Emperor Wears No Fabric

by KarbinCry

Table of Contents

Disclaimer

I am not an engineer, I do not work in IT, I have no practical experience. This text likely contains some mistakes and misconceptions.

If so and you notice them, please inform me.

The point of this article is to bring its topic closer to the general public, introduce it in the general discourse. As such, I welcome any insight from those who know more about it than me.

Introduction

In all but laptop markets, Intel is struggling to remain competitive against a resurgent AMD. What appeared to be impossible but three years ago has happened, and while Intel is stuck squeezing all they can from old processes and even older architectures, AMD caught up and as of now, overtook them.

But how did Intel get into this position?
We all know the usual suspects, which I've mentioned – 10nm woes, decade of architectural stagnation, advent of hardware security flaws…

All of these have been widely discussed and expanded upon, while one culprit curiously escaped scrutiny so far – the fabric.

The wires and signals and protocols pulling a chip, a system, together.

No system can be stronger than the interconnect holding it together, and that is a fact Intel feels more keenly than ever since the Athlon 64.

Let us discuss the protocols and their design, the most complicated element of a fabric.

1 Those Who Don't Learn From History *Will* Repeat It

We hear talk of the one-two punch of Sandy Bridge and Ivy Bridge, or the vision of the potential future repeat of that historic moment with Zen 2 and Zen 3.

But back in the early 00's, the dynamic duo was the K7 (Athlon XP) and the K8 (Athlon 64), and their impact was only limited by Intel using dirty tricks to stem the tide, until it could respond – very quickly[1] – with the legendary Core architecture.

While the success of Athlon XP was in significant part owing to the failure of the infamous NetBurst to get to 10GHz, Athlon 64 and the original Opteron were a true revolution – possibly the biggest in the x86 space, *ever*.
In 2003, AMD, smelling the blood in the water, went in for a kill with the first x86 64-bit CPU (starting a trend of being ahead of software – Windows would have a 64-bit version two whole years later) and, more importantly for us, the adoption of a *fabric* – one which also underlines the success of Zen – **HyperTransport**.

1 It only took 3 years to come up with a radically new design from NetBurst – however that was because the Pentium III was still being developed upon for low-power applications, and could be used for a "new" core.
Today, Intel only has Atom to fall back on.

Before the K8 architecture, the heart of any computing system was not the CPU, but the *chipset*.

The chipset was composed of two chips. The southbridge is what we call chipset today – additional I/O for slower or secondary interfaces, like SATA or USB.
The northbridge was at the center. Not only did it control all I/O like PCI, memory (MCU – memory control unit), and the southbridge, it was the only component directly linked to the CPU.
If we imagine a modern computing system as a diagram, at its center will be the CPU, with many branches leading to the chipset, PCIe, NVME… back then, the northbridge was the center, and even the CPU revolved around it.

K8 changed all this[2].

It had an on-die integrated memory controller (IMC), cutting memory access latency almost in half[3], and for many other traditionally northbridge functionalities, most notable the socket-to-socket communication on server boards and for the link between the CPU and the chipset, HyperTransport was used.

While for mainstream market, IMC was a big deal, it is in server where both novelties provided the greatest benefit, especially in multi-socket systems.

Previously, the two (or more) CPUs talked to each other through the northbridge. While this did have some benefits[4], there were some major drawbacks.
At that time, chipset used a 64-bit wide interface (Front-side bus – FSB) to send data to the CPU. With multi-socket systems, this link was divided across the CPUs, significantly decreasing available bandwidth.
AMD already tried to fix this issue with K7, as the northbridge for the Athlon MP (K7 server chips) was set up to provide separate 64-bit links to each CPU, but this made the chipset much larger. While

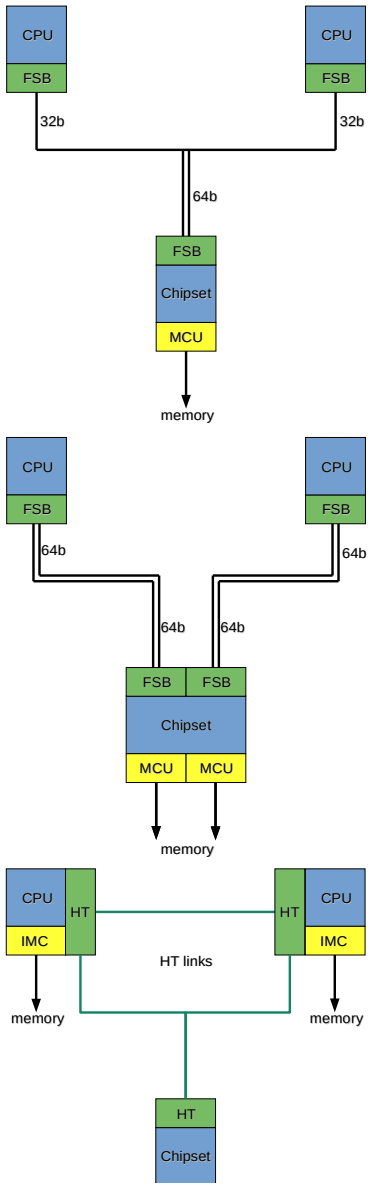

Fig 1: Evolution of 2P. Traditional, Athlon MP and Opteron. KarbinCry.

