Ruche Networks: Wire-Maximal, No-Fuss NoCs

(Special Session Paper)

Dai Cheol Jung, Scott Davidson, Chun Zhao, Dustin Richmond and Michael Bedford Taylor
Bespoke Silicon Group
University of Washington

Abstract—**Network-On-Chip design has been an active area of academic research for two decades, but many proposed ideas have not been adopted in real chips because they have complex behavior or create significant risks in chip implementation. For this reason, many existing chips just employ fast, replicated vanilla dimension-ordered mesh NoCs. However, these networks do not come close to utilizing the full available VLSI wiring capabilities, and propagate packets at speeds that are significantly below the raw speed of wires.**

The ideal network would not require any custom circuits, and would decompose easily into a hierarchical CAD flow consisting of a top-level design instantiating a mesh of identical hardened tiles with short-wire neighbor connections.

At the same time, this ideal network would easily scale to efficiently utilize the majority of the available chip wiring resources, and would offer a mechanism for scaling this wire usage up or down based on available bandwidth. Packets would spend a significant fraction of their time in wire delay rather than router delay. Finally, the NoC would be simple to understand.

This paper proposes Ruche Networks, which fulfill these requirements. They are based on simple 2-D mesh networks but amplify the NoC bandwidth and reduce NoC diameter of tiled architectures by adding long-range physical channels from each tile to other tiles on the same row or column. The more distant the connections, the greater the bandwidth of the network and the lower the diameter. The distance is typically increased until all of the physical VLSI wiring bandwidth have been absorbed.

We explain the rational for this "ruching" and provide a simple methodology for designing and implementing these networks using a standard cell VLSI CAD flow.

In this paper, we show the steps involved in ruching the HammerBlade Manycore's mesh networks; these steps can easily apply to other designs.

I. Introduction

Network-On-Chip design has been an active area of academic research for two decades, but many of the proposed ideas have not been adopted in real chips because they have complex behavior or create significant risks in chip implementation. For this reason, many existing chips [1], [2], [3], [4], [5], [6] just employ a few simple dimension-ordered mesh NoCs without virtual channels.

The ideal network would not require any custom circuits, and would decompose easily into a hierarchical CAD flow consisting of a top-level instantiating a mesh of identical tiles with short wires that connect only neighbor tiles. Each tile would have identical timing to its neighbors.

[1]This material is based on research sponsored by Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement numbers FA8650-18-2-7856 and FA8650-18-2-7852. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) or the U.S. Government. This work was partially supported by NSF SaTC Award 1563767, NSF SaTC Award 1565446, and by the DARPA/SRC JUMP ADA Center.

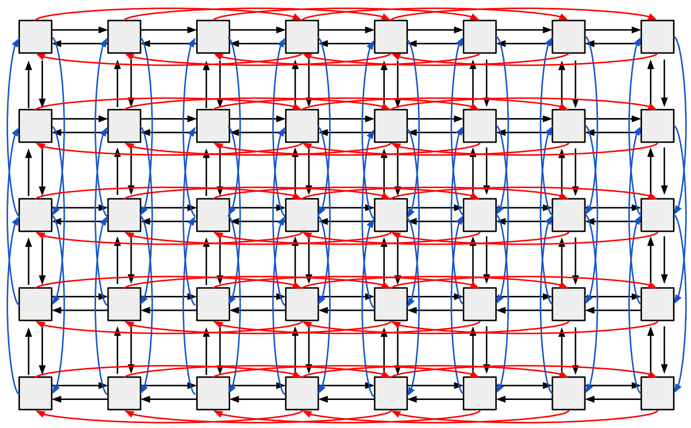

Fig. 1: **A Ruche Network augments a 2-D mesh with additional parallel high-bandwidth links called Ruche Channels.** NoC bandwidth can be scaled to VLSI wiring limits by increasing the hop distance of the links, but node router logic area has constant area, a small increase over a standard mesh. This example employs a X Ruche Factor of 3 and Y Ruche Factor of 2.

At the same time, this ideal network would easily scale to efficiently utilize the majority of the available chip wiring resources, and would offer a mechanism for scaling this wire usage up or down based on available bandwidth. Finally, it would be easy to model in terms of performance.

This paper proposes **Ruche Networks**, which fulfill these requirements. They are based on simple 2-D mesh networks but amplify the NoC bandwidth and reduce NoC diameter of tiled architectures by adding long-range physical channels from each tile to other tiles on the same row or column. The more distant the connections, the greater the bandwidth of the network and the lower the diameter. The distance is typically increased until all of the physical VLSI wiring bandwidth have been absorbed.

In this paper, we explain the rational for this *ruching* and provide a simple methodology for designing and implementing these networks all the way down to VLSI using a standard cell CAD flow. As a running example, we show the steps involved in ruching the HammerBlade Manycore's mesh networks in GlobalFoundries 12nm; these steps can easily apply to other designs.

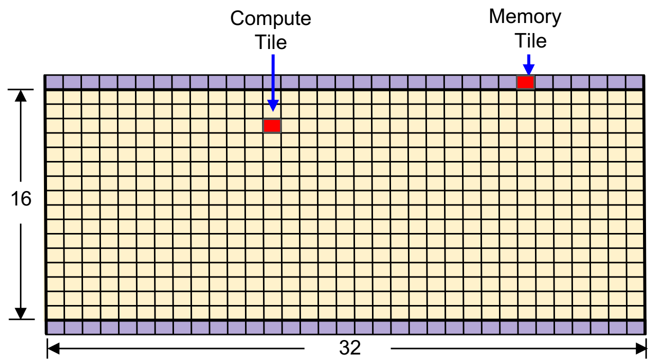

Fig. 2: **Running Example for the Paper: A HammerBlade Manycore Pod.** A HammerBlade Chip is comprised of many pods, one of which is pictured here. The tiles are arranged in a mesh network. In the center are 512 compute tiles, and at the top and bottom are 64 cache tiles, which interface to the off-chip memory system.

