

Parallel 2-D FFT Implementation With TMS320C4x DSPs

Application Report

Rose Marie Piedra
Digital Signal Processing — Semiconductor Group

SPRA027A
February 1994



IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Introduction

Fourier transform techniques are of fundamental importance in digital signal processing (DSP) applications. Among the most commonly used algorithms in image processing is the Fast Fourier Transform (FFT). FFT is used for computation of the Discrete Fourier Transform (DFT).

FFT computations can be used to solve image correlations and convolutions. Two-dimensional convolutions and correlations are used for feature extraction in image processing. For example, applications on fluid dynamics (2-D turbulences) can lead to calculation of velocity vectors and gradients. One important advantage of using frequency domain tools over direct methods is faster execution. The FFT algorithm significantly reduces the computation time of the DFT.

This application note compares serial and parallel implementations of 2-D complex FFTs with the TMS320C40 processor. Special attention is given to parallel implementation of 2-D FFTs. The increasing demands for speed and performance in some real-time DSP applications make sequential systems inadequate. Parallel systems provide higher throughput rates.

The algorithms were implemented on the Parallel Processing Development System (PPDS), a system with four TMS320C40s and with both shared- and distributed-memory support.

This report is structured as follows:

<i>2-D FFT Algorithm</i>	Gives a brief review of the FFT algorithm and its extension to the 2-D case. Describes applications of FFTs in the calculation of correlation and convolution algorithms.
<i>Parallel 2-D FFT</i>	Focuses on parallel implementations of 2-D FFTs. Shared- and distributed-memory implementations are considered, as well as the TMS320C40's suitability for each.
<i>TMS320C40 Implementation</i>	Presents the results of shared- and distributed-memory implementations of parallel 2-D FFT realized on the PPDS. Gives analyses of speed-up and efficiency.
<i>Conclusion</i>	States conclusions.
<i>Appendixes</i>	Lists the code for performing serial and parallel 2-D FFTs in C and 'C40 assembly language code.

The 2-D FFT Algorithm

The Discrete Fourier Transform (DFT) of an n -point discrete signal $x(i)$ is defined by:

$$X(k) = \sum_{i=0}^{n-1} x(i) W_n^{ik}$$

where $0 \leq k \leq n - 1$ and $W_n = e^{-j2\frac{\pi}{n}}$.

Direct DFT computation requires $O(n^2)$ arithmetic operations. A faster method of computing the DFT is the Fast Fourier Transform (FFT) algorithm. If FFT is used to solve an n -point DFT, $(\log_2 n)$ steps are required, with $n/2$ butterfly operations per step. The FFT algorithm therefore requires

approximately $\frac{n}{2} \log_2 n \approx O(n \log_2 n)$ arithmetic operations (which is $\frac{n}{\log_2 n}$ times faster than direct DFT computation). See [3], [6], and [4] for a more detailed analysis of the 1-D FFT case.

Two-dimensional DFT can be defined in a manner similar to the 1-D case [10]. The 2-D DFT is given by:

$$X(k_1, k_2) = \sum_{i_1=0}^{n-1} \sum_{i_2=0}^{n-1} x(i_1, i_2) W_n^{(i_1 k_1 + i_2 k_2)}$$

where $0 \leq k_1, k_2 \leq n-1$ and $W_n = e^{-j2\frac{\pi}{n}}$.

A standard approach to computing the 2-D FFT of a matrix A is to perform a 1-D FFT on the rows of A, giving an intermediate matrix A', then performing a 1-D FFT on the columns of A' [10]. This is the approach followed in this application report.

Timing Analysis

A 2-D FFT of a complex matrix of size $(n \times n)$ requires the execution of a 1-D FFT on n rows, followed by a 1-D FFT on n columns. The number of arithmetic operations required will therefore be as follows:

$$\text{Time} = n * O(n \log_2 n) + n * O(n \log_2 n) \approx O(n^2 \log_2 n)$$

(FFT on n rows) (FFT on n columns)

Application of FFT on Correlation/Convolution Algorithms

Relationships between image and transform domains can be described by convolution and correlation. Convolution is used for linear interpolation or filtering. Correlation plays an important role in feature extraction in image processing. These image operations are computationally intensive; Fourier transforms can be used to enhance speed.

