A Parallel Bit Reversal Algorithm and it's CILK Implementation

Dániza C. Morales Berrios
Advisor: Dr. Jaime Seguel

Mathematics Department
University of Puerto Rico, Mayagüez Campus
Mayagüez, Puerto Rico 00681-5000
danizam@cs.uprm.edu

Abstract

A wide variety of Fast Fourier Transform (FFT) algorithms employ a bit reversal method for the reordering of input or output data. This article shows one of the most frequently used algorithms for calculating the bit reversal permutation. It also presents a parallel implementation in CILK 5.2 and some performance analysis.

1. Introduction

The N-point discrete Fourier transform of an input vector x=[x(j)], is the
N-point vector

$$\hat{x}(k)=\sum_{j=0}^{N-1} x(j)e^{\frac{-2pikj}{N}} \quad ; \quad k=0,1,\ldots,N-1.$$

This computation involves order N^2 operations. There are several different methods for computing the Fast Fourier Transform. The best know among them is the Cooley-Tukey FFT, developed by J. Cooley and T. Tukey in 1965 [1]. The Cooley-Tukey FFt is preceded by a data sorting called bit reversal permutation. For example, if we are interested in computing an 8-point FFT the procedure will consist of two main steps:

$$\begin{tikzcd}
x[\mathbf{0}] & x[\mathbf{0}] & \hat{x}[\mathbf{0}] \\
x[\mathbf{1}] & x[\mathbf{4}] & \hat{x}[\mathbf{1}] \\
x[\mathbf{2}] & x[\mathbf{2}] & \hat{x}[\mathbf{2}] \\
x[\mathbf{3}] \arrow[r, "\text{bit reversal}"] & x[\mathbf{6}] \arrow[r, "\text{FFT}"] & \hat{x}[\mathbf{3}] \\
x[\mathbf{4}] & x[\mathbf{1}] & \hat{x}[\mathbf{4}] \\
x[\mathbf{5}] & x[\mathbf{5}] & \hat{x}[\mathbf{5}] \\
x[\mathbf{6}] & x[\mathbf{3}] & \hat{x}[\mathbf{6}] \\
x[\mathbf{7}] & x[\mathbf{7}] & \hat{x}[\mathbf{7}]
\end{tikzcd}$$

Computing the bit reversal permutation efficiently is a problem on its own. In general, bit reversal methods are divided in two main classes: in-place bit reversals, and indirect addressing methods. The first ones rearrange the input vector x into its bit reversal order. This is normally achieved through data swaps or nested sequences of stride permutations. Indirect addressing methods, in turn, do not reorder x but compute instead a vector representation of the bit reversal permutation. For example, for N=8, the vector representation will be:

$$B=\begin{bmatrix}0\\4\\2\\6\\1\\5\\3\\7\end{bmatrix}$$

In indirect addresing mode, the bit reversal is realized through the calls:

$x[B[j]]$; $j=0,\ldots,N-1$

For example, for N=8:

$x[0]$		$x[B[0]]$
$x[1]$		$x[B[1]]$
$x[2]$		$x[B[2]]$
$x[3]$	$\longrightarrow$	$x[B[3]]$
$x[4]$		$x[B[4]]$
$x[5]$		$x[B[5]]$
$x[6]$		$x[B[6]]$
$x[7]$		$x[B[7]]$

One of the most efficient methods for producing a vector representation of the bit reversal is Elster's algorithm [2]. In this article we introduce a modification of this algorithm and it's Cilk implementation.

