The Incredible Shrinking CPU

This article discusses three problems facing modern high performance MPU architects: decreasing returns from ILP, increasing power dissipation and relatively constant interconnect performance. Each issue is explained and analyzed in detail, as well as the impact of all three on future MPU designs. Finally, an analytical model for MPU performance that incorporates each of these three factors is derived and compared to current and past MPUs and then used to make predictions for future microarchitectures.

Bigger, Better, Wider

The first 20 years of the modern 32 bit era of microprocessor design for computer applications could be described as a quest for bigger, better, wider (BBW) CPUs. This trend is driven by the desire for better performance, and fueled by an exponential growth in transistor budgets. The first MPUs of this era, like the MC68020 and the i386, were partially pipelined scalar designs that took multiple clock cycles to execute the simplest of instructions and averaged 3 to 8 cycles per instruction (CPI). These chips packed several hundred thousand transistors on a die around 100mm^2 in size.

The first generation 32 bit MPUs gave way to fully pipelined scalar designs like the i486, MC68040, and the first generation RISC chips. These processors could execute at burst rates of one instruction per clock cycle and averaged around 1.3 to 2.0 CPI. The CISC chips, equipped with 8 KB of integrated cache used around 1m transistors on a die of about 150mm^2 while the RISC MPUs used external caches which kept their transistor count and die size to under 100k transistors and around 60mm2 respectively.

With multiple millions of transistors within their reach, MPU architects designed the first superscalar chips in the early 1990s. The Pentium and Alpha EV4 could execute up to two instructions per cycle, while the PowerPC 601 and SuperSPARC could execute up to three instructions per cycle. Die sizes ranged from 112mm^2 for the 601 to 294mm^2 for the Pentium. The transistor count was 1.7m for the Alpha and about 3m for the rest. Although these designs all incorporated on-chip cache, from 16KB to 36KB, CPU control logic and datapaths still accounted for the majority of the chip area.

As transistor budgets passed the 10 million mark, architects introduced second generation superscalar designs that could execute up to 3 (x86) or 4 (RISC) instructions per cycle. Although the first integrated L2 cache was introduced in the Alpha EV5, the CPU itself still occupied most of the die area in this generation. In many cases, the extra transistors were consumed to implement out-of-order execution (OOOE) and speculative execution. Indeed, the PA-8000 occupied a 347mm^2 die, without so much as a single bit of integrated cache. But the previously clear blue skies that seemed to stretch endlessly ahead of ambitious MPU architects were already starting to darken. The BBW philosophy of processor design faced three different but interrelated barriers that opposed continued progress down this road with geometrically increasing strength.

Quest for ILP

The first sign that the party was over was diminishing returns from wider and wider superscalar designs. As CPUs went from being capable of executing 1, to 2, to 4, to even 6 instructions per cycle, the percentage of cycles during which they actually hit their full potential was dropping rapidly as both a function of increasing width and increasing clock rate. Execution efficiency (actual instruction execution rate divided by peak execution rate) dropped with increasing superscalar issue width because the amount of instruction level parallelism (ILP) in most programs is limited. Although expensive hardware based techniques like OOOE and speculative execution help somewhat, they are far from being panaceas as often thought [1]. These techniques also require sophisticated compilers to perform code generation and scheduling specific to individual processors to achieve full benefit. The ILP barrier is a major reason that high end x86 MPUs went from fully pipelined scalar designs to 2-way superscalar in three years and then to 3-way superscalar in another 3 years, but have been stuck at 3-way issue superscalar for the last nine years. The Pentium 4 design is arguably a deliberate move to narrower effective issue width in a quest for higher performance through faster clock rate.

It should be pointed out that everything else being equal, a wider issue processor will still generally provide higher performance even if it the increase is unevenly distributed over the space of all applications that users run on a given architecture. If the only problem of going to wider issue superscalar designs facing MPU architects was the diminishing returns from ILP, it is likely that the BBW bandwagon would still be going strong. This is especially true with the advent of thread level parallelism (TLP) based techniques, such as SMT, that soak up instruction issue and execution slots that would go unused in a wide processor executing a program with little available ILP to extract. After all, if the effect of Moore's Law provides geometrically increasing numbers of transistors to throw at problems, surely concerns about wastefulness and efficiency are misplaced? Unfortunately transistors are real physical artifacts that consume power and communicate with each other over wires with non-zero delay and finite bandwidth. This leads us to the more serious, and in the end fatal, two barriers facing the BBW approach.

Peril of Proliferating Power

It is almost paradoxical that while the energy consumed by a single CMOS logic gate switching state has fallen exponentially with shrinking process feature size, the overall power consumption of MPUs has been steadily trending upwards at roughly 22% growth per year. This trend, shown in Figure 1 over the last 23 years, is due to the fact that the combined effect of growth in CPU complexity and operating frequency has exceeded the rate at which the energy used by individual logic elements to change state has fallen.