The correlation of two sequences $x(i)$ and $y(i)$ of length n is defined as [10]:

$$w(i) = \sum_{k=0}^{n-1} x(k) y(i+k)$$

For a 1-D correlation, the common direct approach (in time domain) based on shift-and-multiply operations requires $O(n^2)$ arithmetic operations.

Based on the convolution property of the Fourier transform [3], an efficient way to compute correlation is by using FFT and inverse FFT (IFFT), as illustrated below:

1. Compute FFT $\{x(i)\}$ and FFT $\{y(i)\}$.
2. Multiply FFT $\{x(i)\}$ by the complex conjugate of FFT $\{y(i)\}$.
3. Compute the IFFT of this result.

Similarly, a convolution operation reduces to a simple multiplication in the Fourier domain. FFT correlation/convolution becomes computationally faster than spatial convolution for large images.

Speed-up is approximately $\frac{n^2}{n \log_2 n} = \frac{n}{\log_2 n}$, which is significant when dealing with very large images.

Parallel 2-D FFT

A 2-D FFT is an intrinsically parallel operation; a 1-D FFT is applied separately to each row and column of a matrix.

The Parallel Algorithm

Let $n = qp$, where n is the order of the squared matrix A , p is the number of processors, and $q \geq 1$ is an integer. The basic idea is to allocate a unique working set of rows/columns to each processor. The algorithm consists of three basic steps:

Step 1. FFT on rows: Processor i executes 1-D sequential FFT on rows $qi, qi+1, \dots, qi+q-1$, with $i = 0, 1, \dots, p-1$. Because each processor executes a 1-D FFT on q different rows, this step requires $q * O(n \log_2 n) \approx O\left(\frac{n^2}{p} \log_2 n\right)$ arithmetic operations.

Step 2. Matrix Transposition: Because column elements are not stored in contiguous memory location, row/column transposition of matrix A is necessary prior to executing FFT on columns. Matrix transposition requires $(n-1) + (n-2) + \dots + 1 = \frac{n(n-1)}{2}$ exchange steps, where n is the order of the matrix [7]. The computation delay involved in this operation is therefore $O(n^2)$ for serial execution or $O\left(\frac{n^2}{p}\right)$ for parallel execution.

Step 3. FFT on columns: Same as in Step 1, but by column.

Speed-Up Analysis

The speed-up factor can be calculated as follows:

$$\text{Speed-up} \approx \frac{n^2 \log_2 n}{\frac{n^2}{p} \log_2 n} \approx O(p)$$

The parallelism in the 2D-FFT is suitable for implementation on distributed-memory or shared-memory multiprocessors. Let's consider those cases.

Shared-Memory Implementation

Matrix A is stored in global memory, so each processor has easy access to all the rows/columns. Even when all processors share the same physical data memory, each processor points to a different row/column working set.

Shared-memory systems require careful consideration of the memory-contention problem. Matrix transposition (Step 2) is simpler, but row/column access can create a major bottleneck.

Shared-memory implementation requires at least $2 * n * n = 2n^2$ words of shared memory. If this amount of RAM is unavailable in the system, consider either intermediate downloading of files or distributed-memory implementation as an alternative.

Figure 1 illustrates the shared-memory 2-D FFT implementation for $p = 4$ and $n = 8$.

Distributed-Memory Implementation

Matrix A is partitioned into p regions. Each region contains q rows and is assigned to each processor's local memory. Processors communicate via message passing.

