While much remains to be seen in the ongoing shift towards massively many-core computing, one perhaps pivotal point of focus lies at the intersection of low-level implementation languages (e.g., C/C++) and runtime operating system support. Here, we briefly discuss one issue in this area: concurrent programming models and what may need to happen to languages and operating systems to effectively support audio programming on many-core systems – such that existing and future audio softwares can take advantage of these systems.

The up-to-now dominant model of using preemptive threading to distribute computation among multiple cores is rapidly becoming less solvent for time-critical, high-throughput systems, such as real-time audio engines (the model was never ideal from a programming point of view, and many-core computing further accentuates its shortcomings). Due to their nondeterministic nature, preemptive threads are inherently difficult to reason about, debug, and maintain. Also, the preemptive nature and computational overhead of threads make it hard to synchronize parallel computations in predictable and time-sensitive ways. Furthermore, it is often unclear (to programmers) whether "parallelization" via threads are beneficial to a particular situation – the decision depends on many complex factors such as the nature of the algorithm, timing and synchronization needs, as well as the number of available cores – too few threads will fail to take advantage, too many threads can introduce significant overhead due to unnecessary thread management, context-switching, and scheduling. This can make it extremely challenging to re-architect existing, sequential systems for many-core.

Adopting a developer-centric point of view, it seems imperative that the computing community should produce an agreed-upon, generalized lower-level concurrent programming paradigm (e.g., language extensions at the C/C++ level, plus integrated OS support) that can flexibly specify synchronous, multi-core friendly programs. Such a programming model should allow concurrent code to operate and synchronize with minimal overhead, while leveraging underlying operating system support to abstract the underlying core configuration/availability.

One potentially promising approach lies in Apple’s upcoming 10.6 (Snow Leopard) release of the OS X operating system [1], which is to offer both the addition of “blocks” (parallelizable segments of code) as an underlying programming language feature, along with a centralized mechanism in the operating system kernel, called Grand Central Dispatch (GCD). In short, GCD interprets the blocks specified by the programmer in real-time, and allocates them across available cores (and potentially the GPU). This approach eliminates the need for programmers to account for exactly how many cores are present, while reducing thread-allocation overheads by centralizing blocks and dynamically deciding when to spawn new parallel entities as beneficial. Also importantly, this approach seems to be potentially scalable. Of course, it remains to be seen how this approach actually works, in particular with a wide array of audio systems. At the same time, it is encouraging that a major OS platform has taken concrete, developer-centric steps to try and address the many-core problem.

Hopefully other platforms (e.g., Linux and Windows) will follow suit, and the community can produce a portable, industry-wide many-core programming standard. As designers and implementors of real-time computer music software, it is crucial for us to experiment early and freely with technology as they become available, and to champion audio-friendly solutions at every level.

1. REFERENCES