triggered by PowerMax (measured at 100% injection rate) is lower than the one derived using the fake (un-realistic) approach. The difference between these two maximum power values depends on topology characteristics, and the power expended on the links vs. the power expended within the routers. In any case, the significant conclusion out of this comparison is that fake peak-power scenarios overestimate the true maximum power profile of the NoC and unnecessarily increase the overall system power budget. With PowerMax, worst-case power analysis is brought closer to what is realistically attainable.

VI. CONCLUSIONS & FUTURE WORK

As chips become increasingly more dense and complex, power consumption becomes a primary design constraint. It is imperative for designers to realistically estimate a design's peak power consumption, which directly impacts other salient system attributes, such as performance, implementation costs, battery life, and reliability. This paper introduces PowerMax, the first fully-automated high-level methodology to generate appropriate traffic and data patterns that cause peak power consumption within the NoC. The peak power consumption triggered by PowerMax is, on average, 5.5× higher (up to 8× higher) than what is observed after simulating random traffic and data patterns.

PowerMax guarantees full utilization of every NoC component. Therefore, even if some NoC components are not identical – in terms of their (micro-)architecture – they are still fully utilized, irrespective of their differentiated design parameters. For example, PowerMax imposes full (100%) utilization on every link of the network, irrespective of the link's length. Both short and long wires are equally utilized; this attribute translates into longer wires consuming more power than shorter wires. Differentiated design parameters – e.g., the link length, the presence (or not) of pipelined links, the number of VCs per input port, and the number of buffer slots per VC – can all be handled by PowerMax, since every component (irrespective of its size) is utilized in every cycle.

However, in the case of completely heterogeneous NoCs, where the inter-router link bandwidths can vary (i.e., there is a combined effect of link width and clock frequency), marginal extensions to the problem formulation and the corresponding proposed algorithm are required. Our future work will focus on extending PowerMax to cover such NoC architectures, including those that distribute the NoC components across multiple voltage/frequency domains.

REFERENCES

[1] T. S. Rosing, K. Mihic, and G. De Micheli, "Power and reliability management of socs," *IEEE Trans. VLSI*, vol. 15, no. 4, Apr. 2007.
[2] S. Borkar and A. Chien, "The future of microprocessors," *Commun. ACM*, vol. 54, no. 5, pp. 67–77, May 2011.
[3] N. S. Kim, T. Austin, T. Mudge, and D. Grunwald, *Challenges for Architectural Level Power Modeling*. Springer, 2002.
[4] Y. Jin, E. J. Kim, and K. H. Yum, "Peak power control for a QoS capable on-chip network," in *ICPP*, 2005, pp. 585–592.
[5] V. Kontorinis, A. Shayan, D. Tullsen, and R. Kumar, "Reducing peak power with a table-driven adaptive processor core," in *Intern. Symp. on Microarchitecture*, 2009, pp. 189–200.
[6] K. Ganesan, J. Jo, W. Bircher, D. Kaseridis, Z. Yu, and L. John, "System-level max power (sympo): A systematic approach for escalating system-level power consumption using synthetic benchmarks," in *PACT*, 2010.
[7] S. Polfliet, F. Ryckbosch, and L. Eeckhout, "Automated full-system power characterization," *IEEE Micro*, pp. 46–59, May 2011.
[8] K. Ganesan and L. K. John, "Maximum multicore power (mampo): An automatic multithreaded synthetic power virus generation framework for multicore systems," in *ACM Intern. Conf. for High Performance Computing, Networking, Storage and Analysis (SC)*, 2011.
[9] B. Mathewson, "The evolution of soc interconnect and how noc fits within it," in *Design Automation Conference*, 2010, pp. 312–313.
[10] A. Psarras, J. Lee, P. Mattheakis, C. Nicopoulos, and G. Dimitrakopoulos, "A low-power network-on-chip architecture for tile-based chip multi-processors," in *ACM Great Lakes Symp-VLSI*, 2016, pp. 335–340.
[11] N. Weste and D. Harris, *CMOS VLSI Design a Circuits and Systems Perspective*. Addison Wesley (3rd Edition), 2010.
[12] B. Towles and W. Dally, "Worst-case traffic for oblivious routing functions," in *ACM SPAA*, 2002, pp. 1–8.
[13] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.
[14] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," *J. ACM*, vol. 41, no. 5, Sep. 1994.
[15] J. Flich and J. Duato, "LBDR: Logic-Based Distributed Routing for NoCs," *IEEE Computer Architecture Letters*, pp. 13–16, Jan 2008.
[16] J. Cong, C. Liu, and G. Reinman, "ACES: Application-specific cycle elimination and splitting for deadlock-free routing on irregular network-on-chip," in *Design Automation Conference*, 2010, pp. 443–448.
[17] R. Sedgewick and K. Wayne, *Algorithms*. Addison-Wesley, 2011.
[18] X. Lin and L. M. Ni, "Deadlock-free multicast wormhole routing in multicomputer networks," in *ISCA*, 1991, pp. 116–125.
[19] P. Ren and et al., "A deadlock-free and connectivity-guaranteed methodology for achieving fault-tolerance in on-chip networks," *IEEE Trans. on Computers*, pp. 353–366, Feb 2016.
[20] 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, pp. 466–475, 1993.
[21] J. Duato, "A new theory of deadlock-free adaptive routing in wormhole networks," *IEEE Trans. Parallel Distrib. Syst.*, pp. 1320–1331, 1993.
[22] M. Hsiao, "Peak power estimation using genetic spot optimization for large vlsi circuits," in *Design, Automation and Test in Europe*, 1999.
[23] K. Najeeb and et al., "Power virus generation using behavioral models of circuits," in *IEEE VLSI Test Symmposium*, 2007, pp. 35–42.
[24] B. Daya and et al., "Scorpio: A 36-core research chip demonstrating snoopy coherence on a scalable mesh noc with in-network ordering," in *Int. Symposium on Computer Architecture*, June 2014, pp. 25–36.