2 It was part of a long line of integration. Previously, for example, caches and FPUs were also external.
And today, EPYC did away with even a vestigial chipset, running all I/O directly to the Zeppelin die (Zen1)m, or the I/O die (Zen2).
Intel, on the other hand, decided to use a series of advanced chipsets (PCH) for their Xeon Scalable.
Whereas integrating caches or memory controllers were clear advantages, time will tell if the chipset survives.

3 Compared to K7 and NetBurst, in clocks. NetBurst has similar latency in ns, but the core has to wait double the cycles for needed data, still leading to a performance hit.

4 Notably, it was UMA – uniform memory access. This means all CPUs had the same latency when accessing all memory, because data from memory had to go through the chipset. This makes memory management much easier.
With NUMA – non-uniform memory access – each CPU (or die – on EPYC Naples or Cascade Lake AP) has some memory closer, and some memory farther. Say CPU0 needs data from memory controlled by CPU1 – then it has to go through CPU1, adding latency. Memory management becomes crucial, as you want the program to have data close to it.
Interestingly, EPYC Rome presents a UMA system with the IO die reminiscent of old chipset, but hides the disadvantages of the chipset system by density (all on one package) and close distance from the computing elements.

AMD could have made 4P boards for Athlon MP, they never did, because the building a 4x64b northbridge would simply be too expensive for it to be worth the effort.

With Opteron, each CPU has its own access to memory, chipset is greatly reduced, and each CPU communicates with the others directly, through its 3 HyperTransport links dedicated to socket-to-socket communication. There was no problem to make 4P or even 8P boards, and while the NUMA nature of such a system did require optimization (especially for 8P variant[5]), the benefits in memory bandwidth and capacity were *incredible* for the time, especially compared to multi-socket Xeons.

There was another interconnect – the System Request Interface – on the K8 die, which made the architecture ready to be used in a multi-core design. In 2003, AMD said they are just waiting for a new process node to make such a die small enough to be economical.
Of course, Athlon 64 X2 came out in 2005, together with, also dual-core, Pentium D, however the two NetBurst cores of the Intel offering still had to go through the chipset to communicate, despite being on the same die[6].

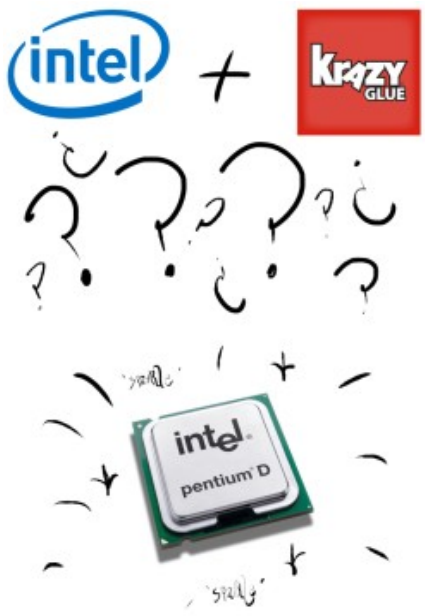

Fig 2: Meditation on the Origin of Pentium D.
A Local Priest of 10GHz, Bloax.

It took Intel another *year* to release Core 2, their first true multi-core product.

All three big innovations of K8 are with us to this day.
AMD64 was the first major expansion to x86 ISA *not* made by Intel, and forced a cross-licensing agreement which protected AMD's access to the x86 license.
Multi-core design replaced the mirage of 10 GHz.
And HyperTransport has been the foundation of AMD's products ever since.

It took Intel years to catch up back then.

Unfortunately for their customers and shareholders, Intel failed to learn their lesson.

2 If You Like It, Put a Ring on It

While the Core architecture was Intel's first true multi-core SoC, it's not very interesting to us. The cores communicated through a wide point-to-point connection to the shared L3 cache. Point-to-point is one of the best performing topologies, but also one of the most costly.

We then skip straight to Sandy Bridge, and its iconic ringbus[7] Sandy Bridge brought more, and larger, cores to the mainstream, and included an integrated GPU, which also needed access to system resources.

