An Efficient FFT-Mapping Method Based on Cache Optimization

ZHU Liang, LIU Tengfei, GAO Lining*, LIU Feng, JIN Ye, SUN Chenguang
School of Information and Electronics, Beijing Institute of Technology
Beijing, China, 100081
Email: gaolining@bit.edu.cn

Abstract—Fast Fourier Transform (FFT) is an important technology in real-time signal processing system, which means the efficiency of FFT algorithm mapping to hardware system has very important significance. At first, we aim at the FFT execution process on processor. And then analyze the memory access process based on cache mechanism, and get that the cache hit directly affects FFT execution time. Hence, an efficient mapping method is come up with, which splits long FFT into multiple segments to make sure every segment is shorter than the cache capacity. Accordingly, the cache hit rate will be improved, correspondingly, the execution efficiency will be better finally. In the end, the new method is experimented on the ADI’s TS201 digital signal processor, and the result shows that the execution time of FFT is improved greatly.

Key words—Fast Fourier Transform (FFT); Cache; Hit Rate

Introduction

FFT (Fast Fourier Transform) is one of the key technologies in modern radar signal processing. Since the FFT is an important part of the radar system, a hard real-time processing system, the processing should be done in a limited amount of time. Therefore, an efficient FFT processing is indispensable.

Contemporarily, there are mainly two methods to implement the efficient FFT algorithm[1]. One of them is by ASIC or FPGA. Yet this way is lacking of flexibility and the development cycle is not short enough. At present, the other more popular method is programmatic implementation based on the processor. This method who achieves FFT algorithm through program code has a very good performance on the features of flexibility and short development cycle. With the increasing of processors’ clock speeds, the execution time of FFT algorithm has been shortened. Thus it can be seen that the method of programming based on the processor has more advantages. However, there is one problem that the structure of processors, which is not designed for FFT algorithm, is fixed, and they don’t match well.

To achieve the targets above, we first analyze the structure of the superscalar processor, parameterize factors associated with the processing, and establish a superscalar processor execution model. And then based on the model, we choose a radix-2 FFT algorithm as an example, to analyze the disadvantage of the conventional FFT algorithm mapping method, and propose solutions of to these problems, that is the splittable FFT-mapping method. This mapping algorithm makes full use of processing resources on the processor. Combined with the features of cache, it greatly reduces the memory accessing time. Therefore, it makes maximum use of processor performance. Finally, based on the mapping algorithm, the radix-2 FFT algorithm is implemented on the ADI’s TS201 digital signal processor, and achieves satisfied results.

1. The structural features of superscalar processor

Superscalar pipelining techniques are used in modern processors. It is a parallel pipeline, which means execution of an instruction is divided into several parts, including addressing, decoding, executing, fetching and writing back. Then through the pipeline technology, it enables multiple instructions at different stages of processing at the same time. With the parallel method of time, it has improved the throughput of the processor. And the processor may use different execution units in running different function instructions, so that the processing of a plurality of instructions can be started in each machine cycle, making the execution efficiency to be improved[1-2]. Figure 1 shows the operating principle of a superscalar processor.
The figure 1 shows the schematic diagram of a superscalar pipeline with four instructions executing. And it is divided into six stages: 1. Reading. This stage is mainly responsible for obtaining execution codes from memory; 2. Decoding. The main task of this stage is to translate execution codes into machine readable operation codes; 3. Scheduling. The task of this stage is assigning the codes to the functional units by corresponding functions; 4. Executing. This stage is to complete instructions’ functions; 5, 6. Completing and exiting. These two stages take charge of modifying the machine state after completing the instructions, and making sure the instructions completed in order. In a superscalar processor’s architecture, stage 1, 2, 3, 5, 6 follow the same mechanism for all the instructions. Thus these five stages often work in a centralized processing way. As for stage 4, different instructions have different ways for executing. Hence this stage is designed as distributed processing\(^{(1,2,3)}\).

