MICROPROCESSOR IMPLEMENTATIONS OF DISCRETE FOURIER TRANSFORM MACHINES

Paul Chow, Zvonko G. Vranesic, J.L. Yen

Department of Electrical Engineering
University of Toronto
Toronto, Ontario, Canada

ABSTRACT

When computing the Fourier Transform with a microprocessor, the speed and complexity of the algorithm which is used become especially important. The most frequently used algorithm has been the Fast Fourier Transform. More recently developed algorithms require fewer multiplications and about the same number of additions as the FFT. A comparison of these algorithms is made and some possible structures of machines are suggested. A description of a machine built to use one of the new algorithms is given and the problems which were encountered are discussed.

1. INTRODUCTION

Methods of computing the Discrete Fourier Transform (DFT) are always of interest. The first practical algorithm, the Fast Fourier Transform (FFT) [1] is still the one which is used most frequently. It allows computation of the DFT

$$X(k) = \sum_{n=0}^{N-1} X(n)W_N^{nk} \quad , \quad W_N = e^{j\frac{2\pi}{N}}$$

in a reduced number of multiplications, of the order of NlogN, where N is the number of points in the transform. The FFT is very suitable for implementation on a digital machine, particularly if special purpose hardware is used. Recent work by Winograd [2,3] has generated much interest in new algorithms which require fewer multiplications and about the same number of additions as the FFT. With current technology, a multiplication operation still takes significantly longer to perform than an addition, leading to the expectation that these new algorithms should provide some speed advantages over the FFT.

This work was sponsored by the National Science and Engineering Research Council of Canada under research grants A3951, A5280.

A comparison of these algorithms will be made, with an emphasis on how a machine might be structured in order to do the computations efficiently. Several possible structures are proposed for computing one of the new transforms and a machine which was built using one of these structures is described. The problems which were encountered are outlined and some solutions are given.

2. COMPARISON OF TRANSFORM ALGORITHMS

The FFT is based on the decomposition of the DFT into successively smaller DFT's. The most common form of the FFT is when the length of the transform is a power of 2. The DFT is reduced to 2-point transforms and the 2-point transform is computed repetitively with different data and constants. Keeping track of the data and the constants to use is fairly straight forward and well suited to a digital machine.

The Winograd Fourier Transform (WFT) [3,4] and the Good-Winograd Transform (GWT, called the prime factor FFT algorithm by Kolba and Parks [5]) have resulted from Winograd's work. They use combinations of a set of short high speed prime length transforms to form transforms of longer lengths. The WFT uses about 20% of the number of multiplications required by the FFT, with about the same number of additions. The GWT requires more multiplications than the WFT, but the number is still significantly smaller than the number required by an FFT.

The WFT and the GWT are much more difficult to compute than the FFT, mainly because the algorithms are not as simply organized as the FFT. Where the FFT is based only on the 2-point transform, the new transforms require several different structures to implement each of the short transforms that make up the final transform. For example, in a 252-point transform which uses the 4-point, 7-point and 9-point short transforms, there must be three different structures to handle the short transforms. These structures are not as simple as the 2-point transform.

The WFT and the GWT differ in the way in which they use the short transforms. The WFT nests the transforms and the GWT computes the short transforms independently. However, the data is ordered the same way for both transforms. Each of the indices of the sample points has a unique value when it is represented by the set of residues modulo the numbers used in the transform. The data is ordered according to these residues with the result that computing the ordering of the data is not as straight forward as in the FFT.

When comparing the algorithms there is a tradeoff of complexity versus speed. The FFT is relatively simple to implement but it uses more multiplications than either the WFT or the GWT. A question of interest is how much the complexity increases relative to the speed which is gained when one of the new algorithms is used.

3. IMPLEMENTING THE NEW ALGORITHMS

The architecture of a machine which can compute the WFT or the GWT efficiently is not clearly indicated. A machine which is to be dedicated to performing a transform at very high speeds must be hardwired controlled. Introducing some programmability will provide some flexibility, but also decrease the rate at which the machine can compute the transform.