---

5 With 3 links, 8P system has 3 tiers of distance: memory directly to the CPU, memory with 1 hop and memory with 2 hops, as 3 links aren't enough to link every CPU to each other. This will come up again later.

6 Curiously, the debates about glued cores began right in 2005: https://hardforum.com/threads/pentium-d-only-true-dual-core.915183/
The debate is curiously prognostic, starting with a speculation of EPYC Naples… except it was to be made by Intel back then. Speculation. Speculation never changes.

7 Nehalem-EX Xeons also used ringbus, but Sandy Bridge brought its refinement into the interconnect Intel uses to this day.

Ringbus is a very simple, elegant, and effective design. While being cheap and easy to route – each packet can only go in two directions – it is very fast, low-latency and scales well up to 6-10 cores.

The single ring is composed of 4 physical circuits – the big one for data, request and acknowledge rings, and snoop ring. Infinity Fabric uses a similar approach, being divided into SDF – Scalable Data Fabric and SCF – Scalable Control Fabric planes. Because network commands are usually smaller size, dividing fabric into a data and command sections allows the command one to run faster, or use a narrower link.

Aside for scaling to higher core counts, what are the disadvantages of the ringbus?

Let's start with one that is a benefit to ringbus itself, but arguably a problem for Intel in general, and that is the separate snoop ring.
Snooping is a method for preserving cache coherence, making sure each unique cache address leads to the same data. If this coherency isn't maintained, core that needs information from cache can end up with two different, conflicting, sets of data. Obviously then, keeping shared caches coherent is crucial, and four different modes of snooping were implemented. Each variant has its pros and cons, and the separate snoop ring allowed use of all four without big overhead penalty. But to remain compatible, other interconnects, some of which could *not* accommodate this protocol so easily, had no choice but to adopt it as well.

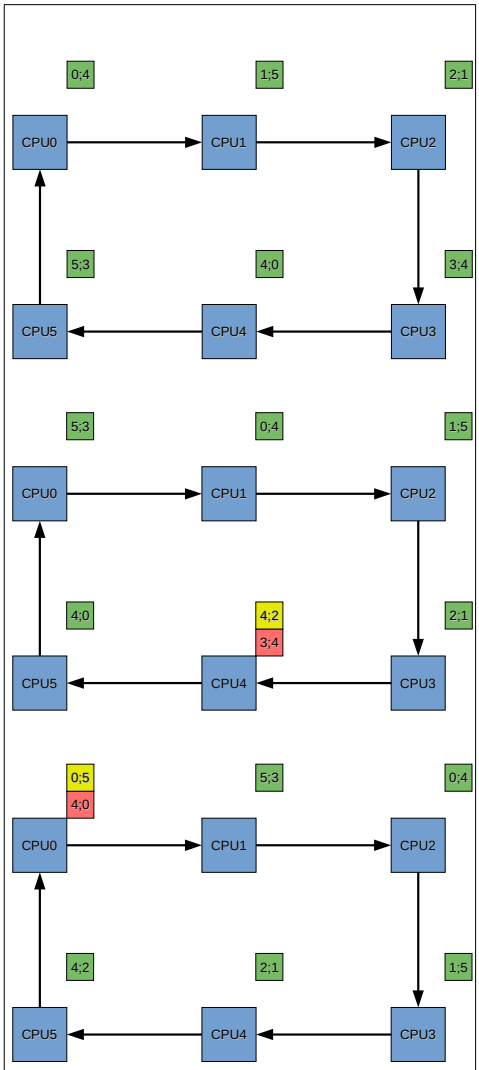

Fig 3: Jammed ringbus.
New packets in yellow, delivered packets in red.
KarbinCry.

Now to the limit inherent and impacting ringbus itself – bandwidth.
Each stop (node) on the data ring can send 32 bytes per clock. If there are two stops, 64 bytes can be "in" the network. If there are three stops, it's 96 bytes.
While it sounds like perfect scaling, in effect it isn't. Many routes will be longer than hops between two adjacent stops, which can cause "traffic jams", and sometimes 32B per stop might not be enough, meaning some data has to be moved in more parts, adding cycles of latency.
Compounded with the latency increase inherent to the number of stops – if you add node C in-between nodes A and B, then latency from A to B (for a 32B packet) goes from 1 cycle to 2, ringbus has severe scaling difficulties, which increase more and more rapidly as you add cores (or other nodes).

The network is also essentially unbuffered (there are very small buffers), which means at any point, there can be only as many 32B packets as there are stops. If enough packets need to travel a long distance, this limits nodes' ability to put new packets on the bus, leading to stalls.
With buffers, a "waiting room", some of those would be prevented – the bigger the buffers, the less stalls. But a proper, solid buffer can take up significant area, while one of the benefits of ringbus was its minuscule footprint.