As mentioned above, the superscalar processor divides the instructions into their functional units, improved processing capacity a lot. Such description is based on the assumption that execution time among each stage are all the same. However, in practical application, the data accessing time would be longer, because the core speed is much lower than the memory accessing speed. It would break the pipeline runs, reduce processing efficiency\(^{(9)}\). To solve this problem, the core processor was introduced with considerable speed cache. Cache takes advantage of locality principle, to store the recently accessed data in cache. Hence when accessing the memory, it would query cache first. If cache hits, the data will be got directly, corresponding, the response will be faster; If cache misses, it will start the memory access, and the accessing result will be cached\(^{(4,5,6)}\). By setting a reasonable cache capacity, the problem of speed mismatch can be solved, and the processor can give full play to its performance.

In conclusion, the modern superscalar processor architecture adopt parallel pipeline to increase throughput and processing efficiency. To solve the problem of mismatch between the memory and the core, cache is brought in. Through the way mentioned above, we can improve the processor’s capacity. In practical application, the real-time capacity of program execution is related to two factors: one is the processor’s processing capacity and the other is the mapping of the algorithm on the processor. In consideration of this, a suitable algorithm is important to make the execution process fit the superscalar architecture feature. Hence it can maximize the processing capacity, and then implement the real-time processing. Next, we will model the execution process for fitting the modern superscalar processor, and analyze the process of FFT based on the model. And finally, propose a new effective method of mapping for FFT.

2. Effective FFT mapping method based on superscalar processor

As mentioned previously, different kinds of instructions are running in different execution units. Thus the processor can be abstracted as a set of several interrelated execution units. Define \(s\) as the number of execution units, \(v_i\) as the execution speed of arbitrary unit \(i\). Then the speed of all the units can form gather \(V = \{v_1, v_2, \ldots, v_s\}\). (The definitions of execution speed are different in different units: the memory units are defined by access bandwidth; and the core units are defined by the number of operations per unit time.) Accordingly, in order to study the real-time capacity, research the relationship between the elements of gather \(T\), we will continue our discussion. Before the discussion, we define the parameter as follows:

[1] Denote by \(\alpha_{ij}\) the degree of overlapping of execution time, which means the percentage of the overlap between task \(j\) and task \(i\) from task \(j\) (The overlapping reflects the parallelism between each execution units. And the percentage is correlated with the mapping algorithm and the processor resources).

[2] Denote by \(t_{(i,j)}\) the total execution time of task \(i\) and task \(j\).

According to definitions above, the total execution time is

\[ t_{(i,j)} = t_i + (1 - \alpha_{ij}) \times t_j. \] (1)
If we regard task \( i \) and task \( j \) as a big entire task \( I \), the total execution time of task \( I \) and one another task in the gather \( C \) is

\[
t_{ir} = t_{i,j} + (1 - \alpha_{i,j}) t_{r}.
\]

(2)

Using recursive algorithm, we can get the total execution time of the tasks as

\[
t = t_{i} + (1 - \alpha_{i}) t_{i} + (1 - \alpha_{2}) t_{2} + \cdots + (1 - \alpha_{n}) t_{n}.
\]

(3)

Formula 3 shows the relationship between the execution time of each task and the total time. Since the elements in gather \( t \) have parity, we can let \( t_{i} \) be the largest one for convenience. Then we can get the relationship as follows:

\[
\begin{align*}
\alpha_{i} & \leq 1 \\
\alpha_{i} & \leq 1 \\
\alpha_{i} & \leq 1 \\
\alpha_{i} & \leq 1 
\end{align*}
\]

(4)

From formula 3 and 4, the execution time will be the minimum when the value of overlapping is 1.

As above, the model is based on execution time of processor. Running on the processor, the FFT algorithm can be divided into four tasks: multiplication, addition, subtraction and writing. In the modern processors, these four tasks run respectively on multiplier, adder, subtractor and memory controller. By rationalizing the pipeline, the FFT execution sequence will be as the figure below (The figure shows that part of the processing is in cache, other part is in memory.)

