PANEL SESSION

Reinventing Audio and Music Computation for Many-Core Processors

Moderated by *David Wessel*
CNMAT, University of California Berkeley
wessel@cnmat.berkeley.edu

Participants
Roger Dannenberg, Yann Orlarey, Miller Puckette, Peter Van Roy, Ge Wang

ABSTRACT

This text serves as an introduction to a problem facing the field of computer music. Multi-core and many-core personal and mobile computers will play a major role in the future of audio and music computation, but it is far from clear how that future will evolve. The hope is that this panel will shed some insight concerning the path to our parallel programming future.

1. INTRODUCTION

We are on the threshold of a sea change in audio and music computing prompted by the computer industry's move towards multi- and many-core computer architectures. Eight-core desktop machines are already widely available and the number of cores is expected to double every two years. The computer music community is now seriously challenged by the problem of writing software for such multi- and many-core processors.

This shift to parallel architectures requires new programming practices and their development appears to be very difficult. Unless we reinvent our software for the many-core future, the performance of our computer-based instruments will come to a standstill as clock rates for mono-processors will cease to increase.

The single core machine with its sequential programming practices was providing seemingly unending increases in performance with high and higher clock rates. As late as 2005 the Semiconductor Research Council predicted continued clock rate increases. A look at Figure 1 suggests that we should be at 11 GHz in 2008. It appeared that we could continue to stay with the sequential programming paradigm and ignore parallelism as a way to improved performance.

The continued speed-ups to single-core architectures came to an end when we hit a limit in the amount of power that could be dissipated by the chip. This power wall and the cramp it puts on clock rates has only recently received serious attention from the general computing world. Witness the fact that only a small percentage of applications can take advantage of multi-core much less many-core systems. In the computer music community our software stable – csound, Max/MSP, PD, SuperCollider, FAUST, CHUCK, STK, Aura, and other musical-domain-specific languages, and the vast majority of musical applications have yet to fully deliver multi-core power. Exploiting parallelism, though it is a concern and attempts are underway, has yet to provide computer musicians with the increased computational power promised by multi-core processors.

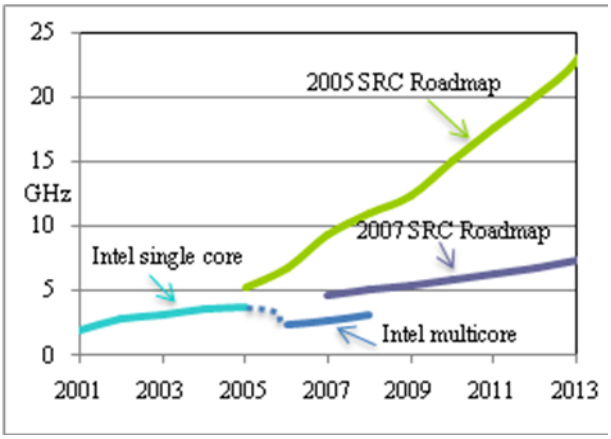

Figure 1. Microprocessor clock rates of Intel products vs. projections by SRC in 2005 and then in 2007[2]

2. OUR MULTI-PROCESSOR PAST AND PRESENT

The computer music community does have a record of past and current successes with parallel architectures. The seventies and the eighties saw the development of a number of systems involving a general-purpose host processor with custom accelerators typically composed of multiple signal processors. Examples include DiGuigno's 4 series sound synthesis engines culminating in the 4x, Freed's Resoneight, multiple DigiDesign Motorola 56000-based AudioMedia Nubus Cards, the IRCAM Signal Processing Workstation, among others. To this day the Kyma System from Symbolic Sound and DigiDesign's DSP-Farm remain highly productive.

Will our future music machines be homogeneous wherein the cores are identical or will they mirror our heterogeneous past when general purpose processors were equipped with accelerators? Will the evolving GPU take on the role of the signal processor?

3. VIRTUALIZATION

One of the advantages of special hardware is that computational resources can be dedicated to the music task. This can make real-time processes more predictable. The general trend in software, however, is to virtualize resources: we allocate threads rather than processors, and virtual address spaces rather than physical memory. Manycore computers will open up many architectural decisions for music software design.

