3D Stacking of High-Performance Processors

Philip Emma, Alper Buyuktosunoglu, Michael Healy, Krishnan Kailas, Valentin Puente, Roy Yu, Allan Hartstein,
Pradip Bose, Jaime Moreno
IBM T.J. Watson Research Center, Yorktown Heights, NY 10598

Abstract

In most 3D work to date, people have looked at two situations: 1) a case in which power density is not a problem, and the parts of a processor and/or entire processors can be stacked atop each other, and 2) a case in which power density is limited, and storage is stacked atop processors. In this paper, we consider the case in which power density is a limitation, yet we stack processors atop processors. We also will discuss some of the physical limitations today that render many of the good ideas presented in other work impractical, and what would be required in the technology to make them feasible.

In the high-performance regime, circuits are not designed to be "power efficient;" they're designed to be fast. In power-efficient design, the speed and power of a processor should be nearly proportional. In the high-performance regime, the frequency is (ever progressingly) sublinear in power. Thus, when the power density is constrained - as it is in high-performance machines, there may be opportunities to selectively exploit parallelism in workloads by running processor-on-processor systems at the same power, yet at much greater than half speed.

1. Introduction

In previous work [1][2][3][4], others have considered dividing up the parts of a processor, and aggregating those parts across multiple planes that are stacked atop each other. The goal was to reduce the lengths of the wiring paths so as to make a faster processor.

Other work [5][6] has examined stacking entire processors atop each other, but dynamically allocating the resources of those processors (registers, queues, functional elements) to different (and possibly co-operating) threads to optimize performance for a fixed set of parts. In those cases, the processors were not high-powered processors.

And many have considered the (now obvious) use of 3D in which the storage of a processor (or processor system) can be greatly augmented by stacking that storage atop that system [7][8][9][10]. Not only does this allow the use of heterogeneous technologies in the same package (e.g, DRAM with high performance logic), but it makes the access path(s) to that storage shorter.

What we present in this paper is the idea of stacking high-performance processors together when the power density is constrained, but allow only one of them to run at full speed at any time. We consider how to facilitate this physically, and suggest different operating modes to help different workloads in different ways. By placing processors atop each other, we facilitate new modes of operation that wouldn't be feasible in 2D.

In Section 2, we describe the 2D processor that we're using, and the four modes of operation that we'll use when we stack two of them together. Of course the pair of chips has twice as many cores, and their physical coincidence allows some new operating modes, each of which is constrained to use the same amount of power. In Section 3, we will show the performance of all each of the 4 modes on 16 different workloads of 6 different types.

While we do not present the multiprocessor performance analysis in this work, in Section 4 we show how 3D allows a much more robust wiring infrastructure that gives the MP system much wider interconnection bandwidth than would be possible with the two chips wired together in 2D.

But when you stack two processors together in 3D, there are many new physical challenges that need to be addressed. While providing richer bandwidth, and enabling better coupling between the caches, the power distribution and the thermal management become more difficult. In Section 5, we discuss the thermal issues, and show how they constrain how thin the layers can be. And in Section 6, we describe the physical properties of the Thru-Silicon Via (TSV), which is not needed in 2D, and which imposes some new constraints that will effect the wiring density and the layer thickness as well.

In Section 7, we show how processor-on-processor chip systems can be used to provide much earlier yield, and therefore earlier shipment of products. And in Section 8, we show how a processor-on-processor configuration naturally allows 100% checking of all of the logic using an "R-Unit," which we will describe.

In Section 9, we explain how the wiring constraints, (specifically the TSV pitch) may make the actual wiring of a chip-to-chip structure much worse than ideal; and therefore, what kinds of wiring densities are likely needed to do some of the finer interconnections (e.g., macro-to-macro) that are frequently discussed. We also describe a couple of new 3D structures that could be done with a finer wiring pitch, including a structure that would simplify the 2D wiring.

In this paper, in addition to showing the realizable processor-on-processor structures in the first sections, we consider the real constraints of 3D technology that need to be addressed to actually do them. We also discuss the basic constraints that render many of the other widely suggested applications of 3D systems impractical today, and we explain the technology problems that need to be solved first. These things are doable, but not yet.

2. The System and Its Operational Modes

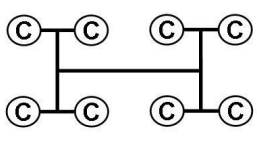

Figure 1. An 8 Core Processor as Connected

For this study, we used 45 nm CMOS with eDRAM for the L3 caches. Each core has its own private L3, and all

cores on the chip are connected in a tree-like configuration, as shown in Figure 1. Note that as we've already pointed out, in high-performance circuit design, the circuits are not designed for power efficiency; they're designed for speed.

To illustrate the power-speed tradeoff, Figure 2 shows a typical power-speed curve that was generated by varying the voltage, modeling the circuit performance subject to the voltage, and determining the operating frequency in which all paths in a core safely meet their timing requirements. We generated these points by starting at our desired operating frequency, which is 4 GHz. What's clear is that the curve is not linear: several design points can be chosen from it to meet different objectives.

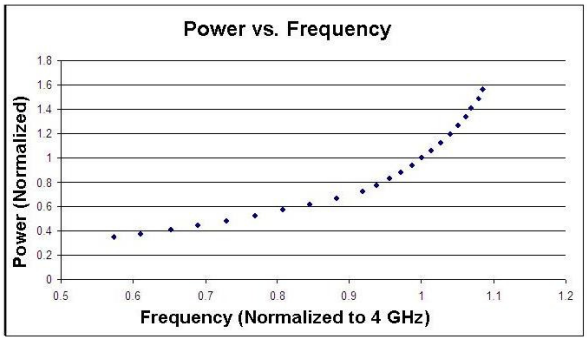

Figure 2. Speed vs. Power

In Figure 3, we've simply marked up the curve of Figure 2 to show what we are doing in this piece of work. Note that at our chosen operating point (Frequency = 1), the Power versus Frequency curve is beginning to curve rather sharply.

At very low voltage, the frequency is (roughly) linear with power. But as we continue to increase the voltage, the frequency stops increasing at this rate (fewer of the paths make their nominal timings), so the slope of the curve starts increasing. There ***is*** a Maximum Frequency at which the processor could possibly run; the slope ***will*** become infinite.

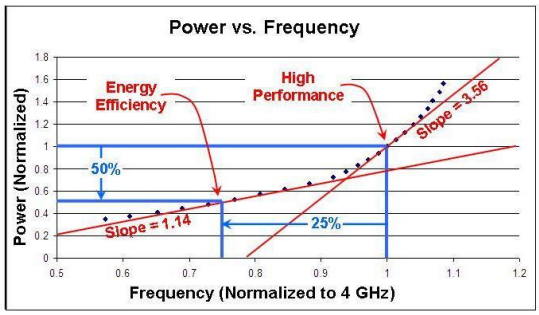

Figure 3. Speed vs. Power - Annotated

At the operating point that we've chosen (4 GHz), the slope is 3.56. Here, the frequency goes as (roughly) the 2.8th root of the power. This isn't efficient, but it gives us a fast processor. What happens if we cut the power in half? We wind up at a point on the curve where power is being used efficiently; here, the slope of the curve is only 1.14, which is almost linear. And note that when the power is cut in half, the speed goes down by 25% (in this case, 3 GHz).

This is what led us to consider 3D systems in which we stack processor chips together, and how by doing so, we could allow pairs of cores to run in some new synergistic modes: modes that that would give us more of a boost than simply "having two cores." Because the cores would be spatially coincident if we used identical chips, these new modes would be unique to 3D. If attempted in 2D, they wouldn't offer "improvements" with certainty.

