Getting the Most Out of the FFTs in the Cray X1 Scientific Libraries

Bracy Elton, Ph.D.
Scientific Library Group
Cray Inc.
411 First Avenue South, Suite 600
Seattle, WA 98104–3847
E-mail: `elton@cray.com`

May 12, 2003

Abstract

We explain how to use the FFT/signal processing routines in the Cray X1 Scientific Libraries, introducing new features in the process. We then discuss performance issues ranging from stride selection to radix-specific considerations. Finally, we present current performance and discuss future directions.

1 Introduction

LibSci is a collection of scientific libraries that contain numerical routines specifically tuned for the Cray X1 system. Part of the library contains routines for signal processing, which includes the fast Fourier transform (FFT), convolutions, and filters. The FFT comprises special cases of the discrete Fourier transform (DFT), cases designed to be fast. This provides users with an introduction to the signal processing portion of LibSci, highlighting issues related to porting programs that use its functionality from other Cray systems as well as non-Cray systems. There are a variety of features and considerations for porting and attaining good performance of these routines.

Section 2 overviews the signal processing portion of LibSci. Section 3 details porting issues. Performance issues are reviewed in Section 4. We present some timing and performance measurements in Section 5, along with comparisons to Cray SV1ex and T94 systems. We conclude by outlining future directions in Section 6.

2 Overview

The Cray X1 system offers hardware support for 32-bit integer and floating point representations, in addition to 64-bit ones. Consequently, LibSci supports these features as well.

In this way LibSci was designed to address certain issues regarding porting code from previous Cray and other non-Cray systems. In particular, LibSci supports more data types, instead of only 64-bit integer and 64-bit floating point representations, as was generally the case on Cray T90, Cray SV1ex, and Cray T3E systems. Indeed, the defaults on the Cray X1 system are 32-bit integer and floating point representations, which help support porting efforts from systems with these as defaults, e.g., workstation-type environments. LibSci supports previous Cray systems by offering routines with 64-bit integer and floating point representations.

At the same time, LibSci also supports features unique to the Cray X1 system, e.g., those related to multistreaming.

2.1 LibSci Variants

The default LibSci contains 32-bit integers and supports single and double precision routines. By default linking is against this version of the library.

The 64-bit variant of LibSci contains 64-bit integers and supports only single precision routines. This version of the library is most compatible with previous Cray systems, e.g., The Cray SV1ex, T90 and T3E systems. Users link to it by using `-s default64` when compiling and linking or using `-lsci64` when linking. Please see [4, 2] for more information.

Table 1 lists how the data types and integer and floating representations relate to the default and 64-bit LibSci variants.

Table 1

LibSci Data Types.

Library	Integer Width	**Floating Point**	
		Precision	**Width**
LibSci (default)	32 bits	Single	32 bits
LibSci (default)	32 bits	Double	64 bits
LibSci (64-bit)	64 bits	Single	64 bits

With the addition of double precision support, the FFT, convolution and filtering routines have adopted the naming conventions for BLAS, LAPACK, ScaLAPACK and BLACS routines.

Generally, a LibSci FFT routine is named depending on the kind of FFT data (real or complex) and its type (single or double precision) with a letter each for the input and output data. "S" and "D" stand for single and double precision real data, respectively, while "C" "C" and "Z" stand for single and double precision complex data, respectively. The convolution and filtering routines now, if they did not in previous renditions of Cray's scientific libraries, have a single letter identifying the data type. [1]

2.2 Contents

The FFT, convolution, and filtering routines of LibSci provide the following:

- One dimensional (1-D), 2-D, 3-D, and multiple 1-D complex-to-complex, real-to-complex, and complex-to-real FFTs/DFTs.
- Convolutions.
  - Directly computed.
  - Computed via FFTs.
- Filters.
  - Correlation of two vector with general coefficient.
  - Correlation of two vector with symmetric coefficient.
  - Weiner-Levinson linear equations solution.

Tables 2 and 3 list the FFT, convolution and filtering routines in LibSci.

2.3 Multistreaming vs. Single-streaming Routines

In addition to supporting a wider variety of data types, LibSci supports multistreaming and single-streaming modes. A program intended to run on a multistreaming processor (MSP) should use the multistreaming version, and a program to run on a single-streaming processor (SSP) should use the single-streaming version. Users can select this with the Fortran by using the compilation and linking flag `-O ssp`.

2.4 Documentation

Further information on LibSci and the FFT, convolution and filtering routines can be found in the manuals [1], [2] and [3], which cover user environment differences, code migration and optimization. Specific information on the FFTs, convolution and filtering routines in LibSci can be found in the following man pages:

- `intro_libsci`
- `intro_fft`

The `intro_fft` man page serves as a starting point for finding man pages on all of the LibSci FFT, convolution, and filtering routines. A reference manual for LibSci on the Cray X1 system is forthcoming.

[1]There are some complex-to-complex FFT routines prefixed with a single identifying letter. One routine has the second letter as the one identifying the type, i.e., either `MCFFT` or `MZFFT`.

Table 2

Single Precision FFT, Convolution and Filtering Routines in LibSci.

FFT Routines

Dimension	Complex-to-complex	Real-to-complex	Complex-to-real
1-D (single)	CCFFT (CFFT)	SCFFT	CSFFT
1-D (multiple)	CCFFTM (MCFFT)	SCFFTM	CSFFTM
2-D	CCFFT2D (CFFT2D)	SCFFT2D	CSFFT2D
3-D	CCFFT3D (CFFT3D)	SCFFT3D	CSFFT3D

Linear Digital Filter Routines

Purpose	Name
Computes a correlation of two real vectors	SFILTERG (FILTERG)
Computes a filter correlation of two real vectors (assuming the filter coefficient vector is symmetric)	SFILTERS (FILTERS)
Solves the Weiner-Levinson linear equations	SOPFILT (OPFILT)

Complex Convolution Routines

Purpose	Name
Computes a standard complex convolution	CCNVL
Computes a complex convolution using FFTs	CCNVLF

Table 3

Double Precision FFT, Convolution and Filtering Routines in LibSci.

FFT Routines

Dimension	Complex-to-complex	Real-to-complex	Complex-to-real
1-D (single)	ZZFFT (ZFFT)	DZFFT	ZDFFT
1-D (multiple)	ZZFFTM (MZFFT)	DZFFTM	ZDFFTM
2-D	ZZFFT2D (ZFFT2D)	DZFFT2D	ZDFFT2D
3-D	ZZFFT3D (ZFFT3D)	DZFFT3D	ZDFFT3D

Linear Digital Filter Routines

Purpose	Name
Computes a correlation of two real vectors	DFILTERG
Computes a filter correlation of two real vectors (assuming the filter coefficient vector is symmetric)	DFILTERS
Solves the Weiner-Levinson linear equations	DOPFILT

Complex Convolution Routines

Purpose	Name
Computes a standard complex convolution	ZCNVL
Computes a complex convolution using FFTs	ZCNVLF

TABLE 4

`TABLE` and `WORK` Sizes in Cray SV1 and SV1ex LibSci FFTs. Note that `ncpus` on SV1 systems is user-controlled. Complex-to-real requirements are the same as for the counterpart real-to-complex ones, e.g., CSFFT, which is not explicitly listed below, has the same requirements as SCFFT. Note: `TABLE` requirements are in units of 64-bit words; `WORK` requirements are in units of 64-bit words.

