The Architecture of the IRCAM Musical Workstation
Author(s): Eric Lindemann, François Dechelle, Bennett Smith and Michel Starkier
Source: *Computer Music Journal,* Vol. 15, No. 3 (Autumn, 1991), pp. 41-49
Published by: The MIT Press
Stable URL: https://www.jstor.org/stable/3680764
Accessed: 03-04-2020 02:26 UTC

Eric Lindemann, François Dechelle, Bennett Smith, and Michel Starkier
IRCAM: Institut de Recherche et Coordination Acoustique/Musique
31, rue Saint-Merri
F-75004 Paris, France
elind@ircam.fr

The Architecture of the IRCAM Musical Workstation

The IRCAM Musical Workstation (IMW) is designed to facilitate experimentation with real-time signal processing and event processing. The main focus is on interactive musical composition and performance algorithms. The hardware architecture of the IMW is based around a general-purpose multiprocessor providing sufficient number-crunching power for real-time musical signal processing and event processing.

A real-time operating system has been written for this multiprocessor as well as a toolbox that provides support for distributed real-time signal processing and event processing applications. The toolbox also defines an object system to support interactive CAD-like applications. Support for musical applications is in the form of MIDI, digital audio I/O, and music-oriented event scheduling. The multiprocessor consists of a number of processor boards, each with two Intel i860 processors (Intel 1989), that plug into a host computer, the NeXT machine. The host is used for its graphic interface and file system. The i860 processors handle all real-time event processing and signal processing.

A number of distributed applications have been written for the system. User interfaces run on the host and communicate with tasks on the multiprocessor. These include *Animal* (Lindemann and De Cecco 1991), an object-oriented data definition and manipulation environment; *MAX* (Puckette 1991), a graphical programming language; *IMW Signal Editor* (Eckel 1990), a tool for manipulating sampled signals in time and frequency domains; and the *IMW Universal Recorder* (Smith 1990), a multitrack real-time data recorder for spooling event and signal streams to disk. This article provides a general overview of the IMW system with a special focus on the hardware architecture.

Computer Music Journal, Vol. 15, No. 3, Fall 1991,

Musical Signal Processing Architectures

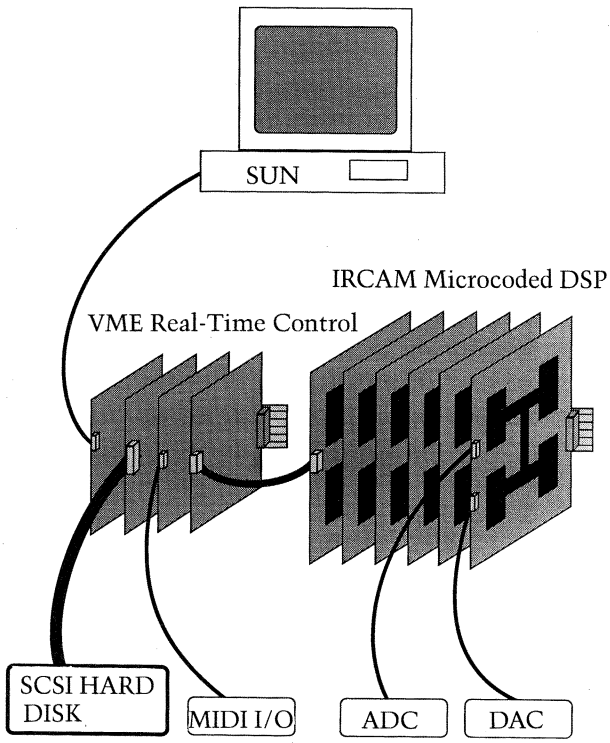
Previous systems for real-time musical signal processing have been divided into several subsystems: a host computer (e.g., Apple Macintosh, IBM PC, or UNIX workstation), serving as a general development environment, graphical user interface, and file server; a real-time control system used for general event handling (e.g., MIDI controllers); and a real-time signal processing system. The control section typically includes a number of microprocessors (e.g., Motorola MC680X0, or Intel 80X86) running real-time kernels. In some low-end systems the task of real-time event management is carried out by the host computer. This tends to work reasonably well with PCs and much less well with workstations. The signal processor can take a number of forms: custom nonprogrammable designs highly optimized for particular algorithms (e.g., Fast Fourier transforms [FFTs] or table lookup oscillators); moderately programmable designs optimized for a certain class of algorithms; or more fully programmable designs, either custom architectures or ones based on off-the-shelf digital signal processing (DSP) processors such as Motorola DSP56001, Motorola DSP96000, AT&T DSP32, or the Texas Instruments TMS32CXX.