2. Parallelizing Elster's Algorithm

The purpose of this algorithm, is to create a vector B with the bit reversal permutation values. Elster computes the N-point bit reversal vector in $\log_2(N)$ steps. For example, for N=8

			B
Initial	+4	+2	+1
0	0	0	0
	4	4	8
		2	4
		6	12

Pseudocode:

```
step 1: Let n=length of x (length of the
                transform)

step 2: B[0]=0

step 3: For i=1; i < length ;i=2*i, n=n/2
          { For( j=0, j<i, j++){
              B[i+j] = B[j] + n; }
          }

step4: end.
```

One way to paralize Elster's method consist in dividing the the total length by the number of parallel processes. For example if we are interested in computing a bit reversal of length 16, the B vector is then the following:

	Vector *B*
block 1	* B[0]
	B[8]
	B[4]
	B[12]
block 2	* B[2]
	B[10]
	B[6]
	B[14]
block 3	* B[1]
	B[9]
	B[5]
	B[13]
block 4	* B[3]
	B[11]
	B[7]
	B[15]

* head

Also, it is possible to calculate the head (first element) of each block because with the Elster's method since it is a bit reversal of the vector [0 1 2 3] this is [0 2 1 3] as shown in the example. So the parallel method will compute each block on a different processor.

1rst block

h	+8	+4 out
0	0	0
	8	8
		4
		12

2rst block

h	+8	+4 out
2	2	2
	10	10
		6
		14

3^{rst} block

h	+8	+4 out
1	1 9	1 9 5 13

4^{rst} block

h	+8	+4 out
3	3 11	3 11 7 15

The CILK's code implement on Elster's method receives as input the size of the bit reversal permutation and the amount of equally sized blocks into which the vector representation is to be subdivided. The Cilk implementation computes each block in parallel after a suitable initialization.

3. Cilk 5.2 Implementation

We used Cilk.2 to implement the parallel Elster's Algorithm. Cilk is an extension of C based on a multithreaded computing model. Calling normal C functions under the spawn directive generates cilk-threads. When a C function "spawns" another function, it continues to execute until a "sync" instruction is reached. This is the main basis for producing logical parallelism in Cilk. Logical parallelism is implemented by allocating parallel task on different processors, and by using a work-stealing mechanism whenever a processor runs out of work. The "sync" directive synchronizes all the outputs of the different tasks returned by each parallel process.

The Cilk environment provide several statistics for the measurement are:

$$T_1 = execution\ \ time\ \ in\ \ one\ \ processor$$

$$T_\infty = execution\ time\ with\ unlimited\ \ number\ of\ \ processor$$

The creator of Cilk (MIT Computer Science Laboratory) have shown that the execution time (T_p) on p processor is approximately:

$$T_p \cong \frac{T_1}{p} + T_\infty$$

And the parallelism:

$$\bar{p} = \frac{T_1}{T_\infty}$$

From them we can also derived the speed-up:

$$S_p = \frac{T_1}{T_p}$$

In fact, since

$$\bar{p} = \frac{T_1}{T_\infty} \quad \text{by} \quad \text{sustituting}$$

$$T_p = \frac{T_1}{p} + \frac{T_1}{\bar{p}} = T_1\left(\frac{1}{p} + \frac{1}{\bar{p}}\right)$$

Therefore,

$$S_p = \frac{T_1}{T_p} = \frac{p\bar{p}}{p + \bar{p}}$$

We run our experiment in a four-processor machine. The above formula was used to estimate speed-up and parallelism.

4. Some Performance Analysis

Some statistics returns by the Cilk's environment are:

length	Parallelism	Speed-up
2^{22}	6.25	2.4
2^{23}	7.11	2.6

These results gave us an idea about the parameters that affect the computation of vector B. One of them is the length of the vector. We observe that parallelism is higher when the length of the vector B is larger. The same observation holds for the speed-up. Also, when the number of blocks is relatively small parallelism and speed-up it’s higher too.

The speed-up takes values between 1 and the number of physical processors, in our case four, which is the optimal speed-up value. Our results indicate speed-ups right in the middle of this range.

5. Future Work

We will continue the study of the FFT's, and it simplification. The computation of this transform is a problem on its own.
We want to design an easier and faster method for calculating this kind of transforms. Also, we are interest in applying a quarter-wave bit reversal and it's implementation in CILK 5.2.

References

1. Cooley J.W, Tukey J.W. “An Algorithm for the Machine Calculation of Complex Fourier Series”. 1965.
2. Elster A.C. “Fast Bit-Reversal Algorithm”, ICASSP 89’ Proccedings.1989