ghaff@
illuminata.com

Gordon Haff
Illuminata

Breaking Up The Microprocessor Monolith

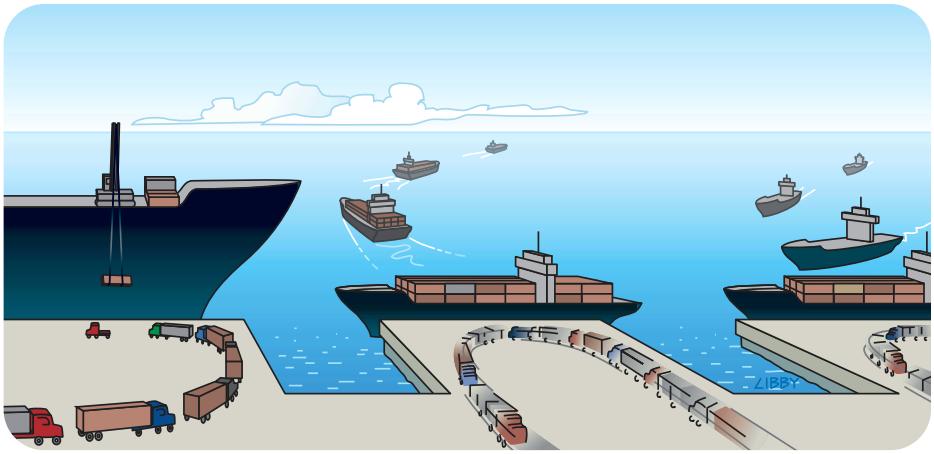
Research Note

Gordon Haff

9 July 2003

Modern cargo ships are about size. Bigger ships can transport large and unwieldy loads such as pre-fabricated bridge trusses or hundreds of truck-sized cargo containers. Big ships are also more efficient. The result is that container ships—which carry their cargo housed in more-or-less standardized boxes for simplified loading and unloading—have grown so large that some can't even fit through the locks of the Panama Canal. A "postPanamax" container ship like the approximately 7,100 TEU[1] "S" class Carsten Maersk carries a load of containers that would stretch 27 miles if placed end-to-end. The shipping industry is considering even larger vessels, whose bulk may be restrained only by the limits of key shipping channels around the world.[2]

However efficient, these beasts are not without limitations. "S" class ships ply a rigidly scheduled route that connects just 21 ports worldwide; New York and other harbors are being forced to dredge deeper channels and invest in larger cranes and docks to handle the biggest ships. Unloading the leviathans also requires enormous terminals with the room to store and sort monstrous piles of cargo from each ship, not to mention even increasing the capacity of roads, rails, and other transport links to move cargo from and to the docks.[3] The real limiting factor, it turns out, is not so much how large a container ship can be built, but whether the associated land-based cargo infrastructure is burly enough to keep it running efficiently.



Odd as the comparison may seem, this is the same problem microprocessors have run into. The load they move is data, not containers, but microprocessors similarly continue to grow in both (die) size and capacity (total transistor

1. Container ship capacity is measured TEUs (Twenty-foot Equivalent Unit). Most boxes in international use are 40 to 45 feet long and count as two TEUs.
2. A 12,000 TEU container ship is the largest that can traverse the Suez Canal and 18,000 TEU the largest that can pass through the Malacca Straits on the way from Singapore and Indonesia to Europe.
3. Maersk's Pier 400 in Los Angeles cost almost $1 billion and sits on a 685 acre site with dedicated rail and highway lines.

Gordon Haff
Illuminata

count). Large processors, like large cargo ships, are highly efficient only when they're loaded quickly and fully, and sent off to do their part of the job.

However, in practice, much of these chips' prodigious processing capacity goes to waste because a healthy percentage of the millions of cycles they run every second go off with less than a complete load of instructions to execute or data to process. Multi-gigahertz CPUs can execute billions of useful instructions per second, but they spend most of their time just waiting for data. Ways to address that waste take many forms and names—throughput computing, thread-level parallelism, multi-core chips, hyperthreading, SMT, CMP, and CMT to name a few. All of these approaches take aim at the same "feeding problem." All are part of a broad design shift away from monolithic single-core microprocessors fed by a single thread of instructions, and toward designs with multiple cores, each of which is rapidly fed by multiple threads. Such throughput-oriented designs are critical as the long-standing performance gap between the processing and the data-holding parts of a system (both memory and disk) grows larger and more imposing every year.