Routine	`ISYS`	**Size Requirements (caption specifies units)**	
		`TABLE`	`WORK`
CCFFT	0	$100 + 8 \cdot n$	$8 \cdot n$
CCFFT2D	0	$100 + 2 \cdot (n1 + n2)$	$512 \cdot \max(n1, n2)$
CCFFT3D	0	$100 + 2 \cdot (n1 + n2 + n3)$	$512 \cdot \max(n1, n2, n3)$
CCFFT3D	1	$100 + 2 \cdot (n1 + n2 + n3)$	$4 \cdot \texttt{ncpus} \cdot \max(n1 \cdot n2, n1 \cdot n3, n2 \cdot n3)$
CCFFTM	0	$100 + 2 \cdot n$	$4 \cdot lot \cdot n$
SCFFT	0	$100 + 4 \cdot n$	$4 + 4 \cdot n$
SCFFT2D	0	$100 + 2 \cdot (n1 + n2)$	$512 \cdot \max(n1, n2)$
SCFFT3D	0	$100 + 2 \cdot (n1 + n2 + n3)$	$512 \cdot \max(n1, n2, n3)$
SCFFT3D	1	$100 + 2 \cdot (n1 + n2 + n3)$	$4 \cdot \texttt{ncpus} \cdot \max(n1 \cdot n2, n1 \cdot n3, n2 \cdot n3)$
SCFFTM	0	$100 + 2 \cdot n$	$(2 \cdot n + 4) \cdot lot$

3 Porting Issues

As discussed further below, the main porting issues regard desired accuracy, table and work space requirements for the FFT and convolution routines, definition of the nth root of unity, routine names, compilation flags, streaming mode, and linking.

3.1 Accuracy

Users should be mindful of the desired accuracy and desired size of integers. Users wanting 32-bit integers and 64-bit floating point precision ought to use the double precision portion of the default LibSci. Code that previously called earlier renditions of Cray's scientific libraries will need single precision names changed to their double precision counterparts.

Users coming from, particularly, previous Cray systems and wanting to maintain the 64-bit floating point precision ought to take a look at the 64-bit LibSci. Indeed, this library is provided to support such situations.

The single precision routines of the default LibSci, for a 32-bit floating point representation, can be used when accuracy requirements do not need as much precision offered by their 64-bit siblings.

3.2 Table and Work Space Requirements

The FFT `TABLE` and `WORK` array requirements may differ from previous Cray systems. Users should consult the online documentation to ensure proper usage of the Cray X1 system's FFT library. Note that the table array needs to be an array of 64-bit words regardless of the setting in which it is used, i.e., independent of the particular LibSci variant.

To illustrate how the sizes differ, Table 4 lists the `TABLE` and `WORK` sizes for the Cray SV1's LibSci FFT routines. Tables 5 and 6 give the sizes for the default LibSci and the 64-bit LibSci FFTs, respectively.

3.3 Sign of Exponent in Nth Root of Unity

Consider the 1-D complex-to-complex discrete Fourier transform,

$$Y_k = \texttt{scale} \cdot \sum_{j=0}^{n-1} \left[X_j \cdot w^{\texttt{isign} \cdot j \cdot k} \right], \qquad \text{for } k = 0, 1, \ldots, n-1,$$

where

$$\begin{aligned} w &= e^{2\pi \cdot i/n}, \\ i &= \sqrt{-1}, \\ \texttt{isign} &= \pm 1. \end{aligned}$$

Table 5

`TABLE` and `WORK` Sizes in default LibSci FFTs. Note that `ncpus` on Cray X1 systems is 4 and 1 for MSP and SSP mode, respectively. Complex-to-real requirements are the same as for the counterpart real-to-complex ones, e.g., CSFFT, which is not explicitly listed below, has the same requirements as SCFFT. Note: `TABLE` requirements are in units of 64-bit words; `WORK` requirements are in units of 32-bit words for single precision in the default LibSci and 64-bit words, otherwise.

Routine	`ISYS`	**Size Requirements (caption specifies units)**	
		`TABLE`	`WORK`
CCFFT	0	$100 + 4 \cdot n$	$8 \cdot n$
CCFFT2D	0	$100 + (n1 + n2)$	$2048 \cdot \max(n1, n2)$
CCFFT3D	0	$100 + (n1 + n2 + n3)$	$2048 \cdot \max(n1, n2, n3)$
CCFFT3D	1	$100 + (n1 + n2 + n3)$	$4 \cdot \texttt{ncpus} \cdot \max(n1 \cdot n2, n1 \cdot n3, n2 \cdot n3)$
CCFFTM	0	$100 + n$	$4 \cdot lot \cdot n$
SCFFT	0	$100 + 2 \cdot n$	$4 + 4 \cdot n$
SCFFT2D	0	$100 + (n1 + n2)$	$2048 \cdot \max(n1, n2)$
SCFFT3D	0	$100 + (n1 + n2 + n3)$	$2048 \cdot \max(n1, n2, n3)$
SCFFT3D	1	$100 + (n1 + n2 + n3)$	$4 \cdot \texttt{ncpus} \cdot \max(n1 \cdot n2, n1 \cdot n3, n2 \cdot n3)$
SCFFTM	0	$100 + n$	$(2 \cdot n + 4) \cdot lot$
ZZFFT	0	$100 + 8 \cdot n$	$8 \cdot n$
ZZFFT2D	0	$100 + 2 \cdot (n1 + n2)$	$2048 \cdot \max(n1, n2)$
ZZFFT3D	0	$100 + 2 \cdot (n1 + n2 + n3)$	$2048 \cdot \max(n1, n2, n3)$
ZZFFT3D	1	$100 + 2 \cdot (n1 + n2 + n3)$	$4 \cdot \texttt{ncpus} \cdot \max(n1 \cdot n2, n1 \cdot n3, n2 \cdot n3)$
ZZFFTM	0	$100 + 2 \cdot n$	$4 \cdot lot \cdot n$
DZFFT	0	$100 + 4 \cdot n$	$4 + 4 \cdot n$
DZFFT2D	0	$100 + 2 \cdot (n1 + n2)$	$2048 \cdot \max(n1, n2)$
DZFFT3D	0	$100 + 2 \cdot (n1 + n2 + n3)$	$2048 \cdot \max(n1, n2, n3)$
DZFFT3D	1	$100 + 2 \cdot (n1 + n2 + n3)$	$4 \cdot \texttt{ncpus} \cdot \max(n1 \cdot n2, n1 \cdot n3, n2 \cdot n3)$
DZFFTM	0	$100 + 2 \cdot n$	$(2 \cdot n + 4) \cdot lot$

Table 6

`TABLE` and `WORK` Sizes in 64-bit LibSci FFTs. Note that `ncpus` on Cray X1 systems is 4. Complex-to-real requirements are the same as for the counterpart real-to-complex ones, e.g., CSFFT, which is not explicitly listed below, has the same requirements as SCFFT. Note: `TABLE` requirements are in units of 64-bit words; `WORK` requirements are in units of 64-bit words.

