ghaff@
illuminata.com

Latency Matters!

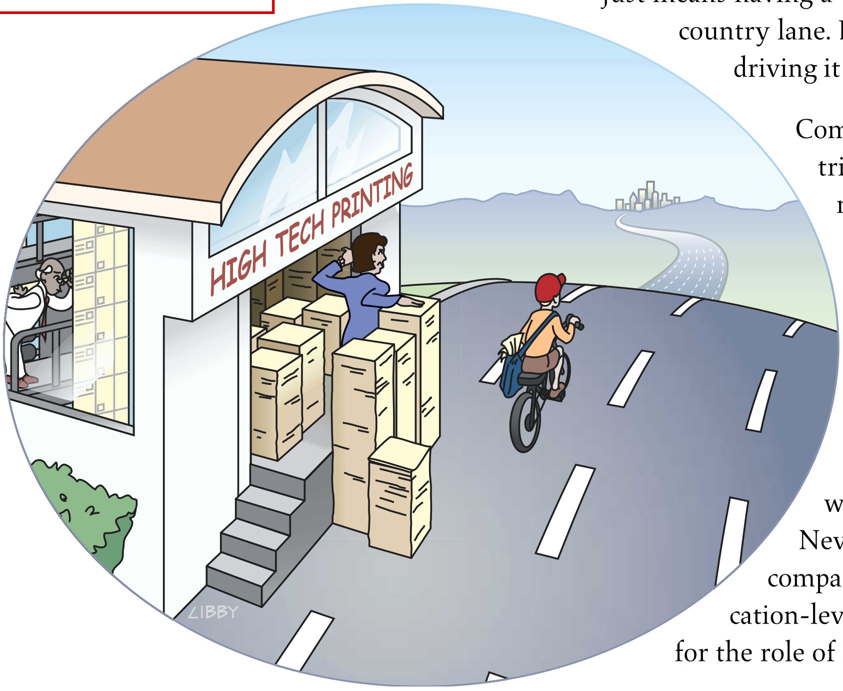
Research Note

Gordon Haff
19 September 2002

What's the best way to estimate travel time? Would you rely on an estimate based solely on the number of lanes in the road and the sound of the engine? Nope. You need to know, at minimum, how far you have to travel, the condition of the road, and how fast you'll likely be able to go. Obvious, right?

You'd think so. But system and networking specs rate computer performance according to bandwidth and clock speed, the IT equivalents of just measuring the width of the road and the engine's revolutions per minute. While they may be interesting, even important, data points, they're hardly the complete story. Not surprisingly, vendor specs show a remarkable number of gigabytes shuttling to and fro every second—numbers derived from theoretical peak calculations, not actual measures of data movement. In the real world, you have to consider the vital factor of *latency*.

Latency is the time that elapses between a request for data and its delivery. It is the sum of the delays each component adds in processing a request. Since it applies to every byte or packet that travels through a system, latency is at least as important as bandwidth, a much-quoted spec whose importance is overrated. High bandwidth just means having a wide, smooth road instead of a bumpy country lane. Latency is the difference between driving it in an old pickup or a Formula One racer.



Comparing the latency of two systems is tricky because access time depends heavily not only on configuration, but also dynamic variables like system load. What's more, vendors often use different definitions. Or they report numbers that are variously best-case, worst-case, or something in-between—without providing enough clarification for customers to evaluate which situation the numbers reflect. Nevertheless, even if precise numerical comparisons are difficult, understanding application-level performance demands an appreciation for the role of latency in different architectures.

Powers of Ten

The first thing to keep in mind is how great the difference in latency can be from one type of component to another, and how designers can minimize delays by making critical data available from repositories with the shortest lag times.

For example: A 1 GHz processor can run an operation through its execution units in a billionth of a second—the same amount of time it takes electricity to travel along a wire about seven inches long.[1] If the processor lacks the instructions or data to complete an operation, and takes one cycle to line up the data from an on-chip cache, that adds maybe a nanosecond or two of delay. If it has to retrieve data from memory that isn't in the cache, it could take microseconds to retrieve and process. If data has to come from a disk drive that has to locate the data and send it across a significant chunk of space to the processor, the delay will probably be in milliseconds. If the data has to come from outside the machine, across the network, the delay could be a second, or more—latency at least a BILLION times greater than if the data were nearby.