The rest of the paper proceeds as follows. First, in Section II, we briefly describe the baseline HammerBlade Manycore architecture, and the design methodology that is employed. We describe architectural trade-offs in Ruche networks in Section III, and evaluated them in Section IV. We present a methodology for implementing Ruche networks in V. Finally, we conclude.

II. Baseline Architecture

As a running example in this paper, we will examine the application of Ruching to a baseline manycore design, HammerBlade Manycore, which was taped out in 12nm GlobalFoundries technology in July 2019 and has been up and running in the lab. HammerBlade is descended from the 1.4 GHz 511-core Celerity [4], [3], which broke several world records, including RISC-V performance and fastest CoreMark performance, but has superior programmability and memory system features. HammerBlade is an agile open source hardware project (https://github.com/bespoke-silicon-group/bsg_manycore) and is continuously evolving—be sure to always find the latest version!

Architecture. The current architecture of a single HammerBlade Pod is shown in Figure 2, which is a 16x32 array of RISC-V compute engine tiles, bounded by memory tiles which contain 32 KB of cache. The tiles are interconnected by a pair of dimension-ordered (X-then-Y) routed 2D mesh single-cycle-per-hop, single-flit-packet networks with two-element input FIFOs and local ready/valid flow control: one 93-bit request network and one 51-bit reply network. Thus, there are 144 wires on a side. Approximately 36 compute tiles fit in a single mm^2 of silicon, and each tile can attain 4 Gflops of performance at 2 GHz.

A HammerBlade Pod easily scales up and down from 16x8 to 64x32, or 128x64. Every tile has a ∼4KB local data scratchpad and ∼4KB I-Cache. All addresses in the caches and scratch pads are mapped into a partial global address space (PGAS), which means that any tile can access any memory address using a standard load or store instruction. If the address is not mapped locally, the compute tile will launch a non-blocking request packet into the network, and continue on to the next instruction; asynchronously receiving the reply packet which contains either load data or store acknowledgment.

Physical Design Methodology. The physical implementation of a design imposes additional constraints that can make it difficult to achieve the expected performance of the system if the physical design is not taken into consideration. The HammerBlade Pod architecture achieves scalability by being a friendly design to realize in silicon.

A HammerBlade Pod is composed of many identical tiles in the array which allows us to take advantage of a bottom-up hierarchical design methodology. The tile will first be fully implemented as a sub-block of the design. At the top level (the full HammerBlade Pod), multiple instances of this tile sub-block will be stamped out in a 2D array and stitched together. During the top level implementation, all internal paths inside the sub-blocks can be ignored and only interfacing logic needs to be considered for timing purposes, greatly reducing the memory overhead and runtime of the EDA tools.

Even still, a HammerBlade Pod with 128 tiles takes over *36 hours* on a 72-core Xeon Gold SP machine to go from SystemVerilog RTL through logic synthesis, automatic place and route (APR), and static timing analysis, when it is working well. When it is not working well, it will run *indefinitely* and never finish. Such large time-to-results reduces the number of design iterations designers can work though to fix and optimize the design, and presents a very strong danger for complex physical designs that problems will not be solved quickly enough to meet tapeout deadlines. Iterations on a sub-block are much quicker (just a few hours) and a sub-block may be instantiated thousands of times; therefore a majority of the design complexity is pushed inside of a sub-block. This gives designers more opportunity to find and implement optimizations that get replicated, amortizing optimization effort.

Because sub-blocks instances must have identical internals, the timing requirements for that block must union all of the timing constraints that are imposed by the many environments it is placed in. If one instance has a particularly slow input signal that for another instance, is overly fast, then the APR tool must assume that the input is *both* slow and fast. This creates a nightmare as the tool tries to make potentially impossible concessions for both timing environments even though no actual instance actually needs to meet that spec.

Taken in this light, a major benefit to a 2D mesh network, with its regular, local wiring, over many other network topologies with irregular wiring is that the timing environment can be made identical between sub-blocks at the top level, eliminating this major source of risk in implementation.

For NoCs, another important physical constraint is the number of wires we can route. Because all of the complexity has been pushed inside the block, the top level routing primarily consists of connections between blocks. The HammerBlade

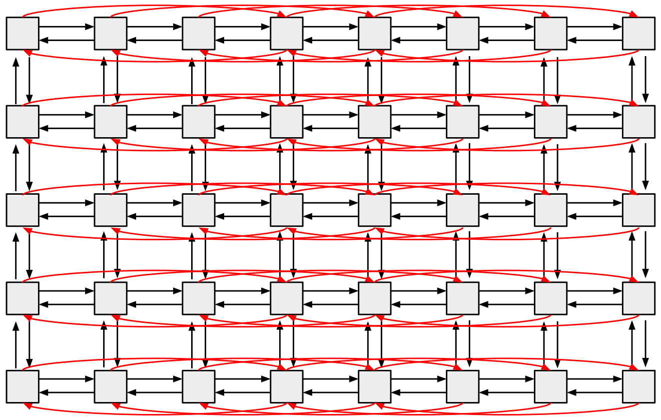

Fig. 3: **The Half Ruche Network only applies Ruche Channels in one direction, and works particularly well with the combination of dimension-ordered routers and networks with aspect ratio 2:1 like HammerBlade.** This example has X Ruche Factor of 3. Each node is connected both to its nearest neighbors, and the nodes three hops to the East and to the West.