- Steps 1 and 3 of the parallel 2-D FFT algorithm are executed in the local memory of each processor. No interprocessor communication is necessary.
- Step 2, matrix transposition, is more complex because matrix A is distributed between processors by row. You must perform message passing of row segments before you execute matrix transposition. This procedure can be described as follows:

- **Total-exchange step:** Processor i sends to processor j columns $qj, qj+1, \dots, qj+q-1$ of each of the rows allocated to it, where $0 \leq j < p$ and $i \neq j$. In such DMA-supported devices as the TMS320C40, this step can be executed simultaneously with Step 1, after computation of each row FFT. Better DMA-CPU parallelism can thus be achieved. This is the approach followed in the parallel 2-D FFT (distributed-memory implementation) presented in this application report.
- **Transposition of submatrices:** After the *total exchange* step, each processor contains all the column elements needed to perform a row/column transposition. Transposition is executed on p squared submatrices of size $(q \times q)$. Submatrix G_k contains elements $(kq + i, j)$, where $(0 \leq i, j < q)$ and $(0 \leq k < p)$. The computation delay involved in this operation is $p O(q^2) = O\left(\frac{n^2}{p}\right)$.

Figure 2 illustrates Step 2 of the distributed-memory implementation, with $p = 4$ and $n = 8$.

- **Memory requirements:** Each processor requires at least $2n^2/p$ words of local memory to store the (n/p) rows allocated to it.
- **Required topology:** This implementation requires a fully connected multiprocessor configuration. Other configurations with rerouting capabilities are also feasible. Refer to [9] for information on *total exchange* techniques for configurations for cube, mesh, and linear arrays.
- **Output result:** Matrix results are stored by column, with column elements stored in successive memory locations in the local memory of each processor. Processor i contains columns $qi, qi+1, \dots, qi+q-1$, where $i = 0, 1, \dots, p-1$.

Figure 1. 2-D FFT Shared-Memory Implementation

FFT on Rows:

Matrix A (Shared Memory)

Processor 0 →	00	01	02	03	04	05	06	07
	10	11	12	13	14	15	16	17
Processor 1 →	20	21	22	23	24	25	26	27
	30	31	32	33	34	35	36	37
Processor 2 →	40	41	42	43	44	45	46	47
	50	51	52	53	54	55	56	57
Processor 3 →	60	61	62	63	64	65	66	67
	70	71	72	73	74	75	76	77

FFT on Columns:

Matrix A (Shared Memory)

Processor 0 ↓	Processor 1 ↓	Processor 2 ↓	Processor 3 ↓
00 01	02 03	04 05	06 07
10 11	12 13	14 15	16 17
20 21	22 23	24 25	26 27
30 31	32 33	34 35	36 37
40 41	42 43	44 45	46 47
50 51	52 53	54 55	56 57
60 61	62 63	64 65	66 67
70 71	72 73	74 75	76 77

NOTE:

Matrix element $A[i][j]$ = ij
Matrix size = $n = 8$
Number of processors = $p = 4$

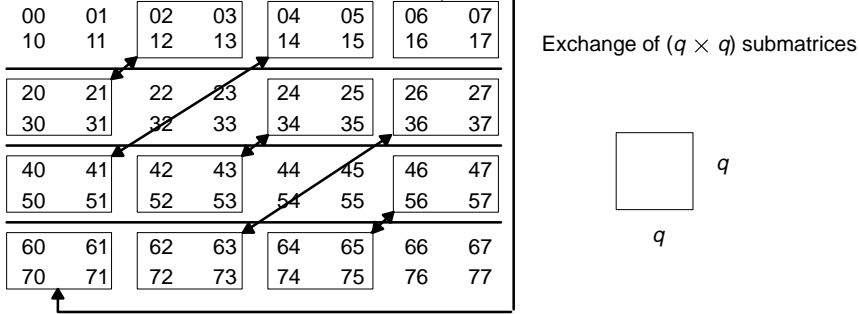
Figure 2. 2-D FFT Distributed-Memory Implementation

(Step 2: Transposition of Submatrices)

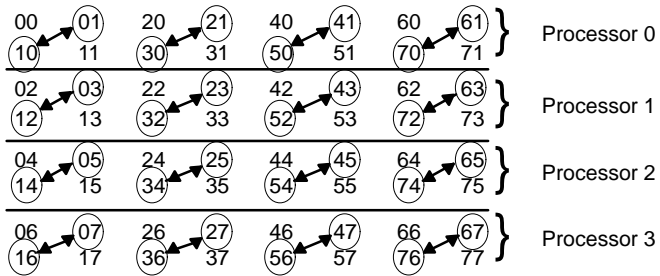
Matrix A After Processors Have Completed FFTs on All the Rows