Routine	`ISYS`	**Size Requirements (caption specifies units)**	
		`TABLE`	`WORK`
CCFFT	0	$100 + 8 \cdot n$	$8 \cdot n$
CCFFT2D	0	$100 + 2 \cdot (n1 + n2)$	$2048 \cdot \max(n1, n2)$
CCFFT3D	0	$100 + 2 \cdot (n1 + n2 + n3)$	$2048 \cdot \max(n1, n2, n3)$
CCFFT3D	1	$100 + 2 \cdot (n1 + n2 + n3)$	$4 \cdot \texttt{ncpus} \cdot \max(n1 \cdot n2, n1 \cdot n3, n2 \cdot n3)$
CCFFTM	0	$100 + 2 \cdot n$	$4 \cdot lot \cdot n$
SCFFT	0	$100 + 4 \cdot n$	$4 + 4 \cdot n$
SCFFT2D	0	$100 + 2 \cdot (n1 + n2)$	$2048 \cdot \max(n1, n2)$
SCFFT3D	0	$100 + 2 \cdot (n1 + n2 + n3)$	$2048 \cdot \max(n1, n2, n3)$
SCFFT3D	1	$100 + 2 \cdot (n1 + n2 + n3)$	$4 \cdot \texttt{ncpus} \cdot \max(n1 \cdot n2, n1 \cdot n3, n2 \cdot n3)$
SCFFTM	0	$100 + 2 \cdot n$	$(2 \cdot n + 4) \cdot lot$

and $w^{\texttt{isign}}$ is the so called nth root of unity. The sign of the exponent of the nth root of unity (which is factored out as `isign` in the above equation) depends on the convention used in the application. Indeed, conventions differ for which of the $\texttt{isign} = \pm 1$ transformations is the forward or inverse transform and what the scale factor should be for each. All the 1-D LibSci routines can compute any of the various possible definitions if you choose the appropriate values for `isign` and `scale`.

From FFT theory, if you take the 1-D FFT with any particular values of `isign` and `scale`, the mathematical inverse function is computed by taking the FFT with $-\texttt{isign}$ and $1/(n \cdot \texttt{scale}$. In particular, if you use $\texttt{isign} = +1$ and $\texttt{scale} = 1.0$, you can compute the inverse FFT by using $\texttt{isign} = -1$ and $\texttt{scale} = 1/n$.

The 2-D and 3-D situations follow similarly.

3.4 Routine Names

Users should ensure their applications use the proper routine names. They should be consistent with the desired accuracy, the data types used in the application hand with the LibSci library to which such code is linked.

3.5 Compilation Flags

With the Fortran compiler, using `-s default32` (or no `-s` typing flags) on both the compilation and link lines reaches the default LibSci, which has 32-bit integers and single and double precision routines. Likewise with the Fortran compiler, using `-s default64` compiles for and links against the 64-bit LibSci. These two options automatically imply the addition of the link flags `-lsci` and `-lsci64` for the default LibSci and 64-bit LibSci, respectively.

3.6 Streaming Mode

Code is compiled by default for multistreaming mode to run on an MSP. To compile code to run on a single stream, i.e., on an SSP (single streaming processor), use (in Fortran) the flag `-O ssp`. This needs to be applied at both compile and link stages.

4 Performance Issues

Using a LibSci FFT routine does not guarantee good performance. Using LibSci FFT routines wisely can achieve great performance.

Three areas that significantly affect performance in the LibSci FFTs:

- FFT length.
- Strides and leading dimensions.
- Implementation choice for 3-D FFTs.

4.1 FFT Length

The FFT length plays a key role in performance. The libraries do well on lengths that contain only factors that are powers of 2, 3 and 5. There are special butterflies for radices 2, 3, 4, 5, 8 for the complex-to-complex FFTs, and 2, 3, 4, 5, 6, and 8 for the real-to-complex and complex-to-real FFTs. Due to the butterfly implementations and FFT theory, lengths containing only factors that are powers of 2 generally do best. A discrete Fourier transform (DFT) algorithm, which is much slower than and FFT, is employed for lengths containing factors that are not powers of 2, 3 and 5.

For real-to-complex and complex-to-real FFTs, it is important that the FFT length in the first dimension be even, otherwise a DFT is used. (This is because the real-to-complex and complex-to-real FFTs are implemented as half-length complex-to-complex FFTs.)

The FFT length plays an additional role in the ability of the Cray X1 to be effective on the problem. This has to do with vectorization and multistreaming. Because having multistreaming processors increases $N_{1/2}$ (the length to reach 1/2 of the algorithmic peak of a particular portion of code of interest), the lengths involved in FFTs need to be commensurately longer to attain like relative performance.

If your application uses mixed radix FFTs, if possible and the application allows, nearby FFT lengths ought to be investigated. It is possible that a longer length may take less time.

4.2 Strides and Leading Dimensions

Another critical area for achieving good performance on Cray X1 systems regards ensuring good strides through memory. As the memory is made up of a power-of-two number of banks, it is important to access memory in such a way so as the bank of interest at any particular moment is not in high demand.

On Cray X1 systems, there are multiple ways of dealing with striding issues. For FFTs there are at least two approaches one can take:

- Try to get odd multiples of 4 and 8 for strides for 64- and 32-bit data, respectively.
- Try to get odd strides.

With complex data, achieving an odd stride is problematic. The next best approach is to ensure an odd leading dimension. If it is a 3-D FFT, then the middle (second leading dimension) should also be odd.

With real data, the FFT routines require an extra two words. So for power of 2 FFT lengths on real data, the striding issue will be, in a manner of speaking, automatically addressed, as the leading dimension will have been already increased. However, for 3-D real-to-complex and complex-to-real problems, you should make sure that the middle dimension (second leading dimension) is indeed odd.

Whichever approach is used in dealing with the first leading dimension whether it be on real or complex data, the second leading dimension of a 3-D problems should be odd.

Figures 1 and 2 illustrate the effect of whether or not leading dimensions are adjusted for better strides for 2-D, 3-D, and multiple 1-D complex-to-complex FFTs, respectively, run in MSP mode with `ISYS`=0. The effects are increasingly apparent with larger FFT lengths.