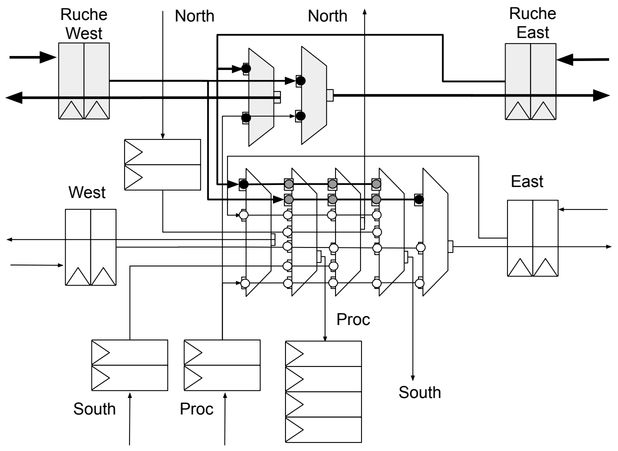

Fig. 4: **Half Ruche Network Router Datapath.** Highlighted logic and thicker wires indicate additional components added to the standard 2D mesh interconnect router in order to support the Ruche Channels. A variant of the Half Ruche Network router design can be constructed by depopulating some of the crossbars as indicated with gray dot connections for the north, south, and proc crossbar multiplexers.

Pod tiles have ports on 4 metal layers: 2 vertical layers and 2 horizontal layers. Between tile sub-blocks only 13% of the available wires are actually used to connect the tiles together. Based on prior experience, 50% wire utilization is a reasonable upper bound as we want to leave every other wire track empty to eliminate noise and signal slowdowns due to capacitve coupling which is particularly bad in long parallel wires that are common in NoCs.

Network Limits. Mesh networks offer great bandwidth when communication has locality; for example, a HammerBlade tile has 4 dedicated links to neighbors but can only inject one memory request per cycle; so there is 4:1 over-resourcing for nearest neighbor communication patterns. Conversely if loads/stores on average travel more than 4 hops, and are sent every cycle, then there is under-resourcing. For codes that share memory across local tiles' memories, software can sometimes organize tiles into *tile groups*, which collectively work on the groups' scratch pads, which creates pockets of locality, reducing contention.

The 2:1 aspect ratio of the HammerBlade Pod is driven by the desire to increase the number of caches on the edge, and with some optimism about the ability to map data structures to caches in a way that improves locality in the mesh. However, as it turns out, ease-of-programming concerns often spread addresses randomly over the caches. Since the caches can service 64 requests per cycle collectively, the bisection bandwidth of 16 requests per cycle means that for random accesses, the system is limited to about half of its peak potential by the section. At the same time, although the achieved single-cycle per hop is fast for the literature, worst case round-trip hop count on an unloaded network is (15+31)×2 = 92 hops, which is a significant on-chip latency.

Thus, there is desire to use HammerBlade's many unused wiring tracks to improve both the latency and the bandwidth of the network, while consuming almost no logic area so as not to reduce the number of tiles on the chip. This led us to conceive of Ruche Networks.

III. Ruched Mesh Network

A *Ruche Network* augments a 2D nearest-neighbor connected mesh network (which we refer to as *the Local Network*) with additional physical *Ruche Channels* to both increase bandwidth and decrease diameter. A *Half Ruche Network* only has Ruche Channels along one dimension (i.e., either X or Y) and a Full Ruche Network has them in both dimensions. In those dimensions, a node will have one Ruche Channel connected to another node in the positive direction and one in the negative direction. An example Half Ruche Network is shown in Figure 3 and an example Full Ruche Network is shown in Figure 1.

We employ the term *Ruche Factor* to describe how far the Ruche Channels span. So for example, a Ruche Network with Ruche Factor zero is just a mesh with single links to nearest neighbors; with Ruche Factor one has dual links to nearest neighbors; and with Ruche Factor two has one link to neighbors and another link to the neighbors' neighbor.

The larger the Ruche Factor, the greater amount of VLSI wiring bandwidth that a design can absorb, and the faster data can travel across the chip, skipping tiles each cycle.

A Ruche Network has only a small increase in tile router size over a standard mesh. This area remains constant even as NoC throughput is increased and NoC diameter is decreased by increasing Ruche Factor. Moreover, *Depopulated Crossbar Ruche Networks*, which restrict routing directions within a Ruche Network, can further reduce this constant area increase to almost nothing.

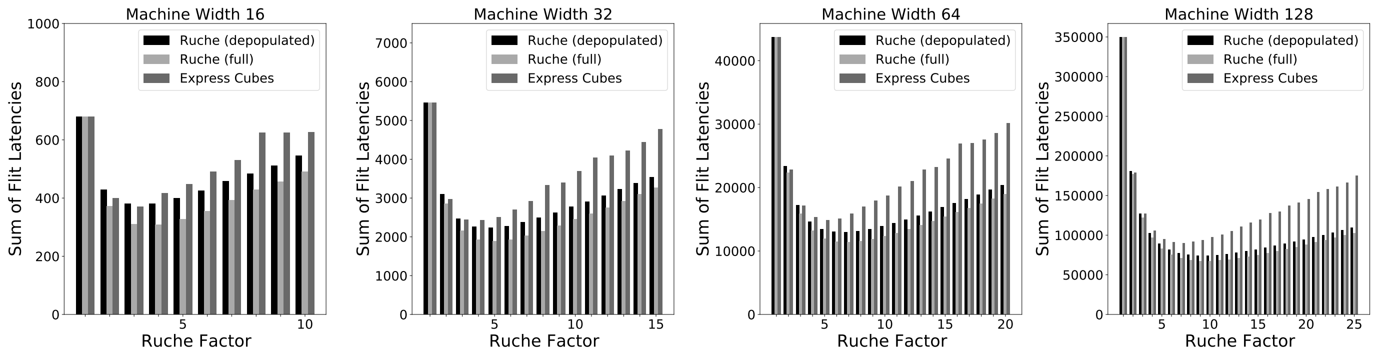

Fig. 5: **Modest Ruche Factors have the best average unloaded latency, but larger ones make more sense with larger machines, and Ruche Networks are generally better than single-level Express Cubes.** The total number of message hops required to perform an all-to-all unidirectional communication for a machine with the given width decreases sharply with only a few hops added to the network; however the benefits start to diminish quickly until eventually additional hops increase the total number of hops.