See Fig 3, which depicts 3 cycles of a very basic ring network, with packets annotated with their origin and destination (0;4 is packet from core 0 sent to core 4). It is very much the worst case scenario, but it shows a situation where most of the cores are stalled, most of the time. In principle, similar thing can happen for any ring topology, just at higher network saturation.

How do you get more data through?
You can make the traffic faster, or you can make the road wider.

Ringbus is very fast already, not much headroom there.

What remains is width.
Because of the low buffering and years of optimizations, increasing ringbus to, say, 64B would require a redesign of the cores and caches.
But what can be done quite easily is to put in a second 32B ring – and that's exactly what Intel did for its Xeon line. This dual ringbus has a disadvantage in more complex routing (packets don't decide to go one way or the other, but also which ring to use), and continuing adding rings quickly diminishes the returns (hence, no triple ring), and so it isn't a long-term solution, but it's something they could do on their mainstream CPUs right now.

Ringbus also quickly gains latency as you add nodes. To move data from one node to the other takes one clock, and so every added node adds latency linearly.

This was a problem for Xeons since Sandy Bridge, and Intel gradually evolved their solution, arriving at a chained two-ring solution with the Haswell architecture.

Each ring services at most 10 cores, and they are connected by two switches, located evenly across the ring. If core 0 needs to communicate with core 19, it doesn't need to go through 19 nodes (and cycles), just, at most, 5 plus the switch[8].

This sounds great, so why couldn't you go to three, or more, chained rings?

We can glean a possible reason could in how a CPU with two rings behaves. It is in many ways more akin to a dual socket configuration.
Despite being on the same die, there was enough of a latency hit going from one ring to the other that Haswell and Broadwell could be set up such that each ring was a NUMA ring.

NUMA configuration was well known at that point, but this added another layer of complexity, and as I understand it, the internal topology of the CPU was not exposed to software in the same *way* as traditional, multi-socket NUMA, which necessitated specific optimizations on the side of the OS or the individual program.

Furthermore, while both rings had half of the PCIe and other I/O, but only one ring had a memory controller, leading to asymmetry. Not only was memory much farther for the secondary ring, the rings also had unequal number of nodes (as the primary ring had extra node – the IMC).

This creates a clear "weak" node.

Take an example from AMD. EPYC Naples 32c was composed of 4 Zeppelin dies, each with some direct memory access.
This allows the program to place itself in memory which is directly connected to the cores

8 This is greatly simplified. There are other elements on the bus than cores, like the I/O and iGPU. The basic principle holds.

executing it. That is the normal NUMA optimization.
But with Threadripper 2990WX, which is the same silicon, two Zeppelin dies have their memory controllers disconnected, leading to performance degradation on those two dies, as all memory accesses they made had to go through the other die.

Optimization then becomes harder, again. Just consider how long it took Windows to properly schedule for 2990WX – and take a CPU with a similar problem, but one in which the OS doesn't immediately see it.

Dual ringbus and chaining allowed Intel to go all the way to 24 cores with Broadwell, it was now a complicated, cumbersome – and most importantly, tapped out – interconnect.

Something new was required to take core count higher.

3 Gridlocked

After reaching the limits of a two-ring topology, and realizing adding another ring would be too complicated and have too many disadvantages, Intel got to work on a new intra-chip interconnect, following a mesh topology.

Mesh is a well-known topology, one of the best understood ones. Even Intel had direct experience using it – starting in 2007 with the Polaris chip, a many-core design which paved the way for Xeon Phi[9].

But the version Intel used for the Xeon Scalable lineup, and for Knights Landing Xeon Phis is significantly different, and we can trace its research as far back as 2014, three years before it was used.

Mesh is a great choice, especially for large core-counts on monolithic dies.
It is a very scalable arrangement, and while distance increases latency, it does so much more gradually and gently than a ring.

Each node in a mesh is called a **tile**[10]. Each tile has 4 connections – North, South, East and West. The farther the destination node is, the more routes a packet can choose from. This increases bandwidth – if one route or connection is full, the packet can go around.
You could even optimize based on latency, having packets of varying importance, with the most crucial ones being prioritized and given the shortest route.

Such extensive flexibility requires much more complex router or routing logic, as each router has to decide where to push a packet to take it to its destination, each having 4 choices[11].

9 And not only that. Polaris used a full mesh, and the single largest element in a tile was not the core, but the router. As a result, it required stacked SRAM as there wasn't enough space for cache.
Polaris was a research chip, but it shows that often the reason a certain technology – 3D stacking – isn't implemented in production chips isn't technical, but economical.