What's the worry? Can't the processor just do something else while it's waiting for data to come in? Yes it can, and modern designs use all sorts of strategies like buffering, pre-fetch, register renaming, out-of-order execution, and multi-threading to do useful work while data is being accessed. But each of these strategies has a cost and adds its own complexities. And such workarounds can only go so far. If data's not handy, it's a fact of life that execution units and other system resources are frequently going idle while waiting for the data to arrive. Latency is therefore a key determinant of what is achievable in system efficiency and performance.

That's why chip designers build in registers—effectively very fast and specialized memory on the CPU itself. It's also why caches are such a pervasive design element at all levels of system design. All communication comes with some degree of latency.

1. Electricity travels down a copper wire at about two-thirds the speed of light in a vacuum.

Each task comes with its own latency requirements. Short latencies enable tasks that require a high level of coordination among different components and need an intimate awareness of each other's actions and status.

For example, the underlying hardware of most computer systems has to constantly check and take action to ensure all the caches and memory are in a consistent state. That helps keep them closely coordinated, but also adds to administrative overhead.

By contrast, the relatively long latencies found on public data networks mean that communications over them have to be relatively autonomous and asynchronous. It's still necessary to check data for errors, but it's not practical to hold up a stream of packets while the system and an application discuss whether the previous packet arrived successfully. Intra-system and on-network communications are like the difference between a telephone call and an exchange through the mail.

The performance-profiles of clusters, systems, chipsets, and chips depend heavily on the distance and latency among components. The closer to the core of the processor one gets, the more intimate and coordinated the communications must be. Physics limit the options designers have to eliminate latency, but within specific domains, their decisions can make a system's performance scream—or whimper.

Consider the problems to be solved in two domains: systems and datacenters.

The Memory Wall Bottleneck

Memory is a relatively low-latency source of data, but the delay that comes from writing to or reading from memory has been improving at a drastically slower pace than have CPUs.[2] Consider that, in 1993, Pentium chips ran at a clock speed of 60 MHz while today a Xeon MP clocks at 1.6 GHz—almost a

2. Wm. A. Wulf and Sally A. McKee popularized this observation with the term "memory wall" in their paper "Hitting the Memory Wall: Implications of the Obvious."

30:1 difference.[3] Not only have the number of cycles per second increased, but the work chips can do in each cycle has also increased with every major new chip iteration.

By comparison, the 60 nanosecond (ns) access time of fast-page-mode DRAM used in the early nineties has dropped to about 7 ns with current PC2100 DDR-SDRAM—about a 10:1 improvement. And this *overstates* the increase because it refers only to the raw memory array access time. Other factors—such as the time data takes to get through the memory subsystem logic and the time to communicate between the processor and the memory—continue to limit how quickly data can be read from and written to memory to an even greater degree than the underlying memory chip hardware. The ratio of 10:1 is also optimistic because DDR-SDRAM is a very fast kind of DRAM that has only recently begun making its way into servers.

So processor speeds have outstripped memory access speeds by a factor of at least six. As a result, today's processors wait an increasing number of cycles for the data they need—potentially sitting idle in the meantime.

This relatively slow memory speed is one reason applications don't get faster at the same pace that processor clock speeds increase. A 2 GHz processor is not necessarily that much faster, in terms of application work accomplished, than a 1 GHz processor. Furthermore, high-end system designs now include as many as 128 processors and more than 512 GB of memory in a single memory image. The infrastructure needed to support this scale—to shuttle instructions and data among all those processors and memory—adds its own latencies to those of the components.

Local caches can minimize the amount of data that has to be moved, and thereby reduce latency to some extent. System designers can often use faster memory chips for caches than they can use in main memory because the caches are smaller than main memory and therefore can economically justify more expensive, but faster, memory types. Caches also help distribute load around the system; in this respect they enhance effective system bandwidth as well as reduce latency bottlenecks.

The small on-chip L1 caches nearest the CPU core can deliver data and instructions to the processor's execution units in just a cycle or two—over 100 times quicker than it can be delivered from main memory. Even multi-MB L2, L3, and L4 processor caches serve up data as much as 30 times or so faster than memory.[4] And the larger the system, the more important caches are to the performance of the overall system design.