Examples of programmable systems that have been used for musical production are the IRCAM *4X* machine (DiGuigno and Gerzso 1986), the WaveFrame *AudioFrame* (Lindemann 1987), and the LucasFilm *SoundDroid* (Moorer 1982).

IRCAM 4X

The system built around the 4X processor shown in Fig. 1 is still in use at IRCAM. It is based on the three-level architecture mentioned above with host, control, and signal processing systems. The signal

Fig. 1. IRCAM 4X system architecture.



processor is a custom discrete design. It is moderately programmable; one can write programs as long as they have no jump instructions. It is particularly optimized for table lookup oscillators, although other algorithms can be implemented with some effort. The 4X is controlled by an embedded VME-based Motorola MC68000 processor running a proprietary real-time kernel. Programs running on this kernel are cross-developed using a Sun Microsystems workstation. Programs running on the 4X signal processor are also cross-developed on the Sun using a compiler developed at IRCAM specialized for signal processing. The Sun workstation is not required in a performance situation.

AudioFrame

The AudioFrame system from WaveFrame is geared less toward research and experimentation, and more toward commercial audio production. It has a four-level architecture with host, first-level event control, second-level event control, and signal processing components. The signal processing system of the AudioFrame is heterogeneous, with specialized modules for sampling synthesis and disk recording, and more general-purpose modules based on multiple Motorola DSP56001 processors. The latter has special-purpose hardware to optimize address generation and memory fetches in reverberation and block-oriented algorithms.

SoundDroid

The SoundDroid system has a similar multilevel architecture. The signal processing section is a large discrete design using emmiter coupled logic (ECL) components. It is fairly general-purpose with parallel functional units for integer multiplication, addition, and address generation. A sophisticated time-tagged parameter update mechanism allows jamming high-priority updates to the front of a normally first-in, first-out (FIFO) update queue. The signal processor is controlled by embedded MC68000 processors, which in turn communicate with a UNIX host.

Smaller Systems

A number of smaller PC-based systems aimed at audio production have appeared in recent years. The most popular are based on one or more Motorola DSP56001 processors. While useful in certain contexts, they tend to lack sufficient number-crunching power for real-time musical performance. In addition, the architecture of commercial DSP processors tends to be optimized for FFTs, nonpolyphase FIRfilters, and standard IIR filter structures. They tend to be very poorly optimized for certain standard musical signal processing tasks such as table-lookup oscillators, sample synthesizer oscillators, variable delay lines, or frequency modulation unit generators. The main problem is lack of support for the fractional addressing necessary in standard phase increment schemes. All of the above DSP designs have integer-based signal processing units. Floating-point units fast enough, and inexpensive enough, to use in musical systems are a fairly recent phenomenon.

Disadvantages of Three+ Level Architectures

As vehicles for experimentation, the multilevel architectures described above can quickly become unwieldy. There is generally a separate development environment for each type of processor—host, control, and signal processing. An application program must generally be written for each processor, and these programs must be made to communicate reliably. The signal processing routines are either microcoded or written in some extended assembler with parallel instruction capabilities. In addition, the signal processing routines may have to be written for integer processors, complicating the task of implementing complex algorithms.

This process tends to be discouraging to researchers. Most research at IRCAM has, in fact, been based on simulation using standard non-real-time systems—workstations and DEC VAXen. Only certain well-suited algorithms have had real-time implementations. These types of systems, with several flavors of processors, also tend to be costly and poorly adapted to time-sharing. As a result, they are scarce resources, ill-suited to casual experimentation.

