≪연구논문≫ Journal of the Korean Magnetics Society, Volume 22, Number 2, April 2012 http://dx.doi.org/10.4283/JKMS.2012.22.2.033

구식 싱글턴 혼합기수 고속푸리에변환 코드에 대한 간단한 동적메모리 할당방법 프로그래밍

김인기*
포항공과대학교 철강대학원, 경북 포항시 남구 효자동 산31, 790-784

(2012년 1월 13일 받음, 2012년 4월 2일 수정본 받음, 2012년 4월 3일 게재확정)

1968년에 발표된 구식 싱글턴 혼합기수 고속푸리에 변환 포트란 서브루틴에 존재하는 일반적인 *N* 문제에 대한 간단한 처치법을 제안하였다. 주어진 문제에 대한 간략한 검토를 마친 후에, 새로운 처치법을 동적메모리 할당방법의 입장에서 최악상황분석법을 통해 논의하였다. 본 논의를 통해 여기서 제시된 프로그래밍 기법은 최소한 다차원 데이터 집합에 대해 지금까지 제시된 여타 처치법보다 우월함을 보였다.

주제어 : 고속푸리에변환, 싱글턴 혼합기수방법, 동적메모리할당, 포트란 90/95

A Simple Implementation of Dynamical Memory Allocation in Old-fashioned Singleton's Mixed-radix Fast Fourier Transformation Code

In Gee Kim*
Graduate Institute of Ferrous Technology, Pohang University of Science and Technology, Pohang 790-784, Korea

(Received 13 January 2012, Received in final form 2 April 2012, Accepted 3 April 2012)

We propose a simple prescription for resolving the general-*N* problem existing in the old-fashioned mixed-radix fast Fourier transformation FORTRAN subroutine by Singleton in 1968. After a brief investigation on the problem, we discuss our prescription with the worst case analysis within the dynamical allocation. The analysis reveals that our implementation is superior, at least for multi-variate data set, than previously proposed data copying methods.

Keywords : fast fourier transformation, Singleton's mixed-radix, dynamical memory allocation, FORTRAN 90/95

The rediscovery of an algorithm for fast Fourier transforms (FFT) [1] has opened a very efficient way for manipulating large data set, especially in scientific and engineering computing area. FFT algorithms are based on divide-and-conquer strategy, which factorizes an input data vector of length N out into the terms of multiples of factors f_i. The factored vector is subjected into the factorization procedure again until the input vector has only single term, and then performs a discrete Fourier transform (DFT) on a single term before the factored terms merged together. This divide-and-conquer strategy improves the speed of calculation of the DFT of $O(N\log N)$ scale instead of $O(N^2)$ scale if one performs the direct DFT. Generally speaking, the algorithms of FFT fall into two categories of the Cooley-Tukey algorithms (CTA) and the prime factor algorithm (PFA), depending on their factorization methods. A full review of various FFT methods is given by Burrus and Parks [2].

Most FFT algorithms in practice have a severe obstacle that the length of the input data set has to follow a given rule, *e.g.*, $N = 2^m$ for radix-2 algorithms. Since many practical problems to be solved restrict the size of data set, for example, $N = 562$, one has to copy the data into a relevant size of vector with a larger ($N = 1024$) or smaller ($N = 512$) elements. The latter case loses the information from input data so it is not appropriate when the problem requires high precision. The former case needs extra memory padded by zeros, and more critically increases the time scale for transforms of $O(2\log(M/N)^{1/2})$ if one adapts next power of 2 for the new size of the zero-

*Tel: (054) 279-9014, E-mail: igkim@postech.ac.kr padded array. In addition, the manipulating data set to fit the existing library routine gives rise to another overhead of $O(N)$, prior and posterior to transform for copying and restoring between the original data vector and the temporary one. To resolve the general-N problem, two algorithms have been proposed: the PFA [3] and the mixed-radix algorithms [4]. The PFA has a very simple indexing scheme, but it only works in the case where all factors of N are mutually prime.

The Singleton's mixed-radix algorithm [4] resolves the general-N problem in such a way that the size N of the data set is factored into prime numbers, and each factored subset is subjected to a well-developed radix-n subroutine, for example, with n equals 2, 3, 4, or 5. The set factored by higher prime numbers is transformed by an $O(N^2)$ algorithm. Because most N below the integer limit of a given machine can be factored into small prime numbers, the mixed-radix method works with reasonable performance, hence so the Singleton's implementation written in the old-fashioned FORTRAN Y has been spread widely to date. However, there were limitations of memory and speed of computers as well as FORTRAN compilers, in the age of the Singleton's code implemented. Such limitations forced Singleton to make his code being limited by maximum prime factor to be 23. Although Singleton's code is translated into FORTRAN 90 by Miller [5], the limitation was not removed, *i.e.*, no dynamical allocation for the temporary storages is applied. Here we suggest a solution to break out of this limitation of the Singleton's codes with a very simple method.