But caches have limitations. They primarily benefit applications whose data sets fit in cache and are repeatedly accessed. Though many applications fit that description, not all do. For example, multimedia applications may call on a server for large chunks of data that they use only once, before requesting a different chunk. This constant flow of fresh data tends to flush the caches almost continually, rendering them nearly useless. Indeed, because many designs require a processor to query the cache first for data it needs, such one-time data dumps can actually increase latency by forcing the processor to delay its memory requests until it has first checked the caches for data that they certainly don't contain.

However much it can accelerate performance, caches must be designed efficiently, a problem that gets more complex as the systems they're in become more powerful.

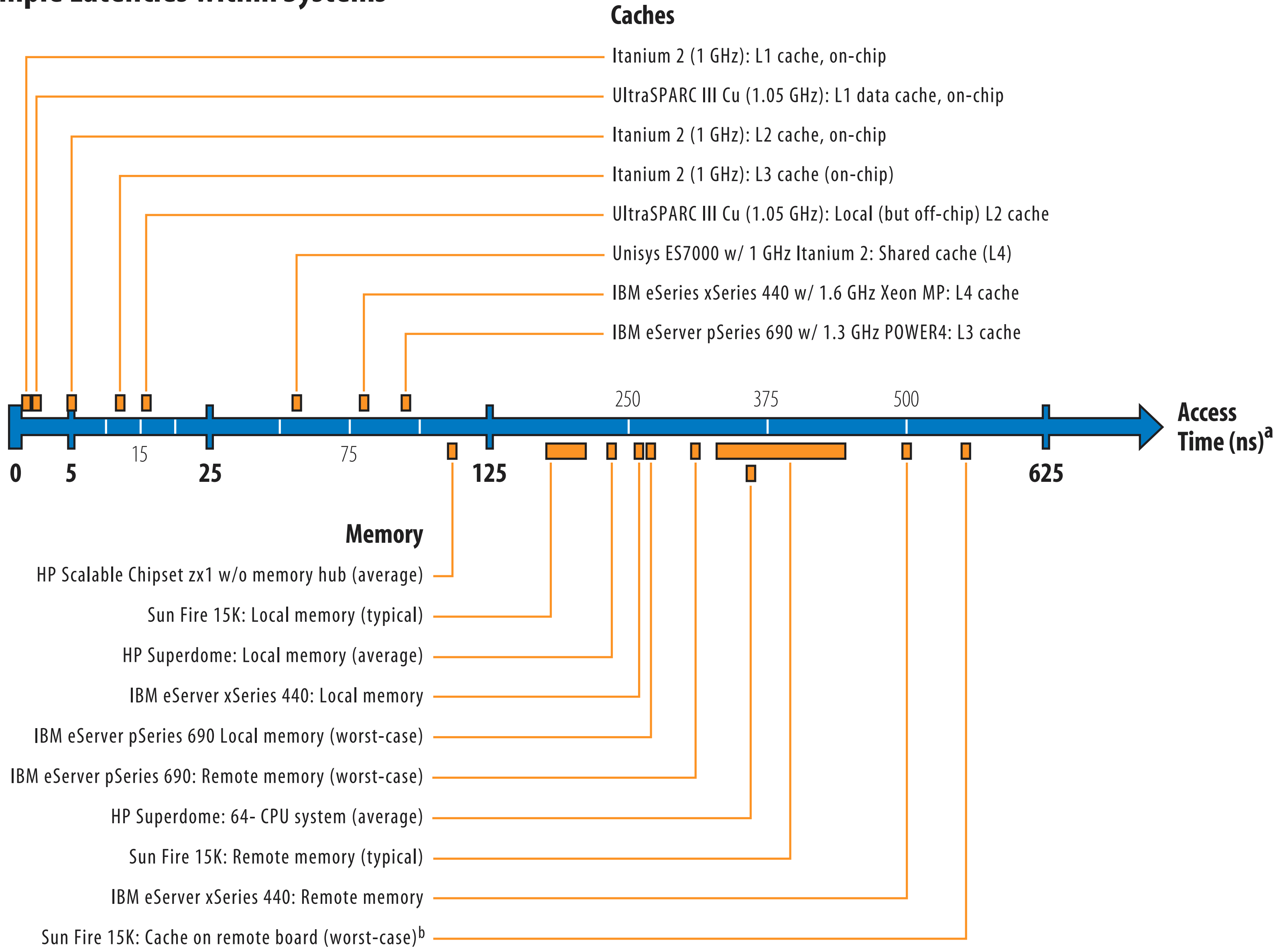
Bigger Systems, Longer Latency?