10 In case of Xeon Scalable, each tile contains a core and every resource assigned to the core.
Specifically: L2 cache, local L3 slice, FIVR (Full Integrated Voltage Regulator), CHA (Caching/Home Agent), CMS (Converged Mesh Stop), AVX-512 unit, etc.

11 Effectively 3 choices – the option to go back is usually culled with little impact on quality of the network.

That is why Intel uses a very rudimentary routing. All packets first go North or South, and then East or West. As simple algorithms go, it is a good one, nevertheless there is a lot of potential left at the table.
Even then, the Caching/Home Agent and Converged Mesh Stop[12] take up around 10%[13] of a tile.

Another quirk of Intel's mesh are the IMC tiles. There are two on each "edge" of the mesh, fully integrated into the grid. This creates a NUMA-like situation, similar to chained ringbus chips, albeit not as significant, as both clusters have direct access to some memory[14].

It doesn't matter how well you build a road network, a couple traffic lights with bad timings can make traffic hell all on their own.

To make mesh as great as it *can* be, Intel needs to beef up routing logic, and since even now it takes up space of 3 cores[15], it's not possible, within a reasonable die size, with a monolithic, planar design.

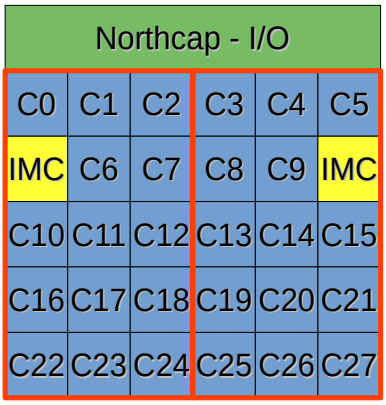

Fig 4: Diagram of Skylake X, showing the two sub-NUMA clusters.
KarbinCry.

4 One Way Highway to Hell

So far we've focused on the intra-chip fabrics, Intel's responses to AMD's multi-core innovation.

Now with all that we've learned, we can get to the main event, red versus blue, HyperTransfer and QuickPath Interconnect (nowadays – Infinity Fabric and UltraPath Interconnect[16]).
This is the class of interconnects which will drive the chiplet and stacked era.

It took Intel only one year after AMD to release a genuine dual- and quad-cores.
But it was 5 whole years before they had a competitor to HyperTransport in inter-chip communication.

Work on YAP (Yet Another Protocol, early name for QPI[17]) begun at least in 2004 (likely at least in 2003 when Opteron launched), but first chips with QPI debuted in 2008 with Nehalem Xeons.

The design was focused on latency (and thus, speed) and sturdiness[18].
Result was a 20-bit wide[19] link, with 16-bit effective data width, and speeds greatly exceeding those of HT. Each clock, two signals are sent, for a total per-clock bandwidth of 4 bytes.
The link is composed of 4 quadrants, each 5/4-bit wide. If the connection is damaged or unstable,

12 CHA ensures cache coherence and CMS is the actual interface with the mesh.
13 Annotated die shots I found only show CHA, at around 10% of the die. I think CMS is included in the CHA in those shots, but if it isn't, then the area would be even more than 10%.
14 Solution could be to have IMC *around* the mesh, interfaced to several "lanes" to decrease both the bottleneck and memory access latency variability.
15 On Extreme Core Count dies.
16 I hope Nvidia renames NVLink to SuperLink to get back into the race for the cringiest, most overblown fabric name.
17 Other provisional names were YAP+ and CSI – Common System Interface.
18 Core architecture Xeon lineup aggressively adopted features improving quality of service to overcome Opterons, which lacked these improvements.
Similarly, EPYC includes extensive security features to exploit lack thereof on the Intel platform.
19 Intel calls it "x20", however data only has 16 bits, and there is extra clock bit.

the interconnect will switch from 4 quadrants to two or even just one, to maintain stable connection[20].

Four snoop modes were implemented, same as with the ringbus.
The UPI, modified version used in Xeon Scalable, only retains directory-based snooping to lower overhead.

Now compare it to Infinity Fabric 1.0[21] and its IFIS (Infinity Fabric InterSocket)[22].

IF 1.0 is also 16-bit wide, but send 8 serialized transfers every clock, for a per-clock bandwidth of 16 bytes.
Compared to QPI and UPI, which have a 20% overhead (4 out of 20 bites are not used for data), IFIS has 11.11% overhead[23].
Infinity Fabric should also be able to dynamically narrow the connection to maintain signal integrity, but I don't know if this is implemented – probably not, however if it is needed, it should be easy to implement.

Per clock, IFIS is clearly superior. But we know QPI was built for speed, can it catch up?

Last version of UPI can run at up to 5.2GHz[24], while IFIS can only clock to 1333MHz[25], making UPI almost four times as fast, and equalizing the two fabric's[26] theoretical performance.