00	01	02	03	04	05	06	07	} Processor 0 (rows 0,1)
10	11	12	13	14	15	16	17	
20	21	22	23	24	25	26	27	} Processor 1 (rows 2,3)
30	31	32	33	34	35	36	37	
40	41	42	43	44	45	46	47	} Processor 2 (rows 4,5)
50	51	52	53	54	55	56	57	
60	61	62	63	64	65	66	67	} Processor 3 (rows 6,7)
70	71	72	73	74	75	76	77	

Matrix A During Total Exchange Step



Matrix A After Total Exchange Step and During Submatrix Transposition



Matrix A After Transpositions of Submatrices (ready for FFT on columns)

00	10	20	30	40	50	60	70	} Processor 0 (columns 0,1)
01	11	21	31	41	51	61	71	
02	12	22	32	42	52	62	72	} Processor 1 (columns 2,3)
03	13	23	33	43	53	63	73	
04	14	24	34	44	54	64	74	} Processor 2 (columns 4,5)
05	15	25	35	45	55	65	75	
06	16	26	36	46	56	66	76	} Processor 3 (columns 6,7)
07	17	27	37	47	57	67	77	

TMS320C40 Implementation

The TMS320C40

The TMS320C40 is the world's first parallel-processing DSP. In addition to a powerful CPU that can execute up to 11 operations per cycle with a 40- or 50-ns cycle time, the TMS320C40 contains six communication ports and a multichannel DMA [8]. The on-chip communication ports allow direct (glueless) processor-to-processor communication, and the DMA unit provides concurrent I/O by running parallel to the CPU. Special interlocked instructions also provide support for shared-memory arbitration. These features make the TMS320C40 suitable for both distributed- and shared-memory computing systems [2].

The 2-D FFT algorithm was implemented on the TMS320C40 Parallel Processing Development System (PPDS). The PPDS includes four TMS320C40s, fully interconnected via the on-chip communication ports. Each 'C40 has 256KB of local memory SRAM, and all share a 512KB global memory [5].

General features of the programs:

- All the programs have been written to be independent of the FFT size and the number of processors in the system. Further optimization is possible for a fixed number of processors.
- Real and imaginary parts of complex numbers are stored in successive memory locations.
- Both C and assembly language versions of the programs are in the appendices. The programs can be downloaded from the TMS320 bulletin board at (713) 274-2323. Set your modem to 8 data bits, 1 stop bit, no parity.
- For the C programs, there are core functions, such as the 1-D FFT and CPU complex moves, and routines to set DMA register values in assembly code to enhanced optimization. For the assembly programs, most of the functions are in-lined to avoid the delay associated with calling routines. But, in order to keep the program flexible, the 1D-FFT has been retained as a subroutine. The new 'C40 LAJ and BUD R11 instructions permit routine calls with just one delay cycle. To make the programs more generic, the (5/4)-cycle sine/cosine table [4] and the input matrix are provided in separate files.
- The radix-2 1-D FFT routine presented in Appendix D is used as the core for the 2-D FFT implementation. But you can use any FFT routine that complies with the calling conventions. Thus, as faster 1-D FFT algorithms are developed, they can be used to implement faster 2-D FFT algorithms.
- The 'C40 timer 0 and the timer routines in the parallel runtime support (PRTS) library, available with the 'C40 C compiler, are used for benchmark measures. The real benchmark timing is equal to the timer counter value multiplied by $2 * (\text{'C40 cycle time})$. For the parallel programs, the total execution time of the parallel algorithm can be defined as $T = \max(T_i)$, where T_i is the execution time taken by processor i . Note that in the programs, T_i is the time between labels $t2$ and $t0$.
- The load-imbalancing case was not considered. Refer to [2] for an example of this case.
- The compiler/assembler tools were run under OS/2 to avoid memory-limitation problems with the optimizer.

Serial Implementation

Serial implementations of the 2-D FFT provide accurate speed-up measures for the parallel programs. Appendix A illustrates single- and double-buffered 2-D FFT serial implementations in C and 'C40 assembly code.

