Parallel Implementation of 1-D Complex FFT Using Multithreading and Multi-Core Systems

Umar Hamid, Haroon Shahzad, and Muhammad Irfan

***Abstract*—Now-a-days most desktop PCs have multiprocessing technology such as Hyper-Threading (HT), Dual-Core, and Quad-Core processors. Technological developments in microprocessor design have enabled hardware vendors to put multiple cores on a single socket. The main idea is to build software applications that can fully exploit the capabilities of a multi-core system using multithreading approach for faster performance.**

The aim of this paper is to show the performance enhancement in terms of execution time using multiple threads. This involves executing sequential software application followed by multithreaded software application on Intel based multi-core systems. 1-D complex Fast Fourier Transform (FFT) function has been taken from an open source library, called FFTw, as an example to prove the concept. Finally results show the decrease in execution time with increase in FFT sizes for a multithreaded software application on a multi-core system.

***Index Terms*—FFT, multithreading, and multi-core systems.**

I. Introduction

Multithreading is the ability of an operating system to execute different parts of a program, called threads, simultaneously. The programmer must carefully design the program in such a way that all the threads can run at the same time without interfering with each other.

A multi-core system is a single processor CPU that contains two or more cores, with each core housing independent microprocessors. Multi-core systems share computing resources that are often duplicated in multiprocessor systems, such as the L2 cache and front-side bus.

This paper provides a basic understanding of Intel processor architectures and multithreading concepts. The aim of this paper is to illustrate the advantages of implementing multithreaded software applications on multi-core systems for achieving performance enhancement in terms of execution time. Both sequential and multithreaded software applications have been developed using an open source FFT library called FFTw and their comparison in terms of execution time, on different Intel processor systems has been presented.

Manuscript received August 13, 2012; revised October 8, 2012.

Umar Hamid is with the Institute of Communication Technologies, ICTECH, Pakistan (e-mail: umar_hamid80@ yahoo.com).

Haroon Shahzad and Muhammad Irfan are with the Harbin Engineering University, China (e-mail: haroonіiui@hotmail.com, mirfan_iiui@yahoo.com).

II. Intel Processor Architectures

A. Multiprocessor Technology

Multiprocessor systems offered by Intel comprise of multiple processor chips on a single board. Today, Intel Itanium and Dual Xeon servers are in use having two or more processors on one physical board and connected through a high speed communication interface [1]. The main disadvantage of multiprocessor systems is that they are expensive.

B. Hyper-Threading Technology

Intel Hyper-Threading (HT) technology allows a single CPU to present itself as a dual CPU to the operating system. An HT processor only has duplicate set of registers and pretends to be two processors. Cache and execution units are shared between these two virtual processors, with appropriate logic to switch between these processors so that only one processor uses each unit at a time. Intel HT processor consists of two APICs (Advanced Programmable Interrupt Controller) that provide multiprocessor interrupt management and distribution across the two processors [1]. Intel Pentium IV processors support HT technology.

C. 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 [1].

D. Multi-Core Processors

Multi-core systems 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 [2]. The main challenge is in the area of software development so that maximum performance can be taken out from multiple processor cores.

III. Multithreading

A. Multithreading Concept

In general, a computer program is a series of instructions that are executed by a CPU. Multithreaded programming takes this idea and replicates it. A multithreaded program has many sequences, each sequence within an independent thread. It is like having different small programs all running together

in parallel. In case of a single processor system e.g. Intel HT processor, the operating system has a procedure called scheduler that provides the illusion that multiple threads are running in parallel. The Windows scheduler provides options to set thread priorities thereby allowing threads to get their time slice in an orderly fashion. For multiprocessor and multi-core systems, 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 CPUs. The main task is to divide the processing load among several threads in order to achieve maximum performance. In most applications, only the CPU intensive sections of code would be multithreaded.

B. Parallel Processing

Parallel processing means executing multiple threads on multiple processing units. The purpose is to decrease the program execution time and is one of the key benefits of multi-core systems. The idea is to identify how many serial applications can run on a multi-core system at once without interference. These applications can be additional copies of the application under test, or they can be entirely different processes. Hence for a multi-core system, as the number of processors increases, a multithreaded application always provides enhanced performance.

C. Modes of Operation

Multithreading helps to create scalable applications that allow the programmer to add threads as and when needed. Managing individual threads is important as they share the same memory and data variables. A thread can be considered as a semi-process with a starting point, an execution sequence and a termination point. One thread shares memory and data with other threads running on a system [3].

