Parallel Implementation of Frequency Domain Beamformer on a Multi-Core Processor

Umar Hamid, Haroon Shahzad, and Muhammad Irfan

Abstract—Multi-channel sensor systems involve frequency domain beamforming techniques to provide adequate spatial coverage. Frequency domain beamformer involves 2D-FFT processing which is a processor intensive operation. This paper presents a parallel programming paradigm to utilize multi-core processors to their full potential in terms of CPU time.

Over the years, Intel architecture has acquired the capabilities of processing the high bandwidth data in real-time. Intel has introduced many powerful processors with CPU clock in excess of 2 GHz i.e. Intel Core2Duo and Core2Quad processors. The challenge is to build software that can fully exploit the capabilities of multi-core processors. This paper emphasizes on parallel implementation of algorithms to have faster performance. Parallel implementation involves multithreading approach for multi-core processors.

The aim of this paper is to show the performance enhancement in terms of execution time using parallel programming techniques. This involves executing sequential beamformer followed by parallel beamformer on an Intel based multi-core processor. Final results show decrease in execution time for a parallel application.

Index Terms—Parallel algorithms, multithreading, multi-core processors, FFT, and beamforming.

I. INTRODUCTION

The trend of increasing speed and complexity in the single-core processor as stated in the Moore's law is facing practical challenges. As a result, the multi-core processor architecture has emerged as the dominant architecture for both desktop and high-performance systems. Multi-core systems introduce many challenges that need to be addressed to achieve the best performance [1]. A multi-core processor design consists of core logic for more than one processor. It means that one processor chip has effectively packaged more than one processor core. The ultimate goal of such a design is to enable the systems run more and more tasks simultaneously in order to achieve enhanced performance. In this context software applications need to be designed in a way to utilize all the processor cores. This paper shows parallel implementation of an advanced signal processing algorithm i.e. beamforming on a multi-core processor in order to maximize the overall processing capability.

II. MULTI-CORE PROCESSOR ARCHITECTURE

These days several variants of multi-core processors are available in the market based on different number of cores, cache organization, and the access mechanism. For a typical multi-core processor multiple caches are shared among the multiple cores.

A. Dual-Core Processors

Dual-core processors contain two identical processor cores on a single chip. Each processor core has its own resources except the on-die cache memory that can be provided to each processor core in three possible ways as designed by Intel; 1) separate on-die cache, 2) on-die cache to be shared between the two cores, 3) each processor core can have a portion of on-die cache exclusively for itself. Both processor cores must have a communication path to the system front-side bus [2].

B. Multi-Core Processors

Multi-core processors can have any number of CPUs on a single chip i.e. 2, 4, and 8. Their architecture is an extension of dual-core processors [3]. This design has enabled multiple cores to run at slightly slower speed and low power consumption. The main challenge is software development so that maximum performance can be taken out from multiple processor cores.

Fig. 1. Multi-core processor architecture [2]

III. PARALLEL PROGRAMMING ON MULTI-CORE PROCESSOR

A. Parallel Programming

Parallel programming means execution of multiple program units i.e. threads on multiple CPUs. The purpose is to decrease the program execution time and is one of the key benefits of multi-core processors. The idea is to identify how

Manuscript received October 12, 2012; revised November 23, 2012.
 Umar Hamid is with the Institute of Communication Technologies, ICTECH, Pakistan (e-mail: alwaysumar@gmail.com).
 Haroon Shahzad and Muhammad Irfan are with Harbin Engineering University, China (e-mail: harooniui@hotmail.com, mirfan_iui@yahoo.com).

many serial applications can run on a multi-core processor at once without interference. The programmer can handle this concern by defining a parallel programming model i.e. dividing an algorithm into different execution threads. These threads can be additional copies of the application under test, or they can be entirely different processes. As the number of processor cores increases, a multithreaded application always provides enhanced performance.

B. Multithreading

Threads are the basic objects that Windows schedule for execution. They contest for and share CPUs. Threads allow a single application to multitask its own execution. On a single-core machine, threads give the appearance of being simultaneous in execution, while on a multi-core machine, multiple threads do indeed execute simultaneously. Threads allow four essential services given below:

- They allow portions of a sequential program to execute in parallel
- They let run multiple copies of a code section
- They make it possible to call functions that are likely to block without keeping your program's user from doing other things i.e. they don't let a program occupy the processor indefinitely
- Only a multithreaded application can take advantage of a multi-core machine

In addition the Windows scheduler provides options to set thread priorities thereby allowing threads to get their time slice in an orderly fashion. For multi-core processors, the Windows scheduler detects the presence of multiple CPUs and allows threads to run in parallel by assigning each thread to a CPU respectively. Hence for an operating system like Windows, a software application must have a multithreaded architecture in order to benefit from multiple cores.