Ruche Channels are a novel technique jointly proposed by our team and by Chris Batten's team at Cornell [7]. In contrast to Express Cubes [8], Ruche Channels do not add additional interposing interchanges between nodes and are better suited for standard-cell based on-chip networks. Unlike Express Virtual Channels [9], Ruche Channels are not a virtual topology imposed on a single physical channel; they are additional physical channels providing additional bandwidth. Unlike MECS [10], they employ identical replicated tiles, scale to large numbers of tiles without tile size increase, and can be implemented easily in standard cell VLSI flows. Moreover, Ruche Channels result in only a constant increase in tile logic gate area despite the addition of many additional physical channels.

Ruche Networks can be used to implement a static form of *NoC Symbiosis [11]* (i.e. coplacement of the network routers with core logic so that the network can leverage unused routing resources) to attain high utilization of VLSI wiring resources, since most of the links passing through a tile do not require routers. However, in this paper, we employ full APR-based *NoC Symbiosis* to enable structured wire interleaving to improve signal integrity, reduce noise, and decrease delay.

What we did to HammerBlade. Since the HammerBlade Pod is rectangular with a 2:1 X:Y aspect ratio, we found that a single-cycle-per-hop Depopulated Crossbar Half Ruche Networks with X Ruche Factor 3 worked extremely well, delivering up to 4× throughput increase and 3× diameter reduction in the most constrained portion of the mesh, while minimizing growth of crossbar area. Since HammerBlade does X-first dimension-ordered routing for request flits, the Half Ruche Networks can be thought to first "punch through" the bisection bottleneck of the original mesh before routing on the more numerous, standard mesh Y channels. Moreover this approach minimizes area increase since Ruche Network router resources scale most favorably in the "first leg" of dimension ordered routing. Reply packets are routed Y-first to retrace the path back to the source, so for the reply network we also X Half Ruche, and experience greater crossbar growth, but the reply network is about 1/3 the width.

Ruche Router Deep Dive. As shown in Figure 4, the router used to implement the Ruche Network is the same design as an optimized dimension-ordered mesh router, but with 2 additional full-duplex ports labeled (without loss of generality, assuming X-first routing) Ruche East (RE) and Ruche West (RW) for a Half Ruche Network and additional Ruche North and Ruche South ports for a Full Ruche Network. Flits are still routed using a XY-dimension ordered routing algorithm; however if the distance to the destination in a dimension is greater than or equal to the Ruche Factor, then the RE or RW ports are employed rather than the local E and W ports. Said another way, the router will favor the Ruche Network unless it would overshoot the X coordinate of the destination, in which case it will use the local network.

The addition of the two full-duplex Ruche ports RE and RW add a small amount of logic to the routers. To drive the RE and RW output ports, each will need a 2-input crossbar (with P and RW inputs, and with P and RE inputs respectively). In addition to the two new crossbars, in a Fully Populated Crossbar Ruche Network, the N, S and P output ports will each need 2 additional input ports for their crossbars. Finally, the E port will need an additional input for the RW port and the W port will need an additional input for the RE port.

Routing Optimizations.. To reduce the router area impact even more, the Depopulated Crossbar Ruche Network, also shown in the figure, employs a modified router that eliminates the ability to turn out of a Ruche Channel; e.g. packets must hop off of the Ruche Channel on to the local network before changing dimensions (i.e. to Y or P port). Then the N, S and P crossbars are depopulated to prevent the area overhead of adding the two Ruche ports. Thus the Ruche Network cannot be taken to the exact X destination, but must hop off the express links earlier and take the local network to the final X destination.

Intriguingly, depopulated variants can have superior bandwidth characteristics to fully populated ones, because it better load balances traffic. In the results section, we will evaluate the performance trade off between these two router designs.

Tuning routing algorithms on Ruche Networks is an intriguing area of research. For example, with Ruche Factor of one, the Ruche routing algorithm will always choose to use the Ruche Channels to route flits. This leads to an under-utilization of the local network negating the additional bandwidth expected from adding the Ruche Network. To combat this, we have the Ruche Network route traffic on the local network for routes with odd hop distances.

IV. Evaluation

To evaluate the Ruche Network, we created a cycle accurate model to calculate the performance characteristics of Half Ruche Networks. For each experiment, we simulated the network characteristics for an all-to-all unidirectional communication on the X axis, which is representative of the memory traffic between tiles and caches in HammerBlade. We also implemented the Ruche Channels using Synopsys Design Compiler and Synopsys IC Compiler II in Global Foundries 12nm. We then ran the DRC-checked, APR'd physical design though static timing, signal integrity and wire noise analysis to establish that practical limits of how densely wires can be packed in VLSI to maximize the number of Ruche Channels.

Network Latency Performance Results. Figure 5 shows that a Fully Populated Crossbar Ruche Network has the best unloaded latencies, followed by the area-optimized Ruche Network with depopulated crossbar and finally the Express Cube. We can see that regardless of the size of the machine, the jump from a Ruche Factor of 1 to 2 yield the largest improvement between two consecutive Ruche Factor values. The optimal Ruche Factor is also quite small for all networks across all machine sizes. For a machine size of 16, the depopulated Ruche Network and the Express Cube are both latency optimized with a Ruche Factor of 3 while the fully populated Ruche Network is optimal with a Ruche Factor of 4, outperforming a Ruche Factor of 3 by only 2 cycles. As the machine size increases, so does the optimal Ruche Factor; however, even for a machine of size 128 the depopulated Ruche Network, fully populated Ruche Network and Express Cube are optimal at Ruche Factors of 9, 10, and 7 respectively.

Network Balance Results. If Local Network routing bandwidth is B then a Ruche Network with a Ruche Factor f increases the cross sectional bandwidth by a factor of $f * B$. However, peak bandwidth improvements require some degree of balancing between Ruche Network and Local Network.