In the FFT, the structure of the algorithm suggests that the basic unit of an FFT machine is the 2-point transform, composed of some adders, a multiplier and including a way of handling the complex operands. If a WFT or GWT machine is to be built along the same lines then there must exist a separate unit for each of the short transforms used. As mentioned previously, the short transforms are much more complicated in structure and they will therefore be more difficult to build. The only advantage they have over the FFT is that the constants used are not complex so that the multiplications will not have complex operands. Even as a hardwired machine, the FFT can be more flexible than these new algorithms. If the basic structure of the FFT machine cycles the data through one 2-point transform unit, then the length of a transform can be adjusted by adjusting the constants and the number of times the data is cycled through the unit. A WFT or a GWT would require the ability to change the short transform units which are being used. This is not as simple a problem to solve.

In a machine which is programmable, the complexity of the new algorithms is not as apparent. Instead of considering the problem as it applies to using a high level language, special purpose programmable hardware will be considered. In order to investigate the problems of implementing the new algorithms a microprogrammable machine structured to compute the GWT in real time was built. A description of the development of this machine, its performance and some conclusions are presented.

4. POSSIBLE STRUCTURES

The GWT was chosen as the algorithm to be used mainly because it is more modular than the WFT. This makes the implementation easier, because the short transforms can be computed in independent sets, instead of being mixed together in the nesting required by the WFT.

Several possible implementations using microprocessors and bit slices were examined. The main criteria used to judge these designs were the ease of implementation and the maximum possible throughput.

A microprocessor implementation with an added hardware multiplier is the most straight forward to build but with current microprocessor technology the machine would be quite slow. A possible configuration is shown in Fig. 1. There are basically two identical halves, each operating on one half of the data. The data is split into the real and imaginary parts. Some communication between the two channels is necessary to handle the case when a constant is imaginary requiring the resulting product to switch channels. An estimated sample rate is around 4-5 kHz for a 252-point GWT using the 8085 [6] microprocessor.

Using bit slices is a way of obtaining more speed. A possible configuration is shown in Fig. 2. This is basically the implementation of yet another computer with its own special instruction set. It is more complex than the microprocessor version because the writing of microcode and the decoding of instructions is required. An estimated sample rate is about 15 kHz maximum using the AM2903 [7] slices as a basis.

By combining a microprocessor with the bit slices the advantages of both technologies can be exploited. The reason for using bit slices is their speed and the reason for using a microprocessor is that it is simple and it does not have to be microprogrammed. When implementing the GWT, the two major problems are managing the data because of the required permutations and performing the computations. The microprocessor provides a simple means of manipulating the data, because it is relatively easy to program. It is then only necessary to microprogram the bit slices to compute the transforms, which is a reasonably easy task to achieve.

The machine that was built contains a microprocessor (an LSI-11 [8]) and a microprogrammable bit slice unit (called FASTOR), under the control of the microprocessor. Its structure is shown in Fig.3.

5. THE LSI-11 AND FASTOR

Using the new algorithms, the problem of handling data becomes significant because of the data permutations required. The host LSI-11 is used only to manage the data and send instructions to FASTOR. All of the arithmetic computation is done by FASTOR under the control of the LSI-11. The basic structure of FASTOR is shown in Fig. 4.

FASTOR has been designed to be fairly simple to implement. The main sections are the central processor, the memory and the microprogram control unit. Communication with the LSI-11 is done through the registers, the instruction register and some status bits.

The central unit uses four AM2903 bit slices to form a 16-bit wide processor. The register files have been organized with two separate files of 32 registers plus the 16 on chip registers. This allows the loading and unloading of one set of registers by the LSI-11 at the same time that FASTOR is using the other set. This gives FASTOR 48 double-port general purpose registers. Double-port registers are used along with a three address architecture so that many of the operations can be performed in a single read-modify-write cycle.

FASTOR executes microcode loaded into a RAM control store. An instruction sent to FASTOR is an address in the microprogram at which FASTOR is to begin execution. This is loaded into the microprogram controller when FASTOR receives a signal from the LSI-11 to start. Therefore, there is no requirement for an instruction decoder which makes the implementation much simpler.

The machine is organized so that the LSI-11 handles the input and output of data and manages the data buffers. There are three data buffers required for real time processing. These are the input buffer, the output buffer and the current work area. The LSI-11 program maintains these buffers and swaps pointers when the input buffer is full in order to start the next transform.

The LSI-11 is also responsible for controlling FASTOR and loading its registers. The system has been structured so that FASTOR can be used to compute any of the short transforms once the data has been loaded in its registers. The LSI-11 signals FASTOR to start by giving it a GO signal and an instruction to tell it which transform to perform. When FASTOR is finished, the LSI-11 is signalled so that a new transform can be started. The LSI-11 program must therefore be able to keep track of the sets of points which make up the short transforms.