On a single processor system, threads can be run either in a cooperative mode or preemptive mode. In cooperative mode, each thread can control the CPU as long as it needs. It has no concept of priorities and multithreading. Windows 3.x operating system used this kind of implementation. In preemptive mode, operating system itself distributes the processor time among threads and decides the running sequence of the threads. This approach exists in majority of popular operating systems today not only Windows. This mode allows fast switching among threads and it appears as if all threads are running in parallel.

On a multiprocessor system, operating system can allocate the individual threads to the separate processor cores and hence improves the program execution. The efficiency of multithreading increases due to the distribution of threads on several processors or cores is faster than sharing processor time on a single processor. It is particularly useful for high speed processing applications like signal and image processing.

D. Designing Multithreaded Applications

Writing multithreaded programs is not an easy task. For example, if one program has many threads, then threads in other programs will naturally get less of the processor time. This means a large number of threads can be a blow to the system memory. Hence we decide number of threads not only according to the amount of processing involved but also according to the system on which the application will run [4].

In this context we propose some important considerations while designing multithreaded software architecture:

1) Keep all individual tasks (i.e. doing mathematical computations or asking for user-input) in separate threads along with different thread priorities
2) A thread that handles time critical tasks should be given a high priority than the other threads running on the system
3) A thread listening for client requests or waiting for a user-input should always remain responsive and should be allotted a high priority

IV. Multithreaded Application Benchmarking

Measuring performance on single core and multi-core systems requires benchmark workloads [5]. This research work involves an open source FFT library called FFTw as benchmark workload.

A. Experimental Setup

We have taken FFT algorithm as a test case. Performance of both sequential and multithreaded applications has been measured in terms of execution time with increase in FFT sizes i.e. 512, 1024, 2048, 4096 etc. Different Intel based systems have been used.

B. Simulation Mechanism

The simulation considers a multi-channel input system. N-point FFTs are performed on each input channel data. FFT transforms the input data from time domain to frequency domain and is a real to complex FFT. Simulations have been developed for both sequential FFT and multi-threaded FFT. Both simulations perform N-point FFTs for all channels with 1000 repetitions. In case of multithreaded FFT simulation, number of threads is according to the number of processor cores in case of multi-core systems. Both simulations provide execution times for several FFT sizes on different Intel based systems.

C. Execution of Sequential FFT Program

The sequential FFT program performs N-point FFTs for all input channels and it is repeated 1000 times. The execution times for several FFT sizes on different Intel systems are given in the tables below:

TABLE I: Intel P-IV (HT) Processor

FFT SIZE	EXECUTION TIME (milli seconds)
512	2356
1024	4868
2048	10507
4096	22548
8192	56820

TABLE II: INTEL CORE2DUO PROCESSOR

FFT SIZE	EXECUTION TIME (milli seconds)
512	502
1024	2285
2048	5370
4096	12310
8192	27820

TABLE III: INTEL CORE2QUAD PROCESSOR

FFT SIZE	EXECUTION TIME (milli seconds)
512	650
1024	2650
2048	5830
4096	13885
8192	29900

D. Execution of Multithreaded FFT Program

The multithreaded FFT program performs N-point FFTs for all input channels and it is repeated 1000 times. The execution times for several FFT sizes on different Intel systems are given in the tables below:

TABLE IV: INTEL P-IV (HT) PROCESSOR

FFT SIZE	EXECUTION TIME (milli seconds)
512	2331
1024	5140
2048	11377
4096	24950
8192	66716

Number of threads = 02

TABLE V: INTEL CORE2DUO PROCESSOR

FFT SIZE	EXECUTION TIME (milli seconds)
512	370
1024	1328
2048	4739
4096	11225
8192	24625

Number of threads = 02

TABLE VI: INTEL CORE2QUAD PROCESSOR

FFT SIZE	EXECUTION TIME (milli seconds)
512	210
1024	634
2048	2881
4096	7110
8192	14919

Number of threads = 04

E. Simulation Results and Analysis

In this paper different Intel processor configurations have been nominated for which both sequential and multithreaded programs have been tested for measuring the performance in terms of execution time. FFTs are the corner-stone of many DSP applications, and DSP libraries put a great effort into optimizing them depending upon the FFT size. In this paper FFT analysis on a multi-channel input system has been taken as an example. Different FFT sizes have been used and FFT was executed multiple times. Both sequential and multithreaded programs do this task with 1000 repetitions to maximize the load on the CPU under test.