Figure 6 shows the number of flits each tile in a 16 tile wide machine routes on the Ruche Network or Local Network. Generally, we would like load to be balanced across the two kinds of networks, maximizing utilization of wiring resources. While the Ruche Network adds an additional $f * B$ cross sectional bandwidth, every tile only has access to one factor of the Ruche Network therefore a load balance of 50/50 per

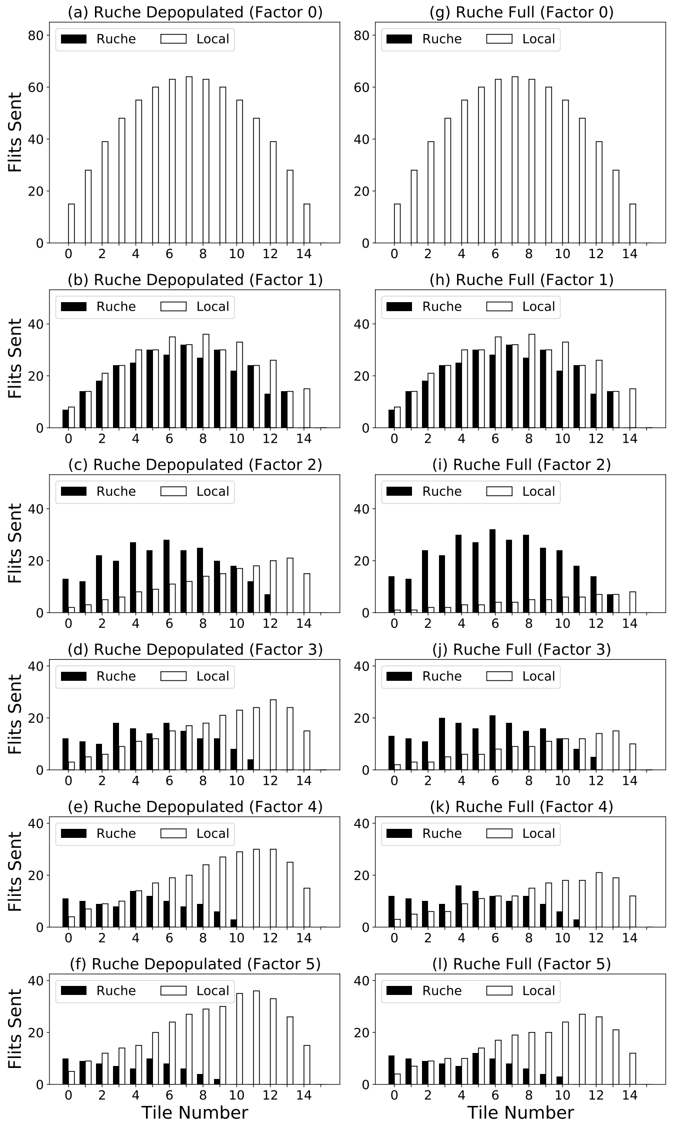

Fig. 6: **As the Ruche Factor increases the balance start to over-burden the Local Network causing a bottleneck in the performance of the system.** The area-optimized Depopulated Crossbar Ruche Network adds additional burden on the Local Network compared to the Fully Populated Ruche Network as every flit must end with a traversal over the Local Network. When the Ruche Factor is equal to 1 as in (b) and (h) we use a different routing algorithm to achieve near equal load balancing between the networks.

tile is desirable. Figure 6 (a,g) are the base cases where the Ruche Factor is 0, i.e. a standard mesh network. Figure 6 (b,h) with Ruche Factor of 1 do not follow the same trend as the rest of the graphs because of the superior routing algorithm describe in Section III.

When the Ruche Factor is equal to 2 or greater we see trends starting to emerge. First, as the Ruche Factor increases we see a shift in utilization from the Ruche Network to the Local Network. This is because flits must travel more hops because

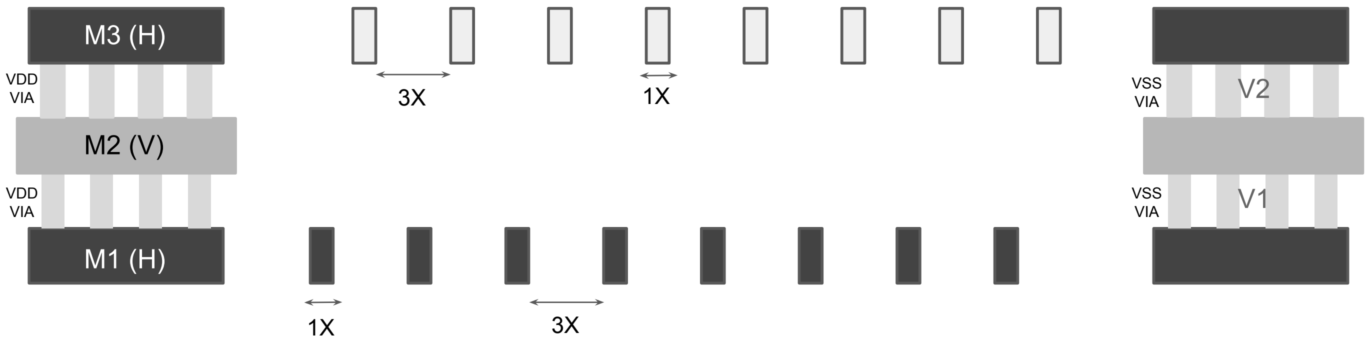

Fig. 7: **Cross-sectional view of metal layers in our extremely high-bandwidth Ruche Network.** Power via arrays are placed periodically, and they block some routing tracks. 16 tracks of wires can be placed between two via stacks. There is a minimum spacing required between the wires and via metal.

the Ruche Channel would overshoot the destination, so fewer and fewer flits are actually put on the network. Furthermore we can see that for the same Ruche Factor, the depopulated crossbar variant of the Ruche router utilizes the Local Network more than the full crossbar Ruche router. This is because the depopulated Ruche router forces all flits to use the Local Network for the final link traversal. When the Ruche Network is heavily utilized like we see in Figure 6 (c) this is actually beneficial compared to the full crossbar Ruche router seen in Figure 6 (i).