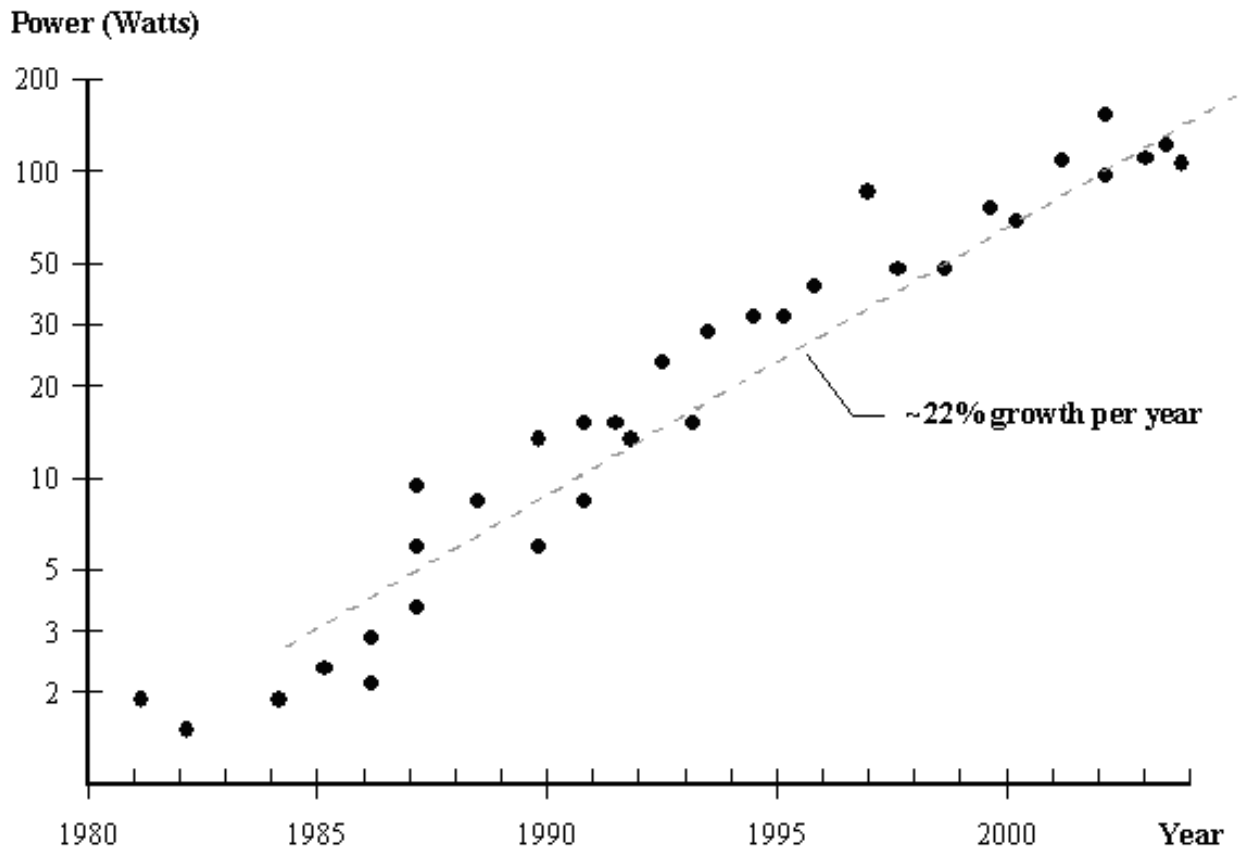

Figure 1 – Microprocessor Power Consumption, 1981 to Present

This trend has taken an ugly turn for the worse with the power dissipated by leakage current rapidly growing as process feature size falls from 180nm to 130nm and below. For example, leakage represents 5% of overall power of the 180 nm McKinley, or about 5 Watts. In the 130 nm Madison leakage grew to about 21% of power or about 22 Watts [2]. Thus in a single process generation leakage power per unit area and per transistor went up by about 5x and 2.4x respectively.

In the distant past MPU power increased with growing design complexity and operating frequency but was offset by power supply voltage reductions that more or less tracked shrinking feature size. In the recent past, MPU power continued to increase with growing design complexity and operating frequency, but supply voltage scaling slowed to avoid performance scaling problems related to transistor threshold voltages. Since power dissipation can no longer be regulated by supply voltage scaling, MPU architects have turned to dynamic power management techniques. By adding tiny bits of extra logic that turn off unnecessary gate switching activity in unused portions of the MPU, designers have been able to constrain power consumption [3][4].