Large systems require multiple large caches, but managing them raises its own set of issues. SMP (symmetrical multiprocessing) systems—including

3. The ratios are much higher still on Pentium 4, for which common speeds approach 3 GHz.

4. The different levels of cache line up in different ways for different chip architectures. For example, Itanium 2's "large" (3 MB) cache is the third level of cache in the architecture while in UltraSPARC III's large (8 MB) cache is the second level. IBM's EXA chipsets and Unisys' CMP architecture, combined with Xeon MP or Itanium 2, offer an unprecedented four levels of processor cache.

Example Latencies within Systems



a. Do not use this diagram to calibrate medical instruments or other devices requiring extreme precision. Logarithmic progression, not to linear scale. HP and IBM memory latencies are "load-to-use" numbers that explicitly include latencies within the processor (~58 ns in the case of PA-RISC) while Sun's "pin-to-pin" latencies do not.

b. When a memory location is owned by another CPU and the current value of the data is in one of that CPU's caches (i.e., memory has the old value of the cache line), the data has to come from the owning CPU, rather than from memory. This operation can take longer than retrieving data from "clean" memory.

both uniform memory access (UMA) and Non-Uniform Memory Access (NUMA) implementations—are "cache coherent."[5] In the early days of multiprocessing, this consistency was achieved by using "write through" caches in which any changes to cache were also made to memory in a single operation, ensuring that both the cache write and the memory write completed without interference by other system activities. While logically simple, such a design approach reduced caching's benefit by requiring all writes to update main memory (and typically locking the bus while the update was underway) before moving on to the next task.

Today, more efficient caching schemes rely on various mechanisms to present a consistent view of the entire cache and memory pool, rather than updating memory every time the data in cache changes. For example, in a typical design based on ownership protocols, CPUs monitor the system address bus to see if the data address on the bus is the same as that in its own cache. If so, they can update other caches and/or memory, depending on whether the data is being shared among several caches, and whether it's been modified relative to other caches or memory.

However, even the most sophisticated caching approaches can only reduce the number of times that main memory is touched. And more processors mean more touches on both cache and main mem–ory, more supporting infrastructure, and more physical distance between processors and memory. Each of these increases the latency of memory access. That's why a dual-processor version of HP's zx1 chipset can have a memory access time of only about 112 ns (including latencies within the pro–cessor) while the 32-processor version of the Unisys ES7000, a "classic" Big Iron crossbar design, needs about 300 ns—almost three times as long—just to traverse the interconnect to memory.

5. By contrast, more distributed architectures—such as MPP (massively parallel processing) computers—are effectively a bunch of smaller computers interconnected with a high-speed network.

Scaling Both Size and Speed

The attack that most vendors have launched against the limits memory access latencies impose on scalability is Non-Uniform Memory Access (NUMA) architectures.[6] NUMA designs are built around building blocks that incorporate some number of processors—typically four—and a portion of the system's main memory.[7] The block diagram looks a lot like a distributed memory or MPP (massively parallel processing) system, but there is a critical distinction. All the processors in a NUMA system can directly access all the memory in the system, no matter where it is physically located. The de–signs give software the same consistent view of memory and caches as does a system that implements memory as a single global pool. By contrast, processors in an MPP architecture only own their local memory.[8]

The advantage of a NUMA architecture is it allows processors to keep some memory close by, where it can be accessed faster than would be the case with centralized memory. For example, processors in both the Sun Fire 15K and the HP Superdome can access local memory on their own building blocks in roughly 200 ns, a full one-third faster than the Unisys ES7000 in which all memory sits in a single centralized pool. This even though the Sun box supports up to 72 processors (and up to 106 with configuration tradeoffs) and the HP system supports 64—compared to 32 for Unisys.