Figure 4 shows a sketch of two cores in each of two configurations. One is the 2D configuration, with the cores side-by-side; the other, the 3D configuration, with one of the cores directly atop the other. One thing should be apparent just by looking at the figure.

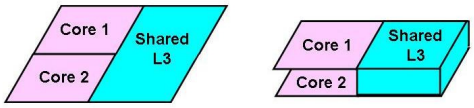

Figure 4. Areal Difference - Shared L3s in 2D & 3D

In the 2D configuration, if we conjoin the two L3 caches to make it one cache that's twice the size, its area doubles. Therefore, its access time will be larger. Since the access time has a significant portion that's proportional to the square root of its area, doubling its size in 2D increases that part of the access time by 40%. In the 3D configuration, the two L3s are spatially coincident. Conjoining them does not add this part of the delay, since the 2D area doesn't change.

In 3D, we've assumed that we need an additional cycle to cross back and forth between the layers, but since the area is unchanged, the basic access time is the same. In 3D, about half of the L3 accesses made by either core will take 2 additional cycles, but the basic access time is the same.

Each L3 has an access time of 24 cycles when the processor is running at 4 GHz. If we double the area of the L3 in 2D, the average access time is 33 cycles. In 3D, half of our L3 accesses are as fast as the original L3, and half take an 2 additional cycles (for two crossings). On average, the access time of the two L3s when conjoined is only 1 cycle more than when each is kept private – 25 cycles.

In two of our operating modes, we combine the pair of L3 caches (4 Mbytes) into a single L3 (8 Mbytes). In 3D, doing this is relatively seamless, and almost always yields higher performance. Doing this in 2D is more difficult, and it would hurt the performance a lot for some workloads.

In our environment, we are primarily interested in very high-speed single thread operation. We can run in this mode simply by turning one of the layers off, and running the other layer at full speed – as it was originally designed. In a second mode, we can turn both layers on, and run two threads at 75% of their original speeds. But note that at 75% of the speed, the number of cycles needed to access the L3 is only 75% of the cycles needed at full speed.

With both processors on, we can choose to keep the L3s independent, or we can choose to run them as a shared

cache, having twice the capacity. This will reduce the back-and-forth misses if they share data. We can also combine the two L3s into a 2X larger L3 when running only one of the processors. Since the second L3 takes some power, we can't run that processor quite at full speed.

For each processor, the L3 cache uses about 6% of the total power. So when a single processor uses both L3s in high-speed mode, to hold the power constant, we need to slow the clock down by 1 - 2.8^{th} root of 1.06, which is 2%.

To summarize, our configurations are shown Figure 5 at the core level:

1) One 4 GHz core with a 4 MByte L3;
2) One 3.92 GHz core with an 8 MByte L3;
3) Two 3 GHz cores each with a 4MByte L3; and
4) Two 3 GHz cores that share an 8 MByte L3.

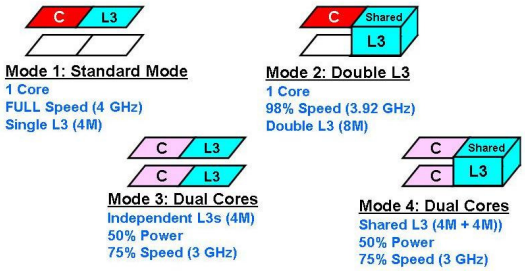

Figure 5. The Four Configurations Considered

3. Performance Comparisons of the 4 Modes

In this section, we evaluate the performance of each of the 4 structures with a simulator called REX (Research Experimental Timer). This timer is a cycle-by-cycle simulator of microarchitectures. The user can proscribe units, structures, interconnections, memory structures and geometries, and timings (in cycles) of the various elements. REX reads an instruction trace, and simulates that program running on the proscribed system configuration.

We've used 16 different benchmarks from 6 kinds of applications to evaluate the four structures described in Section 2. With the exception of SPEC, the ***names*** of the 5 other applications are factitious, but the workloads are real commercial workloads.

The timing parameters used are shown in Table 1. To reduce the speed of the processor, we reduce its voltage. This causes the L1 and L2 to scale with frequency too, so the number of cycles needed to access them stays constant.

The L3 and main memory use different power domains, so they do not have to have their speeds reduced when the processor frequency changes. When measuring those speeds in "Cycles," they'll appear "faster" in cycles at 3 GHz than they are at 4 GHz. And when we double the L3 size by conjoining the two of them, half of the references made to the L3 take 2 more cycles (1 per crossing). This adds an average of 1 cycle to an L3 accesses.

Table 1 shows the access times of all levels in each configuration. We had said that the L2 speed scales with the frequency, so it's constant. When the processor runs 25% slower, we can reduce the access cycles of the L3 and Memory in proportion: 24 → 18, and 200 → 150. Similarly, at 3.92 GHz, the memory access time is 4 cycles less. And as we've already explained, when we conjoin the L3s, we add a cycle to their access time.

# Cycles	4 GHz	3.92 GHz	3 GHz
L2	10	10	10
L3	24	24	18
2 L3s	25	25	19
Memory	200	196	150

Table 1. Access Times for the 4 Main Storage Structures at Different Frequencies

In Figure 6, we show the average number of cycles per instruction for each of the 16 workloads in each of the first two configurations. The first bar chart shows the actual BIPS for the base case (4 GHz with a 4 MByte L3). The second bar chart shows the normalized performance of the 3.92 GHz Processor with the 8 MByte L3, relative to the base case. The base case has a performance of "1."

Note that for SPEC, CJR, and AK6, there's very little difference – the processor that runs at 98% full speed runs about 2% slower. This means that having a larger L3 didn't help these workloads at all; the processor simply took a small performance hit in clock speed.

But on DLM, this system runs about 5% slower. That's because DLM has a relatively high rate of upgrade-misses. Not only does the processor run 2% slower to maintain the power, but because it has a larger L3 cache it retains data longer, which increases the upgrade-miss rate.

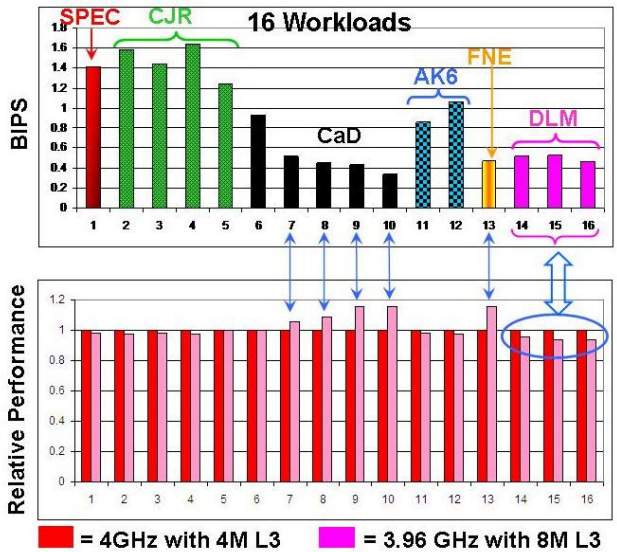

Figure 6. Raw, and Normalized Performance for Six Kinds of Workloads in Two Configurations

Figure 7 shows the log of the number of instructions between successful upgrade-misses (denoted "XI" to connote cross-Interrogation), on average, for five of the workloads (SPEC had no XIs). CJR has a very small XI rate - about 1 in 1000, CaD and AK6 have XI rates of 1 in 120-

125, so there is a minor effect. FNE as a rate of about 1 in 85, so it's similar. But DLM as an XI rate of 1 in 15. You can see in the previous graph that XIs have hurt DLM a lot.