Memory is a Drag

Why is it so hard to keep big processors fed?

It used to be that one of the ultimate goals of microprocessor design was to reach a point where at least some of the simpler instructions could execute in a single cycle. Today's processors are routinely *superscalar*; they use multiple execution units that let the CPU process not just one, but multiple instructions in a single clock tick.

To get a sense of how impressive this advance is, consider that the Intel 8088 in the original IBM PC took two pokey 4.77 MHz clock ticks to perform the simplest data move operation (loading a register with a number). By contrast, a 1 GHz Itanium 2 is designed to simultaneously handle as many as six instructions—many of which it can complete in a single cycle—between its 200x shorter clock ticks. That's over a 2,000-fold increase in processing potential. So what's wrong with that?

One problem is the way processors pull instructions from a stream of data. All major microprocessors use Instruction Level Parallelism (ILP) to help them extract more than one instruction at a time from a single stream of instructions. This accelerates work in the same way that adding additional ticket clerks speeds the line at an airline check-in counter. The line is still single-file, but passengers at the head of the line divide among several ticket stations, reducing the size of the line more quickly, and delivering their requests[4] to the "processor" simultaneously. At least some of the additional agents are likely to be specialists, however; perhaps one handles only first-class customers; another, only passengers who already have tickets. To keep the flow moving to these specialized service areas, additional airline agents start plucking people out of order to direct them to the appropriate location. The result is that processing speeds up *in toto*, but not proportionately to the resources added. The process of selection creates an increasing amount of unproductive overhead as pickers search for the right type of customer to be delivered to the right location.

Microprocessors, being, if it can be imagined, even less effective at handling intelligent sequencing than airline workers, have an even harder time getting the right kind of instructions delivered to the right kind of execution units. They can't arbitrarily grab any random instruction anywhere in the queue to process; a processor's fetching hardware can only go so deep in the line of waiting instructions. What's more, there are the dependencies to deal with. Often the results of one operation are needed to start the next, so those two operations have to be handled in order.

Perhaps worst of all is the huge disparity between the length of time it takes to process an instruction and the length of time it takes to retrieve additional instructions or associated data from memory. These delays are comparable, in airline terms, to walking across a terminal to ask in person how to deal with a complex customer request. Data in a cache on the processor is fed in relatively quickly, but even across the relatively short distances involved,

4. In a processor, a "request" might consist of adding two numbers together or loading some number into a processor register.

getting data from memory chips to the CPU causes a problematic delay that is only getting worse because the speed at which microprocessors can crunch data is ratcheting up more quickly than is the speed at which memory chips can feed it.[5]

The result is a big, sophisticated, wickedly-fast processor that spends much of its time idle, waiting for appropriate instructions or data to be delivered from memory. Imagine if that whole line at the airport had to wait for a passenger who had to drive home first for a forgotten ticket, then back again to collect their ID, and then their luggage, and so on. Ridiculous? Perhaps. But that's exactly what happens when an instruction causes a processor to wait while data comes in on the slow boat from memory. The wait is *hundreds* or *thousands* of cycles.

Inadequate Evolution

One approach to reducing all this waiting is applying a variety of "tricks" that, collectively, try to keep the chip's execution units as busy as possible—even if it means doing some work that may turn out to be unnecessary. For example, processors pre-fetch instructions and data from memory into their caches. This often includes pulling down memory contents before the processor is sure they're actually needed.

This speculative loading is particularly important when there are branches in code ("if TRUE execute this code, else execute that code"). There usually are. Without pre-fetching and other preparatory activity, the processor would have to stop every time a branch was reached and wait to be shown the correct execution path to take, which would kill performance.[6] Therefore, CPUs try to intelligently predict which branch will be followed based on past code patterns and other clues. This isn't infallible, but even if it's often wrong, guessing is better on average than consistently doing nothing.