Preparing for a New Generation

Experience with the types of systems described above led to a statement of goals for a new real-time musical signal processing architecture:

1. Reduction of the number of types of processors and development environments in the system.
2. Minimization of specialized low-level programming requirements.
3. Use of floating-point numbers throughout.
4. Development of a rich set of rapid prototyping tools including graphical programming languages and data visualization environments.
5. Cost reduction.

We began studying hardware architectures for the IMW at the beginning of 1989, keeping these goals in mind. To reduce the number of processors the goal was to unify the roles of embedded control processor and signal processor. At the time, two strategies seemed possible—to use an existing general-purpose reduced instruction set computer (RISC) processor with fast floating-point calculation (e.g., Sun SPARC, MIPS R2000, Motorola MC88000, or Advanced Micro Devices 29000), or to use a floating-point DSP (e.g., Motorola DSP 96000, TI TMS320C30 or ATT DSP32).

The RISC chips had the advantage of sophisticated compiler technology, caches and memory management units, general-purpose integer units with barrel shifters for efficient address manipulation, and good high-level development environments. Their signal processing performance appeared to be about one-fourth that of the signal processing chips, however, generally due to lack of parallelism.

Fortunately, 1989 saw the introduction of a general-purpose extended RISC processor, the Intel i860, capable of very high-speed floating-point number crunching. Its 25-nsec cycle time, its ability to perform a floating-point add, a floating-point multiply, and an integer operation every cycle, and its integrated caches and MMU made it the ideal choice for the IMW. Not only would we have the benefits of a general-purpose RISC architecture processor, but the i860 appeared on paper to considerably outperform the signal processing chips we studied.

The i860 made the unification of the real-time control and signal processing environments possible, turning a three-level system into a two-level system—with only host and real-time systems—while at the same time holding out the promise of a high-level environment for developing signal processing algorithms.

Why not use a one-level system, with a general-purpose multiprocessor host with good real-time response and lots of number crunching power? This is may be reasonable in the next few years. UNIX development is moving more in the real-time direction, with small kernels (e.g., Mach or Chorus), and more attention being payed to guaranteed response time. Much of this has been motivated more by the requirements of efficient transaction processing than by musical or multimedia requirements.

As superscalar architectures succeed the current generation of RISC processors, and parallelizing and pipelining compiler technology becomes more common, hardware architectures capable of supporting a single-level musical signal processing system will become quite realizable.

Hardware Architecture of the IMW

In the next few sections, we describe the hardware architecture chosen for the IMW workstation.