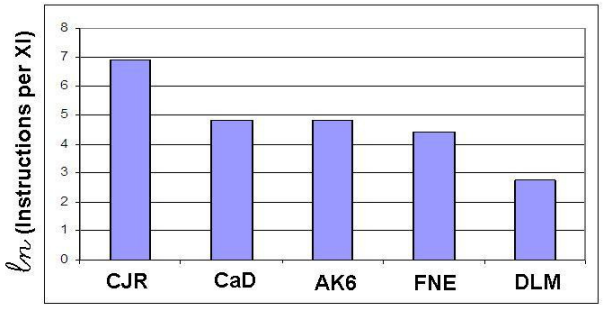

Figure 7. Average Distance Between XIs

By doubling the L3 cache, we've hurt CJR and AK6 a tiny bit, and DLM by only slightly more. But we've helped CaD and FNE a lot. An 8 MByte L3 improves both FNE and CaD by as much as 16%. These data show that the double-L3 might hurt some jobs just a little if they don't need more than 4 MBytes, but it will help others quite a lot.

Figure 8 shows the relative performance of the two dual-core configurations. When running a job on each side, there are additional misses because the two jobs reference some of the same data. This causes some of the data to be pulled from each L1 and L2 when it is referenced by the other core. This will result in additional misses if that data is re-referenced. But there are only additional L3 misses when the L3s are kept separate. If they are run as a single L3, there is no L3-to-L3 traffic.

The first cluster of bars in Figure 8 simply repeats the results from Figure 6, so that it's easy to compare with the second and third clusters. This is an 8-core processor running at full speed (4 GHz).

The second and third clusters of bars show the relative performance of the two coincident 8-core processors (that's 16 cores), where each core runs at 3 GHz. At 75% speed, each of the 3 GHz processors delivers at least 80% of the performance of a 4 GHz processor. So we'll get a minimum or 60% more throughput in this mode.

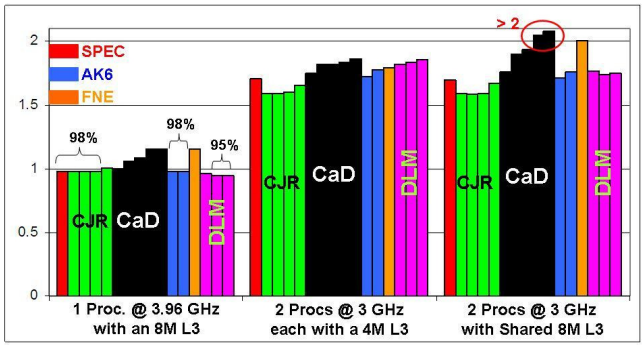

Figure 8. Relative Performances of the 3 Alternate Modes for all 16 Workloads

And a 3 GHz core with a private L3 delivers as much as 93% of the performance of a 4 GHz core having the same L3. While we clearly expected a lower bound of 75% performance (based on cycle-time alone), 93% is very surprising. This workload is CaD. The 3 GHz processor delivers surprisingly high (*relative*) performance because CaD has very high miss rates, so its performance is not tied that strongly to the speed of the processor.

The workloads that show the highest relative performance at 3 GHz are CaD and FNE. These are the same workloads that benefitted the most from doubling the L3 cache. This finding is consistent.

What's really surprising is that with a shared L3 cache, FNE on one 3 GHz core runs just as fast as the base case (4 GHz), and CaD runs as much as 5% faster! This means that a pair of cores at 3 GHz will deliver 210% of the throughput of a 4 GHz core on CaD. While we did expect to see throughputs that were 50% higher at 75% the speed, we did not expect to see throughputs that delivered more than twice the throughput of the base case.

While it might seem obvious that two processors running at 3 GHz is the preferred mode of operation for throughput, there are some applications that need raw speed on a single thread. With the exception of the CaD workloads, the dual core configurations do not deliver the maximum single-thread performance. But in all cases, they do deliver more ***power efficient*** performance, hence higher throughput at the same power; not surprising since this processor was built for speed, not for power efficiency.

4. Wiring Larger SMPs – 2D vs 3D

In Section 2, we showed how the cores on this chip are connected as a tree. What we'll show in this section is that the MP effects can be dramatically different in 2D vs. 3D because of the wiring density. Figure 9 shows two 8-core processors connected in 2D to make a 16-core processor.

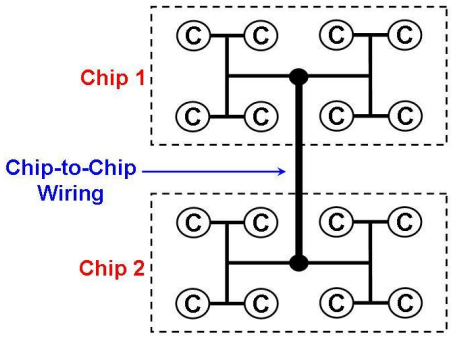

Figure 9. A 16-Core Processor made from Two 8-Core Processor Chips

In 2D, the pair of 3-deep interconnection trees have become a 4-deep interconnection tree, with the root interconnection likely being slower that the other branches. While many applications may be unaffected by this, it can cause severe queueing penalties if MP applications frequently share data among the 16 cores. In Figure 10 we show the same structure in 3D.

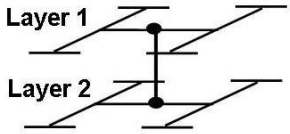

Figure 10. A 16-Core Processor made from Two 8-Core Chips in 3D - The Obvious Configuration

While a 3D 16-core system could be wired this way (exactly like the 2D version), and while these chip-to-chip wires would be even shorter, an even better structure can be made in 3D as shown in Figure 11. It's a much richer structure that's only possible because of the number of interconnections between the layers is much greater in 3D, and the connections are very short because they connect coincident points. Instead of connecting the two planar trees together at a single high-latency point, we can connect each coincident pair of cores together at coincident points in 3D.

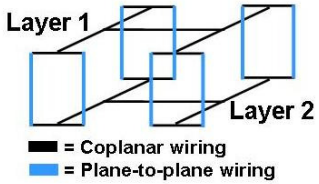

Figure 11. A Richer 3D System that Uses Many More Connections than are Viable in 2D

As a graph, this structure appears complicated, albeit very "connected." One interpretation is that it is still a 4-deep tree but a richly connected one if we keep the busses on each 2D layer (in black) independent. But a simpler way to run this structure is still as a 3-deep tree, but one in which each leaf comprises a ***pair*** of cores.

This is shown in Figure 12. On the left is a copy of Figure 11: a bipartite graph comprising 16 nodes, where the maximum distance between nodes is 4. But in 3D, since core-to-core connections are (at most) a cycle apart, we've combined each pair of coincident cores into a single node, which makes the connection graph look the same as that of the original 8-core processor.

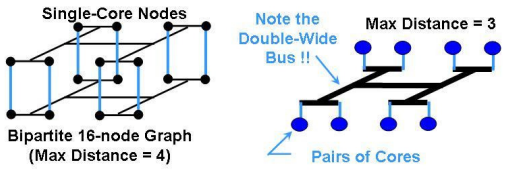

Figure 12. Two Different Interpretations of the 16-Core System in 3D

The only topological difference here is that each node must have (simple) multiplexing to distinguish the two cores. But a big advantage to this structure is that while only one layer controls the bus, the bus wiring can be used on ***both*** layers to make the 8-node processor bus doubly wide. This gives us far more bandwidth than we could get in the 2D configuration, but the topology is the same.

5. Thermal Issues

We've shown four ways to use the processor-on-processor structure, but we need to consider the thermal system that they comprise to understand the constraints. On first consideration, anyone would be wary of running two high-performance processors directly over & under each other in a stack. But on reflection, it's apparent that with each processor running at half power, the total areal power is the same. And that power, hence its dissipation, is now spread across **TWO** layers. Therefore, if the right things are done, the thermal issues should not be that problematic.

