1 History

Although multiprocessor designs have been around for decades, only recently have semiconductor vendors turned to multicore processors as a solution for the so-called Moore's Gap[1] effect. Simply put, modern multicore CPU design enables CPU performance to track Moore's law with attractive CPU power consumption properties. This trend is causing many system designers that previously would use single core designs to design multicore processors into hardware in order to meet their performance and/or power consumption requirements. This has a huge impact on the software architecture for those systems, which must now consider inter-processor communication (IPC) issues to pass data between the cores.

The Multicore Association's Communication API (MCAPI) traces its heritage to communications APIs such as MPI (Message Passing Interface) and sockets. Both MPI and sockets were developed primarily with inter-computer communication in mind, while MCAPI is targeted primarily towards inter-core communication in a multicore chip. Accordingly, a principal design goal of MCAPI was to serve as a low-latency interface leveraging efficient on-chip interconnect in a multicore chip. MCAPI is thus a light-weight API whose communications latencies and memory footprint are expected to be significantly lower than that of MPI or sockets. However, because of the more limited scope of multicore communications and its goal of low latency, MCAPI is less flexible than MPI or sockets.

MCAPI can also be distinguished from POSIX (portable Unix) threads, pthreads. MCAPI is a communications API for messaging and streaming, while pthreads is an API for parallelizing or threading applications. Pthreads offers a programming model that relies on shared memory for communication and locks for synchronization. In multicore processors with caches, efficient pthreads implementations require support for cache coherence. Because they lack modularity, lock-based parallel programs are hard to compose together. MCAPI, with parallelism specified using standard means such as processes or threads, offers an alternative and complementary parallel programming approach that is modular and composable. Because it works both in multicore processors with only private memory or with shared memory, MCAPI allows simpler embedded multicore implementations.

IPC for device software utilizing multicore processors have some new requirements beyond those imposed on multiprocessor designs that utilized discrete processors. Within a multicore chip, the shorter distance for electrical signals enables data to be passed much more quickly, energy-efficiently, and reliably than between discrete processors on a board or over a backplane. In addition, the bandwidth available on chip is orders of magnitude greater. The low latency and high bandwidth offer the potential of ASIC-like (application specific integrated circuit) high-speed communication capabilities between cores. This means that the focus of IPC for multicore processors must prioritize performance in terms of both latency and throughput. Also, some multicore processors will have many cores that rely on chip-internal memory or cache to execute code (to avoid the von Neumann bottleneck which multicore processors can suffer due to the multiple cores contending for external memory accesses). Such cores that execute from chip-internal memory will require an IPC implementation that has a small memory footprint. For these reasons, the two primary goals for a multicore IPC API design are that the implementation of it can achieve extremely high performance and low memory footprint.

In order for a multicore IPC implementation to achieve high performance and tiny footprint, these two goals need to be immutable when in conflict with other desirable IPC API attributes such as physical or logical connectivity topology flexibility, ease of use, OS integration or compatibility with existing IPC programming models or APIs.

The Multicore Association's Communication API (MCAPI) endeavors to meet these goals. Before MCAPI, no IPC API has been designed with these two immutable goals targeting multicore processors.

[1] Moore's Law has hit a wall where increased transistors per sequential CPU chip is no longer resulting in a linear increase in performance of that CPU chip.