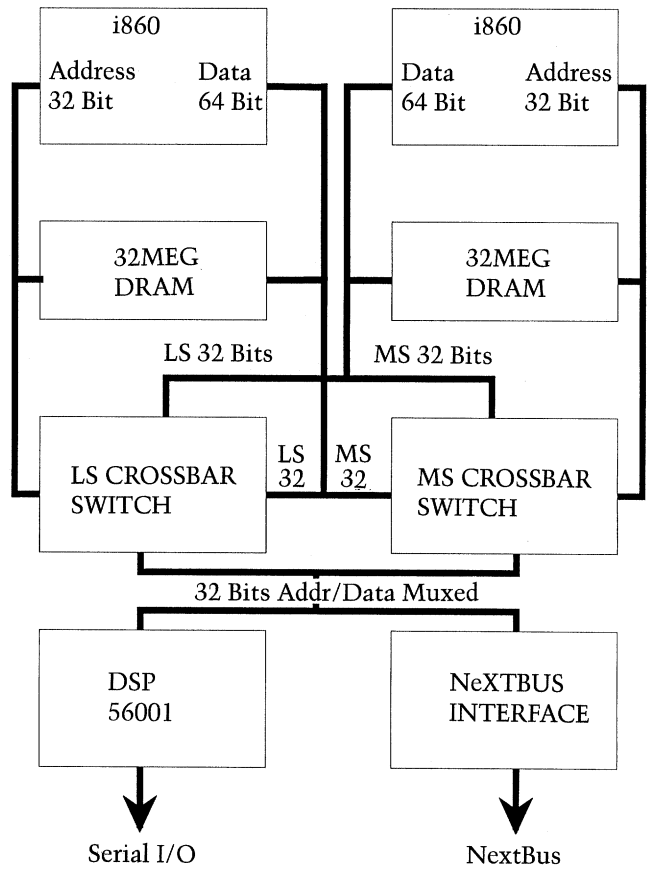
Choice of Host Computer

The hardware architecture of the IMW consists of an Intel i860-based multiprocessor system interfaced to a NeXT host computer. Because almost all of the real-time functionality is provided by the i860s, the real-time performance of the host is not critical. We would like to be able to use the host file system's mass storage capability for real-time recording of digital audio and event streams to disk. This implies not only that the host must have a relatively efficient disk direct memory access (DMA) implementation, but also that the interface between host and i860 systems must be a high-speed (i.e., memory-mapped) interface, as opposed to an Ethernet or token ring interface.

As a minimum, then, the host must provide access to its bus so that a memory-mapped interface can be designed. Ideally, the host will provide plug-in board slots of sufficient number and size so that the i860 system can be implemented in a simple manner without resorting to external chassis that increase the cost and complexity of the system.

The choice of NeXT as host computer was based largely on these considerations. NeXT provides a high-performance implementation of the MIT NuBus that is sufficiently fast to allow direct communication between i860 boards without requiring the design of additional high-speed bus interconnects. Communication between the i860s and the NeXT CPU is also over this bus.

Fig. 2. Block diagram of the IMW calculation engine.



The IMW Calculation Engine

The calculation engine is implemented as a plug-in coprocessor board for the NeXT computer. A block diagram of the board is shown in Fig. 2. Each board has two i860 processors with up to 32 MBytes of local dynamic RAM. The processors communicate with each other by reading or writing each other's RAM. Interconnection between processors on the same board is through a crossbar switch. The crossbar switch is arranged as two blocks, each with four 32-bit ports.

Each block receives the full address of one of the i860s on one of its four ports. One block receives the least-significant 32 bits of each of the two i860s 64-bit data buses on two of its four ports. The other crossbar switch block receives the most significant 32 bits of each of the i860 data buses. When an i860 reads or writes the local RAM of its neighbor pro-

cessor on the same board, 64 bits of data are passed directly through the crossbar switch blocks and the address bus of the "master" processor is passed out through the fourth port of the first crossbar switch block, into the fourth port of the second block, and out through the address port of the "slave" processor. This connection provides no-wait-state 50-nsec burst transfers (160 MBytes/sec) between neighboring processors. After arbitration for control of the crossbar switch, and for control of the neighbor's bus, a "master" processor will keep control until a rupture is signaled. A rupture is generated by a gap in the burst access of the master processor, a demand for access from the MBUS, or a DRAM refresh request. The refresh rupture generator puts a natural and desirable cap on maximum uninterrupted bus mastery.

Since the data buses of i860 processors are tied directly to the crossbar switch, a potential deadlock occurs when the two processors attempt to write each other's memory at the smae instant. This and similar deadlocks involving MBUS access are resolved by "preempting" an i860 write cycle and caching its write data inside the crossbar. The bus connecting the fourth port of the crossbar switch blocks also serves as a 32-bit address/data multiplexed bus (MBUS) for I/O and off-board communications. This bus is connected to a Motorola DSP56001 processor, which is used as an I/O processor. It supports four bidirectional channels of serial digital audio at a 44.1-kHz sampling rate, and an RS-422 serial port for MIDI and other controller interfaces.

The DSP56001 also serves as DMA controller for burst transfers between itself and the local memory of i860 processors, and between the memory of i860 processors on different boards. To perform a DMA read operation from i860 memory, the DSP56001 arbitrates for the MBUS and then latches a 32-bit address into the crossbar switch. The 8 least significant bits of the address are taken from the DSP56001 address bus and the 24 most significant bits are taken from the DSP56001 24-bit data bus. The selected i860 is put in hold and two 64-bit words are read from its memory and cached in the crossbar switch. The DSP56001 is notified that the data are ready. If the DMA transfer is in the direction of the NeXT bus, then the DSP56001 initiates a burst write operation of four 32-bit words to the NextBus Interface Chip (NBIC). It latches the destination address into the NBIC. The cached data are then read automatically from the crossbar by the MBUS state machine controller. The data do not pass through the DSP56001. During the entire process the i860 is interrupted for approximately eight cycles, or 200 nsec. For write operations in the direction of i860 memory a similar process takes place in reverse. If a DMA operation has the DSP56001 itself as target, then, after latching the address, the DSP56001 reads or writes the cached crossbar data directly. It discards the bottom 8 bits of the 32-bit MBUS since its data bus is only 24 bits wide.

