Reducing Memory References for FFT Calculation

Mokhtar A. Aboleaze
Dept. of Computer Science and Engineering
York University
Toronto, ON. Canada
aboelaze@cs.yorku.ca

Ayman I. Elnaggar
Dept. of Electrical and Computer Engineering
Sultan Qaboos University
Muscat, Oman
ayman@squ..edu

Abstract

Fast Fourier Transform (FFT) is one of the most widely used algorithms in digital signal processing. It is used in many signal processing and communication applications. many of the FFT operations are performed in embedded systems. Since Embedded systems is very small processors used in almost all type of appliances from microwave ovens to cars, and many embedded systems are portable and depend on small batteries for power; low energy design is extremely important in embedded systems design. One of the major energy consumption sources in any processor is memory access. Memory access requires more energy than almost any operation in a DSP (Digital Signal Processor), or embedded processor, reducing memory access plays a very important role in reducing energy consumption. In this paper we concentrate on the energy consumption in memory in calculating FFT. we compare between three different techniques in calculating FFT with reference to energy consumption in memory access. We also investigate the effect of the number of registers in the CPU on reducing energy consumption in memory access.

1. Introduction

Fast Fourier Transform (FFT) is probably one of the most used signal processing algorithms in the world. FFT is used in digital communication and in general in digital signal processing and is widely used as a mathematical tool in different areas. In the next few paragraphs, we briefly mention some of the applications of FFT.

FFT is used to reduce the computational time required for solving the problem of electromagnetic scattering from wire antennas and conducting rectangular plates [18]. It is also used in solving a system of Toeplitz normal equation [19]. It is also used in optimal frequency acquisition and measurements in Search and Rescue Satellite Aidded Tracking (SARSAT) [12]. In interpolation techniques for resampling of correlation pulse signals departing from a small number of Nyquist samples [3].

FFT is also used in spectral estimation in [7] and [9], in single tone detection and frequency estimation in [2], and in increasing object detection in radar systems [17]. Also FFT is used in Orthogonal Frequency Division Multiplexing (OFDM) which is the basis for Multi Carrier Code Division Multiple access (MC-CDMA), and its direct spread spectrum version MC-DS-CDMA [6] and [8].

Windowed FFT is used in electric power quality assessment in [11]. While in [5] the authors introduced the generalized sliding FFT to efficiently implement the hopping FFT. Then they showed how to use it to implement the block LMS adaptive filter. Finally the FFT was used in 3-D induction well logging probelm where it could be used in characterization of oil reservoirs.

Power consumption is a very important factor in the design of special purpose as well as general purpose processors [14]. For wireless devices, power plays a crucial role since the device operates on a battery with a limited power supply capability. Obviously that require hardware to use as little power as it possibly could to perform the job at hand. For general purpose processors, increasing power consumption leads to sophisticated cooling techniques which both increase the price and reduce the reliability of the processor.

One of that main sources of energy consumption in any processor chip is the memory (or the cache if there is a cache). It was reported in [16] that instruction cache reference amounts to 43% of the total power consumption of the chip. For small embedded processors without cache, and since the main memory consumes more energy than cache, that figure could be higher in small processors without cache. Minimizing the number of times the processor goes to the memory, helps to reduce the energy consumption in the chip. While there is nothing we can do to reduce the memory access to get instructions (assuming a gernal purpose processor architecture); data access can be reduced in two different ways. The first is using algorithms that require less memory access (reusing the accessed element as many times was possible before discarding it, or reordering memory access). The second is by using a large set of registers, thus accessed elemenst can be temprarily stored in registers for further processing without memory access.

In this paper, we present a comparative study for the memory access in calculating in place FFT. We consider access to both data points and coefficients. We investigate three different techniques to calculate the FFT and show the number of memory references of the different techniques which is an indication of the power dissipated in the memory.

The organization of this paper is as follows. In Section 2, we present a brief overview of FFT techniques. Section 3 discusses related work. In section 4, we present our comparative study for the number of memory references in calculating in-place FFT using a variable number of registers. Section 5 is a conclusion and future work.