Today, architects are faced not only with the familiar upward pressures on power from growth in CPU complexity and operating frequency but also a rapid growth in leakage current power. At the same time power supply voltage scaling is continuing to slow while most of the low hanging fruit in the realm of dynamic power management has already been picked. The final and fatal blow to business as usual is that the practical engineering and economic limits of MPU cooling capabilities at

the system level have already being reached in both high end servers and desktop PCs. That means that MPU architects are facing a situation where power considerations, rather than critical path timing, can limit maximum clock rate, and power budgeting becomes a zero-sum game of trade-offs between CPU complexity and CPU operating frequency. This latter consideration alone spells an end to the past practice of chasing diminishing ILP returns with wanton abandon. One might argue that if the thermal management issue was the only factor standing in the way, perhaps Seymour Cray-style heroic cooling efforts could allow the BBW bandwagon to rumble on for the design of MPUs for large and expensive servers and supercomputers. Unfortunately the third and final barrier facing BBW is as fundamental and immutable as the speed of light. This should hardly be surprising considering both limitations flow from the same set of equations.

Caught on the Wire

The most difficult problem facing MPU architects in the future is the ever widening mismatch between how falling process feature size increases transistor operating speed while doing little for the performance of interconnect in the context of constant or slowly growing die size and increasing CPU complexity. A typical process shrink reduces feature size in all three dimensions by ~30% (i.e. dimensions are multiplied by 0.7). As a first order approximation, transistor scaling causes circuit delay time to also fall by 30%. The propagation delay of most wires on a chip is dominated by its so-called RC delay, the product of its total distributed resistance and capacitance. The capacitance per unit length of a wire changes very little while its resistance per unit length approximately doubles for a 0.7 factor shrink (the wire's cross sectional area drops by 0.7^2).

The effect of process shrinks on wire scaling depends on the type of wire. A local wire is a relatively short wire, internal to a functional block that is shrunk with its circuit and layout topology intact. A wire that tends to run "3 transistors over and 7 transistors up" will drop in length by the same factor as the shrink. Because RC delay is quadratic with respect to length, the shortening of the wire cancels out the increase in resistance per unit length so the RC delay is approximately constant. Although constant wire delay in the context of a 30% reduction in transistor delay is a relative increase, in general local wires are so short that even after a shrink it is usually much less that transistor delay. It is global wires, those that run an appreciable and relatively fixed fraction of die size, that really cause problems. Even if the complexity growth in MPUs was such that die size is kept constant in a shrink the RC delay of global wires approximately doubles in a shrink while transistor delay falls by about 30%. If nothing was done to address the problem MPU designers would face the prospect of dealing with global wires whose propagation delay increased relative to transistor speed by a factor of 2.9 for each process shrink!

Fortunately, there are several methods process engineers and chip designers use to partially counter the wire/transistor scaling imbalance. There are one time fixes that change chip composition to reduce interconnect resistance, R, (copper instead of aluminum) or the capacitance, C, (lower K

materials separating wires) values. However, these have a one time effect that is rather modest given the long term trends.

A common design based approach to combat poor wire performance scaling is the insertion of repeaters. If you take a long RC delay dominated wire of length L and break it into two segments of L/2 by inserting a repeater (inverter or buffer) to amplify the signal, the RC delay of each L/2 segment is 1/4 that of the original wire. The total RC delay reduction obtained is greater than the extra transistor delay introduced by adding the inverter or buffer for even moderately long wires. Breaking wires into shorter segments using repeater insertion also has the benefit of countering inductive effects found in long wires. An example of this practice is found in the Madison Itanium 2. Although this chip is relatively large in size, 374mm^2, all signal wires are broken into segments under 2mm in length using repeater insertion [5]. Unfortunately, repeaters consume power and the numbers of repeaters needed to counter interconnect scaling imbalance rises geometrically with shrinking feature size. Even in the 180nm McKinley and POWER4 processors, buffering global signals consumes 3% of the power and requires about 100,000 repeaters respectively [6][7].

One powerful technique that is used to improve interconnect scaling is sizing by layers. This is done by arranging interconnect so that the thinnest and narrowest layers of metallization are placed at the lowest level (closest to the chip substrate and transistors). The metal layers are gradually scaled up in thickness, width, and separation as the layers are stacked up one at a time during fabrication. The lowest layers have the highest RC delay factor and are only used for local interconnect. The topmost layers are the thickest and coarsest; hence they have the least resistance and RC delay factors. These upper layers are used for power and clock distribution, as well as global signal routing. The real potential of this scheme is realized when adding layers of interconnect in a process shrink. The new finer sized and spaced wires can conceptually be added at the lowest level with a 0.7 scaling factor, while the topmost level(s) keeps the same approximate size and spacing as in the previous generation process. Therefore, the RC delay of the topmost layer(s) of interconnect remain approximately the same after the shrink. This process is illustrated in Figure 2.