Another technique is to forget about trying to divine the code's future and just go ahead and start down both paths. This *speculative execution* may seem like a profligate use of processor resources, but if those resources would have been wasted anyway, there's little harm. Those compute resources aren't usually the bottleneck, anyway; the memory lag is.

Super-fast data stores built right on the processor itself—*caches* that store the subset of main memory that's been most recently accessed—are yet another tool. The caches nearest the CPU core can deliver instructions to the processor's execution units in just a cycle or two—more than 100 times faster than they can be delivered from main memory. Even second and third-level caches, which are farther from the core than a first-level cache, are as much as 30 times faster than memory. Caches are growing in capacity, taking advantage of the inexorable shrinkage of the transistors making them up. They're also getting more complex, distributed in more levels,[7] to better achieve the right mix of raw speed and size.

These tools and techniques centered on reducing memory latency have gotten extremely sophisticated. And they're quite effective. After all, system performance *is* continuing to escalate in spite of the very real drag that memory speeds impose.

But chip designers are paid to look to the future as well as to optimize the present. Pretty much everyone agrees that fairly fundamental changes are needed to keep performance advancing apace. There's even some consensus of the general direction of the changes. The debate is in the details and the degree to which one approach will win out over another.

5. See Illuminata note "Latency Matters!" (Sept. 2003).

6. Stopping processors in their tracks and restarting them is enormously costly. Although the degree varies by design, all of today's sophisticated processors are heavily *pipelined*. They break instructions into smaller micro-operations to simplify the tasks that have to take place during each internal clock cycle. Once the pipeline is filled and churning along, this approach is quite effective; but when a pipeline gets flushed for any reason (such as switching to a branch other than the one executing) the performance hit is severe, because the pipeline needs to fill back up again before proceeding. In the Intel NetBurst microarchitecture, for example, the highly important branch-prediction/recovery pipeline is implemented in 20 stages.

7. For example, Intel's Xeon and Itanium 2 processors both have three levels of cache—each level successively larger, but slower. System vendors building Big Iron from Intel processors, such as IBM and Unisys, often augment per-processor caches with yet another level of cache (L4), shared by several CPUs.

Gordon Haff
Illuminata

Tweaking the ability of processors to wring the most out of a single instruction stream can only go so far. Yet even Intel's Itanium Processor Family (IPF), at least in its current incarnation, has much in common with the traditional high-end RISC approach in that it's a single large processor core connected to large on-chip caches. The *means* by which it strives to keep its multiple execution units fed differs from RISC,[8] but the *end*—high parallelism within a single instruction stream as measured by the number of instructions per cycle (IPC)—is the same.

Expanding cache size is an equally limited option. The 6 MB cache in the upcoming "Madison" version of Itanium 2, for example, consumes more than half the processor's total area. Indeed, Itanium 2 is perhaps the ultimate engine for wringing parallelism out of a single instruction stream—as is particularly well demonstrated by its prowess in high performance computing (HPC) and certain encryption algorithms. In contrast to the less uniformly structured and more I/O intensive code of commercial transactions, HPC is often characterized by repetitive, compute-intensive, and highly-optimized algorithms with plenty of "tight loops"—all characteristics that can translate into lots of instructions per clock, and, therefore, effective ILP.[9]

As optimization tricks become more sophisticated and intricate, however, the gains from incremental advancements diminish. It happens with auto fuel efficiency. It happens with stereo equipment. And it happens with today's high clock rate, super-scalar, out-of-order executing RISC cores.[10] Eventually, big changes are the only way to get noticeable improvements. With apologies to the high-end audiophiles who still think vinyl gives a "warmer" sound, the last big enhancement to consumer audio didn't come from designing a better turntable, it came from introducing the CD, which did away with pops, skips, and worn needles.