2. FFT Algorithm

The Discrete Fourier Transform (DFT) of an N data samples is defined by

$$X_k = \sum_{i=0}^{N-1} x_i \cdot W_N^{ik} \qquad (1)$$

Where $W_N = e^{-j(2\pi/N)}$, are the N^{th} root of unity, x is the original sequence, and X is the FFT of x, $k=0,1,2,...,N-1$ In general, both X, x and W are complex numbers. This can be calculated using matrix-vector multiplication, where W_N^{ik} are arranged as *N-by-N* matrix. That method requires $O(N^2)$complex multiplications. A faster way to calculate the DFT is the Fast Fourier Transform (FFT). FFT algorithm works by dividing the input points into 2 sets (k sets in general), calculate the FFT of the smaller sets (recursively), and combining them together to get the Fourier transform of the original set. That will reduce the number of multiplications from $O(N^2)$ to $O(N \log N)$. Figure 1. shows the 8-points decimation in time FFT. The 8 points are divided into 2 sets of 4 points each, then caculating the FFT for these 2 sets of 4 points each, then combining these 2 sequences to produce the 8 points FFT. The 4-points FFT is recursively done in the same way. The 2 points FFT is known as the butterfly operation, and is shown at the bottom of Figure 1.

As we can see from Figure 1, Both data and coefficients are accessed from the memory. Intermediate calculations are done, then they are stored in the same memory location they were accessed from (in place FFT calculation). By the end of the calculations (the end of the log_2N stages), the memory contains the FFT sequence of the input sequence. As we mentioned earlier, memory access carries with it a heavy price from the energy consumption point of view. Minimizing the number of memory access goes a long way in reducing the energy consumed in calculating the FFT.

For example, in the first stage, we access both *x(0)*, and *x(4)*. That results in calculating new values for *x(0)*, and *x(4)*. Then we proceed to access the rest of the data points. Then at the beginning of the second stage, again, we access *x(0)*, and *x(2)*. If the previous values calculated for *x(0)*, and *x(2)* are stored in a pair of registers, then we don't need to go to the memory in order to access them again. If these values are not stored in registers, then we have to go to the memory to fetch them. the same could be said about the twiddle factor access.

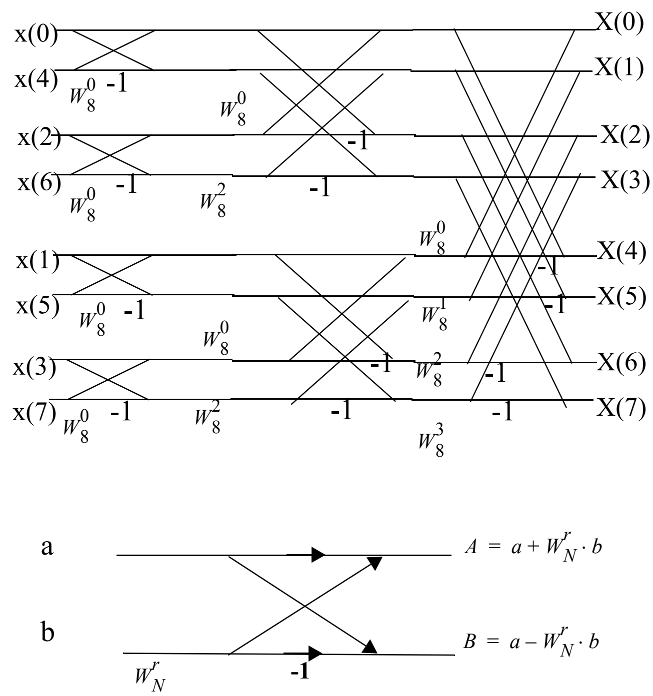

Figure 1. 8-points decimation in time FFT

Whether the required data will be in the memory or in registers depends on the number of registers in the CPU, the architecture of the CPU, the instruction set of the CPU, and the compiler used in generating the machine code. Here we concentrate on the number of available registers in the CPU, we also will make some assumptions regarding the compiler.

In this paper, we calculate the number of memory access in calculating FFT for different algorithms, and for different values of N. Some of these algorithms are designed specifically in order to reduce the number of memory access (by rearranging the calculations), while other are designed to minimize the number of calculations, or to simplify the code.

In the next section, we review some of the previous work in minimizing power consumption for FFT calculations. The power reduction falls in three main categories, reducing memory access for data reference, reducing memory access for address generation, and reducing the number of arithmetic calculations in order to save energy.