C. Threads Synchronization

Writing multithreaded programs is not an easy task. A large number of threads can be a blow to the system memory. Hence selection of number of threads not only depends upon the amount of processing involved but also on the system on which the application will run. All threads of a process share the same address space and also share all objects owned by the process. Hence they are very tightly coupled and their access to any common resource or object must be synchronized to avoid deadlock situation. A shared resource could be a buffer, file, and disk etc. Synchronization between the threads is achieved by using synchronization objects. These objects are semaphores, events, critical section and mutexes. A program having multiple threads that are not synchronized properly, will provide garbage data outputs.

D. Program Parallelization

The transformation of a sequential program into a multithreaded program is called parallelization [4]. This can be done in the following three steps:

1) Sequential algorithm decomposition

The sequential algorithm is divided into small tasks and interdependencies among these tasks are determined so that every thread is assigned a particular portion of the program. The code given to thread is the smallest unit of the same

program which is intended to be running in parallel. This decomposition can be done either at the start of the program during initialization phase or at the spur of time during execution, depending upon the nature of the application. Therefore number of tasks may vary during execution of the program. At any point in program execution, the number of executable tasks is an upper bound on the available degree of parallelism and, thus, the number of cores that can be usefully employed [4]. The goal of algorithm decomposition is therefore to generate enough tasks to keep all cores busy at all times during program execution.

2) Tasks assignment

A thread has a set of instructions that are to be executed by a processor core. Threads execute the instruction one after the other as specified by the programmer. In general, number of threads is equal to the number of processor cores for a multi-core processor. The purpose is to balance the processing load among processor cores by assigning each thread to a particular processor core. The task assignment operation is carried out by the operating system scheduler. However the programmer can set different thread parameters such as thread priority etc.

3) Mapping of threads onto processor cores

In a multithreaded application, the aim is to assign each thread to a separate processor core. There is a trade off between the number of processor cores available and the number of threads to be executed. If there are fewer cores available as compared to threads then each core may be assigned with more than one thread and if there are more cores than number of threads then the processor is under utilized.

The program parallelization steps have been illustrated by the following figure:

Fig. 2. Program parallelization [4]

E. Function Parallelization

In sequential programs, many program blocks may be independent of each other and can be executed in parallel. It could be an expression, a function call or a block of code. These independent program parts are termed as functions and this type of parallelism is called functional Parallelization.

IV. FREQUENCY DOMAIN BEAMFORMING

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 [5].

Using frequency domain techniques, the steering directions are not limited by the sampling period. In this case, number of beams is equal to the number of sensors. Frequency domain realizations minimize system size and cost but the algorithms require more processing power.

The approach adapted to perform 2D-FFT beamforming is as follows:

- Generate the multi-channel sensor data containing desired signal plus noise using MATLAB
- Perform temporal FFT i.e. FFT on the time series data for each channel and accumulate the results. The results have been verified with the FFT performed by MATLAB and are found correct
- Perform spatial FFT i.e. FFT across all the channels and accumulate the results. The results have been verified with the FFT performed by MATLAB and are found correct
- Development of C/C++ codes for sequential and parallel implementation of 2D-FFT beamforming

The parallel application is responsible for dividing the tasks evenly among the processor cores using multithreading approach.

V. PERFORMANCE EVALUATION OF SEQUENTIAL AND PARALLEL BEAMFORMER ON A MULTI-CORE PROCESSOR

This research work involves 2D-FFT beamforming process that uses an open source FFT library called FFTw as benchmark workload.

A. Experimental Setup

We have taken beamforming algorithm as a test case. First we evaluated the performance of both sequential and multithreaded FFT application in terms of execution time in our previous research work [6]. Secondly we evaluated the performance of both sequential and multithreaded beamformer application in terms of execution time in this research work.

