A Multiprocessor Simulation for Advanced Signal Processing Applications

Umar Hamid

***Abstract*—This paper presents a multiprocessor simulation that executes an advanced signal processing algorithm i.e. beamforming for sensor arrays. The simulation model involves several desktop PCs. It demonstrates how an advanced signal processing technique can be implemented on a system that is based on multiple digital signal processors (DSPs) in the absence of actual hardware. It uses MATLAB, a DSP simulator, and an open source MATLAB based message passing interface (MPI). The simulation applies a frequency-domain beamformer to a linear sensor array and the results show efficient utilization of processors for real-time operation.**

***Index Terms*—Multiprocessor, sensor array, beamforming, DSP, and FFT.**

I. Introduction

In all sensor array systems, a signal processing unit is required to handle signals plus noise and provide accurate estimations in both time and frequency domains for signal detection, localization and classification purposes. This requires simulations of the system in its early design stages.

In general, signal processing algorithms are modeled using MATLAB and implemented using DSP simulators. These simulators mimic the DSP core accurately but can simulate one processor at a time in the absence of actual hardware (e.g. a PCI based board having two or more DSPs on it). This paper presents a novel approach for multiprocessor simulation using MATLAB, a DSP simulator and an open source MATLAB MPI. The approach for performing multiprocessor simulation involves desktop PCs where each PC acts as a single processor along with one PC providing central control and data communication. The simulation executes signal processing functions and the results show efficient utilization of the processors for real-time operation. From a signal processing perspective, processing techniques based on a multiprocessor architecture offers design flexibility. With a homogeneous set of processors, dividing tasks evenly among the processors serves to maximize parallel performance.

II. Simulation Environment

The simulation environment comprises of five desktop PCs connected to each other via Ethernet link. Four PCs have a DSP simulator (e.g. VisualDSP++) installed on them and each PC acts as a single processor in the simulation. VisualDSP++ is a DSP simulator by Analog Devices. The scenario is virtually the same as the actual one where there are four DSP processors on a single PCI based DSP board (e.g. TigerSharc TS201 DSP board). On the other hand, fifth PC is not only responsible for generating sensor array data but also provides a central control to all other computers. The individual processors performs signal processing functions on the simulated data in parallel. The fifth PC uses MATLAB and open source MPI software for two important tasks; 1) communicating with other PCs (i.e. DSP simulators) using COM objects and 2) data transfers among PCs using open source MPI.

Manuscript received June 9, 2012; revised July 22, 2012. The research is part of author's thesis for Masters in Telecommunication Engineering.

Umar Hamid is student of Institute of Communication Technologies, ICTECH, Pakistan (e-mail: alwaysumar@ gmail.com).

III. DSP Simulator

A. ActiveX Scripting Framework

VisualDSP++ introduces a language-independent scripting host that utilizes the Microsoft ActiveX script host framework. The scripting host permits the use of multiple scripting engines (VB, Java, MATLAB etc) that conform to the Microsoft ActiveX script engine framework. It is ideal for non-interactive scripting needs such as accessing DSP resources (i.e. reading/writing memory or reading/writing registers), performing repetitive tasks (executing external tools prior to or after a build completes, or setting registers and memory prior to loading a program) [1].

B. DSP Automation API

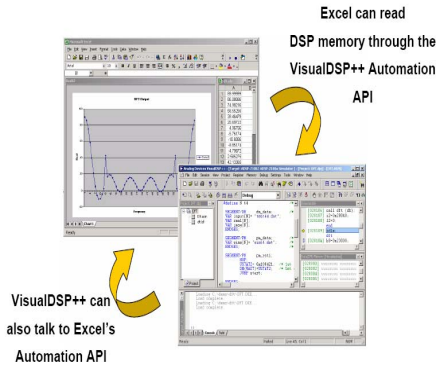


Fig. 1. DSP Automation API [1]

VisualDSP++ includes an Automation API (application programming interface) that provides all script languages with broad access to all of the capabilities of the IDDE in a language-independent manner. It is a Microsoft COM-based technology that makes the VisualDSP++ environment programmable by applications that understands automation and supports the COM interface (VB, Java, and MATLAB

etc). This allows Automation API aware applications to communicate and control VisualDSP++ and vice versa. As an example application, a simple script could be written in Microsoft Excel that would connect to a target board, load a program, set and run to a breakpoint, and then read a block of DSP memory into Excel for data analysis [1] shown in fig. 1.