In practice though, IFIS still has the upper hand.
Imagine a narrow road with traffic going at 75 MPH, and a wide road going at a 50 MPH. While in ideal circumstances both might allow the same amount of cars to pass per hour, if anything goes wrong, the narrow road will be impacted much harder.

And such is the situation with the two interconnects.
QPI/UPI is undoubtedly great for small snippets of data, while IFIS needs to fill a 16B packet, but in modern computing environment with higher and higher I/O bandwidths, this single advantage is not enough, by far.

Now we have been talking solely about the interconnects. Let's talk about how they are used.

There is no Intel CPU which has more than 3 UPI links[27]. However, each Zeppelin die has its own IFIS link, and 3 IFOP[25] links. This means each 8 cores have a link!
Compare this to Cascade Lake AP, where two 28 core dies have to make do with a single link.

---

20 I found mentions this would be used on 8P boards to allow each CPU to connect to every other, by only using 2 quadrants in each link.
However, I never found any evidence.

21 I couldn't find as much information about Infinity Fabric 2.0, but I will provide details on it in footnotes, wherever I can.

22 There is also IFOP – Infinity Fabric On-Package, which is 32-bit wide and makes 4 transfers per clock and has lower overhead

23 Overhead caused by cache coherence mechanisms is not included, as these requests travel through the data plane.

24 Which has a negative. Every time different clock domains communicate, the difference and asynchronicity of clocks causes additional latency – clock domain latency.
If the clock are tied – equal or one is an integer multiple of the other – this penalty is avoided.

25 Based on fabric clock. EPYC Naples has top memory speed of 2666MHz.

26 Infinity Fabric 2.0 can do even more transfers, as it is carried through PCIe 4.0 connections. On Rome, this is about 1.6X the speed of IF 1.0, but for GPUs they use connection more than twice as fast. Rome probably doesn't go as fast because it doesn't need to, and saves power by running slower.

27 Nehalem-EX Xeons had 4 links, but that was an anomaly. Cascade Lake AP processors also tout 4 links, however these CPUs are 2 dies on one package, and each die still only has 3 links, and one is consumed to connect the package, giving a virtual "4 links".

UPI is a bit bigger than IFOS/IFOP, and the whole implementation is different, of course. UPI was never built for these huge core counts, for the massive I/O bandwidths, or to connect chiplets.

While QPI started as greatly superior to HyperTransport, the openness of HT allowed it to change, to adapt, in ways UPI simply couldn't, and likely still can't.

QPI/UPI is too big, too narrow, too inflexible to carry Intel into this year, much less to the future.

5 One Ring to Rule Them All, in Hierarchy Bind Them

Finally, the ground has been laid, the limits of Intel's current fabrics exposed, and we can turn to ways they can fix them, starting with the ringbus.

Ringbus seems quite hopeless, as chaining rings together failed, but hear me out – what if we put in *multiple* rings and *chain* them together?

Of course not quite the way used on Haswell and Broadwell, but with a hierarchical setup.

This topology has several local rings, all separate of each other, servicing their own islands of cores and caches. A global ring would then connect the local rings, and also include all I/O.

The local ringbus could be Intel standard, with the same old structure and protocol, with few necessary alterations, and the global ring could be scaled much more flexibly, free from the constraints of being directly tied to the width of the core.

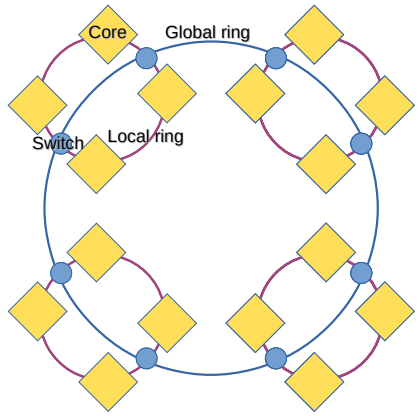

Fig 5: 16-core Hierarchical Ringbus. KarbinCry

Routing obviously becomes more complex, but still manageable, using new packaging techniques.

By which I mean chiplets and 3D stacking.

Global ring, routers and switches would be on a big "interposer" die, along with I/O, iGPU or whatever else, while the cores would be on tiny chiplets of 4 to 8 cores.
Even a configuration similar to EPYC Rome – 8 chiplets with 8 cores each – should be possible even in this basic design.
Using dual ringbus on chiplets would enable another core count explosion, not to mention we could try and make global ring faster or wider, allowing it to handle more local rings. And if we put in more switches per local ring, we can again increase core count on the individual chiplet[28]…

The solution does sound rather expensive, but that impression is, I believe, erroneous.