V. Implementation

In this section, we provide the methodology for physical design of Ruche Networks. A key property of Ruche Networks is that they can be used to convert excess VLSI wiring resources into additional bandwidth and reduced network diameter. However, excessively close, long parallel wires may lead to capacitive coupling and resulting noise glitches and signal integrity based timing delays and non-functioning chips if the proper analysis is not done.

Fortunately, noise analysis is standard in modern CAD tools. Parasitic extraction combined with static timing analysis can accurately calculate the impact of crosstalk; however, since physical design involves lots of effort and time, this paper develops some handy guidelines to implement Ruche Networks in a systematic way.

The central question is, how far can we push it? Although some designs will not need this much bandwidth, our results suggest that you can attain 50% utilization of wiring resources; essentially every other wiring track in the dense, medium range 2X metal layers.

Toolbox of Techniques. We conducted a series of experiments to study the properties of the wires and understand how signals behave under various routing configurations. Conventional wisdom [12], [13], [14] states that timing and energy optimized wiring employs single-width wires that are spaced apart by skipping wiring tracks ("double spacing"), but also that in cases where there is too much noise, adding shield wires may be necessary, slowing down the wires and increasing energy. It also suggests that it is desirable to "schedule" wires so that neighbor wires do not switch at the same time. Finally, it suggests that staggering repeaters is very effective in eliminating both noise and capacitive coupling timing effects. We found that the latter technique was too challenging to implement in a real design, and that double spacing alone was not sufficient to eliminate noise, and that shielding-by-default was at least 2× slower in wiring delay. However, by combining spacing and running the different, delayed segments of long Ruche Channels in parallel, we were able to achieve optimal results. Moreover, we realized that triple spacing of wires would not actually improve things significantly, because metal density rules in VLSI require that the space be filled with metal anyways, essentially eliminating the benefit of the space.

Experiments. Our design pushes the limits of Ruche Networks to the extreme and make use of every other wiring track (minus those blocked by power taps) across two horizontal metal layers across the 178u edge of a tile, as shown in Figure 7, comprising an eye-popping 1100+ wires. This is an 8-20× increase in bandwidth density versus typical NoC designs.

To quantify noise and signal integrity effects, we first created a design with a set of long parallel wires driven by inverters from one end. Generally, wires on the edge that have only one aggressor are less vulnerable to crosstalk. First, we varied the spacing between wires that are all on the same metal layer to determine the maximum density of wires that can be routed on each metal layer without too much signal degradation. For single-spaced wires (no empty track between wires), the net delay can be as much as 1.6× worse than double-spaced wires. Single-spaced wires also have signals that failed static noise analysis. Double-spaced wires did not fail and had about half of the noise margin left.

Second, we vary the input delay on every other wire to find the minimum difference in arrival time required to mitigate the Miller coupling effect, which increases wire delay. Miller coupling reductions improved as the arrival times grew further apart. Therefore we want to make sure that neighboring wires do not switch at the same time. A similar effect can be observed when the wires with opposite signal direction are interleaved. Depending on the design this can make routing more difficult if the wires have to travel longer distance.

Third, we tried different inverter drive strengths and a different number of inverter stages to cover the required distance.