Figure 13 shows the thermal image of a single core running at full power on the left, and the thermal images of two spatially coincident cores running at half power, with one atop the other, on the right.

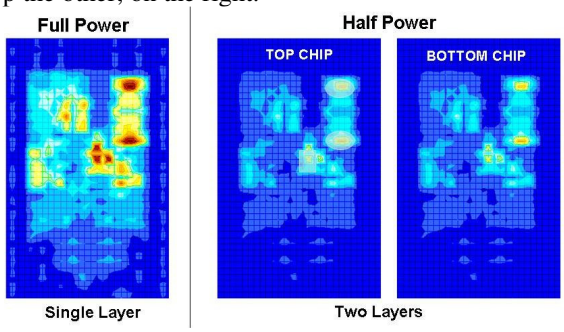

Figure 13. Thermal Maps of a Core Running Full Speed, and of Two Stacked Cores Running at Half Power

The main hot spots in the full-power core still show up in the half-power cores, but the temperatures are lower for both layers. We've found through simulation that the temperature difference between the two layers is about 8^0 C. While this is hardly worrisome, it does indicate that there will be a speed and/or leakage difference between the two processors. But both are cooler than the single layer running at full power. We'll see more on this later.

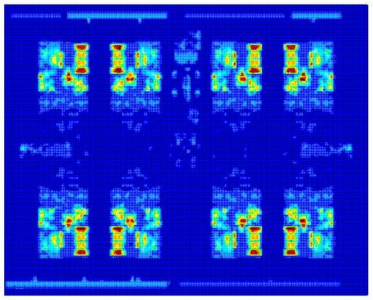

Figure 14. Thermal Map of the 8-Core Chip

Figure 14 shows a full chip that contains 8 cores. In addition to the cores, the chip has lots of infrastructure: busses, I/O, and content that is not (on average) as power-hungry as the cores. What's clear from the thermal map is that the power density isn't uniform. And each core has 3 major hot spots that are apparent.

The chip is 650mm^2, and its power is 250 Watts. The 24 hot spots comprise about 0.5% of the total area of the chip,

each spot being about 1/8 mm^2. Together, the hot spots take about 6% of the total chip power (15 Watts). Therefore, each hot spot takes about 625 mW. But since each spot is roughly 1/8 mm^2, the power density of each of the hot spots is about 5 W/mm^2.

Figure 15 shows the decrease in spot temperature as a function of the thickness of the chip for 5 W/mm^2 (our high-power density). We've shown this for four spot sizes: 200, 400, 600, and 800 μ^2. These curves are drawn relative to 0^0C. That is, if the temperature of those spots were 0^0C when the system's ***not*** running, then these would be the spot temperatures when the system ***is*** running.

To understand what this means in practice, let's assume that the temperature of these spots when the system isn't running is room temperature (25^0C). Now looking at the graph, suppose that we've decided that the 400 μ^2 spot (the second curve from the bottom) can run as hot as 90^0C, but no hotter. To see how thick the chip needs to be, we'd locate the point on the curve corresponding to 65^0C (that's 90^0-25^0). In this case, the curve corresponding to the 400μ^2 hot-spot shows that we'd need the chip to be at least 85μ thick. We've marked this on the graph in Figure 15.

In fact 90^0C isn't sufficient if the system is likely to remain in this state for long, because we haven't taken leakage current and its effect on temperature into account yet. But it the only chip that we are going to run at full power is the top chip, an 85μ "requirement" isn't a problem. But if we had wanted to run the bottom chip at full power too, it would need to be much thicker than we'd like – it would make the vias **much** larger than we'd want.

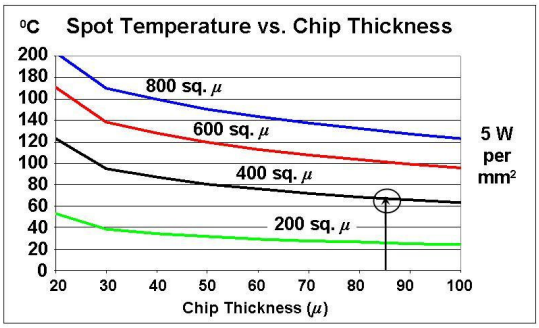

Figure 15. Temperature vs. Chip Thickness at Full Power for 4 Different Spot Sizes

Now we'll consider the same system running at half power. The hot spots are the same size, but their power densities are half as much: 2.5 W/mm^2. Figure 16 shows the thermal graph of the same hot spots running at half-power. And again, the second curve from the bottom corresponds to the 400 μ^2 spot. And again, let's assume that we'd like to operate at a maximum temperature of 90^0C (which corresponds to the 65^0C point on the curve).

In Figure 16, the 65^0C intercept on the 400 mm^2 curve is off the graph; it's slightly to the left of the y-axis. But remember (from our discussion of Figure 13) that the spots on the bottom chip run about 8^0C hotter than they do on the top chip when both are running at half power. So for the bottom chip, the temperature corresponding to 90^0C is actually 57^0C (that's $65^0 - 8^0$) on this graph. We've marked it on the graph with an asterisk. Figure 16 shows that we could make the bottom chip as thin as 25μ, but no thinner. These numbers depend **heavily** on how the chip is packaged; they're not a general result.

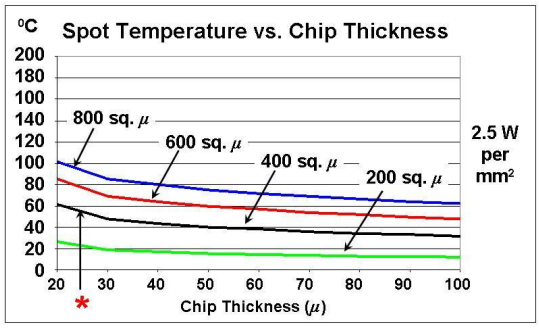

Figure 16. Temperature vs. Chip Thickness at Half Power for 4 Different Spot Sizes

To get a more quantitative view of thermal conduction as a function of power density and chip thickness, we did thermal simulations of power point-sources, for different thicknesses, and for different amounts of power. And for the case of 2 layers, we also varied the relative alignments of the two hot spots.

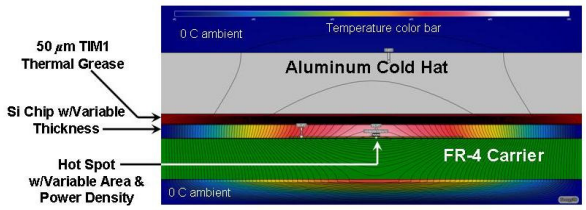

Figure 17. Assumptions Used in Previous Temperature vs. Thickness Graphs

In the 2-layer simulations, we did not include the 20μ solder balls, but as we'll see in the next section these won't make much of a difference. Keep in mind that the main thermal path in both cases is "upwards." Because the direction of heat flow is upwards, the thermal conductivity of the carrier doesn't make much of a difference either. The 2-layer case is shown below.

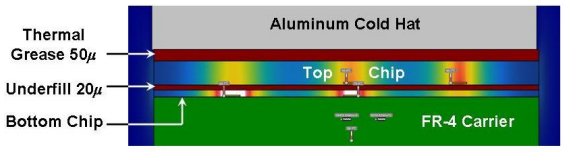

Figure 18. Two-Layer Thermal Assumptions

In the 1-layer simulations, we used the thermal model shown in Figure 17. The chip is face-down on an FR4 carrier, with a 50μ film of thermal grease between the back of the chip and the heat sink. The heat sink is aluminum,

and the other side of that sink is at the ambient temperature; in this case 0^0C.