Figure 2. The FFT execution pipeline based on modern processor

The pipeline is built by three stages: Filling, pipelining and emptying [7]. At the pipelining stage, all the processing including multiplication, addition and subtraction are all hiding in the memory, and partly hiding in filling and emptying stages. Thus the memory execution time can be regarded as the maximum execution time. Now we define the multiplication processing as task 2, addition processing as task 3, subtraction processing as task 4, and the number of times of butterfly computation as \( n \). Hence we can get the degree of overlapping during the stages:

\[
\begin{align*}
\alpha_{i} & = 1 - \frac{3}{2n} \\
\alpha_{i} & = 1 - \frac{3}{2n} \\
\alpha_{i} & = 1 - \frac{3}{2n}
\end{align*}
\]

(5)

The formula 5 indicates that the degree of overlapping is inversely proportional to the number of butterfly computation times \( n \). When \( n \) is large enough, we can infer the value of the degree of overlapping is 1, which means that core calculations are all hiding in the memory access. For the radar system, the number of FFT points can always be thousands, so be the number of times of butterfly computation in every stage. Therefore, memory access becomes the key issue. From the second section, we know the speed of memory access is lower than the core access. That is why we bring in cache. And from the sequence in figure 2, it is obvious that the time of memory read/writes and core calculating match well, when data is in cache. However when data miss, it will cause extra time spend in the pipeline [8,9,10]. So the use of cache is so important to FFT algorithm execution. Next, the memory access process of FFT in the processor will be modeled and analyzed. Let’s symbolize some relevant factor:

[1] Suppose that the hit rate of cache in the data access processing as \( P \), the probability of data in cache.

[2] Suppose that the extra time caused by data missing as \( t_{miss} \). It means the extra time will be cost from the memory obtaining data, when the access data is not in cache.

[3] Suppose that the memory capacity of cache as \( Cache. \) For discussion purpose, we set complex number points as units, and every point includes two data I/Q. Each datum is consisting of several bytes, which number we define as \( b \).

[4] Suppose that the data capacity that one line of cache can store as \( w \). For discussion, \( w \) here is also a complex number, which is consisting of I/Q, and the number of bytes is \( b \).

[5] Suppose that the access time as \( T_{ac} \). It means the
reading and writing time.

[6] Suppose that the length of the data as \( L \). \( L \) is also a complex number and was characterized by integral power of 2. And it is also consisting of I/Q, the number of bytes is \( b \).

[7] Suppose that the access bandwidth as \( B_w \), which is in bytes per second.

The currently used FFT algorithm is the based-2. Hence here we model the memory access processing on the based-2 algorithm as an example. And the result can also be applied to other FFT algorithms\(^{11,12}\). The butterfly computation pipeline of based-2 algorithm is as follows:

![Figure 3. The butterfly computation pipeline of 8 points based-2 algorithm](image)

As figure 3, the FFT process is divided into \( n \) levels (\( n = \log_2(L) \)). In consideration of rotation factor’s access time which is so short comparing with the execution time of FFT, it can be ignored here. Hence the number of data access in each level is \( 2L \) (including writing and reading). According to the definition mentioned hereinbefore, the accessing time of processing an \( L \) long visit data is:

\[
T_w = \frac{4L \times b \times \log_2 L}{B_w} + L \times t_{\text{rest}} \times \sum_{c=0}^{\frac{\log_2 L}{2}} (1 - P_c). \quad (6)
\]

The first item of formula 6 indicates the time cost that a \( 2L \) long data is accessed in cache. The second item means the extra time cost is caused by missing in cache. There won’t be any missing when writing back to the memory. Hence the number of missing in each level is \( L \times (1 - P_c) \). When the FFT algorithm is execution level by level and the method of mapping is common, the following formula can be used to show the accessing time of memory:

\[
T_{\text{eff}} = \begin{cases} 
\frac{4L \times b \times \log_2 L}{B_w} + L \times t_{\text{rest}} \times \sum_{c=0}^{\frac{\log_2 L}{2}} (1 - P_c) & \text{if } L \leq \text{CacheL} \\
\frac{4L \times b \times \log_2 L}{B_w} + L \times \log_2 L - t_{\text{rest}} \times \sum_{c=0}^{\frac{\log_2 L}{2}} (1 - P_c) & \text{if } L > \text{CacheL}.
\end{cases}
\]

The meaning of formula 7 is that there must be missing in the first level of processing. Because the memory controller access data in the length of cache line \( w \), there will be a missing after \( w \) data. Hence the loss rate in the first level is \( 1/w \). And if the processing length is shorter than cache capacity, the loss rate will be 0 in the next levels. If the processing length is longer than cache capacity, the loss rate will always be \( 1/w \) in every level.