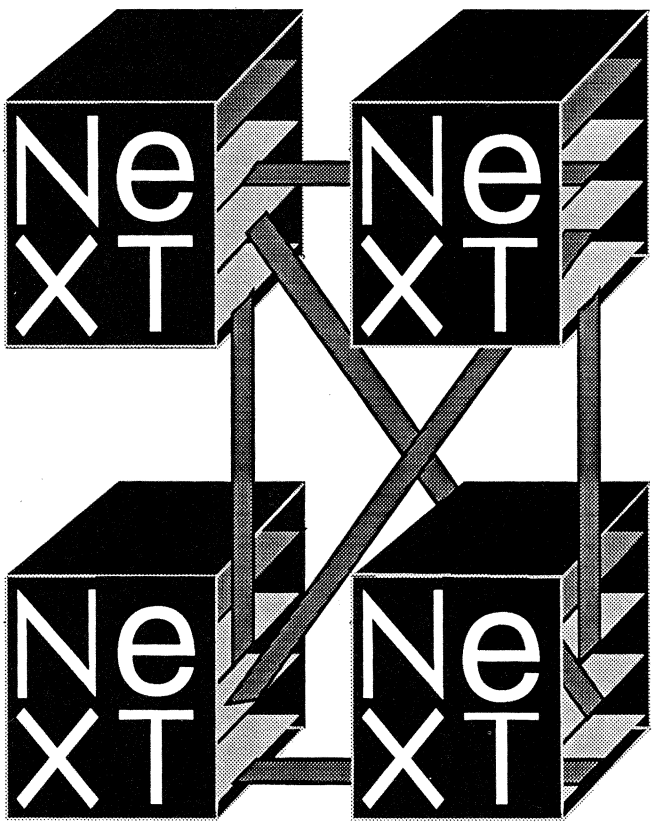
Multiple Boards

The four-slot NextBus permits three dual-processor i860 boards to be plugged into the same machine alongside the host CPU board, which occupies one slot. The NextBus is an extended version of the MIT/Texas Instruments NuBus with a double-speed burst mode that permits 40-nsec 32-bit transfers. This gives a theoretical maximum transfer rate of 100 MBytes/sec between boards plugged into the NextBus. The practical rate after arbitration is somewhat less than half of that.

Multiple Machines

Provisions have been made for an optional daughter board that ties directly into the MBUS. In the future an interface will be designed using this feature, which will provide a high-speed parallel interconnection path between NeXT machines. This interconnection scheme is shown in stylized form in Fig. 3. In any machine, each of three boards is connected to one of three boards in one of three other machines. Each one of these connections is the equivalent of a separate NeXT/NuBus. This connection scheme was chosen over a simpler single bus extended approach because of the added bandwidth it provides and because of peculiarities of the

Fig. 3. Multiple machine interconnection.



NeXT/NuBus bus address decode mechanism, which would have made it difficult to implement a single multimachine interconnect bus and still allow complete memory mapping of all the local RAM blocks into the physical address space of every i860 processor, a symmetry that we wanted to preserve. This connection scheme allows a maximum configuration of 24 i860 processors with 760 MBytes of memory.

Optimized Math and Signal Processing Library for the i860 Processor

An optimized library of commonly used musical signal processing and mathematics routines is being developed in i860 assembly language at IRCAM as part of the IMW project. Until substantial advances are made in compiler optimization for superscalar architectures, this remains a crucial factor in obtaining the desired performance. The library includes standard mathematics routines (e.g., real and complex vector, vector-to-scalar, and matrix operations) and signal processing routines (e.g., filters, FFT, windowing, delay, table-lookup oscillators, envelope followers, sampling synthesis oscillators, envelope generators).