For the 2-layer simulations, in addition to the 50μ film of thermal grease on the back of the top chip, we've set the distance between the chips at 20μ, and put a layer of underfill between the layers.

In the 2-layer simulations, the top chip was made "thick enough" to allow adequate heat diffusion in that layer; usually we used 200μ. The hot spots are assumed to be on the bottoms of both layers. The variables are the thickness of the bottom layer, the sizes and power densities of the hot spots, and the horizontal displacements of the hot spots on the layers relative to each other.

Because we did not include the C4s, or the TSVs in the bottom layer, these models are slightly skeptical. The real limits may be a *little* better than those shown in the figures. But as we'll see in the next section, TSVs and C4s will not change the thermal picture much, because their surface contact areas are small.

Our system is a "processor on processor" system. As first envisioned, the hot spots should be directly over-under each other to make the interconnections between cores easy (the connections are straight through). But we looked at how the thermal gradients would change if we could shift the chips a little to offset their hot spots. Figure 19 shows one example, with a pair of hot spots shifted by 1.8μ.

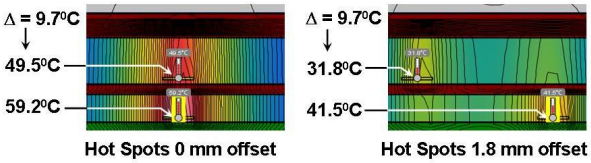

Figure 19: Two 2.5 W/mm^2 Hot Spots

This shows is that by shifting the hot spots, we can reduce their temperatures quite a bit, since the heat spreads laterally. This should not come as a surprise. But how far should we spread them out? And will this hurt the wiring?

Figure 20 shows the answer to the first question. The temperature of the top chip is the bottom curve (because it's closer to the heat sink), and the temperature of the bottom chip is the top curve. The chip on the bottom is hotter than the chip on top.

On the y-axis, we've shown the temperatures of a 400 μ^2 hot spot on the top chip running at full power, and running at half power. These are the two large points on the y-axis. For both of these points, the bottom chip is turned off. Note that the temperature difference between these two points (half power vs. full power) is a little more than 30^0C.

Interestingly, if we put coincident (x=0) half-power hot spots on both layers (the two curves), the temperatures of each of them are less than that of the top layer running at full power. Also, note that at any offset, the temperatures of the two hot spots differ by 10^0C. And as their displacement (x) increases, their temperatures fall dramatically, but their difference remains constant.

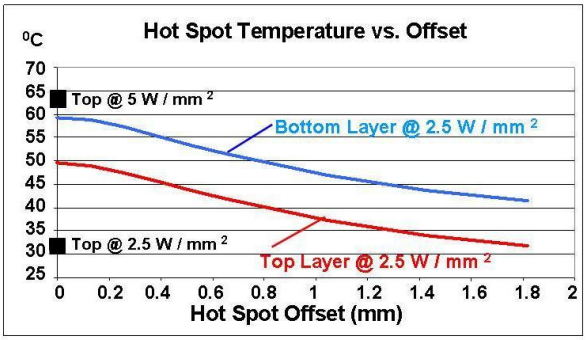

Figure 20. Hot Spot Temperatures vs. Offset

And note that at an offset of 1.8mm., at 2.5 W/mm^2, the temperature of the top hot spot is at the same as it was with the bottom chip turned off (the large dot on the y-axis). Once the hot spots are 1.8mm. apart, they're independent.

6. Properties of Thru-Silicon Vias (TSVs)

We've talked about the performance of 2 layers, and their heat. Now we'll describe the reality of how they are interconnected. To connect two layers together, they can be connected in a face-to-face (FTF) configuration, or in a face-to-back (FTB) configuration. Either way, there need to be silicon thru-vias (TSVs) in (at least) one layer. Of course for more than two layers, the FTB configuration is needed, whether there is a FTF structure in the stack or not.

The advantage of FTF is that the FTF interconnection pitch is not dependent on the TSVs, and it can be finer. For example, an FTF system can be built by soldering the two layers together using μC4 solder balls. And when soldering two silicon layers together, the μC4-pitch can be finer than that for the TSVs. But either way, solder is needed (today) to connect the signals on the two layers together.

The TSV pitch will be greater than the μC4-pitch if the layers are tens of microns thick (e.g., 50-100μ, as we've discussed). Given the length of a TSV (i.e., the thickness of the layer that it permeates), there is a lower limit on its diameter based on the aspect ratio of the hole to be filled.

	CTE ppm / ^{0}K	Thermal Conductivity W / m-K	Electrical Conductivity S / Im	Tensile Strength GPa
Silicon	2.6	130	<semi>	160
Copper	17	400	60	120
Tungsten	4.6	188	18	400
Aluminum	23	235	38	70
Eutectic Solder	25	50	6	40
Thermal Grease	-	<1	-	-

Table 2. Coefficients of the Relevant Materials

In the case of copper (Cu), filling the vias is a plating process, and the aspect ratio is limited to about 10:1. For tungsten (W), the process is a Chemical Vapor Deposition (CVD), and the aspect ratio can be larger (e.g., 30:1).

We'll now discuss several aspects of the systems that have to do with the coefficients of thermal expansion, the conductivities (both thermal and electrical), and the tensile

strengths of each material that we'd use in a chip/package structure like this one, including the thermal grease. Table 2 gives approximations of these four constants for silicon, for the three metals that we can use, and for eutectic solder.

For illustration, consider TSVs with aspect ratios of 5:1 and 10:1. Note that 10:1 is at the upper end of what's possible for copper, which is plated, but not for tungsten, which uses a CVD process. As shown in the table, copper is 3.3 times as electrically conductive as tungsten, so it would seem better to make TSVs out of copper.

To get a better idea of the resistances, Figure 21 shows the interconnection infrastructure as it exists today for thick layers (e.g., >50μ). A bottom layer, 100μ thick, is soldered to the substrate with 25μ C4s, and soldered to the top layer with 15μ C4s. The 4 structures shown include two metals, Cu and W, and two aspect ratios for the vias, 5:1 and 10:1.

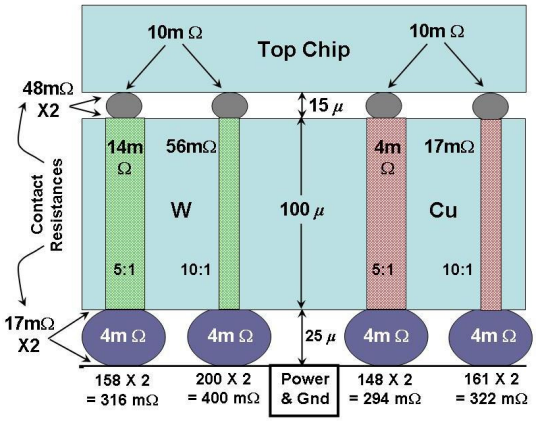

Figure 21. Resistances in a Thick 2-Layer Structure

The resistance of tungsten is 3.3X that of copper, and the resistance of a 10:1 via is about 4X that of a 5:1 via. The resistances of the C4s are also small (proportional to the inverse square of their radius). These are all relatively small components of resistance in the power-to-ground paths.

The largest components in the power-to-ground path are the contact resistances between the solder and the chips (again, the inverse square of the radii). The large C4s are 17 mΩ at each of their two contact points, and the small C4s are 48 mΩ at each contact point. The power-to-ground path has four of each. This amounts to 260 mΩ in the contacts alone. The C4s, by themselves add another 28 mΩ, so the resistance of this path is already 290 mΩ before we count the TSVs themselves.