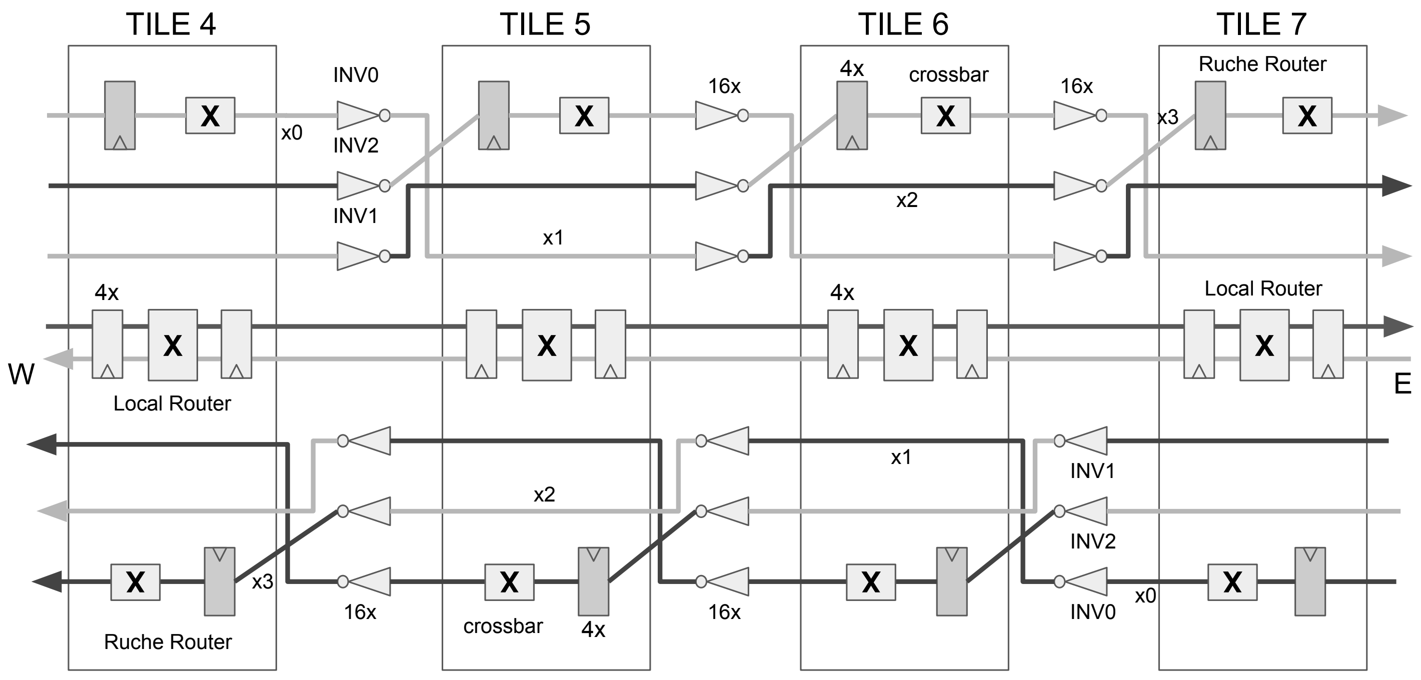

Fig. 8: **Wires that are far apart in terms of signal arrival time are placed on the same layer to minimize the Miller effect.** The color of wires indicates metal layer. Metal layer changes after each inverter stage. The upper half of a tile contains a ruche network going to east. The bottom half goes to west.

Inverters with greater driver strengths are generally good for improving net delay and are more resilient to crosstalk but have greater area, dissipate more energy, have larger cell delays and cause more crosstalk on neighboring signals.

Next, we tried interleaving wires on different metal layers. Between two horizontal routing layers is a vertical routing layer (and two via layers), so the actual distance between wires on different layers is quite far. Wires are usually thicker than they are wide, so most coupling capacitance comes from the adjacent wires on the same metal layer (Figure 7).

We also investigated utilizing upper metal layers which have bigger and much faster, lower resistance wires. Our surprising conclusion is that the noise on these wires prevents single track spacing, and thus the resulting number of usable wires is not worth the trouble. We ended up reserving this layer for routing the long trunks of a clock tree since we will be utilizing many lower layer wiring tracks to route the Ruche Network.

Finally, combining these insights, we implemented a 138-bit wide version of Figure 8 as a proof of concept to demonstrate that the Ruche Network topology can achieve reasonably good noise slack and meet our timing requirement. A set of 4X DFFs is spaced at a distance of one tile width (∼141 um) from another set modeling router launch. These wires are repeatered by 16X inverters. Long wires from different buses are interleaved bitwise on two alternating metal layers, such that wires are double-spaced in each layer. From earlier results, we found that the nets, X1 and X2, were not spaced far apart from each other in terms of arrival time of signals, so we interleaved X1 and X3 on the same layer instead in order to minimize the Miller capacitance.

Timing Arc	Delay (ps)	Timing Arc	Delay (ps)
clk-to-q	117	x0	33
INV0	33	x1	34
INV1	19	x2	35
INV2	26	x3	13
setup	17		
Total Gate Delay	212	Total Net Delay	115

TABLE I: **Under worst-case 2 GHz timing constraint, this interconnect should be able to propagate signals roughly over 3 tiles distance, while reserving half of the cycle time for the router.** Wire-related delay is about 225ps for long wires in Ruche Factor 3 crossing 0.42mm.

Wire delay (bad!) and noise slack (good!) are plotted in Figure 9 and Figure 10 with the X-axis being the position of the wires in each bus type. There are periodic spikes where the noise margin increases on wires occurring on edges of the bus, which confirm that the wires on the edge are more resilient to crosstalk. Even though the input wires to INV1 and INV2 are the longest segment in the interconnect, we can see that the wire delay and noise slack are excellent.

VI. Conclusion

Our future HammerBlade ASICs will incorporate this Ruche Network and we will report on results. We are excited to see more research on this topic and see others take advantage of the insights we have provided in this paper. Please contact us if you have any questions!

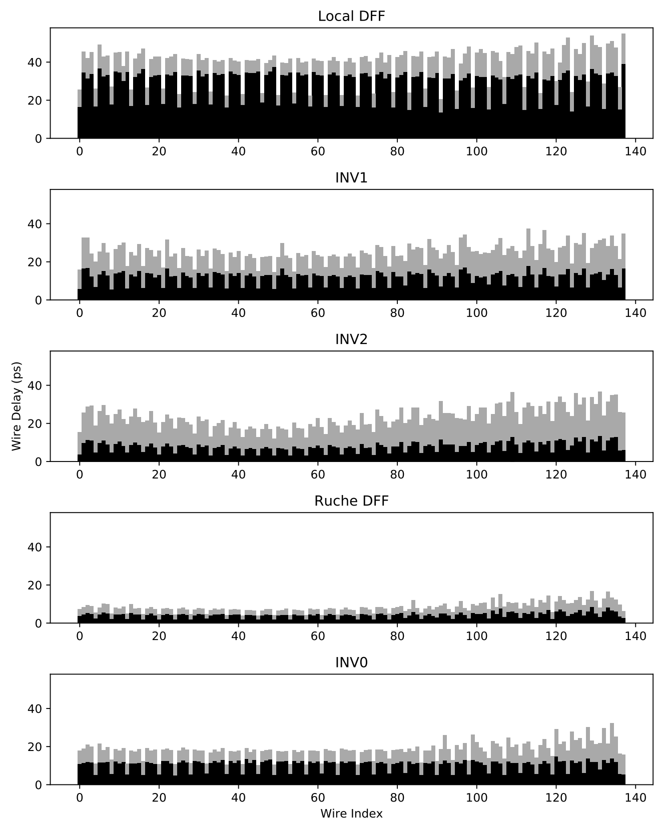

Fig. 9: **Ordering and spacing the interconnect wires was sufficient to drive down wire delay.** The gray part in each bar indicates the portion of delay that is induced by crosstalk. The black part is the delay on a wire itself. It can be seen that crosstalk-induced delay can be larger than the intrinsic wire delay for long wires (INV1, INV2).

