Cores for concern

When, in the mid 1960s, founder Gordon Moore noticed how quickly the transistors on silicon wafers were shrinking, he concentrated purely on how much space circuits would take up over time. A few corrections ensued as the frenetic pace of development of the early 1970s settled down to the long term trend: a doubling in functional density every two years.

After the first Pentiums appeared in the early 1990s, Intel decided Moore’s Law needed something extra; Moore’s Law was then not only about density, but also about performance. And it got faster.

At the time, another observation about scaling was playing its part. At the beginning of the silicon revolution, IBM Fellow Robert Dennard discovered that shorter gates would switch more quickly — one of the factors in what became known as Dennard scaling. Processor makers reaped the rewards in the 1990s as clock speeds accelerated into the gigahertz domain. Architectural enhancements — such as longer pipelines, superscalar execution and fast growing cache memories — helped put the rapidly growing number of transistors to use in getting up the performance curve. And then the music stopped. Moore’s Law is now back to its original density based definition and clock speeds, even on advanced microprocessors, are stuck at around 3GHz.

More speed, more power consumed
As clock speed pushed up, so did average power consumption. In servers, by the early years of the last decade, processors nudged up against the 150W limit that could be sustained by packaging and air cooling. Pure Dennard scaling came to a halt at the 130nm node and designers had to find another way to improve processor performance.

Engineers could deploy ever more complex processor architectures in the hope of exposing more instruction level parallelism to the core. But it’s a law of diminishing returns, encapsulated by Pollack’s Rule: the performance increase of a processor is roughly proportional to the square root of the increase in its complexity. Doubling the number of transistors will, therefore, only increase speed by 40% at best.

As most reasonably complex systems have to run more than one task at a time and many of these can be subdivided into parallel threads, why not multiply the number of processors? In principle — as long as you can feed data to the processors — a near linear increase in performance is possible. And you can use simpler, slower processors to get better performance for a given level of power.

You can apply the inverse of Pollack’s Rule — cut performance by 40% and you can double the number of cores. Anant Agarwal, cofounder and cto at Tilera and a professor at the Massachusetts Institute of Technology, came up with a variation; what he calls the Kill Rule. This states ‘A resource in a core must be increased in area only if the core’s performance is at least proportional to the core’s area increase’.
Analysis using the Kill Rule suggests the best trade off in terms of complexity and area is a dual issue superscalar processor. It is, perhaps, no surprise to find that high end embedded processors, such as ARM’s Cortex-A9, fit that mould.

Right now, most systems only deal with a few processors on a chip. But Moore’s Law scaling suggests an exponential increase in their number as process geometries head below 32nm. The issue will then be to deliver them with progressively lower power consumption.

When Intel demonstrated its 80 core Teraflops Research Chip, the device had good headline performance, sustaining hundreds of gigaflops per second – but only if you want to process the same 3kb of data over and over again. Each core had access to its own local memory, but that was all – the devices had no high speed access to a large slice of memory that might contain useful data. This is because the power problem is shifting away from the processors themselves and into the data movement infrastructure.

Several years ago, Intel’s director of microprocessor technology Shekhar Borkar did the sums and found simply getting enough data on and off a notional 1000 core chip – each of which would demand hundreds of gigabytes per second in total – would cost around 25W in the I/O drivers alone. His proposal was to take the memory to the processor: slide a large dram cache underneath the ‘manycore’ [see fig 1]. Much smaller I/O drivers could be used, reducing the power consumed by interconnect inductance.

Unfortunately, the memory chip could not sit on top of the processor because it would have to transfer almost all of the heat generated through to the heat sink. DRAM does not perform well when hot and the die would provide much higher thermal resistance than the heatsink on its own. The processor would therefore have to run more slowly.

The answer is to have the memory sit underneath the processor and drill vias through the dram to support connections from the system to the processor. Although a future dram could provide gigabytes of storage, this memory would probably act as a fourth or fifth level of cache within the system, with more sitting on DIMMs in the main system – but these would be accessed less frequently over the high inductance paths.

Stacking is an option
The stacked option is one that IBM is investigating for its server processors. The company already uses embedded dram to increase the amount of on chip cache near processors on its existing designs. On a high end system, the presence of this embedded dram saves around 1.5kW, according to IBM fellow Subramanian Iyer. There is some loss of memory density – around 5%, caused by the need to push vias through the die – but more dram closer to the processors should save a lot of power per chip.

The next problem is the on chip interconnect. Although individual cores can easily be powered down to prevent leakage current from dominating overall power consumption, it is hard to shut down parts of an on chip bus. And the number of cores that need to be connected make any sort of bus impractical.
The shift will be to on chip networks – meshes or rings. But networks that allow data to be routed to any on chip processor quickly will be expensive and power hungry. Some savings can be made by moving to a different clocking scheme. The Teraflops chip, and some other recent SoC architectures, used a mesochronous scheme, originally proposed by David Messerschmitt of the Berkeley Wireless Research Centre.