The big base die with the global ring wouldn't cost as much as Cascade Lake SP dies, given it would be much less dense, improving yields, and chiplets would have almost 100% yield on 14nm. Enhanced binning allowed by using tiny core dies would also enable much wider and meaningful product segmentation.

Even if I'm wrong and the silicon and packaging cost would increase dramatically, Xeon's huge margins will sustain it.

28 A ring doesn't need to be a circle, we could zig-zag so that the global ring intersects, say, at 8 points or even more. What matters for latency is the total node distance, and speed (more nodes on fast global ring has lower impact than more nodes on a slower local ring).

As for the consumer market, the switch might take longer, but it will happen. We already see they are willing to absorb big packaging costs with Lakefield.

Hierarchical ringbus is the most Intel solution I have. It uses their most successful and well-known interconnect, retains low latency[29] while gaining enough scalability for at least 5 years.

It is elegant, simple, powerful, and interesting by being a completely different to AMD's approach.

6 Layers and Wormholes

To find a fix for the mesh is a bit harder, considering the routing complexity, and area which needs to be dedicated to it.

Given the advantages and ease of hierarchical rings, why would we even entertain the idea of fixing the mesh?

The obvious answer is redundancy.
Work on both, and you're safe in case one approach proves unviable.

What a position would Intel be in now, if they had one team working on the "overambitious" 10nm, while a secondary group develops a more modest node shrink?
The R&D would go up, but if there's one company in the world that can afford it, it's Intel.

But mesh also has some technical advantages over hierarchical rings.

First is bandwidth. Mesh simply has more paths, more options. Currently they don't do a great work of utilizing that feature, but that can be remedied.

Second is scalability. A mesh, especially the variant I will outline, has almost limitless scalability.
As core counts increase, so does the advantage of mesh over hierarchical rings.
Not to mention the ease of adding cores to a mesh, thanks to its modular, tiled design.

Which brings me to the third advantage, flexibility. Building a heterogeneous system on a mesh is easy – just put in a different tile.
Hierarchical rings simply don't have the incredible granularity of mesh.

All of this, of course, requires intelligent routing, fully utilizing the topology of the network.
And how else to do that, than with 3D stacking.

The solution, again, is active interposer, here completely dedicated to routing. Even I/O can be on the tiles, as long as we get a robust networking logic.

Then, every tile can be a single tiny chipset.

You could add AVX-512 tiles (as Intel seems intent on pushing AVX-512), removing clock penalty for AVX workloads, GPU tiles, AI tiles… all working nicely in a "menu" style, semi-custom business model. Need AVX? Order more AVX tiles, etc., whereas with a hierarchical ring, you would be more limited by the size of chipsets[30].

---

29 This depends on the switches and routers, but given they will have so much space on the interposer die, they should be able to make something very fast and efficient.

30 So why not make a single-core chiplets for hierarchical ring? Well, because a single-core chiplet can't have its own ring. The global ring would become the local ring, it would be the same as current ringbus.
You could make a big interposer with global die, a small interposer with local ring and then put in these tiny chiplets, but that adds complexity and cost.

When we enter the age of thread aggregation, where the number of logical cores will be lower than that of physical cores, Intel could just add a couple scheduling tiles to handle the added complexity of distributing single-threaded code to many physical cores.
Interspersed with more specialized computing elements, like GPU, AI or AVX units, or even FPGA and ASIC tiles, the raw power and efficiency would be magnificent, and it would all be possible on a proper mesh.

A big problem is again, latency. And, if we have I/O tiles, then the connections of these would be under more load than other connections, while nodes along the edges have fewer connections and routing options.
This can be mitigated with clever scheduling, but fundamentally, it isn't solvable in a mesh topology.

That's why I'd add *wormholes*.

Connect each parallel edge with a special, longer link. It will have bigger latency than a standard link, but much lower than hopping all the way using the normal links.
These wires could just be placed in a lower layer of the interposer, shortening the distance. Just like a wormhole.

And we're not limited just to the edges.

A very beneficial application would be to add extra connections to IMC tiles, giving the whole system uniform and low memory access latency. We would go beyond 4 links, necessitating extra routing logic, but the benefits would be big.

Mesh is bulky, brute-forcing its way straight through any scaling limits.

But just as the beauty of a hierarchical ring has its place, so does a massive, complicated sledgehammer, perhaps.

7 If You Can't Beat 'em, Join 'em

Despite the rising per-package core count, multi-socket will still have its place for years, just like high-performance off-board connections, and connection between separate dies (or stacks) on a package.

And it is here that Intel finds itself in a particularly bleak position. QuickPath Interconnect is simply obsolete, and renaming it to UPI doesn't change that it is at the very edge of what it can do – and yet it's not enough even now, much less in the future of larger, more heterogeneous systems.