Note that in graphs, the data are connected with lines for visual clarity; no interpolation is implied. Furthermore, we present both timing and performance graphs in both log vs. log and log vs. linear formats to reveal more readily behavior at large and small scales.

4.3 3-D FFTs Implementations

Another area that can affect performance of 3-D problems is the choice of implementation. On the Cray X1 system there are presently two implementations for 3-D complex-to-complex, real-to-complex and complex-to-real FFTs. Generally, one gets better performance than the other due to effect use of the (required) extra `WORK` array.

Which one is utilized depends on the `ISYS` parameter, which can take on the values 0 or 1. Each has a different requirement for the `WORK` array.

For the `ISYS`=0 implementation, in each of the three 1-D FFTs, one dimension is used for managing the `WORK` array and is processed serially, one dimension is for vectorization and parallelization a la multistreaming, and one dimension is for the transform. At each stage parallelization and vectorization lengths are limited. Consequently, performance can suffer.

For the `ISYS`=1 implementation, in each of the three 1-D FFTs, one dimension is for parallelization a la multistreaming, one dimension is for vectorization, parallelization a la multistreaming, and one dimension is for that stage's (1-D) transform. The increased `WORK` array offers avenues for parallelization on one the other dimension. Hence, performance improves with this approach.

Figure 3 depicts the performances and timings for the `ISYS`=0 and `ISYS`=1 cases for the 64-bit LibSci CCFFT3D run in MSP mode with odd leading dimensions. It is apparent that the extra memory for when `ISYS`=1 translates into extra performance.

(Note that the data in the graphs are connected via lines for visual purposes and are not intended to imply any sort of interpolation. Also, the graphs exhibit results only for sizes that contain only factors that are powers of 2, 3, and 5.)

5 Performance and Timing Measurements

It would be useful to investigate performance and timing measurements across all variants FFTs in LibSci, such a work spanning the possibilities outlined in Table 7. While exploring all these avenues is a worthy endeavor, there simply is not enough room here for all the possibilities. We limit our exhibition to comparisons across systems with the 64-bit LibSci complex-to-complex routines, CCFFT, CCFFTM, CCFFT2D, and CCFFT3D, to provide ideas for how the Cray X1's LibSci routines fare in MSP vs. SSP mode and against the Cray SV1ex and T94 systems. Furthermore, we restrict the cases, to `ISYS`=0 for CCFFT, CCFFTM, and CCFFT2D, and to `ISYS`=1 for CCFFT3D, and with odd leading dimensions for CCFFTM, CCFFT2D and CCFFT3D.

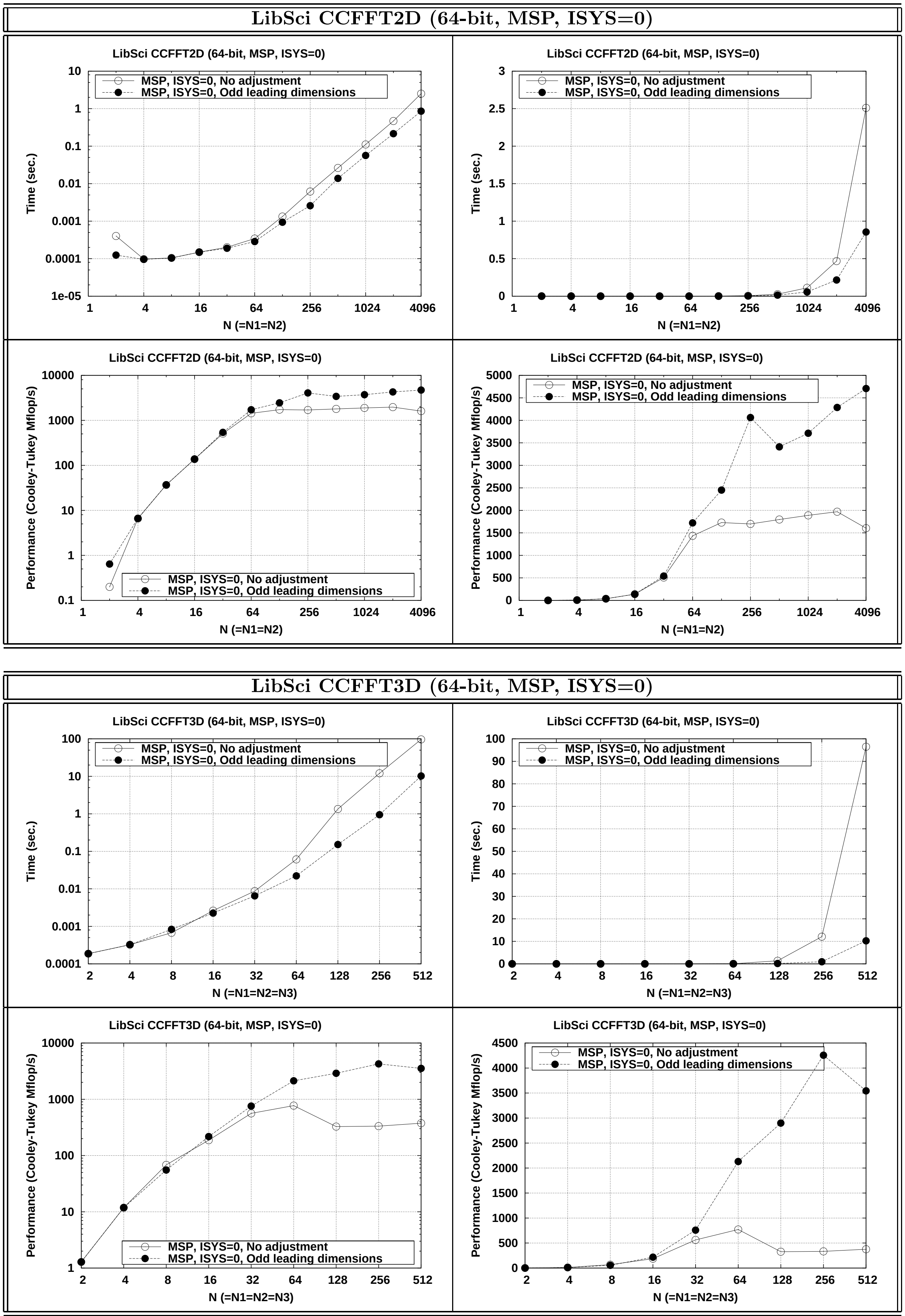

FIG. 1. *Time and Performance of LibSci CCFFT2D and CCFFT3D (64-bit, MSP mode, ISYS=0). Effect of adjusting leading dimensions.*