Observations:

- Although the 1-D FFT can be executed directly in off-chip RAM, the preferred method is to transfer the row/column to on-chip RAM first. This fully exploits the dual-access single-cycle characteristic of the 'C40 on-chip RAM for some parallel instructions. This transfer delay can be minimized with double-buffering techniques.
- Double-buffering technique:
 - CPU operations are confined to computing 1-D FFT (1 row at a time). The DMA processor performs the data transfer between on-chip RAM and external RAM, providing the CPU with a new set of data. This requires a continuous CPU-DMA synchronization.
 - While the CPU is computing FFT on row/column i , the DMA processor transfers the vector result of row/column $(i-1)$ bit-reversed and the next row/column $(i+1)$ from external RAM to on-chip RAM to have it ready for the next FFT computation. This technique is called double buffering.
 - The $2K \times 32$ -bit-word on-chip RAM constantly holds 2 buffers. Each buffer must be located in a different on-chip RAM block. Because each on-chip RAM block has an independent bus path, CPU/DMA access conflict is minimized. You can compute up to 1K-point real FFT or 512-point complex FFT in on-chip memory. If the double-buffering technique is not used, the system can compute up to 2K-point real FFT or 1K-point complex FFT.
- For DMA bit-reversed complex transfers, you can use autoinitialization to transfer the real part of the FFT vector result first and the imaginary part later. You must set the “read bit-rev” bit (control register bit 12) to 1 and the source address index to the FFT size (n). Given the 'C40 architecture, no extra delay occurs with CPU/DMA bit-reversed addressing.
- For DMA column transfers, autoinitialization is also used to transfer the real and imaginary parts of each complex vector.
- Given the offset addressing capabilities of the 'C40, the transposition step requires no extra cycles when moving columns from off-chip to on-chip RAM.

Shared-Memory Parallel Implementation

Appendix B contains single- and double-buffered versions of the 2-D FFT (shared-memory version) in C and 'C40 assembly code.

Observations:

- A node ID (`my_node`) is allocated by software to each processor. In this way, each processor automatically selects its associated row/column working set.
- Each row/column is initially transferred to on-chip memory to minimize memory access conflict among the processors. Using the DMA for double-buffering minimizes not only this access delay but also the effect of a nonzero wait-state global memory similar to that of the PPDS.
- Interprocessor synchronization is required before you execute FFT on columns. Synchronization is implemented via a counter flag in global memory. Every processor increments the counter by 1 after completing the execution on the rows allocated to it. In this way, the processors begin executing FFT on columns only after the counter equals p (number of processors).

- The transposition step that is necessary prior to executing FFT on columns is implemented simultaneously with the transfer of columns to on-chip memory, with no delay penalty.
- The global memory of the PPDS can contain a complex matrix with a maximum of 256×256 elements. Because of the need for an extra location for the synchronization counter, the program has been tested with a maximum of 128×128 elements.
- For benchmarking of shared-memory programs, a global start of all the processors is absolutely necessary; otherwise, the real-memory-access conflict resolution will not occur. To facilitate this process, a C-callable routine (*syncount.asm*) is provided in Appendix D for debugging systems lacking global start capability. Rotating priority for shared memory access should be selected by setting the PPDS LCSR register to 0x40.

Distributed-Memory Parallel Implementation

Two distinct single-buffered 2-D FFT implementations were used:

- Use of DMA only for interprocessor communication (See *dis1.c* in Appendix C).
- Use of DMA for interprocessor communication and matrix transposition (See *dis2.c* and *dis2.asm* in Appendix C).

Observations:

- The six-channel DMA coprocessor is used for interprocessor communication during the *total-exchange* step as follows:
 - As soon as 1-D FFT is completed on a row, the DMA coprocessor will be in charge of transferring (n/p) complex points of this result already stored in local memory, from one processor to the other in *total exchange* fashion.
 - Each processor will transmit a total of $(n/p)(p-1) \approx O(n)$ complex numbers per row. In the PPDS, a memory-to-memory interprocessor transfer operation of an integer number requires seven clock cycles—four to transmit the 4-byte word, two to write it to/from memory, and one to set up the communication channel. The communication delay will therefore be approximately $7 * 2 * n = 14n$ clock cycles.
- Careful consideration of the communication delay involved is necessary to achieve true CPU-DMA parallelism. If the *total exchange* step requires more time than the 1-D FFT computation, the application will slow down.
- DMA channels are set in split mode [8] with source and destination synchronization using the ICRDY and OCRDY port signals, respectively. In this way, DMA will be interrupted when there is new data to read in the input FIFO. A value will be written to the output FIFO if the output FIFO is not full. Transferring is done in a linear fashion (not bit-reversed).
- Because the interprocessor communication occurs in an *exchange* fashion, no extra memory is needed for temporary buffers. Source address and destination addresses are set to the same address values. Destination node IDs help to determine the location of the data to be exchanged. Data will never overlap, because the communication port FIFOs act as data buffers, **as long as the communicating processors start executing the *exchange.asm* routine at approximately the same time**. This can be achieved by using a common system reset or the parallel debugger manager (PDM), which is part of the 'C4x emulator.

For systems without common reset, use the *exch2.asm* routine instead of *exchange.asm*. The *exch2.asm* routine can be downloaded from the TMS320 bulletin board at (713) 274-2323. Set your modem to 8 data bits, 1 stop bit, no parity.

- The destination node is selected in such a way that each pair of processors are synchronized to *talk* to each other at approximately the same moment, thus facilitating communication scheduling and avoiding a system lock that could occur if a processor sent data with no processor ready to receive it. You select the destination node by using a XOR operation: $(my_node) \text{ XOR } (i)$, $0 < i < p$. In the case of a 4-processor system, the following situation exists during the first step ($i = 1$):

Processor 0 : $(my_node = 0) \text{ XOR } 1 = 1$

Processor 1 : $(my_node = 1) \text{ XOR } 1 = 0$

Processor 2 : $(my_node = 2) \text{ XOR } 1 = 3$

Processor 3 : $(my_node = 3) \text{ XOR } 1 = 2$

Processors 0,1 and 2,3 select each other for the first data exchange. Similar analysis can be done for the other steps.

- Matrix *port* is the connectivity matrix and shows the connectivity between the processors. Processor *i* is connected to processor *j* through *port*[*i*][*j*]. This matrix is the only system-specific part of the program.
- To attain as even a transmission as possible between processors, shifting priority among DMA channels has been selected.
- To synchronize DMA/CPU operation, each processor must know whether transferring is complete on a DMA channel before initializing the DMA with a new set of values. In unified mode, you can check for completion either by determining whether the start bits (DMA control register) are set to 10 or by checking the IIF register (if TCC was previously set to 1 in the DMA control register). In split mode, even when the corresponding bit in the IIF register is set, there is no guarantee that transfer is complete on both the primary and secondary channels. For this reason, the preferred method is to determine whether the start/aux_start bits (DMA control register) are both set to 10. In the programs, both methods have been used.
- DMA can be used for matrix transposition also. While the CPU is performing a 1-D column FFT (in on-chip RAM), the DMA is doing the next row/column transposition in off-chip memory.
- Double-buffered distributed-memory implementations were not described, but the approach is similar for shared memory.

Implementation Results

The following 2-D FFT programs were implemented and tested on the 'C40 PPDS (see Appendices for source code):

- SER.C/SER.ASM: Single-buffered serial implementations (C/assembly language code versions)
- SERB.C/SERB.ASM: Double-buffered serial implementations (C/assembly language code versions)
- SH.C/SH.ASM: Single-buffered shared-memory implementations (C/assembly language code conversions)
- SHB.C/SHB.ASM: Double-buffered shared-memory implementations (C/assembly language code versions)

- DIS1.C: Distributed-memory implementation, with DMA being used only for interprocessor communication. (C code version)
- DIS2.C/DIS2.ASM: Distributed-memory implementation, with DMA being used for interprocessor communication and matrix transposition (C/assembly code versions).