6. RESULTS

The machine which was built is capable of computing the 252-point GWT with a sample rate of 6 kHz. FASTOR can handle a sample rate of about 18 kHz but the LSI-11 is not capable of handling the data at this rate. The data permutation is implemented by using a set of tables containing the permuted addresses. This requires the use of a fairly slow addressing mode, which combined with the amount of data that must be moved in and out of FASTOR's registers, precludes faster operation. The major problem is how to organize the data in memory so that computing the addresses can be done easily. Microprogramming the short transforms is not very difficult.

The speed can be improved by loading and unloading FASTOR's registers using direct memory access (DMA) into main memory. However, this DMA transfer is not as straight forward as the normal block transfers that are usually associated with DMA. The interface must be able to compute the addresses of the data required, because the data is not necessarily stored sequentially in memory.

Possible schemes for this interface include having the address tables stored as part of the interface. The tables can be initialized by the controlling program. The main disadvantage of this method is that it requires a lot of memory. A slower variation of the method is to give the controller the starting address in memory where the table is stored. The controller first accesses the table by DMA to get the required addresses. A small memory is required to store these addresses so that they can be used to return the data. In both of the above schemes, the LSI-11 is relieved of the task of loading and unloading FASTOR's registers. Its main control over FASTOR is to tell it which data table to use and which transform to perform on the data. The remainder of the time can be spent doing the input and output.

Another possible way of building the DMA interface is to give it the intelligence to compute the required addresses. By arranging the data in memory in a particular order, and given the first address (base address), it is possible to access the data for a short transform by stepping through memory with a fixed address increment determined by the prime factors in the transform being computed. For example, in a

252-point transform, the data can be organized so that for the 9-point transforms the base addresses are 0,1,...,6, 63,64,...,69, 126,127,...,132. For the base address of 1, the nine points are at 1,8,15,22,...,57. Calculating the base addresses is also fairly easy. Therefore, with an adder and some controls it is possible to build an interface which can compute the addresses required to access the correct data.

In all of the schemes mentioned above for improving the speed of performing the transforms, the common characteristic is a DMA interface which has the intelligence to determine the addresses required to access the permuted data. This reduces the number of memory accesses required by the LSI-11 by 60% which will improve the speed significantly.

7. CONCLUSIONS

The experience gained in this work has shown that the effort to reduce the number of multiplications has made the accessing of the data much more difficult and the structure of the algorithms much more complex in comparison to the FFT. In an implementation where it is possible to program the new algorithms, it is much easier to cope with these problems, but for a special purpose hardware implementation these factors are important considerations because it is always desirable to build a machine with as simple a structure as possible to make debugging much easier.

In comparing the algorithms, the GWT was found to be easier to implement than the WFT because it is more modular. Several possible machine structures were studied in an effort to find one which would be easy to build and suitable for performing the GWT.

The selected design uses a microprocessor (an LSI-11) controlling a unit (FASTOR) built using bit slices. The LSI-11 is programmed to manipulate the data and FASTOR is used to do the arithmetic computations. It was found that the slowest part of computing the transform was the LSI-11. This was because of the amount of data which was being moved and the difficulty in accessing the data. Several ways of handling the problem by the use of DMA interfaces were discussed. These interfaces would be capable of computing the addresses required to access the permuted data.

The new algorithms do provide a faster way of calculating the DFT compared to the FFT even though the programs are longer and more complex. By providing a fast processor to do the arithmetic it was found that the speed of computing a transform using a microprocessor becomes limited by the number of memory accesses required and the problem of calculating the addresses to do the accesses. A future direction might be to study the problem of accessing the data when these transforms are used.

8. REFERENCES

1. J.W. Cooley and J.W. Tukey, "An Algorithm for the Machine Computation of Complex Fourier Series," Math. Computation, Vol.19, Apr. 1965, pp.297-301.

2. S. Winograd, "On Computing the Discrete Fourier Transform," Proc. Nat. Acad. Sci., U.S.A., Vol. 73, No. 4, April 1976, pp. 1005-1006.

3. S. Winograd, "On Computing the Discrete Fourier Transform," IBM T.J. Watson Res. Ctr., Yorktown Heights, N.Y., IBM Res. Rep., RC-6291, Nov. 1976.