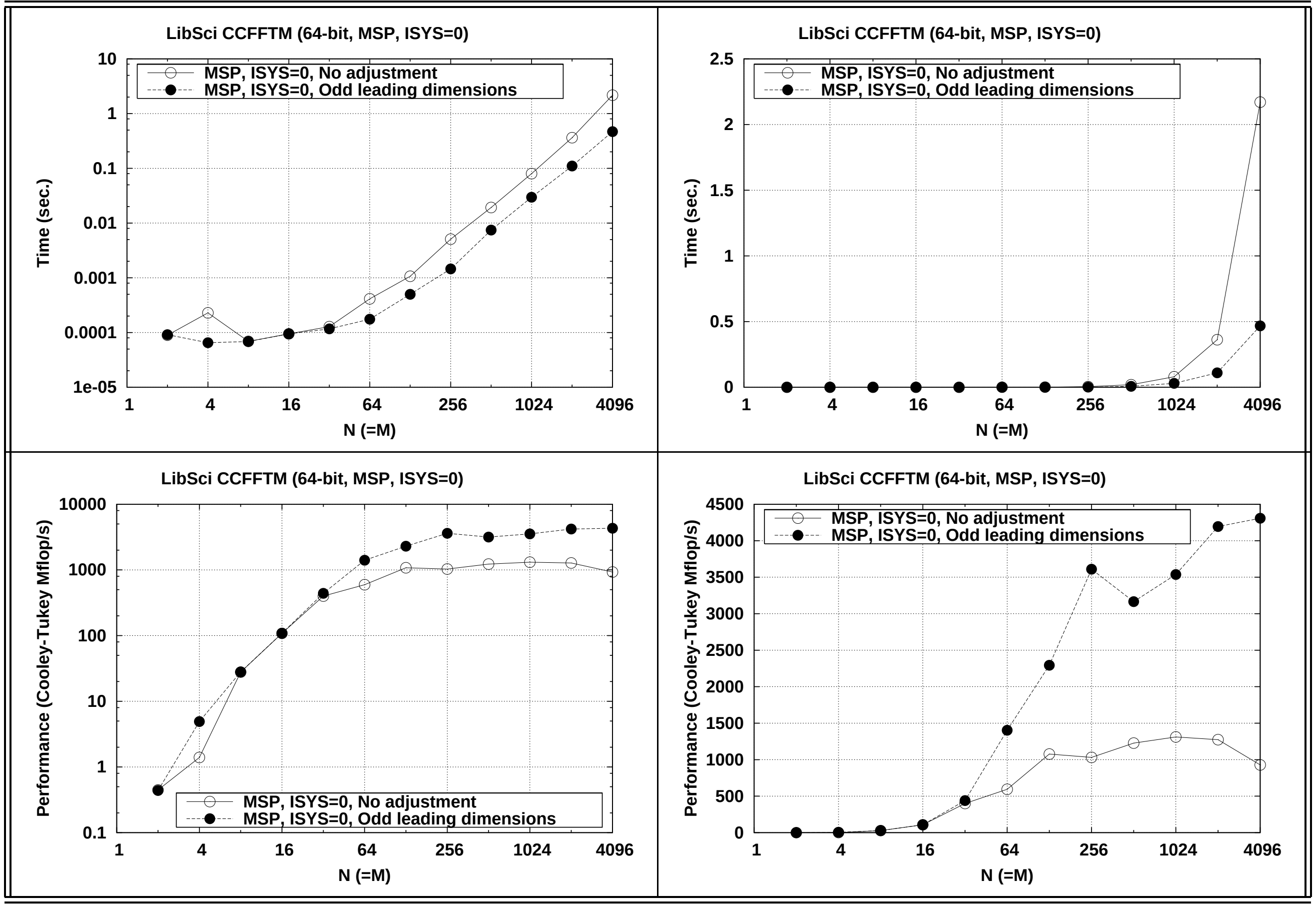

Fig. 2. *Time and Performance of LibSci CCFFTM (64-bit, MSP mode, ISYS=0). Effect of adjusting leading dimensions.*

Note that for the Cray X1 system, results are given for all sizes in the given ranges that have only factors that are powers of 2, 3 and 5. As with previous graphs, any data connected via lines is for visual purposes with no intended interpolations.

Figures 4–7 show timing and performance measurements for single 1-D, multiple 1-D, 2-D and 3-D complex-to-complex routines CCFFT, CCFFTM, CCFFT2D and CCFFT3D. The data show for the cases depicted that the Cray X1's LibSci FFTs for an MSP run faster than on the other systems or on an SSP for larger sized problems. On the Cray X1 system, users may want to use the SSP mode variety of FFTs for smaller sized problems.

6 Future Plans

The future of FFTs on the Cray X1 system, include further optimizations and possible new algorithms and functionality improvements.. Optimizations may include more effective use of the data cache, better instruction scheduling for complex-to-complex radix 3 and radix 5 butterflies, providing the runtime choice of using allocating vs. non-allocating vector load and store operations.[2] Additionally, we may explore coding the real-to-complex and complex-to-real mixed radix butterflies in assembly language.

Regarding functionality a Fortran90 module interface block for LibSci is anticipated for 2004. This will help manage using LibSci in Fortran90 programs across all the types given. The most significant improvement will the be addition of 2-D and 3-D distributed memory parallel FFTs, anticipated for release by the end of 2003.

[2]Non-allocating vector memory operations are used currently.

Table 7

Possible Factors to Compare Among LibSci FFT Routines.

Dimension	**Transform Type**	**Systems**	**Implementation**	**Data Type**	**Stride Adjustments**
1-D	Complex-to-complex	Cray X1 (MSP)	`ISYS`=0	Single precision default LibSci	None
2-D	Real-to-complex	Cray X1 (SSP)	`ISYS`=1	Double precision default LibSci	Odd leading dimensions
3-D	Complex-to-real	Cray SV1ex		Single precision LibSci (64-bit)	Odd multiple of 2 leading dimension/odd 2nd leading dimension
Multiple 1-D		Cray T90			

7 Conclusions

We have discussed the fast Fourier Transform (FFTs), convolution, and filtering portions of LibSci for the Cray X1 system. Users should be mindful of porting issues regarding chosing the desired LibSci variant in conjunction with the data types for the situation and desired accuracy. The `TABLE` array must always be an array of 64-bit words. The `WORK` array may need adjusting as the size requirements may vary from previous Cray systems. For users not familiar with FFTs, the FFT lengths should be chosen wisely to get actual FFT implementations (lengths should contain only factors that are powers of 2, 3 and 5). If possible, users of the 3-D FFTs may wish to provide a larger `WORK` array to achieve increased performance. On the Cray X1 system, larger problem dimensions fare better than smaller ones and the leading dimensions should be adjusted for good strides.

8 Acknowledgements

The author thanks Mary Beth Hribar for reviewing code and general consultations, Rick Hangartner for general consultations, Catherine Knutson for reviewing and running FFT test code, Neal Gaarder and Jim Schwarzmeier for CAL support, and Wendy Thrash for help optimizing the the radix 3 and radix 5 complex-to-complex butterflies.