From the discussion above, we know that when the processing length of FFT is longer than cache capacity, there will be missing every time of reading. Then cache will lose efficiency, and the processing efficiency will be sharply reduced as well. Therefore we need to find a way to reduce the processing time of large number of points, and improve the hit rate in the access processing. One solution is to divide the long piece of points into shorter pieces. In this way, cache can become fully effective in dealing with the shorter ones. Then the processing time will become shorter\(^{13}\). To accomplish this goal, we can start out from the discrete Fourier transform (DFT).

\[
X(k) = \sum_{j=0}^{2^L-1} x(j)e^{-j2\pi k j/2^L}, \quad k = 0, 1, 2, \ldots, L-1. \quad (8)
\]

Decomposed the variables \( l, k \) in formula 8 as follow:

\[
l = Y \times f + y \\
k = S \times m + s \\
f = 0, 1, \ldots, F-1; y = 0, 1, \ldots, Y-1 \\
Y \times F = L \\
m = 0, 1, \ldots, M-1; y = 0, 1, \ldots, S-1 \\
M = Y \\
S = F
\]

Here \( Y, F \) are integral power of 2.

Substituting formula 9 into formula 8, we can get

\[
X(m, s) = \sum_{y=0}^{S-1} \sum_{f=0}^{F-1} x(f, y)e^{-j2\pi \frac{fM+yS}{F}} = \sum_{y=0}^{S-1} \sum_{f=0}^{F-1} x(f, y)e^{-j2\pi \frac{fM+yS}{F}}
\]

\[
= \sum_{y=0}^{S-1} \sum_{f=0}^{F-1} x(f, y)e^{-j2\pi \frac{fM+yS}{F}}
\]

\[
= \sum_{y=0}^{S-1} \left( e^{-j2\pi \frac{yS}{F}} \sum_{f=0}^{F-1} x(f, y)e^{-j2\pi \frac{fM}{F}} \right)
\]

(10)
The second summation item means a DFT is accordance of column trend. Then we get the decomposing method:

[1] Translate a single-dimensional data into an \( F \times Y \) matrix.


[3] Multiply the result of transformation with the coefficient \( e^{-i2\pi 2nY} \) (It is called compensating factor).

[4] Perform DFT according to the row.

According to the mapping method here to implement FFT, the execution time of every step can be obtained by formula 7.

Now the total time spent is

\[
T = \frac{4L \times b \times \log_2 L}{B_l} + \frac{3L \times b \times \log_2 L}{B_l} \leq \frac{4L \times b \times \log_2 L}{B_l} + \frac{4L \times b \times \log_2 L}{B_l} \cdot (11)
\]

According to the formula 11, the FFT algorithm after adopting the mapping method doesn’t improve a lot when the processing length is shorter than cache capacity. However if it is longer, the improvement is obvious \(^{[14]}\). To verify the effectiveness of this method, we will implement the radix-2 FFT algorithm on ADI’s TS201 digital signal processor.

3. An effective mapping method for FFT based on the TS201 processor.

According to the mapping algorithm, when FFT is mapped to TS201, the size of the two-dimensional matrices after partitioned should be determined first. Here the goal is to match the two dimensions processing. Hence it is partitioned by this principle:

\[
F = 2^{(n-1)2}, Y = 2^{(n+1)2} \quad \forall \text{is an ODD NUMBER}
\]

\[
F = Y = 2^{n2} \quad \forall \text{is an EVEN NUMBER}
\]

After data partitioning, the problem how to achieve data should be considered. In practical application, the data always be huge, thus it usually be stored in external storage space of DSP (usually SDRAM) \(^{[14]}\). Therefore we can use DMA of DSP processor to access data from SDRAM according to the partitioning method showed in formula 12, and remove it to on-chip memory. Considering the first step of processing is by column, the removing can get data according to the rule of “consecutive in column, jumping in row”. And now all kinds of DMA support this method. After achieving, the data can be processing. And in order to make full advantage of the storage resource, here the twiddle factor and data are put in different banks so that we can access twiddle factor at the same time, when reading data. Besides, the length of twiddle factor just needs to be set according to the value of \( Y \). From the reference 6 here, the length is:

\[
N = \frac{Y}{2}.
\]