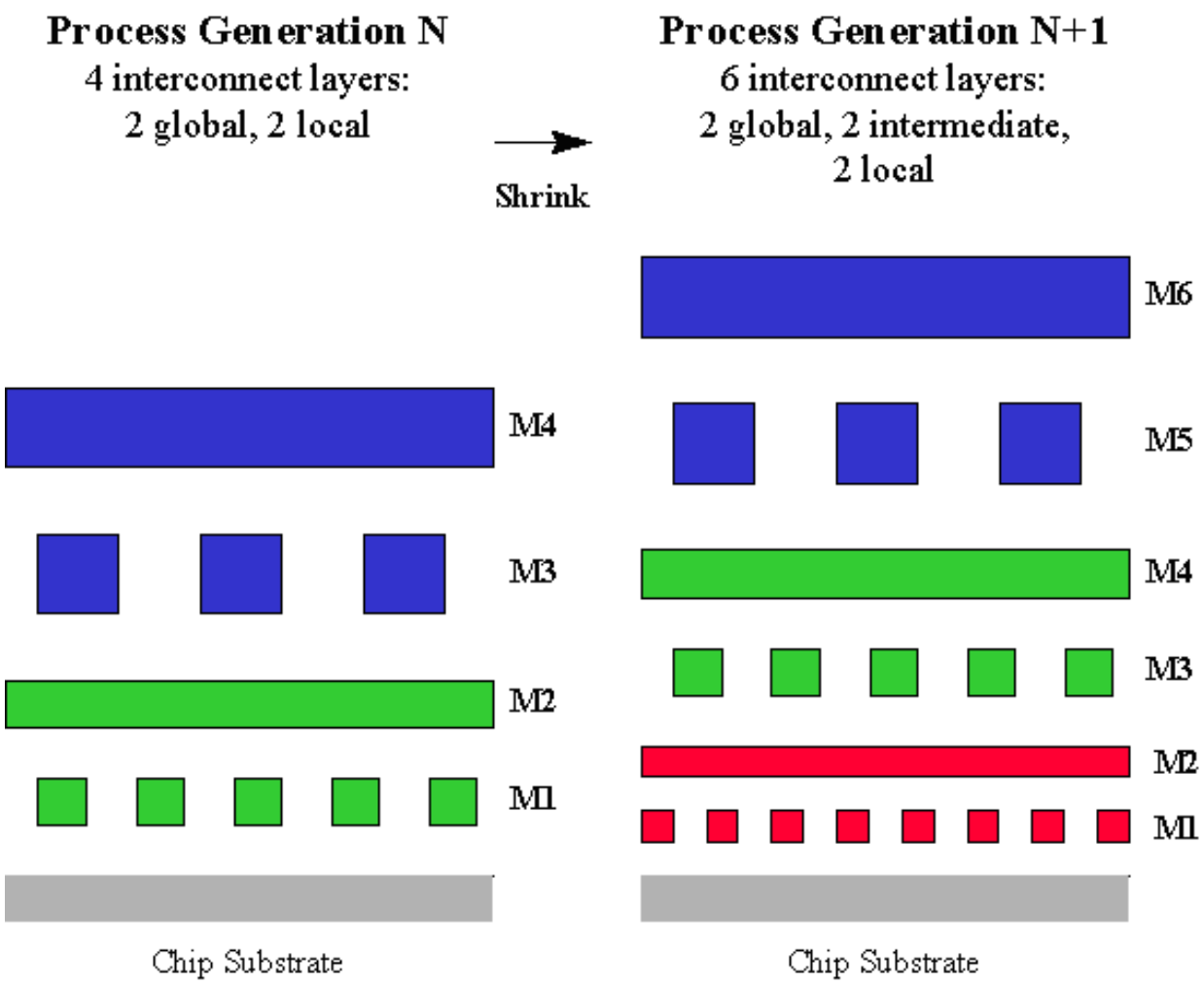

Figure 2 – Interconnect Scaling in an Ideal Shrink with Extra Metal Layers

The interconnect layers shown use alternating orthogonal routing (M1 wires run perpendicular to the image while M2 wires run across the image and so on). This technique preserves the global interconnect performance despite process shrinks. Ideally, this technique will slow down the growing difference between global signal propagation delay and transistor switching speed from an approximately cubic relationship with the linear scaling factor to approximately linear with scaling. That is, wires are still getting slower with respect to transistors but not nearly as fast (1.4x slower instead of 2.9x slower per shrink).

Unfortunately even this amelioration cannot be kept up indefinitely. Adding extra layers of metal to a process increases manufacturing costs and reduces yields. In addition, the number of high quality wires per unit area stays constant in this scheme while the number of transistors per unit area doubles with each shrink. This provides MPU architects with an unpalatable choice between severely limiting the growth of global signal flow within their processors or increasingly using the slower intermediate interconnect layers. The impact on wire performance from using intermediate or local interconnect layers is shown in Figure 3. The red, green, and blue colored lines indicate the performance of local, intermediate, and global signal layers in an ~90nm process with a conventional

oxide dielectric. The thick black line indicates the point at which a signal propagation in a wire changes from being RC delay limited to being limited by the speed of electromagnetic wave propagation along a transmission line [8] (sometimes incorrectly called the “speed of light” limit; the propagation speed is really about half the speed of light in air or a vacuum due to the relative permittivity of the oxide dielectric separating the conductors in this example).