Let us investigate, first of all, the Singleton's original code [4, 5]. The Singleton's code has the interface of `sfft(a, b, ntot, n, nspan, isgn)`, where `a` and `b` are the real and imaginary part of the complex data set of length `ntot`, `n` is the dimension of the current data values, `nspan/n` is the spacing of consecutive data values while indexing the current variable, and `isgn` controls the direction of the FFT. For a single-variate transform, the number of complex data is equal to `ntot(= n = nspan)`, and calling it by

```
call sfft(a, b, n, n, n, isgn) !%
```

A tri-variate transform with `a(n1, n2, n3)` and `b(n1, n2, n3)` is computed by a series of calls

```
call sfft(a, b, n1*n2*n3, n1, n1, isgn) !%
call sfft(a, b, n1*n2*n3, n2, n1*n2, isgn) !%
call sfft(a, b, n1*n2*n3, n3, n1*n2*n3, isgn) !%
```

The interface greatly simplifies the user routines in which multi-variate data set has to be transformed.

However, the fixed-memory allocation functionality in old-fashioned FORTRAN languages limits on the dimension of the current data set values `n`: (i) the number of factors of `n` should be smaller than a given maximum number of factors `maxn`, (ii) the maximum prime factor of `n` has to be smaller than a given prime number `maxf`, and (iii) the square-free portion of `n` must be smaller than a given prime number multiple `maxp` minus one.

Regarding the limitation (i), the smallest number with 16 factors is 12,754,584 ($= 2^3 \times 3^{13}$) [4], so that it does not occasionally occur such situation, but we would like to give freedom on it. On the limitation (ii), Singleton fixed `maxf` as 23, because of most numbers allowed in usual 4-byte long integer can be factorized into the prime numbers less than 23. However, if `n` is a large prime number itself or it contains a larger prime factor than 23, the data set of this size is not allowed. The limitation (iii) makes the situation messy as described below.

Suppose a dimension of the data set values N is factored by several prime factors, *i.e*, $N = p_1p_2\cdots p_i\cdots p_j\cdots p_{m-1}p_m$, where p_i is the ith prime number factoring N. We may factor out the squared factors, *i.e.*, we factor out $p_i\,p_j$ from N to get a new number M, if $p_i = p_j$; and we perform this procedure repeatedly on M until we obtain the final number $M = P = f_1f_2\cdots f_m$ where no factor f_i appears more than twice. The number P is then called the 'multiple of square-free factors' of N, and f_i is called the 'square-free factor'. Let us consider two positive integers r and s, such that $r > s$, which contain the square free factors $P_r < P_s$. If P_s is greater than the given maximum square free portion `maxp`, the data set of larger size r is allowed if P_r is smaller than `maxp`. A simple example on this situation is given by Kirby [6].

The simplest and best way to avoid these problems without modifying the existing code is to predict the size of the data set before calling the FFT routine. However, it is difficult to know *a priori* and optimal size of data sets in many practical situations, because the size of data set is occasionally evaluated dynamically be the given problem.

There are three ways to resolve the general-N problem with dynamical memory allocation (DMA) ability in FORTRAN 90/95, or with the other programming languages supporting the DMA. First of all, the ultimate method is that one may invent wheel again, *i.e.*, one may implement a new mixed-radix FFT subroutine with the same interface and functionality as the Singleton's code does. To date there is at least one FORTRAN 90 implementation for the one dimensional case by Steffens [7], which treats the higher dimensional data set by decomposing it into one dimensional one prior to performing a mixed-radix FFT subroutine. Although the Steffen's implementation has many desirable features of software engineering, *e.g.*, data encapsulation, structured programming, *etc*., overheads of $O(N)$ required again for copy-in-copy-out procedures prior and posterior to manipulating the given multi-dimensional data sets. Second one is proposed by Kirby [6], such that one may add a routine that calculated a safe size of the data set within the only 59 lines long subroutine, allocates a temporal data vector with the safe size, and then pads zeros at the attached dummy elements of the data set before calling the Singleton's subroutine. However, the treatment of Kirby increases complexity in caller routines when the multi-dimensional data set is subjected to the FFT, as well as the requirement for the data copying overhead. The third way, what we describe here, is just to make the Singleton's code dynamic itself. Surprisingly, this very simple solution has never been introduced in public domain so far!