The other side of that coin is that the Sun Fire 15K takes almost one-third longer than the ES7000 to access memory on a different board, which makes it even more important that applications be structured to use NUMA systems wisely. If an application can locate a chunk of logic and its associated data within the same block, latency for that process will be

6. We have also called these nuSMP, or non-uniform SMP.
7. Sometimes I/O is also included as part of the building block; in other cases it is a centralized resource.
8. Distributed memory architectures sometimes incorporate mechanisms such as RDMA to access non-local memory, but the memory's owner must always grant explicit permission.

much lower than if the data and logic occupied separate blocks. IBM's x440 NUMA system takes more than 50 percent longer to access memory outside a local block, for example.

Data General's AV 20000, which together with Sequent's NUMA-Q pioneered the use of NUMA for commercial applications, had local-memory access speeds that are comparable to current boxes but had a remote access latency of 2.3 to 3.1 *micro*-seconds. That's about a 15:1 ratio between remote and local accesses. By contrast, today's systems typically range from approximately 2:1 to 3:1.

Some NUMA designs are better at minimizing the non-local access times, however. HP's Superdome, for example, averages access times on a 32-way system that are within 10 percent of the ES7000's approximately uniform memory access times; and the worst-case remote-access times in IBM's p690 distributed switch design are essentially a wash with those of the Unisys platform.

IBM's p690 stands out by its use of a distributed-switch architecture to deliver almost uniform memory access times between local and non-local memory. This miniscule latency delta (about 1.15:1) shows that the "penalty" associated with accessing physically distributed memory is getting smaller with each design iteration.

From an application perspective, the key metric is the average latency rather than the best or worst case. Vendors apply a combination of techniques to drive this number down, primarily by cutting down on the number of remote calls to memory.

One approach is to put on each board a large cache that can store the data from recent remote-memory requests, so they are available locally for the next similar request. These caches, which typically range from 32 MB to 128 MB, can deliver data about as fast as can local memory. IBM's x440 has a particularly speedy version; cached remote *and* local references can be accessed in just 80 ns.

Applying OS intelligence to the problem is the other common approach. Scheduling techniques such as processor affinity and memory affinity can help keep processes that are being executed and the data they need physically close to one another, maximizing the percentage of accesses that are local. Sophisticated commercial Unix operating systems have had these capabilities for years—in part because they bring some benefit even to large-scale systems with uniform memory access. By contrast, Microsoft is just in the process of adding that ability to Windows .NET Server. Thus, at least for now, systems that run Linux or Windows—such as IBM's x440—are more dependent on hardware optimizations than their RISC/Unix counterparts, which have software to share the burden.

The Datacenter Latency Bottleneck

One useful, though not wholly accurate,[9] distinction between access inside-the-box and access outside-the-box used to be summarized as tightly coupled access vs. loosely coupled.

Tightly coupled access means that the processor, cache, and memory interactions within a shared memory system all add up to a single, consistent memory view. That way, when a processor accesses any address in memory, it can be sure that it gets back data in the order that it was written—whether it comes from in cache or memory. It's up to the hardware to make sure that even simultaneous actions by different processors don't break this view.

By contrast, loosely coupled access in distributed systems, such as clusters, still coordinate memory and data exchanges, but at a comparatively superficial level. They do have a common view of databases and other shared data, but don't need to worry about the internal details—such as cache coherency—of other systems. These designs must tolerate greater latency than tightly coupled systems, though there are ways to minimize latency even in this design.

9. Some relatively long distance links such as SCI (Scalable Coherent Interface)—formerly used in Data General and Sequent NUMA systems—do support the coherent memory semantics associated with "tight coupling."

Even more loosely coupled are client/server in– teractions, which typically must cross TCP/IP networks—anything from a local LAN to the entire Internet. Applications designed to run in this environment *must* be latency-tolerant because even local hops can take 60-120 microseconds. Internet round trips even to nearby hosts are on the order of 20-50 milliseconds; a traceroute to Timbuktu or Kathmandu often approaches a full second.

However, loose coupling isn’t the same as no coupling. Clusters—multiple systems that access a single database, or share processing functions—place a particular premium on streamlined coordination. Yet latency for the TCP/IP-over-Ethernet nets that most cluster configurations use for transport is on the order of 10 to 50 times that of optimized high-performance cluster connects—such as those that use the very low-latency Memory Channel interconnect in HP TruClusters. It’s no coincidence that TruClusters are also the only Unix-based clusters routinely scaled to more than two nodes in a shared-database configuration. Past Compaq tests using Oracle Parallel Server (OPS)[10] showed about a 45 percent increase in transaction throughput by substituting Memory Channel for Ethernet-based connections.