All library operators are vector routines, which means that signals are processed as vectors of samples instead of on a sample-by-sample basis. This is necessary to offset the setup time associated with pipelined architectures such as the i860. The vector length is typically 64 samples. Control sources that require subvector precision, such as a break-point envelope generator, are coded as vector modules themselves.

A New Set of Problems

The choice of a general-purpose processor for the IMW multiprocessor architecture introduces new problems of efficiency caused by the presence of caches, a memory management unit, and dynamic RAM with page-access mode (which could be ignored with classical real-time signal processing static RAM-based, parallel architectures). One of the challenges in designing library modules is to take these problems into account.

Our optimization of memory access uses a model whereby input and output vectors are assumed to be in the data cache when an operator is called and are sized at compile time and aligned to support 128-bit data transfers. Wavetables and coefficient tables are generally not cached and are accessed using special pipelined floating-point load instructions available on the i860 processor. Attempts are made to optimize placement of these tables in 4-kByte DRAM pages. DRAM access within a page is several times faster than interpage access.

Preliminary Benchmarks

Benchmarks based on the first implemented modules give the following peak performances. At a 44.1-kHz sampling rate, a two-processor board can

Fig. 4. Impact of cache misses on vector addition.

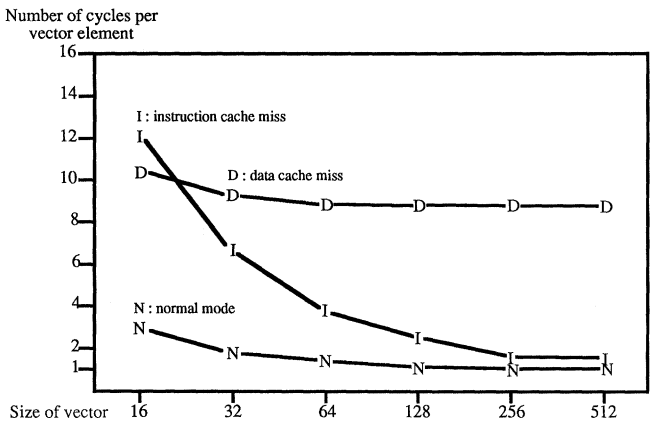
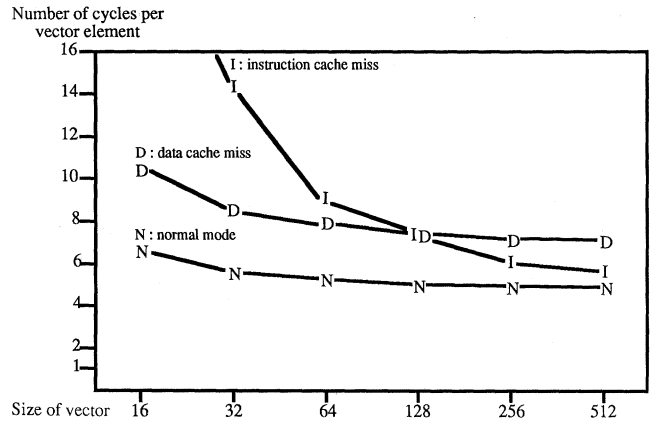


Fig. 5. Evaluation of cache misses cost for vector absolute value.



compute 400 simple table lookup oscillators, 450 two pole recursive filters, 50 voices of a 7-point interpolating sampler oscillator, or 1800 basic vector operations. A 1024-point complex FFT is computed in 1 msec on one processor.

Impact of Cache Misses

Unlike classical signal processors, the general-purpose i860 processor, used in a real-time context, introduces new problems of efficiency caused by the presence of instruction and data caches. Cache misses (e.g., data or instructions not in their respective caches when a library operator is called) add an extra cost—the external memory accesses necessary to fetch data or instructions.

To evaluate this impact, a diagnostic has been developed that forces cache misses and supports measurement of execution time. Figures 4 and 5 show this execution time as a function of vector length for two library operators. Execution time is given in number of processor cycles per vector element. Three cases are presented: normal execution respecting memory model, instruction cache miss, and data cache miss.

This simulation shows that, as one would expect, the cost of data cache miss per vector element is an additive quantity quite constant over vector size, and that instruction cache miss cost is a constant independent of vector size.