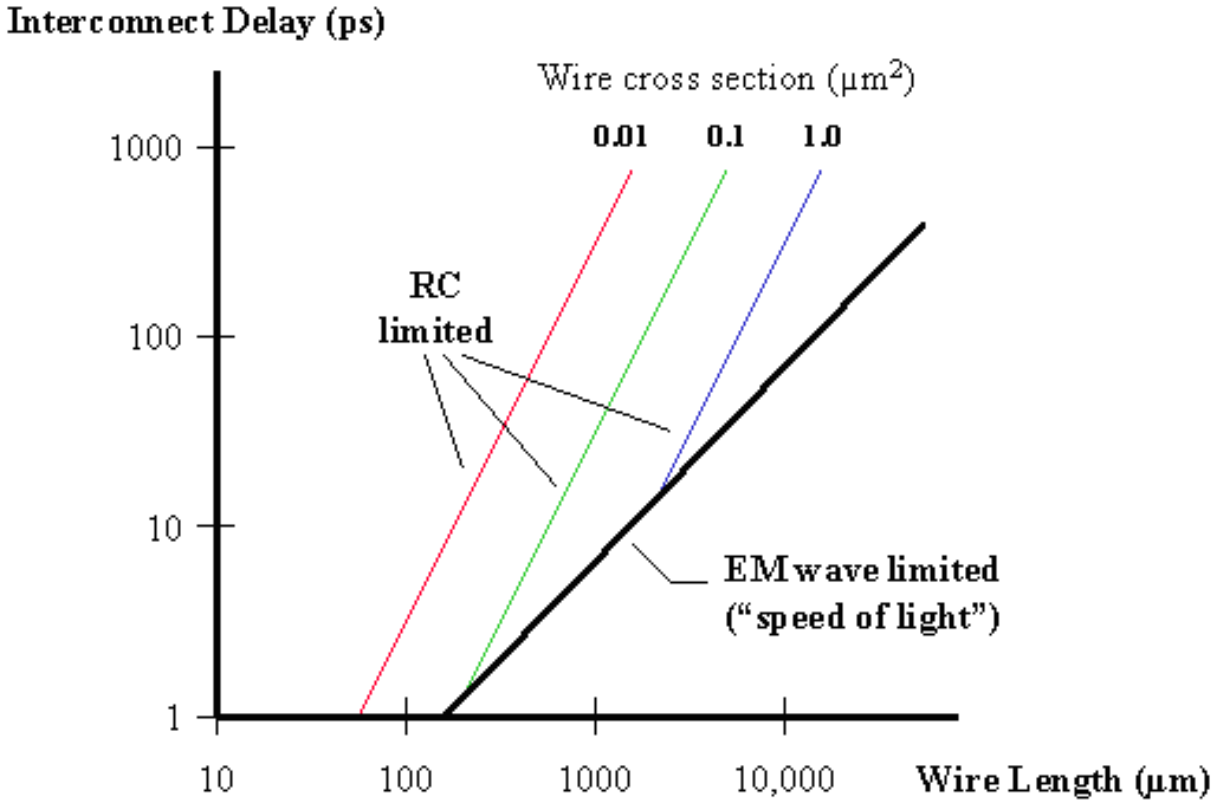

Figure 3 – Wire Performance versus Size

In reality, the interconnect performance issue is more complex than presented here. Real processes don’t shrink all features, either minimum sizes or minimum separations, on all layers by the same ideal 0.7 factor. More importantly, there are numerous tricks that physical chip designers can use to optimize interconnect performance, especially for bused signals. For example, simply increasing the physical separation between a critical signal wire and its closest neighbors can significantly improve performance by reducing parasitic capacitance per unit length. However, this technique is of little practical use on a large scale because it tends to push up MPU size which in turn increases average wire length. The general trend is as clear as it is unavoidable. The performance of interconnects will fall increasingly behind that of transistors as process feature size shrinks and this factor will more strongly influence the design of each successive generation of MPUs.

Goldilocks and the Three CPUs

I am going to put forth a performance optimality hypothesis for modern monolithic VLSI based MPU implementation without attempting rigorous proof but rather using an accessible, and hopefully convincing argument in its support. Our hypothesis is that for any given ISA and semiconductor process, there is an optimal physical size/complexity of CPU that will give a maximum execution speed or single thread performance for general purpose workloads. That is to say that CPU performance given by the equation:

```
Perf = (IPC * Freq)/Path
```

attains a maximum value for a design of finite size and complexity. In this equation Perf represents the performance in the form of the reciprocal of the time to complete a given computational task, IPC represents average instructions executed per clock cycle, Freq is the maximum CPU clock rate, and Path represents program path length in instructions to execute the given computational task to completion. For the sake of simplicity I will consider the program path length constant. In reality, one can add new instructions or addressing modes etc. to an ISA to reduce path length but that has similar effect on CPU complexity as an increasing IPC, not to mention it violates the original premise that an optimal CPU size exists for any given ISA.