In order to make the Singleton's code dynamic, it is required to calculate, in an additional subroutine, the limit parameters, maxn, maxf, and maxp before allocating the temporary storage. The additional routine is very simple. The dynamic Singleton subroutine invokes, at the beginning, another subroutine primfact(n, maxn, maxf, maxp), which calculates maxn, maxf, and maxp from the input number of n of the data set size. The subroutine primfact

(1) tabulates the prime numbers below n by using the Sieve of Eratosthenes,

(2) factorizes n into prime numbers and stores the prime factors into another table,

(3) and obtains the square-free factors and calculates their products.

As seen from the procedure given above, maxn and maxf are obtained at the stage (2), while maxp is evaluated at the stage (3). With these values sfft allocates the corresponding temporary storages in the dynamic Singleton routine. One has to be careful that the Singleton's code assumes that the number of prime factors is odd. So if maxn is even, maxn must be increased by one to be odd.

Once the prime numbers below a given N are tabulated at the stage (1), the procedures for factorization and calculating the multiples of square-free factors are direct and negligible in time. Note that the time scale of the Sieve of Eratosthenes is $O(\sqrt{N})$, which is superior to the other treatments with copying input vectors of $O(N)$ for the one-dimensional case. However, one may ask that "How long does it take with the overhead?"

The worst case for the above overhead in practice occurs when the data set is extremely large and it is one-dimensional. For a 4-byte long integer variable, the maximum number is $2^{31}-1 = 2{,}147{,}483{,}647 = O(10^9)$, which is even much larger than the smallest size with 16 factors of $O(10^7)$. The Sieve of Eratosthenes takes steps of $O(10^5)$ for this size. A two-dimensional case with the size of $N \times N$ data set, the treated Singletons code invokes primfact with n = N~65,536 = $O(10^4)$, where the Sieve of Eratosthenes takes only $O(10^2)$ steps. The same argument for the three-dimensional cases gives $O(10)$ steps for the Sieve of Eratosthenes. Hence, the overhead for higher-variate data sets is negligible. Since the Singleton's code is applied for FFT of multi-variate–mostly in three dimensional complex data sets in most practical applications, this treatment will work in reasonable speed.

Another type of worst case arises by the mixed-Radix algorithm itself. Below the maximum integer number of 4-byte long, the largest prime number and the multiples of square-free factors is $2^{31}-1$. Note that no multiple of square-free factor can exceed the largest prime number. Such a large data set with the length which contains a large prime factor in one dimension, the time scale is dominated by $O(N^2)$. In this case, one may cut-off the last datum to make the size of the data set being $N-1$, which is no more a prime number, hence so is less dangerous, because Fourier transform is not susceptible to the inclusion of the last datum for large N.

The suggested subroutine was coded in the full-potential linearized augmented plane wave (FLAPW) method implemented in the QMD-FLAPW package [8]. During the implementation, we modified the main Singleton's FFT subroutine sfft within the modern structural programming paradigm, *i.e.*, we eliminated the goto statements from the subroutine. The realistic test was done on ferromagnetic body-centered cubic iron as given in Ref. [9]. We confirmed that the new subroutines, primfact and sfft reproduce exactly the same results of Ref. [9]. It is also found that there are no significant increments in running time with these new subroutines. This result is essential in developing new subroutines.

In summary, we have proposed a simple method to allow general-N data set for the well-known mixed-radix FFT method developed by Singleton in 1968. It seems like a prescription of an oxygen-mask to a dying patient. However, the analysis has shown that this prescription will work efficiently until an ultimate solution appears.

Acknowledgements

This work was supported by POSCO through Steel Innovation Program. Helpful discussions with professors B. I. Min and A. J. Freeman are greatly appreciated.

References

[1] J. W. Cooley and J. W. Tukey, Math. Comput. **90**, 297

(1965).
[2] C. S. Burrus and T. W. Parks, DFT/FFT and Convolution Algorithms, Wiley, New York (1984).
[3] C. Temperton, J. Comput. Phys. **52**, 198 (1983).
[4] C. Temperton, J. Comput. Phys. **58**, 283 (1985).
[5] R. C. Singleton, IEEE Trans. Audio and Electroacoust. **17**, 93 (1968).
[6] A. Miller, http://users.bigpond.net.au/amiller/fft.f90
[7] J. F. Kirby, Computat. Geosci. **28**, 999 (2002).
[8] M. Steffens, http://users.bigpond.net.au/amiller/singleton.f90
[9] See http://www.flapw.com
[10] S. W. Seo, Y. Y. Song, G. Rahman, I. G. Kim, M. Weinert, and A. J. Freeman, J. Magnetics **14**, 137 (2009).