3. Related Work

Low power design for digital signal processing in general, and FFT in particular has attracted a lot of attention. Low power could be implemented on many levels. The algorithm level, the architecture level, and the hardware level.

At the algorithm level, many attempts have been done to minimize power consumption, higher radix FFT, mixed radix FFT [20], Other attempts are made on both the architecture level and the circuit level.

In [10], the authors considered the problem of coefficients address generation for special purpose FFT processors. They designed the hardware required for access of the Fourier coefficients from the coefficients memory. They showed that their scheme can result in power saving of 70-80% compared to Cohen's scheme (note that saving in the energy consumption of the address generation unit only, not the entire processor power).

In [15], the authors proposed an address generation scheme for FFT processors. Their proposed hardware complexity is 50% less than Cohen's scheme. Also their scheme activate only half the memory (they used memory banks) thus saving more energy than previously known techniques.

A low-power dynamic reconfigurable FFT fabric was proposed in [22]. The processor can dynamically be configured for 16-points to 1024-points FFT calculation. The overhead for dynamic configuration is minimal while the power reduction compared to general purpose reconfigurable architecture is in the range 30-90%.

In [13] the authors proposed a novel FFT algorithm to reduce the number of multiplications as well as the number of memory accesses. Their algorithm depends on re-arranging the computation in the different stages of the Cooley-Tukey algorithm in order to minimize the Twiddle factor access. Their algorithm clusters together all the butterfly operations that use a certain Twiddle factor, access that twiddle factor to perform the operations, and never access it again. That result in 30% reduction in twiddle factor access compared to the conventional DIF FFT. They also implemented their algorithm on TI TMS320C62x digital signal processor. Their implementation shows a significant reduction in the memory access on the TI TMS320C62x digital signal processor.

4. Comparison

The number of memory access depends on many factors. The first is the produced machine code. The machine code depends of course on both the high level source code (if written on high language source code; otherwise the assembly code), and the compiler used. It also depends on the instruction set of the target processor and the number of available registers in the processor.

The compiler plays a very important role in the speed of the executed code. An efficient use of the resources (including the available registers) can speedup the execution.

In this paper, we simulated three different algorithms for FFT. We did not produce the assembly code, but rather got the data access pattern from the C code. As mentioned before there is no direct correspondence between the data access pattern and the memory access pattern. data access could be directed to memory if the data are in the memory, or a register if the requested data reside in a register.

Another important factor is writing to the memory. If a data element is changed, we can proceed to write the changes to the memory, or we can keep it in a register to be used in the computations and written to the memory either at the end of the code, or when we need to use the register to store another data item.

In this paper, we assumed that the compiler uses the available registers to access and store the data. Once a data element is stored in a register, it stays there until the compiler has to use the register to store new data. In that case the registers are released in a FIFO fashion. We also assumed a register-register architecture, in which the operands are assumed to be in a register which is typical of a RISC architecture. That puts an upper bound on the performance, and depends to a great extent on the compiler to produce an optimal code.

The first algorithm is the regular radix 2 DIF FFT, the pseudo code in Figure 2. can be found in [1]. note that the last stage is separated from the rest of the stages since we do not need to multiply by the twiddle factors in that stage, by separating the last stage from the rest of the stages, we save on memory access as well as on multiplications.

Radix-4 FFT algorithm is shown in Figure 3. Also taken from [1]. Note that in both figures, that is a pseudo code where the elements of x are complex numbers. In reality it will take more than a single addition/multiplications to represent the addition or multiplications of a complex number.

The third algorithm we considered is the one shown in [13], that minimizes the Twiddle factor memory access.

```
fft_dif(x[],m)
n=2^m;
for(i=m; i>1; i--) {
   m1=2^i;
   mh=m1/2;
   for(j=0;j<mh;j++) {
      e=exp(2*PI*i*j/m1);
      for(r=0; i<n; i=i+m1) {
       u=x[r+j]; v=x[r+j+mh];
       a[r+j]=u+v;
       a[r+j+mh]=(u-v)*e;
       }
      }
   }
for(r=0;r<n;r=r+2) {
   x[r]=x[r]+x[r+1]; d a[r+1]=a[r]-a[r+1];
   }
```

Figure 2. DIF FFT Algorithm

Table 1. shows the number of memory access for the three above mentioned algorithms for different number of registers ranging from 4 to 20. We also considered diffent sizes for FFT sequences with length $N = 2^n$ points. One can see that for architectures with a small number of registers, the regular Radix-2 FFT performs best over a wide range of FFT sizes (from 16 to 1024). For architectures with large number of registers, Radix-4 FFT performs the best over the same range of sizes. For architecture with a moderate number of registers (8,12,16) Radix-2 FFT and the reduced memory access method perform equally good with a slight edge for the simple radix-2.