I will not attempt to describe this optimal CPU in any way; I will simply posit that it exists and that any attempt to design a higher performance CPU by brainiac means (increasing issue width, OOOE capability, branch prediction accuracy etc), or speedracer means (simpler design, deeper pipelining etc) is doomed to failure. Keep in mind that I am considering only the CPU size and complexity, not that of the entire device. For the sake of analysis I will consider first level (L1) caches as an integral part of the CPU as per widely used convention. L2+ caches are usually considered external to the CPU since they do not generally influence processor clock rate and can be altered independently of the CPU. In contrast, altering the L1 cache would likely require modifications to the CPU.

Let’s start with the concept of a processor’s critical loops. A critical loop is an elementary processor function that must complete in a limited number of clock cycles and cannot be further pipelined due to cost or performance constraints [9]. Some examples of critical loops include integer ALU result forwarding, data cache access, next PC generation, register renaming in OOOE processors etc. A basic processor consists of mechanisms to fetch a stream of instructions from an instruction cache, decode them, schedule them for execution, execute them (possibly accessing data cache or determining branch prediction accuracy in the process), and retire them while preserving necessary ISA requirements for serial program execution behavior and exactness of exception handling. Because of the existence of critical processor loops, it is necessary for some internal signals to travel a certain significant portion of the total CPU physical size within a single clock period. Some special purpose computational devices such as systolic arrays and graphics processors have far less demanding critical loops but these are not designed to handle general purpose applications which are often intensely control and data dependent in nature.

Since top level CPU organization and topology change little with CPU complexity, we will assume that global signal wire lengths for critical loops will increase proportionately to the square root of CPU complexity measured in area or transistor count:

$$\text{Critical Signal Length} = \text{Comp}^{1/2}$$

RC wire delay increases quadratically with length, hence global signal wire delay increases linearly with CPU complexity (area or transistors). The maximum operating frequency of a processor is defined by the reciprocal of the critical path timing plus extra time related to flip-flop setup and clock skew and jitter. Skew is timing variation in the clock distribution system that occurs due to geographical location (i.e. point A is 5 mm away from point B). Jitter is period to period variation in the timing of the clock edges, and can sometimes be modeled as a random variable. Let's define maximum clock frequency as:

$$\text{Freq} = 1/(T_{clk} + T_{tran} + T_{wire}) = 1/(T_{clk} + T_{tran} + K_1 * \text{Comp})$$

where T_{clk} is the sum of flip-flop setup time and clock skew and jitter, T_{tran} is the transistor portion of critical path delay, and T_{wire} is the global wire portion of critical path delay which can expressed as the product of a constant K_1 and CPU complexity Comp. Observation of the history of microprocessor development into the superscalar age suggests that IPC improves at best as the square root of CPU complexity [10]. So we can write IPC and Performance, Perf, as:

$$\text{IPC} = K_2 * \text{Comp}^{1/2}$$

$$\text{Perf} = (\text{IPC} * \text{Freq})/\text{Path} = (K_2/\text{Path}) * \text{Comp}^{1/2} * (T_{clk} + T_{tran} + K_1\text{Comp})$$

With this formulation maximum performance is achieved for a design whose complexity is such that the global wire delay component of critical path timing, K_1 Comp, is equal to the sum of the latch overhead and transistor portion of critical path timing, $T_{clk} + T_{tran}$. The performance of CPU designs less complex than this is sub-optimal because IPC falls faster than maximum frequency increases. For more complex designs performance is sub-optimal because IPC increases more slowly than the maximum frequency falls. This concept is shown in Figure 4.

