U.D.I : A Unified D.S.P. Interface for sound signal analysis and synthesis
Ph. Depalle & X. Rodet
IRCAM - Dept. Analyse/Synthèse
31, rue St-Merri
75004 PARIS

ABSTRACT : In this paper, we present a new machine independent programming
environment, called UDI (a ”Unified Dsp interface”), which allows a coherent software
approach for developing and maintaining digital signal processing algorithms. UDI is devoted to
sound signal analysis and synthesis. It runs on ordinary computers or on host-array processor
configurations and guarantees that software can survive machine changes. UDI has already been
ported on nine different machine configurations. It consists of a programming standard (based on
a vector approach, a data abstraction and a routine-call abstraction), a library of vector routines
and configuration-specific drivers. Many applications have already been written in UDI.

INTRODUCTION

For sound signal analysis and synthesis, many Digital Signal Processing (DSP)
algorithms have to be developed, which are sometime general and some time more specific to a
given application. This represents an important effort for several reasons. Writing such programs
usually necessitates a specific specialized programmer, and debugging and testing are particularly
difficult. Since the computation cost is high, fast processors are often used. DSP processors are
all the more efficient if the library calls provided by the manufacturer are invoked. But since the
technology of memory-processor is evolving very rapidly, one must often maintain at the same
time a large number of DSP programs on several different DSP machines. This is a heavy task
which must be simplified by a coherent software approach.

Consequently, we have developped a programming environment called U.D.I. ”a
Unified DSP Interface” which is machine independent, running on ordinary computers or on
host-array processor configurations. U.D.I. consists of a programming standard, a library of
vector routines and configuration specific drivers.

THE PROGRAMMING STANDARD

The programming standard is based on a vector approach, a data abstraction and a
routine call abstraction.

VECTOR APPROACH

The vector approach is motivated by the structure of DSP algorithms which can often
be decomposed in basic vector operations. It is also motivated by the architecture of DSP
Processors which are particularly fast on vector operations.

DATA ABSTRACTION

ICMC GLASGOW HK PROCEEDINGS
223
The data abstraction consists of two structures. First the BUFFER structure contains all the informations necessary for describing an array of data (type, size, address, etc...). Different vector routines apply on BUFFERS which determine array of data of different types such as long, float and double. Therefore at run-time, the vector routines verify the coherence between their type and the type of BUFFERS on which they apply. They also verify the coherence between the number of operations desired and the number of data elements described by the implied BUFFERS. Secondly the WINDOW structure describes a way of looking at a portion of a BUFFER (BUFFER pointer, offset, size, index-increment, etc...). The index-increment allows one to address interleaved data in the BUFFER starting at offset. As an example, they can be used to process one channel of a multichannel sound signal. WINDOWS can be modified at run-time and this is useful to process different portions (vectors) extracted from a BUFFER. For instance, processing a whole BUFFER using a sliding WINDOW can be implemented by using successive incrementation of the offset. Processing of data described by BUFFERS and WINDOWS is provided for by machine independent standard routine calls.

LIBRARIES OF ROUTINES

There exist two sorts of libraries of routines. The "simulation library" is composed of C coded vector routines and is aimed at running on the host. On the contrary, the array processor specific libraries are usually hand coded.

Three types of situations are in view:

In the first situation, the user runs a program P on the host and the vector routine calls are redirected to the array processor and generally execute the specific vector code provided by the manufacturer.

In the second situation the same program P should run on the host alone, with vector routines and driver routines and driver running on the host : this case is the case when there is no array processor or when the array processor is unavailable (monosketch processor for instance).

The last situation is the development-debugging stage where vector routines and driver routines on the host can be considered as simulated on the host : this situation occurs because on one hand it is easier to develop on the host and on the second hand this situation allows machine independent development.

DRIVER

The driver part of U.D.I. performs processor control, code loading, allocation and data exchanges. In particular BUFFERS and WINDOWS are allocated (and modified) by driver calls and the driver verifies the coherence of the location of BUFFERS and of WINDOWS within BUFFERS. The data exchange routines also take care of all specific features of the DMA of the attached array-processor such as block-size and address alignment.

IMPLEMENTATION

U.D.I. has already been ported on nine different configurations (SUN, SUN-Mercury MC 32, SUN-Mercury ZIP 32, DEC 3100, Next, PC, Ianl I 869 board and Macintosh). The port on a new configuration is very easy: it consists only in rewiring of some fifteen driver routines. The software layer between the normalized U.D.I. routines and the machine specific routines, is generated by a U.D.I. utility program. It is easy to add a new routine to the library and the documentation is generated automatically from simple specifications.

ICMC GLASGOW 1990 PROCEEDINGS

226
U.D.I. has been implemented, tested and used at IRCAM. We have written many analysis and synthesis programs using the U.D.I. system such as phase vocoder, filtering, or digital spectrograms.

As an application example, a program called process file has been written. It performs on one or several data files any algorithm described by one or a sequence of U.D.I. library or driver calls. For instance a cepstrum computation on a data file is obtained by giving to process file a script of eight routine calls only (cf. appendix).

CONCLUSION

As a conclusion we can mention some points that have been found particularly important:

- common programming errors are often prevented by the system,
- U.D.I. code is guaranteed a long life surviving machine changes,
- programming and maintenance is easier and faster, due to the large number (150) of available routines,
- efficiency is optimal because of the vector approach.

Some extensions are being studied such as the automatic execution of the simulated routines on the host when an array processor routine code is not available.

One of our goal is that U.D.I. becomes available for researchers and musicians and be used on many machines and in many centers: in this way a large number of long-life programs can be written and exchanged.

Appendix

; Script to be given to process_file
; to perform Cepstrum computation

length = 1024
half_length = length/2

; Sends data from the host to the attached processor
send_data file signal buf_1 1 $\text{length}

; Compute hamming window and apply it to the signal
hamm buf_wind_2 1 $\text{length}
vmul buf_1 1 buf_wind_2 1 buf_1 1 $\text{length}

; Perform FFT on the windowed signal
rfft buf_1 1 int1 1 $\text{length}

; Compute the logarithm of the magnitude of the FFT
cvmsgs buf_1 2 buf_out 2 $\text{half_length}
vlog buf_out 2 buf_out 2 $\text{half_length}

; Perform inverse FFT on the log magnitude
tfft buf_out 1 int1 -1 $\text{length}

; Send data from the attached processor to the host
receive_data buf_out 1 file fenetre $\text{length}

ICHIC GLASGOW 1990 PROCEEDINGS

127