The idea behind mesochronous clocking is that, at more than 1GHz, delivering a consistent clock to all parts of a chip becomes increasingly difficult because of the delay across 1cm of die. Mesochronous clocking avoids having to distribute the same clock signal across the chip. A single clock need only be delivered to tiny islands the size of a few individual processors. You then use asynchronous techniques to join up the islands in a scheme that is also known as globally asynchronous, locally synchronous (GALS).

As cores will be powering up and down dynamically, this scheme has the benefit of allowing them to decouple from one master clock, although there is the increased verification overhead of dealing with asynchronous schemes.

There is still the issue of how much power it takes to move data arbitrarily around the chip. Reducing the degrees of freedom will hurt interprocessor communication performance, but provide big energy savings. So, working on the principle of locality – that most data will move between closely coupled threads running on neighbouring processors – designers may choose to subdivide the manycore chip into clusters inside which are high speed routers, which talk to the outside world using higher latency connections. This is an approach used in the Elm architecture developed at Stanford University (see fig 2).

Alternatively, processors might be restricted to nearest neighbour connections, in which data gradually ripples from source to destination: this is reminiscent of the wormhole routing employed by the Inmos Transputer architecture, revived by Intel for the Teraflops chip.

Not every processor has to be the same, however. Even supercomputers are moving towards heterogeneous architectures in which conventional general purpose processors are supplemented by more specialised engines. Embedded hardware has long depended on a mixture of general purpose processing and dedicated hardware. To provide greater flexibility — as standards change, for example — this hardware is becoming more programmable and turning, as Robert Brodersen of the University of California at Berkeley predicted in the late 1990s, into a ‘sea of processors’.

An architecture proposed by Stanford University Professor Bill Dally, chief scientist at nVidia, combines a small number of conventional processors with an array of ‘stream processors’.

Prof Dally devised the idea of the stream processor while at MIT and the concepts have been embraced most
enthusiastically by makers of graphics processing units, such as his current employer. In contrast to conventional processors, which Prof Dally calls latency optimised architectures – they are designed to turn around the calculations for individual pieces of data quickly – stream processors are concerned primarily with throughput. The answer to a calculation could come hundreds of cycles after it is first initiated – this massive number of cycles hides the time it takes to get data to and from the registers and works well if you have enough calculations running in parallel. This is generally the case in 3D shading, where each pixel can be calculated independently.

If you run hundreds or thousands of threads, each with its own stream of pixels, the processor can timeslice between them to ensure that execution units and data fetch engines are being used efficiently. These shader engines were originally quite primitive, but have mutated into floating point units capable of running a much wider range of code. There is still a need for latency optimised code, but this can be run on a smaller number of conventional processors.

A link to dedicated hardware might be provided by additional slices of programmable logic: an architectural approach favoured by Satnam Singh of Microsoft Labs in Cambridge. FPGAs are poorly suited to floating point intensive code because they cannot be implemented efficiently on the reprogrammable fabric – but that is where gpu-like stream processors can shine. With short word lengths and logic intensive code, FPGAs can come into their own because they allow thousands of custom processors to be built in parallel, used and then scrapped when no longer needed. The problem with embedded FPGA blocks is their high area demand compared to their performance. But on some problems, such as search, they are many times more efficient than software programmed processors.

At this month’s International Solid State Circuits Conference (ISSCC), Sung Bae Park, vice president of processor R&D at Samsung Advanced Institute of Technology, will describe an architecture that combines a number of architectural elements proposed for manycore processors. He says it will be capable of 1TLOPS performance while consuming just 100mW – once technology is ready to deliver it in 2020.

The objective, according to Park, will be to reduce switching activity through massive clock gating and microthreading; similar to stream processing. But these processors will widen to become ultra long instruction word (ULW) machines, running many instructions in parallel in one cycle. To provide them with data, Park proposes a ‘super mux’ – the building block for a 128 x 128bit crossbar network that can relay bits between processors quickly under compiler control to ensure that wide instructions are stuffed with useful data.

It will take several process generations to get to Park’s version of a teraflops manycore processor. A growing problem will be one of reliability. Increasing variability in processing will make it more probable that some parts of the chip will be out of tolerance. It will be too expensive to scrap the parts. So why not use the fact that many cores will be replicated across the die to provide a level of redundancy? Redundancy could be at individual processor level, or it could be finer grained. Researchers at Oregon State University have proposed a stream processor called Sycntium that allows spare execution units to be used as replacements for failures by using what they call ‘pipeline weaving’ (see fig 3) – each unit is connected to two downstream units, one of which is selected as usable after a self test.

**Processors in memory?**

If the processor can be broken into pieces, it may be possible to embed executions into memory. A technique proposed by Dave Patterson at the University of California at Berkeley and used to a limited extent by the Tabula FPGA architecture. As transferring data is more energy intensive than processing, it may make more sense to replicate processing elements throughout the memory and then use marshalling elements to bring the results back – the data stays more or less where it is and the processing moves around as different parts of the data is needed.

Naturally, the programming model for this would look very different even to that needed for stream processing. But in a world where the primary constraint will be energy, it may be the ultimate architectural switch.