Is processor virtualization still a good thing? What approaches seem most viable: (1) dedicate cores to tasks, even if it means low efficiency, to maximize simplicity, (2) partition tasks carefully and map them to various cores to optimize performance, or (3) write software in terms of many threads, and rely on languages and operating systems to map threads to cores to maximize portability.

4. VOICES, STREAMS, CHANNELS, TRACKS, AND LINES

Most music consists of coordinated sources operating in parallel. The traditional western orchestral model involves a number of individual musicians, a score for each to follow, and a coordinator in the form of a conductor. Our early languages, the so-called Music N languages, adopted this convention – a score and a virtual orchestra. With the advent of real time, the computer music performer began to function as a kind of conductor who regulated the tempo, adjusted the dynamics, and fine-tuned the coordination among the voices. Even when the score is abandoned by more interactive and improvisatory approaches to computer music performance, the notions of separable voices, streams, channels, and tracks remain operable and ripe for parallelization. Given music's highly exposed parallelism it seems natural to assign voices to different cores. In fact, if we think back to the heyday of the patchable analog synthesizer voices were added by adding more hardware resources. In these analog synths, sound generation was not time-shared as it is our current mono-processor approach. There was no operating system to do time-slice management of concurrency, just "bare metal" in the form of multiple analog hardware resources. As many-core processors evolve we will likely revisit this "bare metal" approach of dedicating separate processors exclusively to voices.

Version 5 of Max/MSP takes a step in this direction. The poly~ abstraction for managing multiple copies of a process specified by a patch now uses multi-threading and allocates the process copies to separate cores. This mechanism allows the Max/MSP programmer to take advantage of multi-core processors with significant gains in performance. However, the performance gain is not a simple multiple of the number of cores but quite mysteriously depends on a number of features of the process itself and overhead incurred by the multi-threading mechanism. One would hope that given the exposed parallelism that predicting performance would be simpler. After a bit of experimentation with parallelism in Max/MSP it becomes clear that we need better tools to evaluate performance and aid the programmer in locating bottlenecks. This will require computer architectures with more program counters and timing traces.

5. DESIGN PATTERNS AND COMPUTATIONAL MOTIFS

Our community is extremely diverse in terms of programming skills. Many successful computer music composers and performers have limited technical education but have learned enough programming to be musically creative with a musical-domain-specific language. There are also highly skilled professional programmers in the field capable of dealing with difficult race condition bugs and writing highly optimized code. Writing efficient and correct parallel code is hard and more bug-prone than sequential programming so it would seem unreasonable to ask the less skilled programmers to deal with the difficulties of parallelism. One hope is that the expert programmers will develop frameworks and libraries of the key and consumptive components that take advantage of parallel architectures. Such libraries would go a long way towards the goal of providing a software environment that will automatically map programs to multiple cores.

The approach we are pursuing at Berkeley [2] examines musical applications and identifies the key computational components. These design patterns are things like *map-reduce* and *pipe-and-filter* while the computational motifs are things like *spectral processing*, *unstructured grids* as used in finite-elements models, *dense* and *sparse linear algebra*, and others. Parallel libraries serving our key design patterns and computational motifs would provide for more productive programming by all.

The move to parallelism is rich with opportunity and it is critical that our musical and audio applications drive the innovations. Meeting our need for the responsiveness of a fine musical instrument will enrich the user experience throughout the entire landscape of computing.

6. REFERENCES

[1] Semiconductor Research Council, International Technology Roadmap for Semiconductors, Executive Summary, 2005 and 2007, /public.itrs.net/

[2] K.Asanovic, R.Bodik, J.Demmel, T.Keaveny, K.Keutzer, J.Kubiatowicz, E. Lee, N. Morgan, G.Necula, D. Patterson, K.Sen, J.Wawrzynek, D. Wessel and K. Yelick. "The Parallel Computing Laboratory at U.C. Berkeley: A Research Agenda Based on the Berkeley View," *Tech. Re.UCB/EECS-2008-23*, EECS Department, University of Clifornia, Berkeley, March 21, 2008.