The most resistive TSV is 10:1 tungsten. These add 110 mΩ to the path. The most conductive TSV is the 5:1 copper. These add 8 mΩ to the path. So while the thick copper TSV is 14X more conductive than the thin tungsten TSV, the full paths are 300 mΩ and 400 mΩ, respectively. Which is better? It's a *little* better to have a bigger TSV, and it's a *little* better to use copper. But the difference isn't much.

Before continuing our discussion of electrical resistance, let's return to the point we made Section 5 about "C4s not providing much thermal coupling between the layers." Why not? Table 2 shows that the thermal conductance of solder is 8X less than that of copper. But a much more important factor is that the coupling surface-area of the C4s is tiny. Let's take the surface area of a 15μ C4 to be 100 μ^2 where it bonds to each chip. If the C4s are on 100μ centers, there is ***one*** C4 for every 10,000 μ^2. Then the thermal conductance of the C4 balls is only 1% of the value shown in Table 2. That's 0.25 PPM / ^{0}K, which is nearly nothing.

Now, back to our discussion of electrical resistance. Even though the path difference between the extremely different TSVs is small (100 mΩ), since the TSVs are used to supply power, the lowest resistance will give us the least voltage drop, and the least ringing on surges. How much?

In Section 5, we found that the hot spots were about 1/8 mm^2 (about 350μ by 350μ) with a power density of 5 W/mm^2. The power for each hot spot is 5/8 Watts. If TSVs are on 100μ centers, and we use ***all*** of them for power and ground within hot spots, the situation shown in Figure 22.

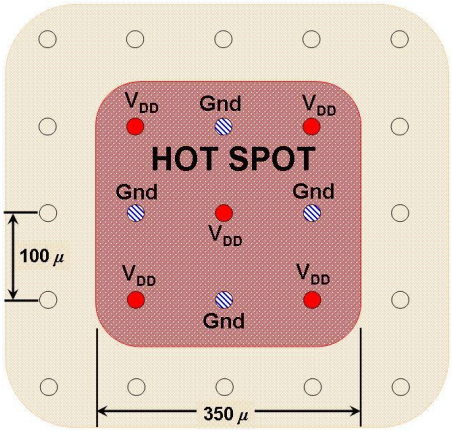

Figure 22. Our 350 μ X 350 μ Hot Spot

Since the hot spot uses 625 mW, and covers multiple vias, we need to know how much current a via could pass so that we can determine the importance of its resistance. In our case, V_{DD} is 1 Volt. An easy way to do this calculation is to simply look at the diamond formed by the ground vias, and to assume that the power via in the center provides all of the power within that diamond. The area of the diamond is 14,140 μ^2. The area of the hot spot is 122,500 μ^2. So the power via in the center must source 11.5% of the current to the 625 mW hot spot. This is 72 mA. The same is true of the ground vias.

If we use the highest resistance TSV path, which in this example is 400 mΩ, 72 mA will cause a loss of 29 mV. That's a 3% loss - which means a 3% increase in current. With the 300 mΩ vias, the loss is 21.6 mV. So, as before, the difference is not that significant.

A final comment on copper and tungsten: Table 2 shows that the CTE of silicon is 2.6, the CTE of tungsten is 4.6, and the CTE of copper is 17. Therefore, there is a slight mismatch between silicon and tungsten, and a ***huge***

mismatch between silicon and copper. So it would seem that tungsten would be a better choice of metals if we were expecting thermal stress. But silicon is about 30% **harder** than copper and 5X softer than tungsten. If the copper tries to expand slightly, it can't. It's too soft. But if the tungsten expands much at all, it will crack the silicon. Which is "best" depends on how hot the system will actually get.

7. Using 3D to get High Early Yield

In the proceeding sections, we've discussed the nominal operation of the system, and the realities of making that system. Unrelated to these issues, there are two other advantages of processor-on-processor systems. The first is early yield, which we'll discuss here, and the second is reliability, which we'll talk about in the next section.

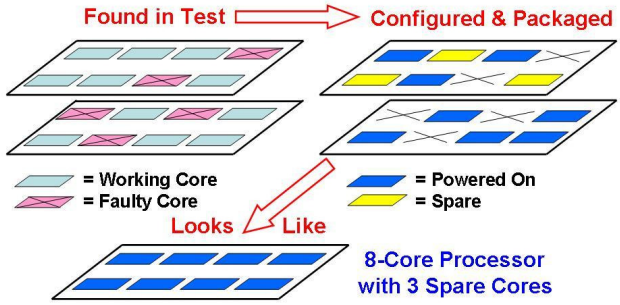

Figure 23. Configuring 2 Layers to Get Early Yield

Figure 23 shows the first concept (early yield) for an 8-core chip. When chips are first made with a new technology, their yields are too poor to be able to ship products right away. That means that systems can't be sold until the technology matures. In our Early Yield model, we put two of the chips together so that there are two cores in each x-y position, and then look at the probability that there is at least one working core in each location. If there is, we can turn that one on, and the other one off. From the outside, the pair of chips appears to be a single chip with a working core in every location.

Figure 24 shows the yield of an 8-core chip as a function of time (Quarters of a Year), in several configurations using 4 different values of the initial yield of a single core, Y. We've modeled the yield of a core as an exponential function of time that happens with process maturity. If Q represents Quarters (three months), the Yield is:

$$Y(t) = 1 - e^{-\beta Q}$$

The upper left plot takes the initial yield of a core to be Y = 0.5. The other plots show different values of Y. The middle curve in each graph is the yield of an 8-core processor chip by itself. For example, if the initial yield is Y(1) = 0.5 (the upper left plot in Figure 24), then the chip yield for 8 cores is $0.5^8 = 0.0039$. The chip could not be shipped with this yield. But from the point at Q=1, we get:

$$\beta = -\ln(1 - 0.5^8) = 0.0039$$

This allows us to construct the rest of the curve. In this case, you can see that the chip yield (the middle curve) doesn't get to 50% until the 4th Quarter (1 year), and it doesn't get to 90% until the end of the 6th Quarter.

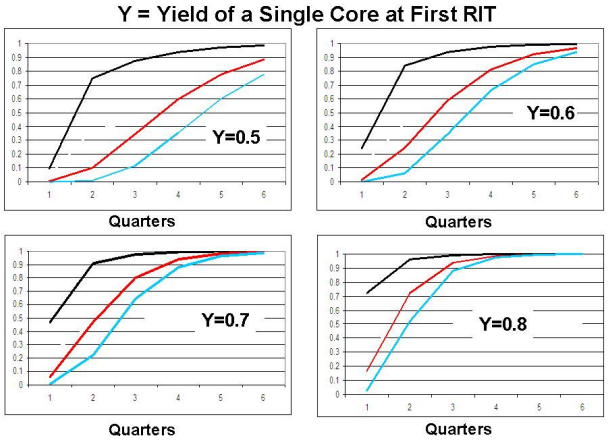

Figure 24: Yield Curves of an 8-Core Processor

But if we put any two chips together, then the probability that there's a working core in each of the 8 positions is plotted as the top curve in each graph. The curves show that even with Y=0.5, the yield of a working 8-core processor is nearly 80% at the beginning of the 2nd Quarter. Note that the yield of a single chip processor is only 10% at this time. This method would allow systems to be shipped a few Quarters earlier than standard yield would allow.

The bottom curve in each graph is the yield of two-chip systems in which all 16-cores work. Even with an initial core yield of 50%, the yield of a 16-core system becomes 50% in the 4th Quarter. These curves show that we could start shipping 8-core processors at the beginning of the 2nd Quarter, we could start shipping 16-core processor "chips" about half a year later. How's that for Moore's Law?