Reference \([15]\) also shows how to set the jumping step size of twiddle factor in each level based on radix-2 FFT. Here \( step \) means the size; \( n \) means the number of level. Hence when we deal with a \( Y \) length data, the relationship between \( Y \) and \( n \) is

\[
\begin{align*}
\text{step} = 0 & \quad n = 0, \\
\text{step} = \frac{Y}{2^n} & \quad 0 < n \leq \log_2 Y - 1.
\end{align*}
\]

When deal with an \( F \) length data, formula 14 can still be used to get the value of step if \( F \) is equal to \( Y \). If not, the relationship is as follows:

\[
\begin{align*}
\text{step} = 0 & \quad n = 0, \\
\text{step} = \frac{Y}{2^{n+1}} & \quad 0 < n \leq \log_2 F - 1.
\end{align*}
\]

From the front three sections, the core computing will hide in memory computing, if we arrange the pipeline in our way. It can be our goal to build the software pipeline of the butterfly operation. Reference \([16]\) shows the method to arrange the pipeline in DSP, not tired in words here.

The testing result after optimizing is shown in the table below:

<table>
<thead>
<tr>
<th>Points</th>
<th>(2^{10} \times 10 \times 10)</th>
<th>(2^{12} \times 12 \times 12)</th>
<th>(2^{14} \times 14 \times 14)</th>
<th>(2^{16} \times 16 \times 16)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution time (us)</td>
<td>In traditional method</td>
<td>In new mapping method</td>
<td>In theory</td>
<td>In application</td>
</tr>
<tr>
<td>In theory</td>
<td>In application</td>
<td>In theory</td>
<td>In application</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>20.053</td>
<td>20.283</td>
<td>23.893</td>
<td>24.830</td>
</tr>
<tr>
<td>2048</td>
<td>43.520</td>
<td>44.350</td>
<td>51.200</td>
<td>51.425</td>
</tr>
<tr>
<td>4096</td>
<td>225.280</td>
<td>328.787</td>
<td>121.173</td>
<td>135.680</td>
</tr>
<tr>
<td>8192</td>
<td>488.107</td>
<td>741.047</td>
<td>256.630</td>
<td>269.413</td>
</tr>
<tr>
<td>16384</td>
<td>1051.307</td>
<td>1646.217</td>
<td>539.307</td>
<td>547.520</td>
</tr>
<tr>
<td>32768</td>
<td>2252.800</td>
<td>3555.367</td>
<td>1133.227</td>
<td>1224.373</td>
</tr>
<tr>
<td>65536</td>
<td>4805.973</td>
<td>7866.683</td>
<td>2375.067</td>
<td>2539.520</td>
</tr>
</tbody>
</table>

Table 1 shows the comparing results. For the small points (less than 2048), the processing time here is longer than the traditional one. The reason is that the processing of small points can all be in cache (TS201’s cache can store 2048...
floating-point pluralities), and in this condition, the mapping method here will have the extra time of multiplying the compensation factor. While for the big points, result has changed. In this condition, cache can be taken full advantages. The processing speed has been doubled at least. And the table I also shows that the processing time in theory is almost the same as the time in application. For the traditional method, there is a big deflection for big point condition. It is because DSP’s memory adopts DRAM. There will be an extra time of activating. When activating and missing exist together, the extra time will be longer than the one in theory.

Conclusions

In this paper, a new method of mapping from the FFT algorithm to the processor is studied. The first task is analyzing the structural features of modern superscalar. And based on the result of the processing capacity, relevant factors are parameterized. We build the program execution model underlying the modern processor. Then based on this model, the execution processing of the FFT algorithm is analyzed and we learn that the key factor of real-time capacity is the memory access. Based on the result, we continue to study the accessing feature of FFT and build the analyzing model, furthermore, discover that the hit rate of cache directly affects the algorithm’s real-time capacity. The traditional mapping method doesn’t make a full play to cache, which causes the low efficiency of big-points. To improve the hit rate, a separable FFT mapping method is implemented. It divides the big-points into several small-points, by which can make a full use of cache, and push the processing capacity improved a lot. In the end, a radix-2 FFT is implemented on the ADI’s TS201 digital signal processor, by guiding of the new mapping method. The result shows that the implementation time of FFT is improved greatly and verifies the model’s correctness.

References