In [13], the authors compared between their algorithm and the Radix-2 FFT running on TI TMS320C62x and show that their algorithm require less memory access than Radix-2. We believe that this may be because of the VLIW architecture of the processor, the number of registers available, or the compiler used. However, we our work shows that with a very good compiler, radix-2 and 4 FFT can outperform the reduced memory access algorithm.

Table 1. The number of memory accesses for different number of registers, and FFT size =2^n

# of Reg	4	8	12	16	20	n
Rad2	504	380	275	266	256	
Rad4	567	390	339	215	187	
Reduced	645	364	279	253	248	n=4
Rad2	1312	988	711	676	672	
Rad4						
Reduced	1681	942	731	673	668	n=5
Rad2	3232	2428	1731	1652	1648	
Rad4	3407	2350	2067	1351	1187	
Reduced	4137	2352	1795	1657	1652	n=6
Rad2	7680	5756	4071	3892	3888	
Rad4						
Reduced	9817	5600	4243	3913	3908	n=7
Rad2	17792	13308	9347	8948	8944	
Rad4	18175	12542	11091	7335	6483	
Reduced	22713	12992	9779	9001	8996	n=8
Rad2	40448	30204	21095	20212	20208	
Rad4						
Reduced	51577	29568	22131	20329	20324	n=9
Rad2	90624	67580	46979	45044	45040	
Rad4	90879	62718	55635	37031	32851	
Reduced	115449	66304	49395	45289	45284	n=10

```
fft_rad4
n=2^ldn;
for(ldm=ldn; ldm>1;ldm=ldm-2) {
   m=2^ldm; mr=m/4;
   for(j=0;j<mr;j++) {
      for(r=0;r<n;r=r+m) {
       u0=a[r+j]; u1=a[r+j+mr];
       u2=a[r+j+2*mr]; u3=a[r+j+3*mr];
       t0=u0+u2+u1+u3;
       t1=u0+u2-u1-u3;
       t2=u0-u2+(u1-u3)$
       t3=u0-u2-(u1-u3)$
       a[r+j]=t0;
       a[r+j+mr]=t2*W;
       a[r+j+2*mr]=t1*W
       a[r+j+3*mr]=t3*W
       }
   }
}
```

Figure 3. Radix 4 DIF FFT

5. Conclusions

In this paper, we compared between three different algorithms in the number of memory access required for FFT calculations. We generated data trace from the C code and we use approximate measures to estimate the number of memory access required to complete the computations assuming a variable number of registers available to the compiler. We did not assume a specific compiler, just the data trace and a FIFO register set to be used with the compiler. Our work shows that although the results depends on the number of registers, and the problem size, however the simpler Radix-2 FFT performs well compared to the other 2.

6. References

[1] Arndt, J. "Algorithms for Programmers", could be found at www.jjj.de/joerg.html August 2005.

[2] Chan, Y.T.; Ma, Q.; Inkol, R.; "Evaluation of various FFT methods for single tone detection and frequency estimation". *IEEE Canadian Conference on Electrical and Computer Engineering*. Vol. 1 25-28 May 1997 pp 211-214

[3] Coenen, A.; De Vos, A. ""FFT-baed interpolation for multipath detection in GPS/GLONASS receivers". *Electronic Letters*, Vol. 29, No. 19. Sept. 1992 pp 1787-1788.

[4] Cohen, D. "Simplified control of FFT hardware", *IEEE Transactions on Accousitc, Speech, and Signal Processing*. Vol. 24, No. 6, Dec. 1976 pp577-579

[5] Farhanq-Boroujeny, B.; and Gazor, S. "Generalized Sliding FFT and its application to implementation of block LMS adaptive filters". *IEEE Transactions on Signal Processing*. Vol. 42, Issue 3. March 1994. pp 532-538

[6] Frederiksen, F. B.; Prasad, R. "An Overview of OFDM and related techniques towards development of future wireless multimedia communications". *IEEE Radio and Wireless Conference RAWCON2002*. 11-14 Aug. 2002. pp 19-22

[7] Gan, R. Eman, K.; Wu S. "An extended FFT algorithm for ARMA spectral estimation", *IEEE Transactions on Accoustic Speech, and Signal Processing*, Vol. 32, No. 1 Feb. 1984 pp 168-170.