A New Plan

But there's an alternative design approach. Rather than devoting a chip's entire area to a single large processing engine and its associated caches, it's possible to split the chip into two or more smaller engines, each of which is capable of handling a separate instruction stream. This multiprocessor-on-a-chip approach—often called Chip Multi Processing (CMP) or just multi-core—attacks the problem of single-minded processors that are growing in size faster than they can be fed with instructions and data by cutting them into pieces, each of which can work in parallel. It's the logical equivalent of replacing one big chip with several smaller ones. However, physically consolidating them onto a single die typically cuts the per-processor-core cost. What's more, multiple cores on a single chip can communicate and coordinate with each other more quickly than if they were on separate chips, often leading to better overall scalability.

Chip designers also have a complementary technique up their sleeves, called Simultaneous Multi Threading (SMT). SMT makes a single processor core appear to software as two or more separate CPUs, even though there is only one core and set of execution units physically present. SMT requires some incremental circuitry; each logical processor (or thread) needs its own copy of certain resources to let it present to software the illusion that it is an independent CPU. However, these resources—which include the instruction pointer, Instruction Translation Lookaside Buffer (ITLB), and various registers—take up a relatively small number of transistors. Simple implementations, for example, take up only about five percent of the total chip area, far less than duplicating the entire processor core. Intel is delivering (and heavily promoting) SMT under the trade name "Hyperthreading" on all its current 32-bit processors today, [11] but it's also on the near-term roadmaps of both IBM and Sun.

8. IPF is based on what Intel calls EPIC (Explicitly Parallel Instruction Computing), a derivative of VLIW (Very Long Instruction Word) architectures, in which much of the parallelizing work gets passed off to compliers, rather than having to be performed in real-time by the processor.
9. Although some types of HPC, such as life sciences, film processing and rendering, and petroleum reservoir simulation, are becoming quite data intensive as well, which makes I/O at least as important as raw compute power of any type.
10. Including a chip like Intel's Pentium 4 that, in spite of supporting a complex instruction set to maintain backward software compatibility, is optimized to run a relatively small number of instructions very efficiently.

E Pluribus, Power

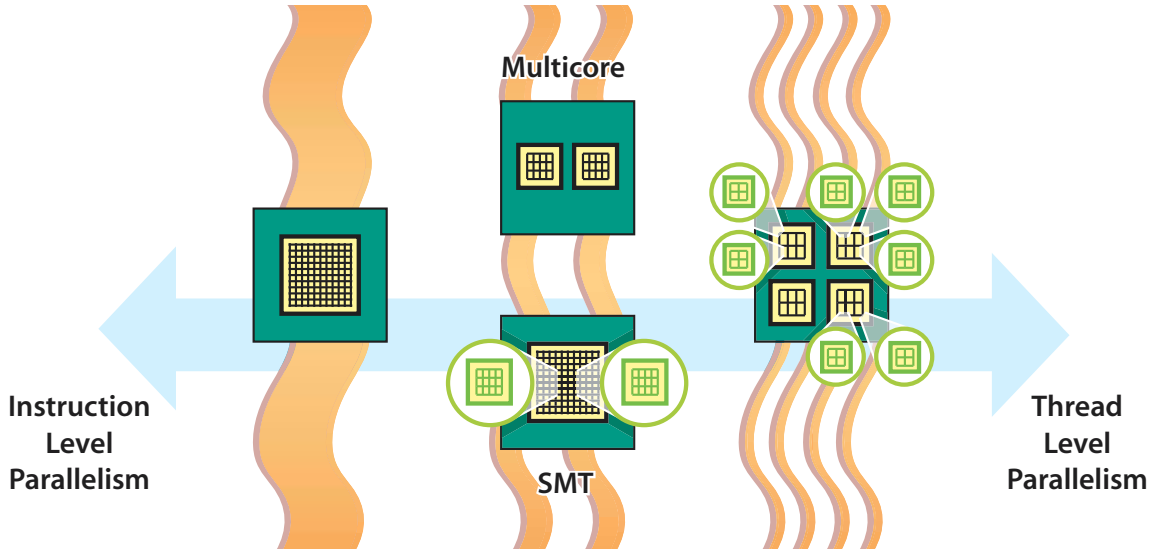
These alternative approaches tilt the space-planning portion of chip design toward smaller and simpler cores and more threads per core, and away from monolithic layouts. They take a thread-level parallelism (TLP) approach—in contrast to the instruction-level parallelism that's been the focus of microprocessor design to date. Rather than relying on a series of tricks to pump a single instruction stream through as quickly as possible, TLP techniques are designed to let the chip handle several chains of instructions at once, efficiently switching away from tasks that are waiting for data to arrive from memory, and replacing them with tasks that have their data ready for processing.