References

[1] Cray Inc., *Cray X1(TM) User Environment Differences*, 2003. S-2310-50.
[2] ———, *Migrating Applications to the Cray X1(TM) System*, 2003. S-2378-50.
[3] ———, *Optimizing Applications on the Cray X1(TM) System*, 2003. S-2315-50.
[4] M. B. Hribar, *Cray X1 Scientific Libraries*, in *Proceedings* of the Cray User Group Conference, May 2003.

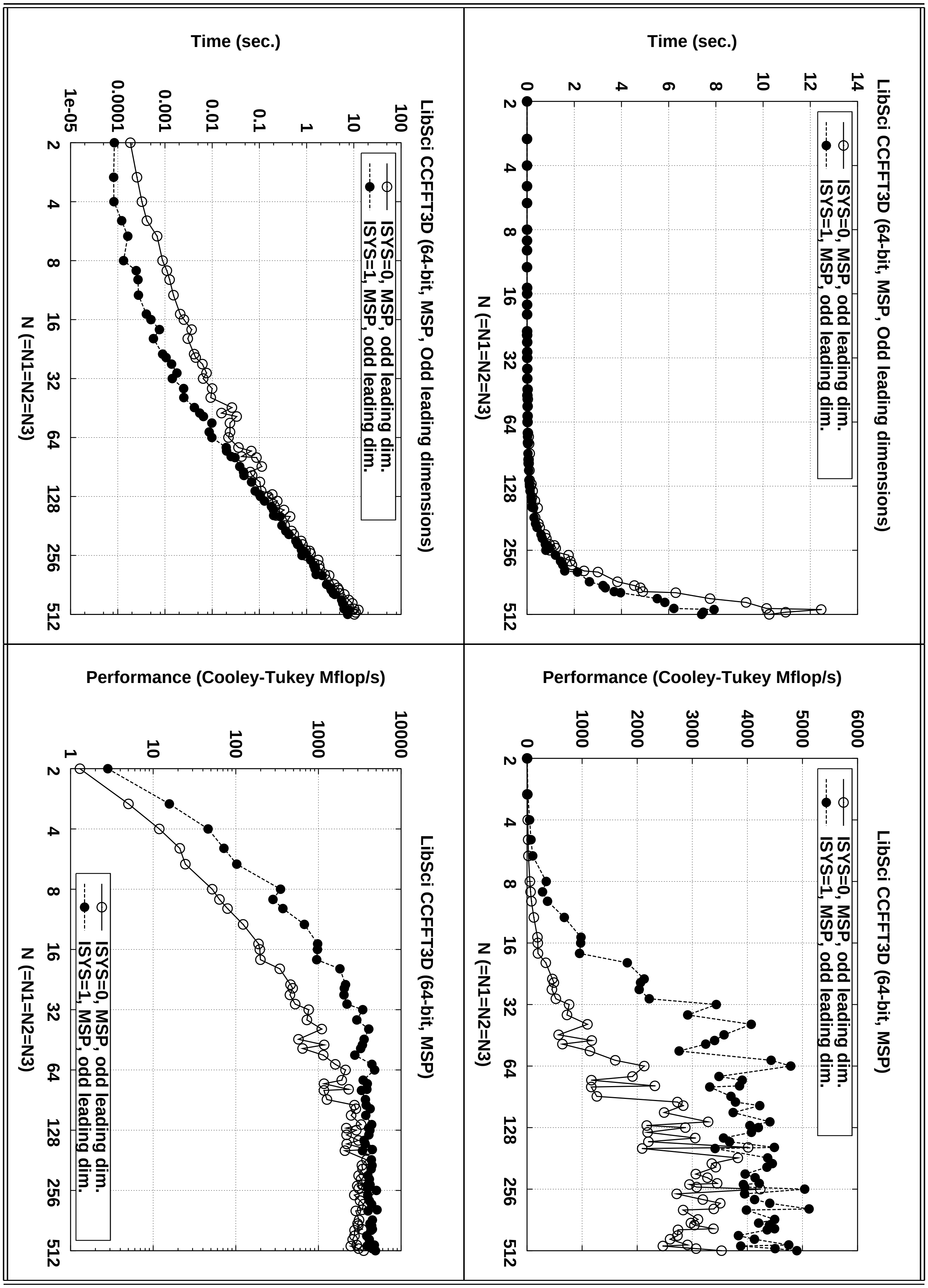

FIG. 3. *Time and Performance of LibSci CCFFT3D (64-bit, MSP mode, Odd Leading Dimensions). Effect of using alternate algorithm: ISYS=0 and ISYS=1.*