[8] Ghorashi, S. A.; Allen, B.; Ghavami, M.; Aghvami, A. H.; "An overview of MB-UWB OFDM". *IEE Seminar on Ultra Wide Band Communications Technologies and System Design*. * July 2004. pp 107-110.

[9] Gough, P. T. "A fast spectral estimation algorithm based on the FFT", *IEEE Transactions on Accoustic Speech, and Signal Processing*, Vol. 44. No. 8. August 1996 pp 1317-1322.

[10] Hasan, M.; Arslan, T. "Coefficients memory addressing scheme for high performance processors" *Electronic letters*. Vol. 37, No. 22. Oct. 2001 pp1322-1324

[11] Heydt, G. T.; Field, P. S.;, Pierce, D.; Tu, L.; and Hensley, G. "Applications of the windowed FFT to electric power quality assessment" *IEEE Transactions on Power Delivery.* Vol. 14, Issue 4. Oct. 1999 pp 1411-1416

[12] Holm, S. "Optimum FFT-based frequency acquisition with application to COSPAS-SARSAT" *IEEE Transactions on Aerospace and Electronic Systems*. Vol. 29, No. 2 April 1993 pp 464-475

[13] Jiang, Y.; Zhou, T.; Tang, Y.; Wang Y. ""Twiddle-factor-based FFT algorithm with reduced memory access" *Proc. of the 19th IEEE International Parallel and Distributed Processing Symposium IPDPS2002* April 2002. pp 70-77.

[14] Lahiri, K.; Raghunathan, A.; Dey S.; Panigrahi, D.; "Battery-driven system desig: A new frontier in low power design". *proceeding of the 7th Asia and South Passific Design Automation Conference ASP-DAC 2002*.7-11 jan. 2002, pp 261-267.

[15] Ma, Y.; Wanhammar, L. "A Hardware efficeint control of memory addressing for high-performance FFT processors" *IEEE Transactions on Signal Processing* Vol. 48, No. 3 March 2000 pp917-921

[16] Montanaro, J. et al "A 160-MHz, 32-b 0.5-W CMOS RISC microprocessor" *Proc. IEEE Intl. Solid-State Circuit Conference.*, Feb. 1996 pp 214-229.

[17] Prasad, N.; Shameem, V.; Desai, U. B.; Merchant S. N.; "Impreovement in target detection performance of pulse codded Doppler radar based on multicarrier modulation with fast Fourier Transform (FFT)". IEE Proceedings on Radar, Sonar, and Navigation. Vol. 151. No. 1 Feb. 2004 pp11-17.

[18] Sarkar T.; Arvas, E.; Rao, S. "Application of FFT and conjugate gradient method for the solution of electromagnetic radiation from electrically large and small conduction bodies," in *IEEE Transactions on Antennas and Propagation*. Vol. AP-34, No. 5, May 1986 pp 635-640.

[19] Yarlagadda, R.; Babu, B. "A note on the application of FFT to the solution of a system of Toeplitz normal equation", *IEEE Transactions on Circuits and Systems*, Vol. 27, No. 2 Feb. 1980 pp 151-154

[20] Yeh, W.-C.; Jen, C.-W. "High-speed and low-power split radix FFT" *IEEE Transactions on Signal Processing*. Vol. 51, No. 3 March 2003 pp 864-874.

[21] Zhang, Z. Q.; and Liu, Q.; H. "Applications of the BCGS-FFT method to 3-D induction well logging problems. *IEEE Transactions on Geoscience and Remote Sensing*. Vol. 41, Issue 5. May 2003. pp 998-1004

[22] Zhao, Y.; Erdogan A.; Arslan T. "A low-power and domain-specific reconfigurable FFT fabric for system-on-chip applications" *Proc. of the 19th IEEE International Parallel and Distributed Processing Symposium IPDPS05* April 2005 pp 169a-172a