learned how to optimize their hardware to perform well on the standard benchmarks. An application that behaves substantially differently from a benchmark test may not show the same level of performance as advertised by the manufacturer.

The microprocessor's memory interface is a critical contributor to its performance. Whether a small 8-bit microprocessor or a 64-bit behemoth, the speed with which instructions can be fetched and data can be loaded and stored affects the execution time of an application. The necessary bandwidth of a memory interface is relative and is proportional to the sum of the instruction and data bandwidths of an application. From the instruction perspective, it is clear that the microprocessor needs to keep itself busy with a steady stream of instructions. Data bandwidth, however, is very much a function of the application. Some applications may perform frequent load/store operations, whereas others may operate more on data retained within the microprocessor's register set. To the extent that load/store operations detract from the microprocessor's ability to fetch and execute new instructions, they will reduce overall throughput.

Clock frequency becomes a defining attribute of a microprocessor once its instruction set, execution capabilities, and memory interface are understood from a performance perspective. Without these supporting attributes, clock speed alone does not define the capabilities of a microprocessor. A 500-MHz single-issue, or nonsuperscalar, microprocessor could be easily outperformed by a 200-MHz four-issue superscalar design. Additionally, there may be multiple relevant clocks to consider in a complex microprocessor. Microprocessors whose internal processing cores are decoupled from the external memory bus by an integrated cache are often specified with at least two clocks: the core clock and the bus interface clock. It is necessary to understand the effect of both clocks on the processing core's throughput. A fast core can be potentially starved for instructions and data by a slow interface. Once a microprocessor's resources have been quantified, clock frequency becomes a multiplier to determine how many useful operations per second can be expected. Metrics such as instructions per second (IPS) or floating-point operations per second (FLOPS) are specified by multiplying the average number of instructions executed per cycle by how many cycles occur each second. Whereas high-end microprocessors were once measured in MIPS and MFLOPS, GIPS and GFLOPS performance levels are now attainable.

As already mentioned, memory bandwidth and, consequently, memory architecture hold key roles in determining overall system performance. Memory system architecture encompasses all memory external to the microprocessor's core, including any integrated caches that it may contain. When dealing with an older-style microprocessor with a memory interface that does not stress current memory technologies, memory architecture may not be subject to much variability and may not be a bottleneck at all. It is not hard to find flash, EPROM, and SRAM devices today with access times of 50 ns and under. A moderately sized memory array constructed from these components could provide an embedded microprocessor with full-speed random access as long as the memory transaction rate is 20 MHz or less. Many 8-, 16-, and even some 32-bit embedded microprocessors can fit comfortably within this performance window. As such, computers based on these devices can have simple memory architectures without suffering performance degradation.

Memory architecture starts to get more complicated when higher levels of performance are desired. Once the microprocessor's program and data fetch latency becomes faster than main memory's random access latency, caching and bandwidth improvement techniques become critical to sustaining system throughput. Random access latency is the main concern. A large memory array can be made to deliver adequate bandwidth given a sufficient width. As a result of the limited operating frequency of SDRAM devices, high-end workstation computers have been known to connect multiple memory chips in parallel to create 256-bit and even 512-bit wide interfaces. Using 512 Mb DDR SDRAMs, each organized as $32M \times 16$ and running at 167 MHz, 16 devices in parallel would yield a 1-GB memory array with a burst bandwidth of 167 MHz $\times$ 2 words/hertz $\times$ 256 bits/word = 85.5 Gbps! This is a lot of bandwidth, but relative to a microprocessor core that operates at 1 GHz or more with a 32- or 64-bit data path, such a seemingly powerful memory array may just barely be able to keep up.

While bandwidth can be increased by widening the interface, random access latency does not go away. Therefore, there is more to a memory array than its raw size. The bandwidth of the array, which is the product of its interface frequency and width, and its latency are important metrics in understanding the impact of cache misses, especially when dealing with applications that exhibit poor locality.

Caching reduces the negative effect of high random access latencies on a microprocessor's throughput. However, caches and wide arrays cannot completely balance the inequality between the bandwidth and latency that the microprocessor demands and that which is provided by SDRAM technology. Cache size, type, and latency and main memory bandwidth are therefore important metrics that contribute to overall system performance. An application's memory characteristics determine how costly a memory architecture is necessary to maintain adequate performance. Applications that operate on smaller sets of data with higher degrees of locality will be less reliant on a large cache and fast memory array, because they will have fewer cache misses. Those applications with opposite memory characteristics will increase the memory architecture's effect on the computer's overall performance. In fact, by the nature of the application being run, caching effects can become more significant than the microprocessor's core clock frequency. In some situations, a 500-MHz microprocessor with a 2-MB cache can outperform a 1-GHz microprocessor with a 256-kB cache. It is important to understand these considerations because money may be better spent on either a faster microprocessor or a larger cache according to the needs of the intended applications.

I/O performance affects system throughput in two ways: the latency of executing transactions and the degree to which such execution blocks the microprocessor from performing other work. In a computer in which the microprocessor operates with a substantially higher bandwidth than individual I/O interfaces, it is desirable to decouple the microprocessor from the slower interface as much as possible. Most I/O controllers provide a natural degree of decoupling. A typical UART, for example, absorbs one or more bytes in rapid succession from a microprocessor and then transmits them at a slower serial rate. Likewise, the UART assembles one or more whole incoming bytes that the microprocessor can read at an instantaneous bandwidth much higher than the serial rate. Network and disk adapters often contain buffers of several kilobytes that can be rapidly filled or drained by the microprocessor. The microprocessor can then continue with program execution while the adapter logic handles the data at whatever lower bandwidth is inherent to the physical interface.

Inherent decoupling provided by an I/O controller is sufficient for many applications. When dealing with very I/O-intensive applications, such as a large server, multiple I/O controllers may interact with each other and memory simultaneously in a multimaster bus configuration. In such a context, the microprocessor sets up block data transfers by programming multiple I/O and DMA controllers and then resumes work processing other tasks. Each I/O and DMA controller is a potential bus master that can arbitrate for access to the memory system and the I/O bus (if there is a separate I/O bus). As the number of simultaneous bus masters increases, contention can develop, which may cause performance degradation resulting from excessive waiting time by each potential bus master. This contention can be reduced by modifying the I/O bus architecture. A first step is to decouple the I/O bus from the memory bus into one or more segments, enabling data transfers within a given I/O segment to proceed without conflicting with a memory transfer or one contained within other I/O segments. PCI is an example of such a solution. At a more advanced level, the I/O system can be turned into a switched network in which individual I/O controllers or small segments of I/O controllers are connected to a dedicated port on an I/O switch that enables each port to communicate with any other port simultaneously insofar as multiple ports do not conflict for access to the same port. This is a fairly expensive solution that is implemented in high-end servers for which I/O performance is a key contributor to overall system throughput.