Fig. 1 shows the performance comparison of different Intel based systems in terms of execution time and FFT size for sequential FFT program. It is evident from the bar graph that dual-core and quad-core systems offer better processing speed as compared to single core system. In case of sequential processing, core2duo processor shows better performance than core2quad processor as its CPU clock is higher i.e. 2.8 GHz.

Fig. 2 shows the performance comparison of different Intel based systems in terms of execution time and FFT size for multithreaded FFT program. Number of threads has been decided according to the number of CPU cores in case of multi-core systems. It is evident from the bar graph that as the FFT size increases; multi-core systems show significant improvement in execution time. It can be seen that core2quad processor gives the best performance as compared to other two processors by using multithreading approach.

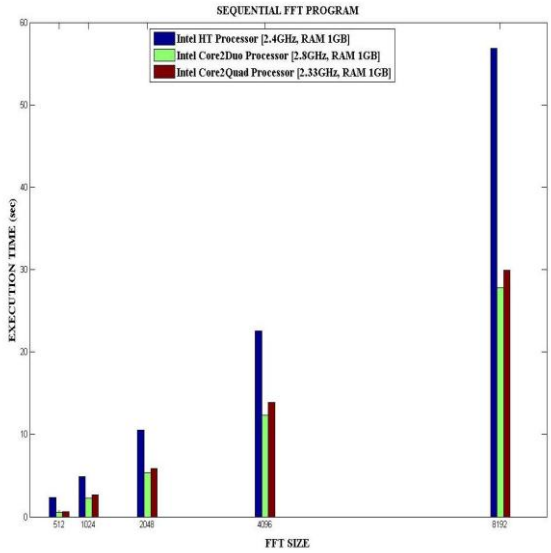

Fig. 1. Comparison of processors using sequential processing

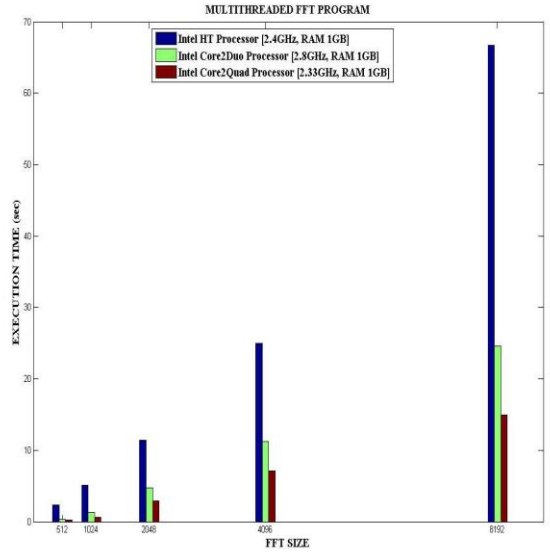

Fig. 2. Comparison of processors using multithreaded processing

V. Conclusions

Parallel programming techniques are common in modern software applications, and are used to enhance the scalability and performance of large jobs. Regardless of the development environment, multithreading presents many important advantages to software developers. Multithreaded applications can use their computer based systems to the fullest potential as companies continue to standardize on multithreaded operating systems or acquire increasingly affordable multiprocessor computers to boost performance. This paper presents our experience with multithreading on multi-core processors and convinces us that multithreading a single DSP application (e.g. FFT analysis) provides a relatively straightforward and very effective route to increased performance. Intel's current processor roadmap offers 1, 2, and 4 core processors. This means that a highly parallelized, multithreaded DSP application can be designed assuming a computer with 4 processor cores, each of which is HT enabled.

References

[1] T. Martinez and S. Parikh, "Understanding dual-processor," *Hyper Threading Technology and Multi-Core Systems*, Technical White Paper, 2005

[2] Intel Corporation, *Migrating Industrial Applications to Multi-Core Processors*, Technical White Paper, 2008

[3] National Instruments, *Differences between Multithreaded and Multitasking for Programmers*, Tutorial, 2008

[4] Intel Corporation, *Developing Multithreaded Applications*, Technical White Paper, 2005

[5] Intel Corporation, *Measuring Application Performance on Multi-core Hardware*, Article, 2007

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.