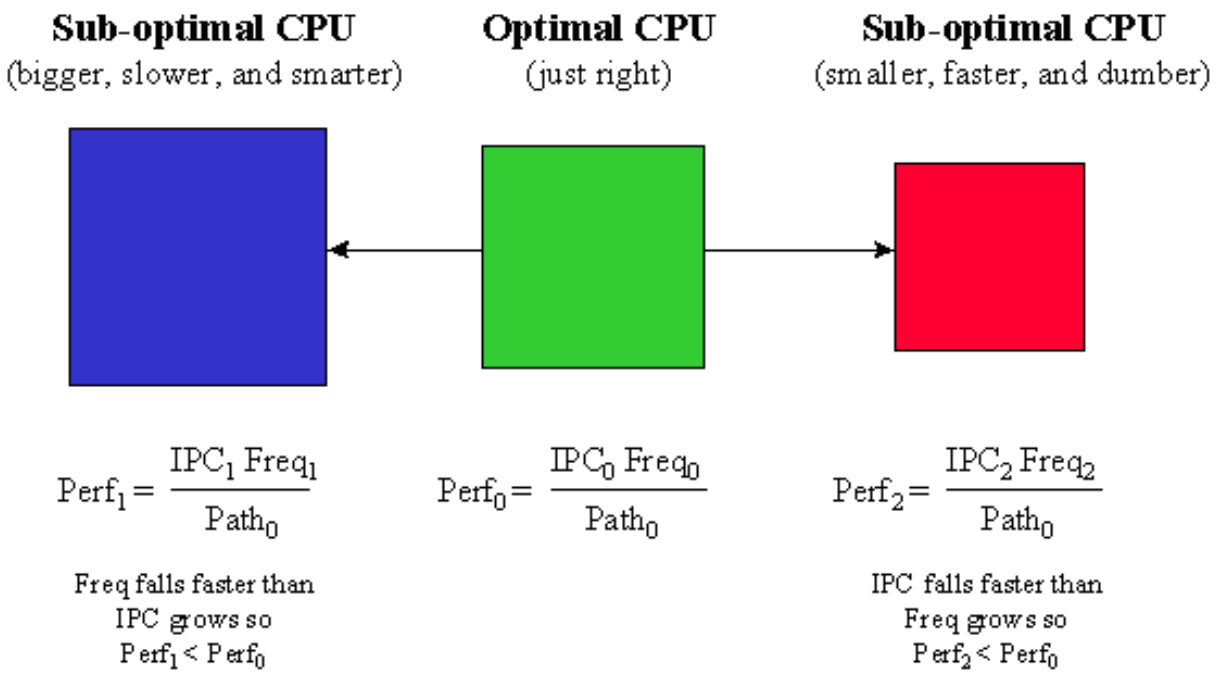

Figure 4 – Optimal Performance CPU Size/Complexity

Although useful to demonstrate the principle of optimal CPU size, the previous derived formulation was greatly simplified from reality. For example, the transistor based portion of critical path timing would also tend to increase as with CPU complexity (although much slower than linearly, perhaps logarithmically). And IPC is not a continuous function of complexity but is rather discontinuous. For example you can build a 2 way or 3 way superscalar processor, but you cannot build a 2.7 way superscalar processor. Nevertheless many aspects of CPU design have much finer granularity than issue width yet also influence performance. For example the number of TLB entries, length of instruction queues, the capacity of branch target buffers and return address shadow stacks and so on.

So where is the state of the art in MPU design today with respect to the hypothetical optimal performance CPU? Indications are that current CPU designs are still much simpler, smaller, and slower than optimal performance CPU designs for their respective ISA and manufacturing processes. Real world commercial MPU designs face constraints like economic and manufacturing brakes on die size growth, practical restraints on power budgets, EDA tool limitations, and time to market concerns that keep them from even approaching the maximum single thread performance potential at the present time. However the BBW trend not only brings current designs closer to the point of optimal performance, but process scaling trends are quickly bringing the optimal performance CPU complexity limit closer to current designs as I will show in the next section.

Smaller, Simpler, Faster

Having formulated a simple model to demonstrate the concept of the optimal size or complexity of CPU to maximize single thread performance, it is interesting to ask the question of what happens when the design is shrunk. As previously mentioned optimal performance is achieved when CPU size and complexity is such that the global wire delay portion of the critical path is equal to the sum of the latch overhead and transistor delay portion of the critical path:

$$K_1 Comp = T_{CLK} + T_{TRAN}$$

Consider the effect of a process shrink by a linear factor of S (S = 0.7 for a typical single generation process shrink) while CPU complexity is kept constant. The latch overhead and transistor portion of critical path delay will both fall by approximately a factor of S while global wire delay constant of proportionality more or less stays the same. This implies that the complexity of the CPU needs to be reduced by a factor of S to maintain the equality relationship shown above that defines the optimal performance processor. The implications of this effect are shown in Table 1.

	Arbitrary	Typical Shrink
Feature Size Shrink Factor	S	0.7
Complexity (transistors)	S	0.7
Area	S^3	0.34
Linear dimension	$S^{3/2}$	0.59
Clock Frequency	S^{-1}	1.43
IPC (complexity$^{1/2}$)	$S^{1/2}$	0.84
Performance (IPC * Clock Frequency)	$S^{-1/2}$	1.20
Dynamic Power (voltage scaled by S)	S^2	0.49
Dyn. Power Density (voltage scaled by S)	S^{-1}	1.43

Table 1 – Effect of Process Shrink on Optimal Performance CPU Design

The optimal performance CPU must shed about 30% of its transistor count for a typical 30% linear process shrink to reduce its size enough to make up for higher RC delay constant of the interconnect. But because transistor density doubles as a result of the process shrink the overall effect translates into an optimal performance CPU design of barely more than 1/3 of the die area of that in the previous generation process. Hence was derived the title of this article!

Notice that as a result of the shrink, the optimal performance CPU's clock frequency rises faster than its IPC falls. Thus there is still a net increase in single thread performance benefit from process

shrinks even in a heavily interconnect dominated MPU design regime. Dynamic power scales approximately as the product of CPU complexity, clock frequency and the square of supply voltage. Thus the optimal performance CPU design complexity reduction cancels out clock frequency increase so power goes as the square of the supply voltage. Supply voltage is ideally scaled proportionally with process linear dimension (i.e. constant field scaling) so dynamic power will fall by about half for a typical process shrink. The power density of the optimal performance CPU design increases as the inverse of the process scaling factor or by about 1.43 for a typical process shrink. Unfortunately, the change in the leakage component of CPU power consumption due to a process shrink has too many complex and technology specific factors for general analysis. One should keep in mind that leakage is not one single effect but rather the sum of more than a half dozen different leakage mechanisms even in a basic bulk CMOS process[11]. The various components respond differently to changes in supply voltage and physical scaling of transistors and associate source and drain active areas and are very sensitive to changes in materials and manufacturing processes. Also leakage is strongly dependent on circuit implementation details which tend to change over time. About the most that can be said is that reduction in CPU complexity, die area, and supply voltage tends to reduce leakage current and power by varying degrees while the shrinking of device and interconnect critical dimensions tends to increase leakage strongly.