The remaining graphs show the same system with initial core yields of Y = 0.6, 0.7, and 0.8. If we got initial core yields of 75-80%, an 8-core chip would hit acceptable yields by the second quarter, and at 80% yields, there would be very good yields on 16-core systems by the 3rd Quarter. The problem is that we can't know the yield before the first chips are actually produced.

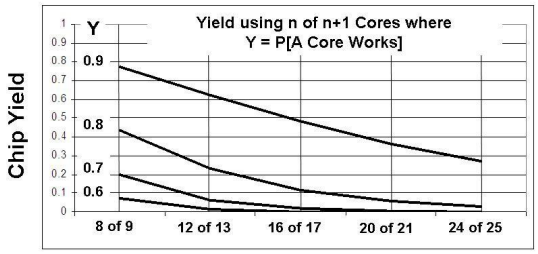

Figure 25. Yields of Several N-out-of-(N+1) Cores

You might ask whether a chip-on-chip configuration is a "necessary" yield play. After all, an alternative that's probably cheaper is to put N+1 cores on a chip (if they fit), and to start shipping products when N of them work. In

Figure 25, we show the yield of N-out-of-(N+1) core chips as a function of N for various initial yield points.

What's clear from this plot is that yield drops off **very** quickly as N grows. While the 8-core chips have decent yield, the 16-core yields are terrible except at Y = 0.9. And according to our yield model, this will happen over time; but it does not give us an acceptable ***initial*** yield.

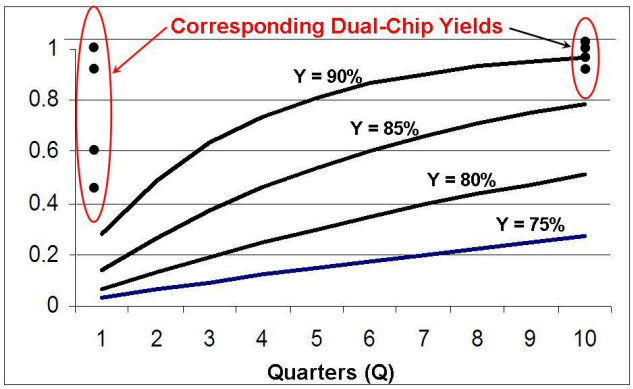

Figure 26. Chip Yield by Quarter for a 12-Core Chip, Based on Initial Single-Core Yields

The N-out-of-(N+1) method and the 2-layer method that we've described are not *competing* yield enhancers; they're **complementary**. We can use both for even better results. We have shown that the yield is very good for 8-out-of-9 core processor chips. We have also shown that in the 2-layer system, an 8-core processor will start yielding fairly soon, and it will become a 16-core processor sometime later. Both schemes suffer much more than linearly as N grows larger.

Figure 26 shows the yield for a 12-core processor system using two layers. There are four curves to depict the chip yield for initial core yields of Y = 0.75, 0.8, 0.85, and 0.9. In addition to having chosen values of Y that are much higher than what we used before, you should notice that the yield growth for a 12-core processor is ***much*** slower than it was for the 8-core processor. In this figure, we've not drawn the full curves for the two-layer systems; we've only shown the beginning and ending points.

The solid curves in the figure are the standard yield curves. The graph shows that if the initial core yield is 75%, we can't even yield 25% of the 12-core processors until the 10th Quarter. But while the initial single-chip yields are only about 2% (the lowest curve), nearly half of the chip pairs would be fully working 12-core processors immediately (see the lowest dot at 1st Quarter). In other words, with a full chip of only 2%, we could ship nearly half of the chip pairs as fully working 12-core processors in the 1st Quarter.

Therefore, in addition to the performance modes that we discussed, a processor-on-processor system can be used to yield products much earlier than we do today. And once the basic core yields are high enough, we can get pairs of chips with all cores working with pretty good combined yields.

8. Using 3D for Super Reliable Computing

After yielding the full system, i.e., once all of the cores on both layers are working, in addition to performance, the core-on-core configuration can be used to do super-reliable computing, with 100% checking of all circuits. This is done using an R-Unit, which was a mechanism designed for IBM's first zSeries CMOS mainframes.

What the R-Unit does is: 1) it holds all architected state for a core (mostly register values) in a small set of ECC-protected buffers; and 2) it compares the results of two cores, each of which has its own copy of that state, that are running the same program at the same time. At the completion of each instruction (which happens on the same cycle on both cores), the R-Unit performs a comparison.

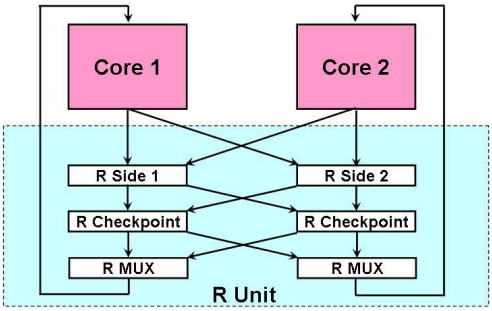

Figure 27: The R-Unit for 100% Checking

If the results are the same, then the R-Unit generates ECC for those results, and it stores the new value of the state that was changed in its own private copy of the state. If the results are not the same, then the R-Unit stops both cores, and it restores all of the architected state in both cores using its hardened copy of the state. Then the R-Unit restarts the two cores at the point in the program at which the last instruction successfully completed.

Figure 27 shows the basic structure of the R-Unit. While the R-Unit stands on its own in 2D, we can think of the R-Unit as if each core has its own half of the R-Unit. Both cores send their results to both halves of the R-Unit. Both halves generate ECC, and both halves compare both results. If both halves agree that both results are the same, then the new results are checkpointed.

The R-Unit is quite small, and can provide 100% checking. While it cannot determine what caused an error, it will catch any error and transparently restore the states of both cores simultaneously.

In 2D, the outputs of the two cores can be physically far apart, so the checking and checkpointing processes are delayed by several cycles. In addition, the wiring for all of the cross-compares and ECC generation takes lots of area. In 3D, the outputs of the two cores are spatially coincident, so all of the comparisons can be done immediately on completion of each instruction - and certainly within a cycle. And in 3D, because of the spatial coincidence, comparisons are not as complicated to wire.

9. Interconnection Density

This section is here to discuss the final piece of reality: interconnection density. To gauge the possible granularity of logic macros in layer-to-layer interconnections, you'll need to know how many signals you can put in a given area of the chip to see what's feasible, and whether the wiring of those connections will cause timing problems. In high-powered systems, about half of the connections are needed for power & ground, so the number of signals that can be run is only half of the total.

For example, if connections can be put on a 100μ pitch, then there are only 100 per mm^2, only 50 of which can be used for signals. But at a 10μ interconnection pitch, 10,000 fit – good for about 5,000 signals in the same mm^2.

The thickness of the thin layer might also limit the number of vias that can be placed. The aspect ratio needed for a TSV might force you to make much larger vias than you'd like. For example, if the thin layer is 100μ, and you can only metalize holes with an aspect ratio of 5, then the diameter of each via has to be 100/5 = 20μ. With a keep-out circumference around it, the area used by the via must be large, e.g., 25μ X 25μ. For these numbers, you could only put 400 X 400 vias per mm^2 if you used 100% of the area. Instead, if the vias were on 100μ centers, using 1/16 of the chip area, then there could only be 10X10 of them per mm^2, and they'd take 1/16 of the chip's area.

The via-pitch can be devastating if the number of signals that you need to connect is large. For example, let's assume that a single macro is 2.5mm. by 2.5mm. If the via pitch is 100μ, then there are 625 vias within the macro, about 300 of which are for signals. Figure 28 shows the vias in this area. Signals can use half of them.