Underlying TLP is the fact that modern server programs[12] tend to be extensively multi-threaded, throwing off many streams of code that execute autonomously. This is the result of optimizing for SMP servers that range from two up to 32, 64, or more processors. Without such multi-threading, a program could only run on a single processor—which would rather defeat the purpose of SMP. The TLP approach provides those many threads with lots of cores (or logical cores in the case of SMT) on which to execute, while cutting down on the number of discrete physical CPU chips and interconnects, as well as the infrastructure needed to support them. TLP can't speed up memory. But if the core waiting for data is one of many lightweight units, rather than one of a small number of heavier editions, the overall performance cost to the system is much reduced.

This thread-based approach demands more system bandwidth than ILP because more of the work is being done outside the processor; TLP requires that the system maintains more simultaneous instruction streams between memory and CPUs. In addition, TLP processors would tend to have smaller individual cache sizes because space-consuming caches exist primarily to reduce average memory latency—an overriding concern for current ILP designs, but not as much for TLP. Smaller caches, however, tend to further increase system bandwidth demands when they are too small to hold the data or code that is reused, and which must then be retrieved from memory.

TLP doesn't eliminate system design challenges, but it definitely reduces them. Goosing up a system's bandwidth is relatively straightforward; building

11. These "Netburst" micro-architecture CPUs include both the Pentium 4 and Xeon families.
12. Desktop computers also run multiple threads. A quick visit to the Windows XP task manager shows Microsoft Word using 10 threads, the Mozilla browser using 15, a Symantec virus-scanner using 36, and all programs using 452 threads in aggregate. However, although multi-threading is becoming more widespread within heavyweight desktop applications like Adobe Photoshop, it remains less pervasive than on servers.



fatter pipes throughout the system is more a question of cost than theory or architecture. Latency, however, is more the result of fundamental limits of technology and physics—things like the speed of light that are not so easy to overcome.

Of course, TLP vs. ILP is not an all-or-nothing proposition. A processor core used within a TLP-oriented approach still wants to run a given thread as quickly as possible—and that means using some ILP techniques like caches and branch prediction. The object isn't to yank all the ILP optimizations out of the processor, but rather to find that right mix that delivers the best overall performance for the target application mix. So TLP cores will still have ILP optimizations, though fewer of them, less intensively cultivated; truly heroic measures like out-of-order execution, speculative execution, and proliferating execution units are therefore much less likely to permeate TLP designs, while straightforward optimizations like caching and multiple register I/O ports are highly likely.

The Thread Wars Heat Up

The advent of TLP is upon us. Everyone from IBM to Intel to Sun understands that it's coming, though their approaches and advocacy come at different levels.

Sun is the leading advocate for TLP, explicitly weaving the idea of thread-level parallelism into all its statements about future processor directions under the banner of "throughput computing." And it's not just talk. Sun's applying a combination of experience developed internally (such as that obtained developing its MAJC graphics engine) and acquired externally (through its purchase of Afara Websystems) to make TLP very much the centerpiece of its future SPARC microprocessors. The most exuberant example of this is "Niagara," a 32-simultaneous-threads CPU due to appear in systems late in 2006.[13]

13. See Illuminata note "Sun: Better Computing Through Threads" (July 2003).

Major server microprocessor plans for multiple cores and multiple threads per core