The Narrowing Road Ahead

What would an optimal performance MPU look like in a 130nm process? Indications are that Intel used various design methods such as repeater insertion and dedicated pipeline stages for global signal propagation to keep global wire delay to a little under 20% of critical path timing for the 3.4GHz top speed grade of Northwood P4. This x86 CPU core uses about 28m transistors and occupies about 100mm^2. To achieve the optimal performance CPU criterion of global wire delay equaling latch overhead plus transistor delay portion of critical path timing would require a CPU design about 5x larger and more complex than the Northwood. It would clock at 2GHz, and consume about 180 Watts of dynamic power. With IPC approximately 2.2 times higher than Northwood its performance would be about 1.3 times greater with the same 512KB L2 cache. Obviously such a design proposal pushes die area and device power far beyond any reasonable high volume desktop MPU design. Nevertheless applying the process scaling regime for optimal performance CPU design derived in the previous section to this hypothetical optimal performance 130nm x86 CPU and a remarkable trend is revealed as is shown in Table 2. Keep in mind that integrated L2+ cache size will scale rapidly with shrinking feature size which would accelerate performance faster than shown.

	130nm	90nm	65nm	45nm
Complexity (transistors)	140m	98m	69m	48m

CPU Size (mm^2)	500	170	58	20
Clock Frequency (GHz)	2.0	2.9	4.1	5.8
IPC (relative to 3.4 GHz P4)	2.2	1.9	1.6	1.3
Performance (relative to 3.4 GHz P4)	1.3	1.6	1.9	2.2
CPU Dynamic Power (W)	180	88	43	21
CPU Power Density (W/cm^2)	36	52	74	105

Table 2 – Optimal Performance x86 CPU scaling with Process

It seems like the current trajectory of x86 MPU design will crash into the global wire delay scaling barrier around the 65nm process node. Beyond that one might expect future design advancement to approximately follow the optimal performance CPU scaling trends I derived earlier, listed in Table 1, and applied in Table 2. It is clear is that CPU cores will have to shrink very rapidly in size beyond 65nm to keep clock frequency and performance moving ahead at the best possible pace. This frees up die space for both larger L2+ caches, more highly integrated system level functionality, and also for the inclusion of multiple CPU core instances on the same device, i.e. chip level multi- processing or CMP. This latter trend will be constrained mainly by power budgeting rather than die size considerations.

Conclusion and Summary

The quarter century old practice of designing bigger, better, and wider CPUs for attaining MPU performance beyond process driven clock rate increases is about to be radically curtailed by the twin barriers of power budget limitations and the scaling mismatch between interconnect and transistors. The latter effect is so severe that beyond 90nm the physical size of individual CPUs will drop start falling dramatically even compared to traditional die sizes for high volume desktop PC oriented MPUs. The decrease in CPU size will be used to increase the integrated functions and capabilities and to integrate an increasing number of CPU cores per device. The trend will be especially noticeable in server chips but will also spread to MPUs for desktop computers and eventually even mobile and embedded control applications. Research into highly CMP server chips in the past has been driven largely by a desire to trade off single thread performance per CPU for higher total device level throughput [12]. In the future such an approach may be driven by necessity from the increasing cost of cross die communication rather than by architectural choice. The good news is that the reduction in single thread performance compared to what could otherwise be accomplished will disappear over time.

The need to shrink overall CPU size faster than natural transistor density growth provides means that the complexity of individual CPU cores will need to fall. In the absence of significant advances in implementation efficiency this will accelerate the downward pressure on architectural ("IPC") performance that already exists from the growing mismatch between CPU clock rate growth and main memory access time. Nevertheless the single thread performance of each CPU will continue to rise, driven by clock rate scaling with reduced feature size not that dissimilar from historical levels. The trend towards reduced CPU complexity will also help temper the growth of leakage current power consumption. The need to reduce CPU complexity over time plus the ability to integrate an increasing number of cores on a device to exploit TLP probably means that usage of CPU multi-threading techniques like SMT will diminish over time rather than grow. Although multi-threading techniques tend to contribute only slightly to CPU physical complexity it tends to consume a disproportionate amount of processor design and verification time. With rapidly shrinking CPU size it will become far easier to exploit TLP through increasing degree of CMP. The long term trend to CPU simplification may eventually bring renewed interest in non-x86 ISAs that can provide high levels of ILP exploitation with relatively low logic transistor count for computing markets now dominated by x86 processors.