Interconnect latency is an even greater factor with the Oracle 9i RAC database—which introduced cache fusion, a new feature designed to eliminate much of the disk I/O associated with shared database clustering.[11] With cache fusion, the in-memory cache of a nearby node serves the cluster data, rather than having the processor acquire data from disk as was the case with OPS. That’s a significant architectural improvement because it takes about 10-20 milliseconds to pull data off a typical disk array. (Even data present in a disk array’s read cache takes about 2 milliseconds to retrieve.) However, it also means that the importance of the connection speed among clustered systems is even greater—given that the bottleneck at the disk is so much less than it was.

10. The parallel version of the Oracle database prior to the Oracle 9i release. Oracle 9i RAC replaces OPS, and seeks to make parallel database implementations much less rarefied.

11. Polyserve uses a similar technique in its large-scale cluster file system.

Memory Channel isn't the only high-performance datacenter interconnect available; the current mar– ket for interconnects is enormously fragmented. Others include Myrinet (Myricom), ServerNet (HP), SP Switch2 (IBM), BYNET (NCR), QsNet (Quadrics), HyperFabric 2 (HP), GSN (SGI), and cLAN (Emulex). Other than in a few specialized, performance-oriented niches, Ethernet remains the standard for system-to-system connections. IBM, Sun, and others vendors have been promoting the emerging InfiniBand standard as a broader-based high-performance interconnect, but there are few production sites as of mid-2002.

What makes the speed of these interconnects differ from each other isn’t so much the type of media or the specifics of the hardware interface. A signal takes about the same length of time to get through the low-level physical layers of one as another. As standard networks grow toward 10 Gbit/sec speeds, the physical hardware underlying 10 Gigabit Ethernet, Fibre Channel, and InfiniBand looks increasingly similar. Indeed, the trend is toward making components like connectors common across as many different interconnect variants as possible.

What gives the high performance datacenter interconnects dramatically shorter latencies than standard TCP/IP-on-Ethernet is their lack of overhead from software-stacks and link protocols. TCP/IP was designed for and remains oriented toward wide-area communications over inherently unreliable links. For example, the TCP protocol performs a checksum for each packet, whereas an interconnect such as InfiniBand can pass the job of ensuring reliable transmission to the hardware in the lower layers. All this adds up to a huge latency difference between TCP/IP-over-Gigabit Ethernet compared to interconnects and protocols that are optimized for datacenter roles such as clustering.

Example Latencies within Datacenters

Protocol/Transport	Typical latency (one-way)	Bandwidth per link (unidirectional)	Notes
HP Memory Channel	2-6 microseconds (as low as 2.2 micro-seconds with native messaging; 6.4 micro-seconds with MPI messaging)	100 MB/sec (sustained point-to-point)	A key technical ingredient for getting the best performance from Tru64 clusters.
SGI Gigabyte System Network (GSN)	< 10 microseconds (adapter) < 30 microseconds (with MPI messaging)	790 MB/sec (typical application)	Offered by SGI as a premium, higher performance alternative to Myrinet.
InfiniBand	3-20 microseconds (native hardware RDMA)	2.5 GB/sec for 4X links (theoretical peak)	Being promoted as a new interconnect standard that can replace both high-performance interconnects and Ethernet within the datacenter.
Myricom Myrinet	7-9 microseconds (small messages)	245 MB/sec (sustained)	The standard for high-perfor-mance compute clusters.
IBM SP Switch2	1 microsecond (raw) 18 microseconds (with MPI messaging)	500 MB/sec (raw) 350 MB/sec (sustained with MPI)	IBM's high-performance switch focusing on large, shared-nothing clusters such as its SP systems.
HP Hyperfabric2	22 microseconds	320 MB/sec (raw peak)	Adds reliability features (e.g., reliable datagram) on top of Myrinet physical and switching layers.
Emulex cLAN	7-40 microseconds	160 MB/sec (raw peak)	Hardware implementation of VI (TCP/IP bypass) protocol.
Fast and Gigabit Ethernet (with TCP/IP)	60-200+ microseconds	128 MB/sec (raw peak)	The default commodity stan-dard for system-to-system communications.