	Today		*Future (Public Roadmap)*
Vendor/Architecture	**Multi-core**	**SMT**	**Multi-core and/or SMT**
AMD / AMD64	No	No	No
Intel / IA-32	No	2 threads (aka hyperthreading)	Hyperthreading continues.
Intel / IPF	No	No	Dual-core (Montecito in 2005)[a]
HP PA-RISC	No	No	Dual-core (PA-8800 in late 2003)
IBM POWER	Dual-core (POWER4)	No	Dual-core/dual-thread-per-core (POWER5 in 2004) "Ultra high frequency cores" (POWER6 in 2006)
Sun SPARC	No	No	Dual-core (UltraSPARC IV in 2004)[b] Dual-core (Gemini in 2004)[c] 8 cores/4 threads-per-core (Niagara in 2005)[c]

a. It has been reported, but not confirmed by Intel, that a post-Montecito generation of the Itanium Processor Family (Tanglewood) will have additional cores and possibly SMT.

b. The UltraSPARC V (2006) will also support up to two total threads in a design that will be able to effectively work as a single large core or two smaller ones.

c. Unlike the UltraSPARC IV, which is designed for use in Sun's large SMP servers, Gemini and Niagara are targeted for blades and other scale-out systems.

Gordon Haff
Illuminata

Other vendors are not as directly focused on TLP, nor do they feature it so centrally in their product roadmaps. But that's not to say that they are ignoring the trend toward thread-level (rather than instruction level) parallelism. Indeed, both IBM and Intel have already incorporated considerable TLP capabilities into their own chips, whereas Sun has yet to do so.

IBM, for example, began shipping dual processor cores in its POWER4 processors in late 2001. Forthcoming POWER5 designs continue down the multi-core path, adding multi-thread support as well. HP's PA-8800 processors due in late 2003 will be multi-core designs.

Even Intel—the company Sun likes to hold up as producing the industry's ultimately monolithic, maniacally ILP chip, Itanium—is a heavy promoter of multi-threading in its Pentium 4 designs. It began introducing "HyperThreading" (a form of SMT) to Pentium 4 in mid-2002, and has extended HT to most Pentium and Xeon chips since. To be sure, SMT will continue to serve somewhat different purposes depending upon the complexity of the processor core with which it's paired. With a super-scalar core such as that found on Intel's 32-bit Pentium 4 and Xeon CPUs, SMT aims to keep the multiple execution units busier than is possible with ILP techniques alone by layering thread-style parallelism atop ILP; it's not exclusively concerned with switching away from instruction streams that have been held up by memory latencies. By contrast, Sun is looking to pair SMT primarily with simple cores, where it will act as more of a fast thread-switching technique that directs CPU resources toward those threads that have their associated data in from memory and ready for processing. The overall concept—multiple logical processors per physical processor—is the same in both cases, but in one case SMT very much augments ILP while in the other case it largely offers itself as an alternative.

However, Intel also sells a hugely multi-threaded network processor, albeit of a specialized architecture and design,[14] and it has acquired a significant body of SMT intellectual property and a number of designers from the now-ended Alpha processor family. [15]

Everyone's heading toward TLP. Most or all of the major processor options will gain TLP optimizations over the next three years. Some are just heading there faster and with a more determined step than others.

More Ports, More Ships

Which sounds more efficient: a single big cargo ship that waits for every last container to arrive before it can leave port or a fleet of small boats that can leave individually from a variety of different ports as each container shows up? Sure the big boat is pretty efficient once it's fully loaded and underway, but feeding it with cargo takes costly and rarified infrastructure. And anything that causes it to sit idle is enormously wasteful of resources.

Microprocessor design is shifting from monolithic to modular as it shifts from an ILP-centric approach to a more TLP-centric one. ILP won't go away entirely—think a fleet of mid-sized cargo ships rather than an ocean full of tiny motorboats. But no longer will the ultimate processor be the one with the biggest, baddest single core.

---

14. The IXP2800 family puts 16 multi-threaded processing engines onto a single chip. IBM also sells network processors under the PowerNP label; they also have 16 "pico processors."
15. Intel will reportedly mine much of this "EV8" IP for future generations of Itanium.

Gordon Haff
Illuminata