B. Results of Sequential FFT Program

The execution times for several FFT sizes on different Intel systems are given in the Table I below:

TABLE I

FFT SIZE	EXECUTION TIME (milli seconds)		
	Intel P-IV	Intel Core2Duo	Intel Core2Quad
1024	4868	2285	2650
2048	10507	5370	5830
4096	22548	12310	13885

TABLE II

FFT SIZE	EXECUTION TIME (milli seconds)		
	Intel P-IV	Intel Core2Duo	Intel Core2Quad
1024	5140	1328	634
2048	11377	4739	2881
4096	24950	11225	7110

C. Results of Multithreaded FFT Program

The execution times for several FFT sizes on different

Intel systems are given in the Table II below:

D. Beamformer Implementation Mechanism

Temporal FFT is the first step in beamforming process. It is performed on time series data (i.e. N samples) for all input channels (i.e. M sensors). In order to utilize the computational power of all processor cores, number of threads is kept equal to number of processor cores. The task of temporal FFT is divided in such a way that each thread computes temporal FFT for equal number of channels and the input data matrix is assigned to all threads accordingly.

For equal division of input data to all threads, number of channels required per thread is calculated as follows:

$$\text{ChannelsperThread} = \text{Total Channels} / \text{No. of Threads}$$

Each thread computes temporal FFT of N time samples for ChannelsperThread and stores output in respective buffers in order to avoid any error. In order to obtain accurate data outputs these threads have been synchronized properly. Next step is the computation of spatial FFT. It is essential for all threads to complete their temporal FFT computations before further processing. For spatial FFT, data matrix is rearranged so that data is evenly distributed among all threads. In spatial FFT computations each thread is tasked to compute FFT of M points for N/4 iterations. Again all these threads have been synchronized in order to obtain accurate results of spatial FFT.

E. Results and Analysis

In this paper frequency domain beamforming on a multi-channel input system has been taken as an example. Both sequential and multithreaded implementations of beamforming have been tested on an Intel based multi-core processor for measuring the performance in terms of execution time.

It is evident from the results mentioned in section B and C that Core2Duo and Core2Quad processors offer better processing speed as compared to a single core processor for implementing parallel algorithms. In addition it can be seen that Core2Quad processor gives the best performance as compared to other two processors by using multithreading approach.

Keeping this in view, Intel Core2Quad processor was selected for parallel implementation of frequency domain beamformer. The sequential beamforming application on Core2Quad processor executes in 200 to 220 msec time (for M=128 and N=4096). On the other hand, parallel beamforming application on Core2Quad processor executes in 100 to 125 msec time. This results in almost 50% reduction in execution time for computing beamformed data outputs using parallel programming techniques.

VI. CONCLUSIONS

Parallel computing has enabled software developers to design applications in a way to efficiently utilize the processing capabilities provided by the latest multi-core processors. It facilitates in handling complex algorithms and provides significant increase in system performance. This paper is a manifestation of our success in achieving reduced execution time for a frequency domain beamformer on a

multi-core processor using parallel processing techniques. This conclusion was drawn on the basis of comparative analysis of execution times for both sequential and parallel implementations of beamforming algorithm.

REFERENCES

- [1] K. A. Yao, Y. E. Ghazawi, and T. Newby, "Experimental Evaluation of Emerging Multi-core Architectures," *IEEE Parallel and Distributed Processing Symposium*, 2007.
- [2] T. Martinez and S. Parikh, "Understanding Dual-Processor, Hyper Threading Technology and Multi-core Systems," Technical White Paper, 2005
- [3] Intel Corporation, "Migrating Industrial Applications to Multi-Core Processors," Technical White Paper, 2008
- [4] T. Rauber and G. Runger, "Parallel Programming For Multi-core and Cluster Systems," Springer Book, 2010
- [5] T. E. Curtis, "Principles of Sonar Beamforming, Copyright Curtis Technology," (UK) Ltd, 1998
- [6] U. Hamid, H. Shahzad, and M. Irfan, "Parallel Implementation of 1-D Complex FFT using Multithreading and Multi-core Systems," *International Conference on Intelligence and Information Technology*, Pakistan, 2010

Umar Hamid was born in Sialkot, Pakistan. 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.

Haroon Shahzad was born in Rawalpindi, Pakistan. He has done BSc (Hons) and Masters in Computer Scienc from International Islamic University, Islamabad. At Present he is pursuing Masters Degree in Software Engineering from Harbin Engineering University, China. His current research interests include Software Engineering, Parallel and Distributed processing.

Muhammad Irfan was born in Sargodha, Pakistan. He received the Masters degree from International Islamic University, Islamabad, in 2004. He is currently pursuing Masters degree at the Harbin Engineering University, China, in the field of Computer Applied Technologies. His research interests include Software design and architecture, Parallel processing techniques for Signal processing algorithms.