The recent history of multiprocessing in computer music has presented us with an interesting reversal. In the early 1980s, crunching audio samples seemed a much more computationally intensive task than musical decision making, and research at MIT, Stanford, and IRCAM focused on figuring how to get a few hundred MIPS out of a machine for doing audio synthesis and processing. The only possible way forward was multiprocessing, and the Samson box, the 4X and the ISPW were responses to that. The ISPW, which came fully on line in about 1991, had up to six processors. Carefully optimized applications could attain some 300 MIPS of throughput.

Since that time, the processors themselves have radically sped up, to the point that only a very small number of computer musicians complain that they can't now realize their synthesis and processing aims with a uniprocessor. On the other hand, musical decision making, which frequently makes heavy use of learning algorithms, combinatorial optimization, and/or data-intensive searching, emerges as a bottomless pit of demand for more MIPS, storage, and communications capacity.

It turns out that audio crunching isn't too hard to parallelize out to small numbers of processors (fewer than 10, say). The same techniques we used in the 80s and early 90s remain valid today. Audio "signals" and "control messages" (the dominant abstractions of the moment for describing audio computation) can simply be packaged and routed among processors for computation via a static network of unit generators each assignable (either manually or automatically) to an available processor. For example, the modern DAW station does this, in that plug-ins seem to be where the MIPS are truly needed; and because of the way the plug-in paradigm is designed, individual instances are easy to farm out to available processors.

My own project of the last ten years, Pd, is an explicit acknowledgement that we no longer truly need multiprocessors to do computer music; my own belief is that the important thing now is to lower the cost of entry, and low-cost computers are still mostly uniprocessors. (Nonetheless, because of a musical project I'm now working on, I just recently ended up having to re-implement the old ISPW multiprocessing approach, which is now done in an object I call "pd~". This sprouts a pd sub-process -- from within Pd or Max, take your pick -- connected by audio and control message pathways to the calling program. I anticipate that it will be most useful for solving interoperability problems between Max and Pd, although in principle one could use it to load a modern several-processor machine efficiently.)

But the real promise of highly parallelized computation (that is, computation requiring more than two to four processors) is probably in the decision-making tasks that are used in computer-aided composition and improvisation - that is, the learning, optimization, and search problems. I don't know how to do it, but I think it would be interesting to try to make multiprocessor implementations of engines for solving these classes of problems.

I don't think that the way we program computers today will ever change. The bulk of applications to which we now turn uniprocessing machines (or dual processors with one processor idling, as seems to be the norm today) will stay useful, and nobody will figure out how to parallelize them. Instead of attempting that, we should look for new styles of computing, more adaptable to multiprocessors that could complement or augment what we do today.

Computer music seems to be an excellent source of interesting computing problems, and we should always remind our friends in computer science departments and in industry of this. In the same way that real-time techniques were invented and exercised by computer music researchers in the 1980s, today we can offer problem spaces and innovative approaches to problems outside the reach of conventional computers. Highly parallelizable algorithms seem likely to be one such problem area, rich in challenges and in promise.