4. H.F. Silverman, "An Introduction to Programming the Winograd Fourier Transform Algorithm (WFTA), IEEE Trans. Acoust. Speech, Signal Processing, Vol. ASSP-25, April 1977, pp. 152-165.

5. D.P. Kolba and T.W. Parks, "A Prime Factor FFT Algorithm using High-speed Convolution," IEEE Trans. Acoust. Speech, Signal Processing, Vol. ASSP-25, August 1977, pp. 281-294.

6. "MCS 85 User's Manual," Intel Corporation, Santa Clara, Cal., 1977.

7. "Am2903 Four-Bit Bipolar Microprocessor Slice, Am2910 Microprogram Controller Technical Data," Advanced Micro Devices, Inc., Sunnyvale, Cal., January 1978.

8. "Digital Microcomputer Handbook," Digital Equipment Corporation, Maynard, Mass., 1977.

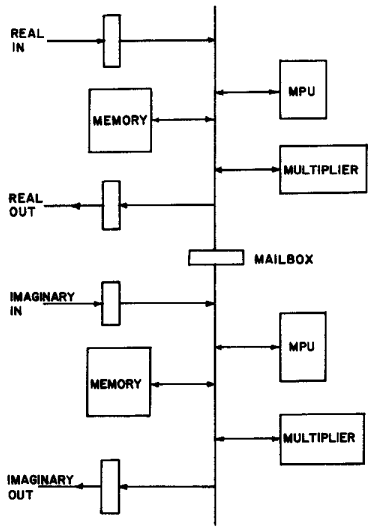

Fig. 1 Using Microprocessors

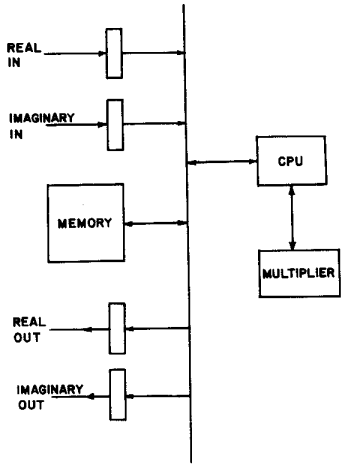

Fig. 2 Using Bit Slices

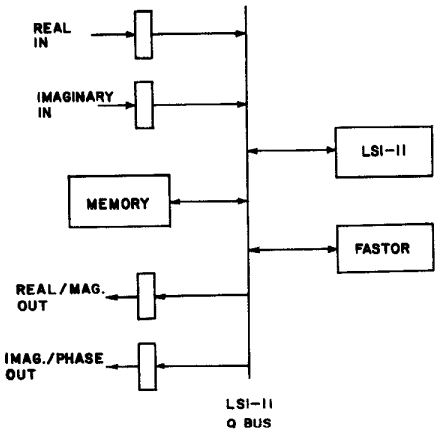

Fig. 3 Using an LSI-11 and FASTOR

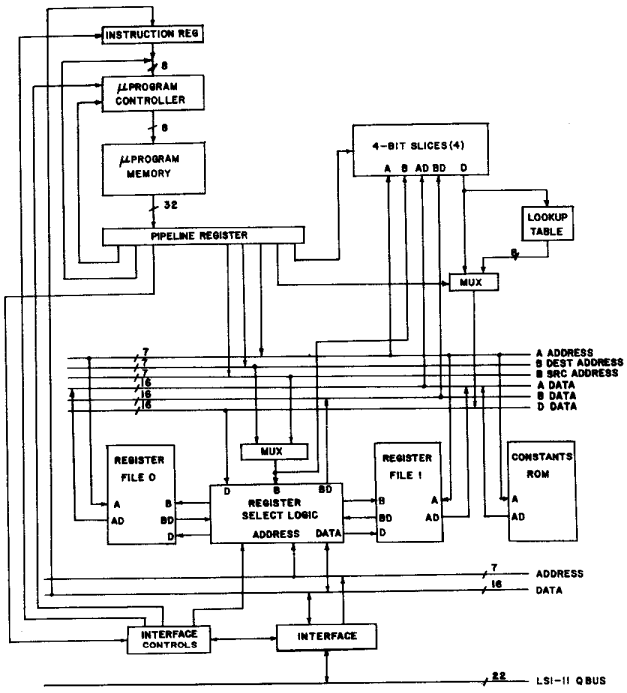

Fig. 4 Basic Structure of FASTOR