Overview of IMW Software Architecture

Figure 6 shows a global view of the various software components of the IMW. The bottom shows components that run on the i860 multiprocessor. The top shows components that run on the host and communicate with the multiprocessor. Most of the software components are described in the accompanying articles in this issue of *Computer Music Journal.*

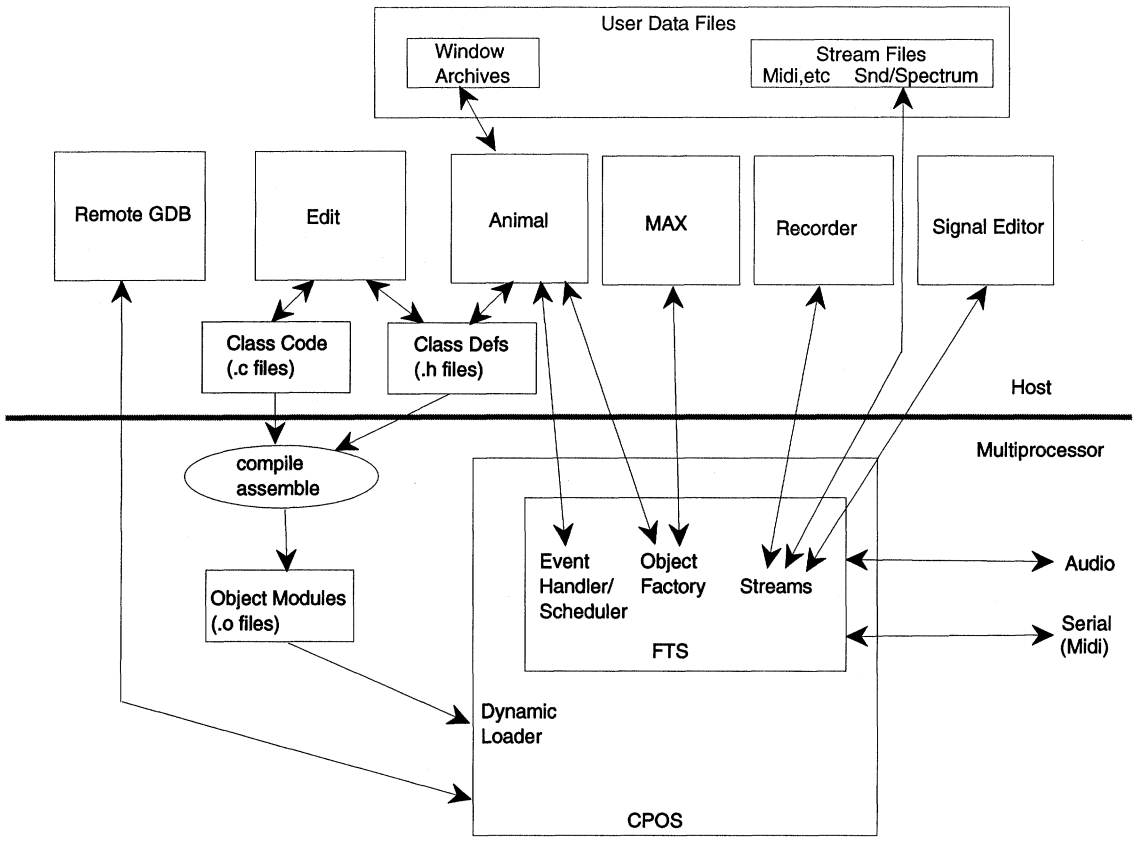
CoProcessor Operating System (CPOS)

CPOS is the operating system running on the i860 processors. There were several goals in its design: to provide an effective real-time kernel capable of supporting event and signal processing; to provide a "reasonable" degree of UNIX compatibility so that users not familiar with the IMW could come up to speed rapidly; to make it relatively easy to port existing non-real-time applications; and to make available third-party development tools such as GDB, the Free Software Foundation's GNU debugger.

Faster Than Sound (FTS) Toolbox

Running on top of CPOS, the *FTS* toolbox provides support for musical event processing as well as scheduling and mapping of signal processing "graphs" across the multiprocessor. FTS imple-

Fig. 6. Software components of the IMW.



ments these functions in the context of its object-class system which supports message-passing, method invocation, and run-time metaclass structures. This class system is used by the MAX and Aminal applications to support interactive and incremental class definition, object instantiation, and dynamic linking of code.