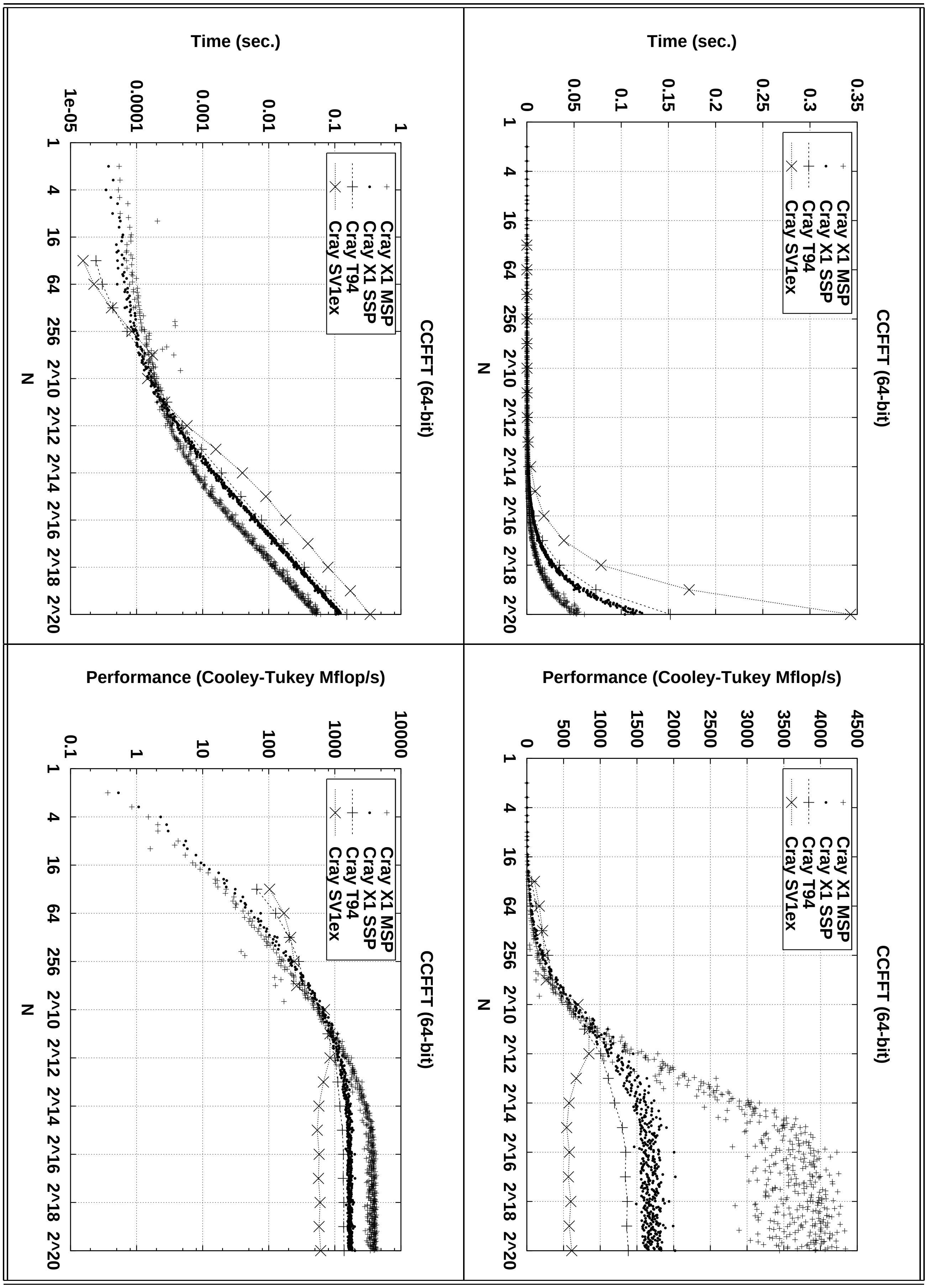

Fig. 4. *Time and Performance Comparisons of LibSci CCFFT (64-bit).*

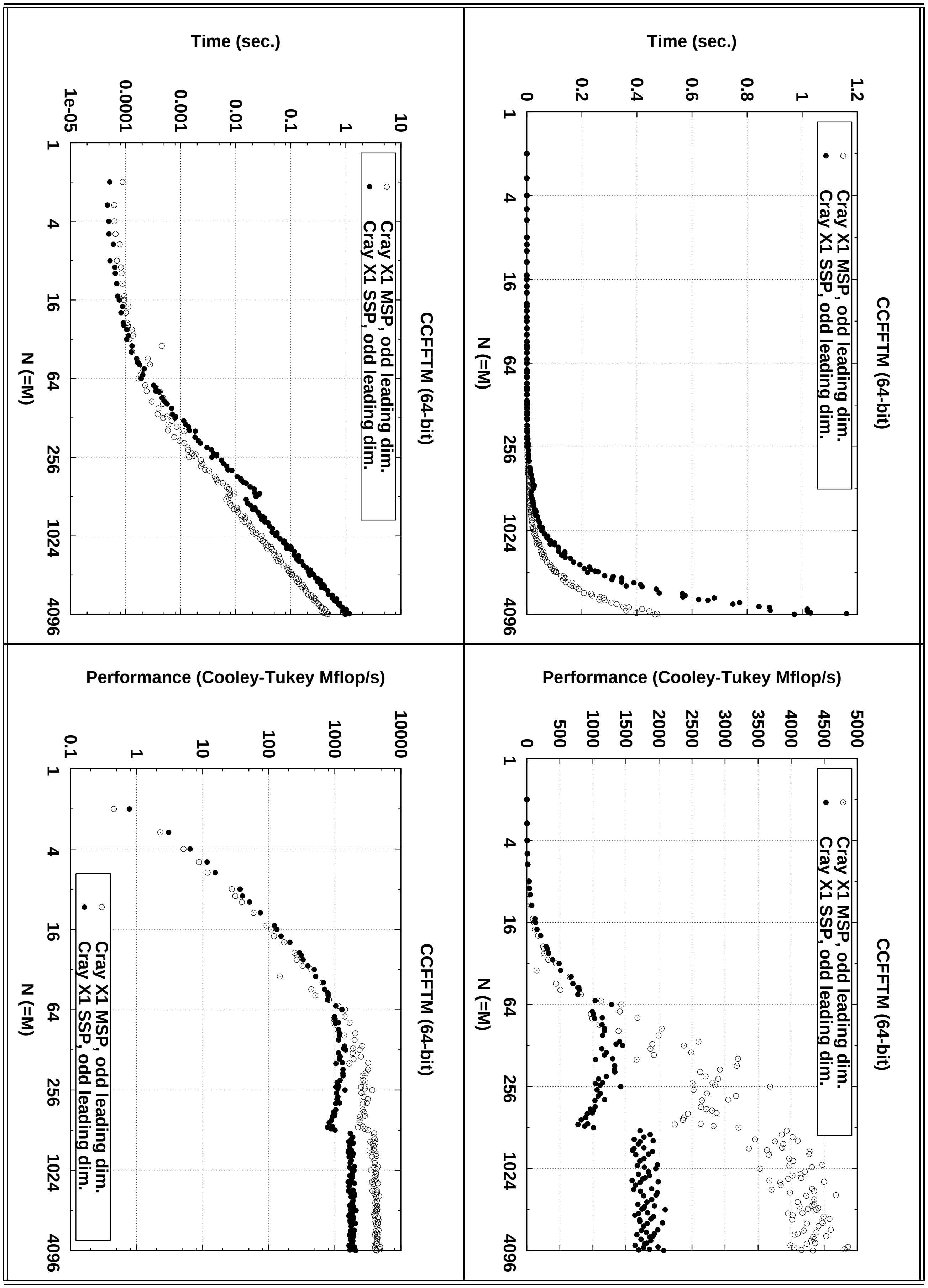

Fig. 5. *Time and Performance Comparisons of LibSci CCFFTM (64-bit).*