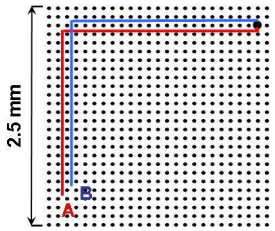

Figure 28. Thru-Via Spacing Can Cause Problems

If we need to connect 280 signals between layers with 300 signal vias, some could have unexpectedly long routes. In Figure 28, we show a connection between points A and B, which are on different layers. While A and B are very close in their x,y coordinates, the wiring between them is nearly a centimeter long if they have to use a via that's far away (the routing is shown in the figure).

In Section 6 (see Figure 19), we had wondered whether moving one of the layers over by a millimeter would cause wiring problems. The answer seems to be that at the current interconnection pitches, there are ***much worse*** wiring problems. So skewing the layers by another millimeter or two doesn't add to these problems in any significant way.

3D technology integration has improved significantly in the past ten years. The main focus of recent work has been reducing the size and pitch of the TSVs. The literature has examples of TSVs diameters <1μ. From the perspective of manufacturing, a big limitation of these techniques is the requirement for wafer-to-wafer bonding, which compounds the yield limitations, since not all die will be good.

Additionally, little attention has been paid to the die-to-die interconnect (micro-bumps). As a result, even when fine-pitch TSVs are available, the micro-bump pitch can be the limiting factor. Several previous works have examined requirements for die-to-die interconnection pitch given various stacking granularities. For example: processor-on-memory, core-on-cache, core-on-core, or various types of core folding. In general, our analysis has produced results in line with these previously-published estimates.

3D Integration Type	Pitch Needed
Core-on-Memory	200-400 um
Core-on-Cache	30-50 um
Core-on-Core	40-70 um
Functional Unit on F.U.	10-20 um
Functional Unit Folding	2-10 um

Table 3. Interconnection Pitch Needed for Various Levels of Integration

We produced our estimates by examining the number of connections between sets of functional blocks in a real processor, including core-to-core connections, core-to-cache connections, connections between functional blocks, and connections within functional blocks. Table 3 is a summary of those estimates for the required interconnection pitch to enable each degree of 3D integration.

But from the previous section, we cautioned against extremely thin layers when using high-powered circuits, because the lateral thermal conduction becomes nonexistent. To talk about a 2μ via pitch (with a 1μ via) is not realistic if the via has to go through a 50μ layer.

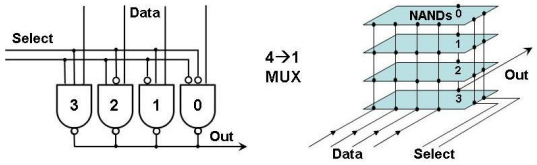

Figure 29. Thin-Layer Stacking of Certain Logic

On the other hand, the dataflow of a computer contains mostly multiplexors, decoders, and various gating to move data around, select it, shift it, gate it, etc. Lots of this logic is "1-hot" logic, meaning that one gate of many gets turned on, and the others stay off. While the logic for these parts is very simple, the wiring is not. For example, take the multiplexor in Figure 29. The "Select" inputs choose one of 4 gates; the input to that gate passes through to the output.

Since only one of the four gates can actually become live, we could stack these gates up in four layers as shown, without violating point-power-density limits. While the logic area that's saved is not significant, stacking the gates

up makes the wiring trivial. The savings is not merely areal; that's not why we'd do this. We'd do this because it greatly simplifies the planar wiring. This could possibly reduce the number of metal layers needed to make a processor.

And finally, as stated in Table 3, we do see a new opportunity if the via (and/or μC4) pitch becomes small enough to accommodate wiring between functional units on different layers (e.g., 10-20μ). This opportunity would be a practical extension to our processor-on-processor study, and is shown in Figure 30.

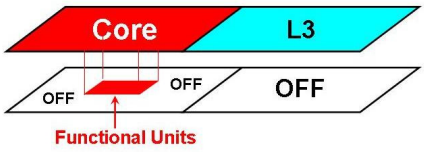

Figure 30. Augmenting the Microarchitecture of a Core by Using Parts of Another Core

Given the ability to do finer wiring between layers, new modes of operation could be made from the core-on-core structure by using functional parts of the inactive core to augment the microarchitecture of the active core. This enables us to dynamically instantiate two or more microarchitectures (e.g., a heterogeneous system of cores) from the same set of parts. An obvious example is to double the number of floating-point units to make a core that has better performance on superscalar applications. This would not be reasonable to do in 2D.

10. Conclusion

We have shown that there are several completely new advantages that can be had by putting a pair of processors directly over/under each other in 3D. Today, this pair of processors can be run in four different performance modes, offering considerably more throughput than a single fast processor. Several of these modes aren't practical in 2D.

And with the processor-on-processor configuration, we showed that in addition to performance, there is a high-reliability mode that allows a pair of cores to check 100% of the processing done. It also offers a straightforward way to ship products early when the actual chip yields are very poor. This allows a considerable acceleration of product cycles – likely worth Moore than a technology generation.

We've said a lot about the layer-to-layer interconnection density, and showed that if it's not sufficiently fine, layer-to-layer signals can become very long. It's also important to note that if there is a need for lateral heat conduction, the layers cannot be made too thin, so the TSVs cannot be arbitrarily small.

While 3D offers some completely new modes of operation for a processor, the current technology doesn't yet allow enough wires between layers for cores to share their logic. But, putting one processor over another processor proves enormously useful in the high performance arena.

11. References

[1] Y. Xie, G. H. Loh, B. Black, and K. Bernstein, "Design space exploration for 3D architectures," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 2, no. 2, pp. 65–103, Apr. 2006.

[2] G. H. Loh, Y. Xie, and B. Black, "Processor Design in 3D Die Stacking Technologies," *IEEE Micro*, vol. 27, no. 3, pp. 31–48, May 2007.

[3] Y. Xie, "Processor Architecture Design Using 3D Integration Technology," *23rd International Conference on VLSI Design*, no. 3, pp. 446–451, Jan. 2010.

[4] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. Loh, D. McCaule, P. Morrow, D. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb, "Die Stacking (3D) Microarchitecture," in *39th AnnualIEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, 2006, pp. 469–479.

[5] H. Homayoun, V. Kontorinis, A. Shayan, T.-W. Lin, and D. M. Tullsen, "Dynamically heterogeneous cores through 3D resource pooling," in *IEEE International Symposium on High-Performance Comp Architecture*, 2012, pp. 1–12.

[6] T. Zhang, A. Cevrero, G. Beanato, P. Athanasopoulos, A. K. Coskun, and Y. Leblebici, "3D-MMC: A Modular 3D Multi-Core Architecture with Efficient Resource Pooling," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2013, pp. 1241–1246.

[7] M. Wordeman, J. Silberman, G. Maier, and M. Scheuermann, "A 3D system prototype of an eDRAM cache stacked over processor-like logic using through-silicon vias," in *IEEE International Solid-State Circuits Conference*, 2012, pp. 186–187.

[8] B. Zhao, Y. Du, Y. Zhang, and J. Yang, "Variation-Tolerant Non-Uniform 3D Cache Management in Die Stacked Multicore Processor," *42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 222–231.

[9] A. K. Mishra, X. Dong, G. Sun, Y. Xie, N. Vijaykrishnan, and C. R. Das, "Architecting on-chip interconnects for stacked 3D STT-RAM caches in CMPs," *38th Annual International Symposium on Computer Architecture*, 2011, pp. 69–80.

[10] G. Loh and M. D. Hill, "Supporting Very Large DRAM Caches with Compound-Access Scheduling and MissMap," *IEEE Micro*, vol. 32, no. 3, pp. 70–78, May 2012.