MAX

MAX is a grpahic programming language for creating and controlling real-time event and signal processing algorithms. MAX was first developed for the Apple Macintosh computer and has been used in many real-time performance situations in the context of real-time MIDI control. The port to the IMW adds the ability to design real-time signal processing algorithms using MAX.

Animal

Animal is an interactive system for creating graphic representations of FTS classes and for interacting with networks of objects created from these classes. It provides for persistent storage of objects on disk and object libraries.

The IMW Signal Editor

The Signal Editor supports viewing and editing of audio signals in time and frequency domains. The frequency domain tool, called "SpecDraw," shows sonographs—"two-and-a-half dimensional" representations of time versus frequency with amplitude mapped onto gray levels—on which the user can design dynamic digital filters by drawing arbitrary polygons in pitch-time space. The user can fil-

ter out the contents of the polygon or everything outside the polygon. In the future, arbitrary cut-and-paste operations on polygon regions will be supported.

IMW Recorder

The "Universal Recorder" application is intended to provide a user interface using the multitrace recorder paradigm for spooling real-time event streams and signal streams to disk or memory. The recorder uses message and signal "probes" that can be attached to FTS objects to tap onto their message streams and, on playback, to regenerate these streams in a time-accurate manner.

Conclusion

The IRCAM Music Workstation is a large project,the first stage of which is being completed at the time of this writing. A contract has been signed with Ariel Corporation in the USA for the commercialization of the hardware and the base-level software (CPOS, mathematical library, and basic development enviornment). Other software packages will be made available separately. It is hoped that these tools will provide an excellent environment for computer music performance and research.

The distinguishing feature of the IMW architecture is its reliance on high-level computing techniques for real-time musical signal processing and event management. These techniques include a uniform general-purpose multiprocessor with set associative caches, memory management units, floating-point computation, and high-level compiler support; a UNIX-compatible operating system (CPOS); a unified environment for signal processing and discrete event management (FTS); and tools for high-level object management and user interface design (FTS, MAX, and Animal). From the point of view of the "average hacker," these features help to make the IMW look like a "normal" computer. The programmer is presented with standard development and debugging tools rather than being forced to enter into a lengthy apprenticeship in order to master DSP microcoding, arcane scheduling techniques, and so forth. With the addition of the high-level development tools MAX and Animal, the IMW application development becomes accessible even to nonprogrammers, such as technically oriented musicians, or to scientists in a hurry.

It is hoped that the IMW architecture will prove to be relatively portable. Longevity and portability are not qualities often associated with high-performance real-time systems. The basic IMW system software can survive several generations of hardware upgrade, especially as successors to the i860 processor are introduced. These features contribute to a system that can function as both performance instrument and general-purpose research tool.

References

DiGuigno, G., and A. Gerzso. 1986. *La Station de Travail Musical 4X.* IRCAM Technical Report. Paris: IRCAM.

Eckel, G. 1990. "A Signal Editor for the IRCAM Musical Workstation." In *Proceedings of the International Computer Music Conference.* San Francisco: Computer Music Association.

Intel Corporation. 1989. *i860 64-Bit Microprocessor Programmer's Reference Manual.* Santa Clara, California: Intel Corporation.

Lindemann, E. 1987. "DSP Architectures for the Digital Audio Workstation." *AES Preprint—83rd Audio Engineering Society Convention.* New York: Audio Engineering Society.

Lindemann, E., and M. de Cecco. 1991. "Animal: Graphical Data Definition and Manipulation in Real Time." *Computer Music Journal* (this issue).

Moorer, J. A. 1982. "The LucasFilm Audio Signal Processor." *Computer Music Journal* 6(3):22–32.

Puckette, M. 1991. "Combining Event and Signal Processing in the MAX Graphical Programming Environment." *Computer Music Journal* (this issue).

Smith, B. 1990. "A Universal Recorder for the IRCAM Musical Workstation." In *Proceedings of the International Computer Music Conference.* San Francisco: Computer Music Association.