Additionally, it is a proprietary standard, narrowing the use and ecosystem of UPI, while accelerators, FPGAs and ASICs rise in importance.

Given how far behind Intel is, and the need for a more open standard, adopting HyperTransport would make sense.

Of course, they could use other symmetrical cache-coherent interconnects – OpenCAPI stands out, as it has already proven itself on the POWER platform, but HyperTransport has the heritage, x86 track record, and ecosystem to make it the best choice, not to mention a license friendly to proprietary supersets (like Infinity Fabric).

Plus, it would be oh so poetic for Intel to resort to HT.

And it's not like they are above such a move. If pushed hard enough, Intel is willing to simply follow AMD.
With x86-64, they had trouble making their own 64-bit extension to the ISA, and eventually just licensed AMD64, giving x86 license in return, squashing any, however small, chance that Intel could take their last standing competitor's license away[31].

Adopting HT, in comparison to losing the one way for complete domination of the x86 market, wouldn't be all that bad.

As for AMD, which retains influence over the HT Consortium, they have reason to allow Intel to use HT. The ecosystem around it will grow much faster with both big server chip makers using the protocol, benefiting AMD as well.
Even if they were opposed, the add-in card manufacturers invested in HT are strong enough to push it through.
And so, both x86 companies benefit from an explosion of HT-compatible hardware, and those making all this hardware will obviously move more units, now that even on the Intel platform, their products would reap full benefits of HT.

Of course not everybody *will* benefit, but everyone – including the rejuvenated, confident AMD – will *think* they will win.

And if that's not enough, there is historical precedent.
Transmeta, the last successful startup of the dot-com bubble, used HT in their x86, VLIW (what?!) Efficeon architecture launched in 2003.

And, curiously, after a complicated history[32], Intel ended up with a lot of Transmeta IP, perhaps including their implementation of HyperTransport, giving Intel a solid base to work from, to build something great upon.

Certainly – something greater than UltraQuickPath.

8 Conclusion

That's it, congratulations, you made it! Well, unless you just skipped to the end – in which case, boo[33]!

As stated in the Disclaimer, there were probably some mistakes I made along the way, and Intel almost surely won't go with one of my ideas, but what I wanted most of all was to show how important fabrics are now, and how their role will expand even further in the future.

And, I wanted to make you *excited* about interconnects. They should be just as much a part of our discourse as cores, dies, chiplets, packages and memory.

Because that which binds them all together is just as, if not more, important.

---

31 There is, of course, VIA/Zhaoxin, but their license deal was also re-structured by the Intel-AMD settlement. Just years before, VIA was buying out other x86 companies in part to protect their access to the x86 license.

32 Transmeta turned into an IP licensing company, bought by a company dissolved later that year, and their IP portfolio ended up with Intellectual Ventures, dubbed the "ultimate patent troll".
Before that, in 2007, Transmeta sued Intel for breach of patents pertaining to power saving features, and the two companies settled, with Intel forced to license IP from Transmeta

33 It's certainly not as if I'm one of those people who always read the last page of the book first, nuh-uh, not me, sir!

Case in point – CLAP[34].

I really think Intel wanted to make something more than just a prestige, world-record breaking chip. They could've use super-binned lower clocked dies to make Xeon optimized for throughput – area where AMD is ahead the most.
The 400W meme which now rests in the kitchen mausoleum along with Bulldozer hotplate and Fermi grill would still be made, for those records, but the lower SKUs could've been very interesting.

But when you have 28 cores connected to another 28 cores with a link slightly weaker than the one AMD uses for communication of 8c dies, you just can't scale, *especially* in those high throughput workloads.
And if your interconnect isn't strong enough, no matter how good the silicon is, your system just won't work.

If we want to understand what comes next, we need to understand fabrics, we need to talk about them.

Because the more we talk, the more *actual* experts, not self-taught theoreticians like me, come forward, to show us what comes next.

Just when I was writing this article, there were two new pieces of information with great relevance to this piece.

We got more information on Intel's next offerings for server – 14nm Cooper Lake and 10nm Ice Lake-SP (and some on Ice Lake-AP).
Both use UPI.
Even worse, the high core count Cooper Lake is almost certainly built, just like CL-AP, from two dies on one package.

We also learned that in some workloads, EPYC 7742, the 64 core beast, is slightly bottlenecked by Infinity Fabric.

There is only one difference.

AMD has the *answer* to their small problem, while Intel is unable to deal with their deep, *old issues*.

9 Acknowledgments

WikiChip – invaluable resource, I just wish articles on older stuff were more fleshed out.

AnandTech – for great breakdowns, and especially of older architectures.

ServeTheHome – for their accessible coverage of server platforms and hardware

34 Cascade Lake Advanced (heating) Performance.