C. Interface with MATLAB

Following steps have been followed in order to interface MATLAB with VisualDSP++ correctly:

- MATLAB is treated as a client application and VisualDSP++ acts as a server application
- Open the VisualDSP++ COM object
- Get the current directory path and add the desired VisualDSP++ project
- Activate the session for TS201 DSP processor and load the desired program
- Accessing the memory type object in order to gain access of DSP memory
- Write data to the input section of DSP memory, execute the algorithm and read the results back from the output section of the DSP memory
- Reset the processor and quit

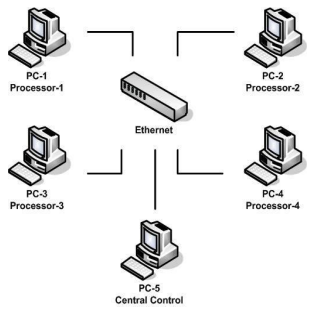
All these steps are performed using the commands and functions available with the Automation API for COM objects [1].

IV. Simulation Mechanism

The simulation model consists of four PCs or processors. The program executing the signal processing functions on the simulated data resides in each of the four processors and is called target application. In addition, it consists of a PC providing central control and data transfers to all other PCs. The program residing in this PC is called host application.

The host application performs the following tasks:

- Establish client-server connections among PCs
- Generates simulated data for sensor array using MATLAB
- Executes DSP simulator sessions using COM components on the four processors
- Simultaneous execution of beamformer routine on the four processors using COM components
- Gathers processed data from four processors and provides output



- Fig. 2. Multiprocessor simulation model

The host PC generates linear array data. The simulated data consists of target bearing, frequencies and additive noise. The host application is responsible for dividing the data evenly among the processors in such a way that each processor handles its tasks respectively.

The block diagram for multiprocessor simulation is given below in Fig.2,

V. Beamforming on A Single DSP

A time delay in time domain corresponds to a phase shift in frequency domain. The frequency domain beamformer uses this principle. The multiple-beam beamformer uses 2D-FFT processing on the multi-channel sensors data. This result in multiple beams formed simultaneously [2].

This paper has adapted the following approach to perform 2D-FFT beamforming [3] is as follows:

- Generate the linear array data having desired signals plus noise using MATLAB beamformer model
- Establish interface between MATLAB and VisualDSP++ and load the simulated data into the DSP memory
- Change the FFT sizes in the FFT routines according to the incoming data
- Perform temporal FFT i.e. FFT on the time series data for each sensor and accumulate the results. The results have been verified with the FFT performed by MATLAB

Perform spatial FFT i.e. FFT across all the sensors and accumulate the results. The results have been verified with the FFT performed by MATLAB.

VI. Beamforming on Multiple DSPs

The simulation model uses an automation API involving four PCs or processors each having its own VisualDSP++ simulating a single processor. All the processors perform FFTs on their part of data coming from host PC and then one processor accumulates the data and performs conventional matrix handling techniques for appropriate data storage and outputs. The approach for dividing the array data among four processors has been taken from [3] and is given below:

For example, the data size for a linear array is 64x1024; where 64 is the number of sensors and 1024 is the number of time samples for each sensor. For performing temporal FFTs, the data is divided among four processors such that each is having a data size of 16x1024. In this way all the processors perform the task simultaneously (i.e. each processor executes 1K point FFT for 16 sensors) and the results are accumulated. Now the output data size becomes 64x512 because of symmetric properties of FFT. For performing spatial FFT, the data divided among four processors such that each is having a data size of 64x128. In this way all the processors perform the task simultaneously (i.e. each processor executes 64 point FFT for 128 frequency bins) and the results are accumulated.