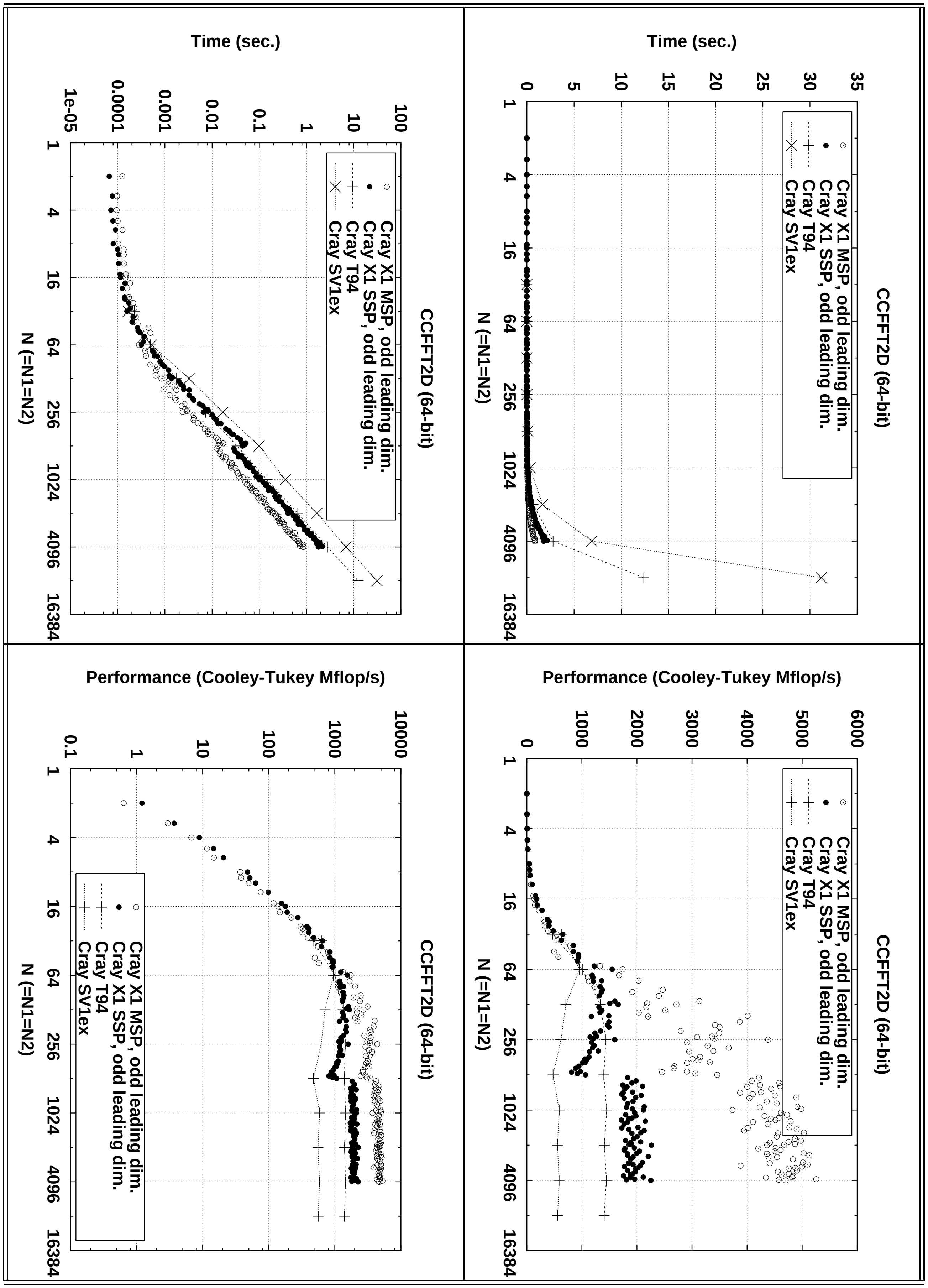

Fig. 6. *Time and Performance Comparisons of LibSci CCFFT2D (64-bit).*

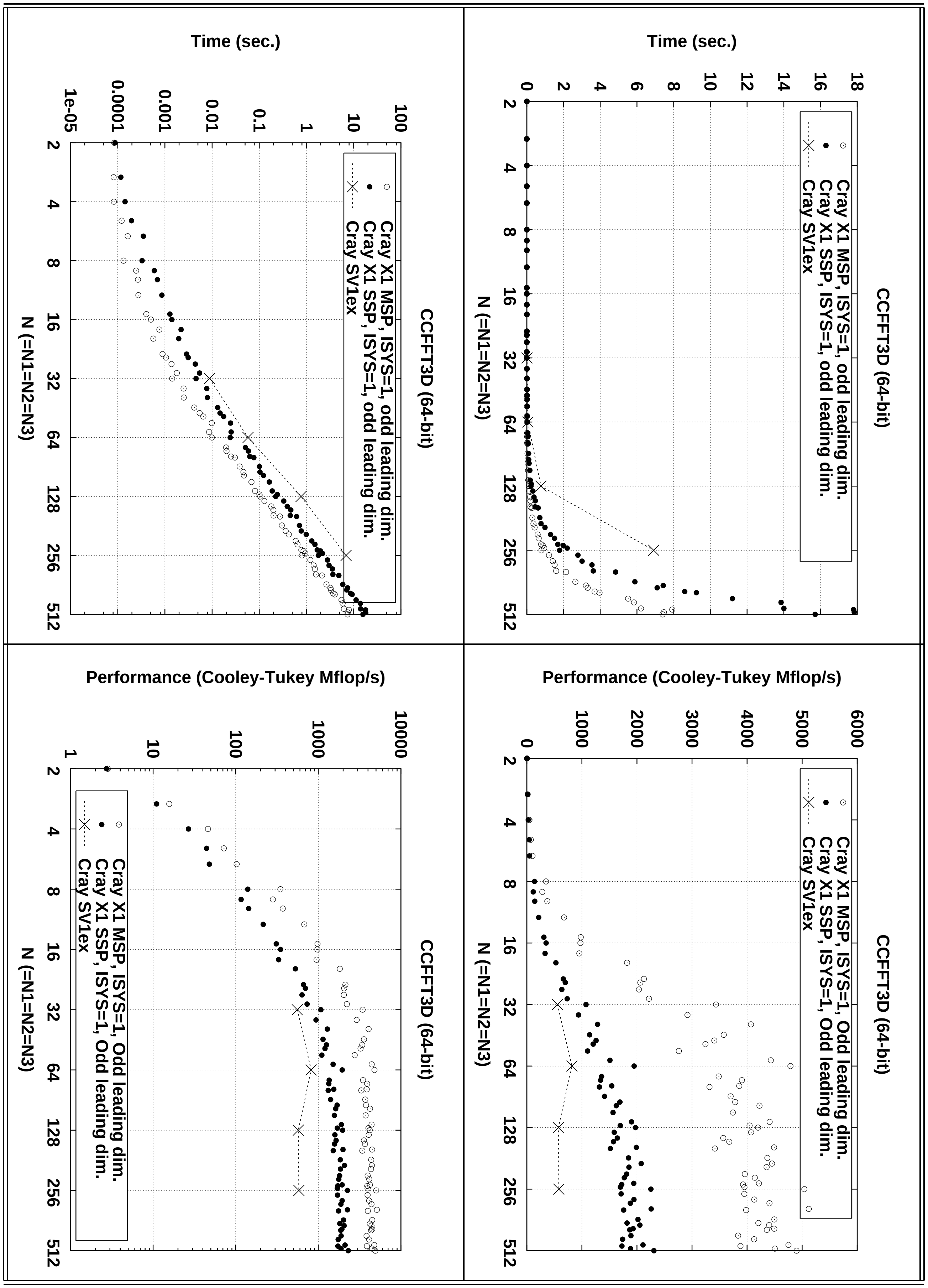

Fig. 7. *Time and Performance Comparisons of LibSci CCFFT3D (64-bit).*