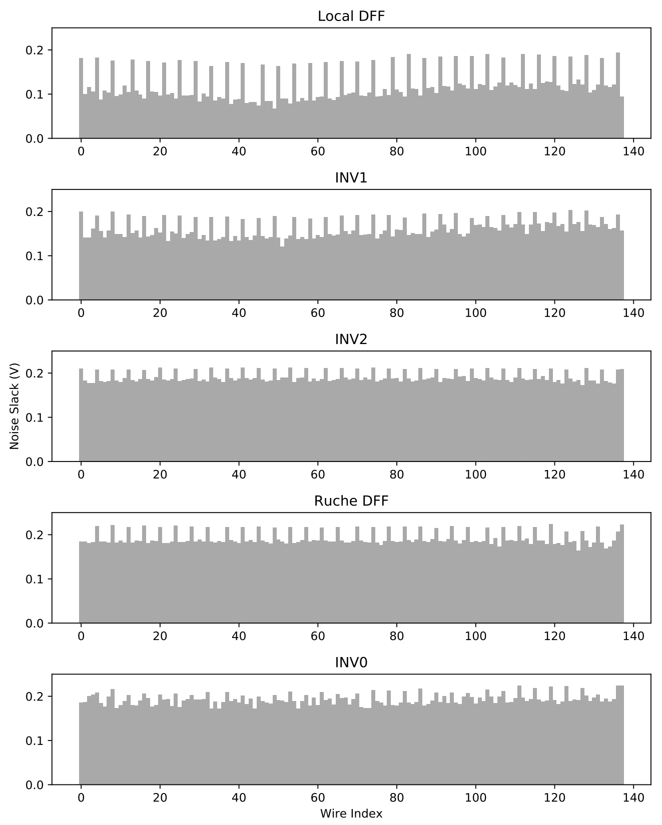

Fig. 10: **Spacing the interconnect wires was sufficient to contain noise due to capacitive coupling of densely packed, high-bandwidth NoC wires.** Peaks are for wires near power/ground via arrays.

References

[1] M. B. Taylor, J. Psota, A. Saraf, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, A. Agarwal, W. Lee, J. Miller, D. Wentzlaff, I. Bratt, B. Greenwald, H. Hoffmann, P. Johnson, and J. Kim, “Evaluation of the Raw microprocessor: an exposed-wire-delay architecture for ILP and streams,” in International Symposium on Computer Architecture, 2004, pp. 2–13.

[2] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C. Miao, J. F. Brown III, and A. Agarwal, “On-Chip Interconnection Architecture of the Tile Processor,” IEEE Micro, vol. 27, no. 5, pp. 15–31, 2007.

[3] S. Davidson, S. Xie, C. Torng, K. Al-Hawai, A. Rovinski, T. Ajayi, L. Vega, C. Zhao, R. Zhao, S. Dai, A. Amarnath, B. Veluri, P. Gao, A. Rao, G. Liu, R. K. Gupta, Z. Zhang, R. Dreslinski, C. Batten, and M. B. Taylor, “The Celerity Open-Source 511-Core RISC-V Tiered Accelerator Fabric: Fast Architectures and Design Methodologies for Fast Chips,” IEEE Micro, vol. 38, no. 2, pp. 30–41, Mar 2018.

[4] A. Rovinski, C. Zhao, K. Al-Hawaj, P. Gao, S. Xie, C. Torng, S. Davidson, A. Amarnath, L. Vega, B. Veluri, A. Rao, T. Ajayi, J. Puscar, S. Dai, R. Zhao, D. Richmond, Z. Zhang, I. Galton, C. Batten, M. B. Taylor, and R. G. Dreslinski, “Evaluating Celerity: A 16-nm 695 Giga-RISC-V Instructions/s Manycore Processor With Synthesizable PLL,” IEEE Solid-State Circuits Letters, vol. 2, no. 12, pp. 289–292, 2019.

[5] M. McKeown, A. Lavrov, M. Shahrad, P. J. Jackson, Y. Fu, J. Balkind, T. M. Nguyen, K. Lim, Y. Zhou, and D. Wentzlaff, “Power and Energy Characterization of an Open Source 25-Core Manycore Processor,” in International Symposium on High Performance Computer Architecture (HPCA), Feb 2018, pp. 762–775.

[6] K. S. Shim, M. Lis, M. H. Cho, I. Lebedev, and S. Devadas, “Design tradeoffs for simplicity and efficient verification in the Execution Migration Machine,” in International Conference on Computer Design (ICCD), 2013, pp. 145–153.

[7] Y. Ou, S. Agwa, and C. Batten, “Implementing Low-Diameter On-Chip Networks for Manycore Processors Using a Tiled Physical Design Methodology,” in NOCS, 2020.

[8] W. J. Dally, “Express Cubes: Improving the Performance of k-Ary n-Cube Interconnection Networks,” IEEE Trans. Comput., vol. 40, no. 9, p. 1016–1023, Sep. 1991.

[9] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha, “Express Virtual Channels: Towards the Ideal Interconnection Fabric,” in International Symposium on Computer Architecture. New York, NY, USA: Association for Computing Machinery, 2007, p. 150–161.

[10] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu, “Express Cube Topologies for on-Chip Interconnects,” in International Symposium on High Performance Computer Architecture, 2009, pp. 163–174.

[11] D. Petrisko, C. Zhao, S. Davidson, P. Gao, D. Richmond, and M. B. Taylor, “NoC Symbiosis,” in NOCS, 2020.

[12] A. B. Kahng, S. Muddu, and E. Sarto, “Tuning strategies for global interconnects in high-performance deep-submicron ICs,” VLSI Design, vol. 10, no. 1, pp. 21–34, 1999.

[13] A. B. Kahng, S. Muddu, E. Sarto, and R. Sharma, “Interconnect tuning strategies for high-performance ICs,” in Design, Automation, and Test in Europe. Springer, 2008, pp. 359–376.

[14] P. Gupta and A. B. Kahng, “Wire swizzling to reduce delay uncertainty due to capacitive coupling,” in International Conference on VLSI Design. IEEE, 2004, pp. 431–436.