PC1 to PC4 shown in fig. 2 remain in wait mode at the start of simulation. PC5 performs the following tasks:

- Open connections with remote PCs i.e. from PC1 to PC4
- Load simulated data from MATLAB into remote workspaces
- Open VisualDSP++ sessions using COM components

for remote PCs
- Simultaneous execution of Temporal FFT routines on remote PCs
- Access results from remote workspaces for central control using PC5
- Load data (output of temporal FFT) from MATLAB into remote workspaces
- Simultaneous execution of Spatial FFT routines on remote PCs
- Access results from remote workspaces for central control using PC5
- PC5 gathers data for output of 2D-FFT beamformer

VII. Simulation Results and Analysis

Data for a linear sensor array has been simulated using MATLAB and includes information such as array size, array frequency, target frequencies, incoming direction of the target relative to array broadside, noise generation, and other environmental characteristics.

Beamforming algorithm using 2D-FFT has been tested both on a single DSP and on multiple DSPs as mentioned earlier. Both temporal and spatial FFTs have been executed on the DSP simulator VisualDSP++, their results have been compared with MATLAB FFT results and are found correct. The beamformer output from MATLAB model and multiprocessor simulation model is the same and is given below:

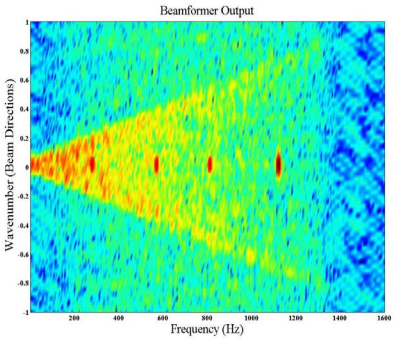


Fig. 3. Beamformer output [4]

Fig. 3 shows different frequencies radiated by the target on the x-axis and incoming beam directions termed as wavenumber on the y-axis. The values of wavenumber [-1, +1] corresponds to bearings [-90, +90] relative to array broadside [4].

VIII. Simulation and Real-Time Implementation

The simulation model can be compared with real implementation in a sense that the central control PC is acting like a module in which the PCI based DSP card is plugged having four DSPs onboard.

The simulation is handling the issue of synchronization manually for four processors whereas in real implementation it is possible to declare one processor as the master and the other three processors as slave processors driving their clock from the master processor.

Each processor in the simulation is handling the data and program code into different memory sections according to the processor address space. All the input and output variables have been assigned labels so that it becomes easy for external applications (i.e. host application) to get access of these DSP memory addresses appropriately.

The routines carrying out the signal processing functions have been implemented using both C and assembly language code. These routines execute on DSP simulator. They have been tailored according to the beamformer requirements. The same routines will be used for real-time implementation.

IX. Conclusion

This paper has discussed a multiprocessor simulation model that will facilitate the researchers in many ways i.e. familiarization with DSP simulator, coding for DSPs in both C and assembly language, handling of DSP memory and registers, placement of data and program in DSP address space, handling of multiple DSPs with proper synchronization and timing. In addition, code testing and debugging can be carried out using DSP simulator. With this approach, the application programmers can develop the code in less time without waiting for the actual hardware to arrive. The same code can be executed in parallel on multiple DSPs with little effort for final implementation. Finally the beamformer simulation results have been found correct along with efficient utilization of processors for real-time operation.

References

[1] J. Pound, "An Introduction to Scripting in VisualDSP++," EE-235, Analog Devices, 2004
[2] T. E. Curtis, "Principles of Sonar Beamforming, Copyright Curtis Technology (UK) Ltd," 1998
[3] U. Hamid, H. Shahzad, and M. Irfan, "Parallel Implementation of Frequency Domain Beamformer on a Multi-core Processor," *International Conference on Computer and Electrical Engineering*, China, 2010
[4] R. Judd, k-Omega beamforming Example, TASP VSIPL, 2003

Umar Hamid was born in Sialkot, Pakistan, in 1980. He received BSc degree in Electrical Engineering from University of Engineering and Technology Taxila, Pakistan, in 2003. His current interests are data acquisition and signal processing.