In the Blink of an Eye

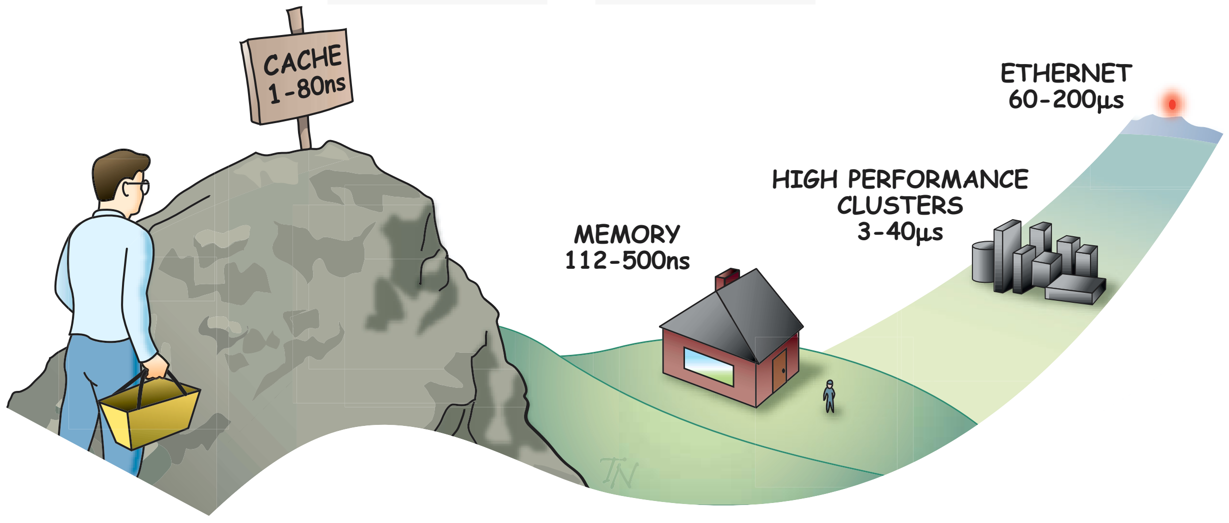
In so many ways the IT industry is obsessed with speed. Frequency dominates processor discussions in the public eye to such a degree that AMD has even taken to naming processors with a megahertz equivalency rating—which just happens to be a higher number than the chip's actual physical clock rate.[12] It's therefore a bit of an oddity that latency is so subordinate to bandwidth in discussions of speeds and feeds.

But make no mistake—latency is a critical performance element with implications that extend upwards through the software stack. To scale effectively, system architectures whose memory designs are highly non-uniform require workarounds to compensate for the lower-end of their response ranges, including hardware features such as caches, and OS features such as processor affinity. And average memory latencies that are consistently long can become the primary system bottleneck across applications that make frequent changes to data—a typical characteristic of transactional environments.

12. See Illuminata Note, "Server Technologies Perspectives #1", May 2002.

The situation is no different outside the box. Some types of information exchange—such as the serving of static Web content—can tolerate even long public network latencies. And remote data-synchronization can function with one-way latencies of up to about 1 millisecond—equivalent to about 125 miles—as an upper limit. However, clusters and multi-tier applications benefit from the speediest and lowest overhead communications—especially if they are updating data frequently as opposed to just reading. This requirement has spawned a plethora of high-speed networking links that are essential parts of the biggest, fastest clusters.

That's not to say that bandwidth, clock speed, and bus width are not important. Of course they are. Bandwidth alone ensures that the bits have wide open lanes down which to fly. But latency is at least as important. So enjoy the impressive bus and bandwidth numbers your vendors show you, and use them as part of your decision-making. But be sure to also ask about and understand the latencies involved—within a system block, among memory blocks, and between systems. Without knowing how long your processors have to wait while the data is being assembled, you'll never know how fast they'll actually work.


CACHE 1-80ns
MEMORY 112-500ns
HIGH PERFORMANCE CLUSTERS 3-40µs
ETHERNET 60-200µs
TN