Speed-up of a parallel algorithm is defined as $S_p = T_s/T_p$, where T_s is the serial time ($p = 1$) and T_p is the time of the algorithm executed using p processors. Efficiency is defined as $E_p = S_p / p$, where $0 < E_p < 1$ [2]. An efficiency below 50% reflects poor parallel implementation performance. Figure 4 through Figure 13 show speed-up and efficiency figures obtained for the shared- and distributed-memory programs implemented on the PPDS. The figures are based on the TMS320C40 2-D FFT timing benchmarks shown in Table 1. Execution time for program i is denoted as T_i .

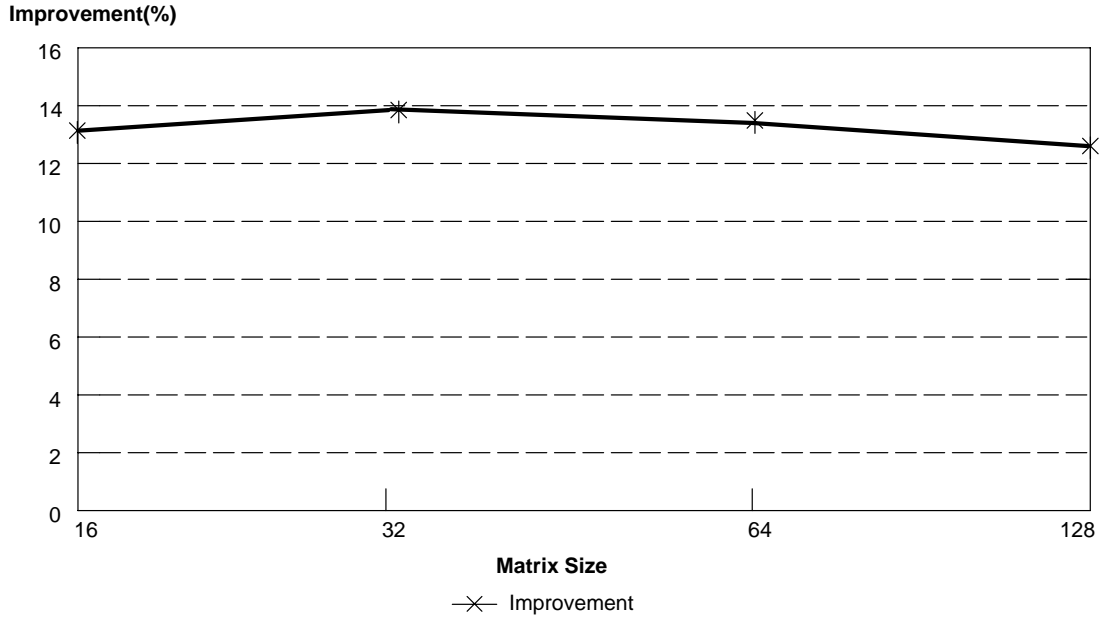
The following analysis shows the effect of the matrix size and the number of processors in the system.

- Serial implementations:
 - There was a 13% improvement using double-buffering in the serial program. (See Figure 3)
 - Using registers to pass function parameters had a positive effect on the performance of the C implementations (using assembly language core functions). As seen in Table 1, the timing difference between C and assembly code is minimal for large matrices.
- Shared-memory implementations:
 - The double-buffered shared-memory implementation displays a better performance than the single-buffered shared-memory version for $p = 2$ (see Figure 7). The DMA helps to reduce the data transfer delay. For $p = 4$, however, the performance declines because of the increase in shared-memory arbitration. In this case, the single-buffered shared-memory implementation is more beneficial (see Figure 11).
 - Shared-memory programs are strongly affected by the design of the shared-memory arbitration unit. For example, in the case of the PPDS, a processor will not release access to shared-memory during back-to-back reads. The speed of the single-buffered shared-memory implementation is thus increased because of the reduction in the delay penalty for continuous switching.
 - Efficiency decreases with more processors because the memory conflict delay increases. This effect can be seen in Figure 13, where efficiency figures are plotted against the number of processors in the system.
- Distributed-memory implementations:

The distributed-memory implementations show an excellent performance. Speed-up/efficiency for large matrices is high, and the decline in the efficiency when the number of processors increases is very slight (Figure 13). Performance also improves when DMA is used to help with the combined task of matrix transposition and interprocessor communication. See Figure 4 and Figure 5.

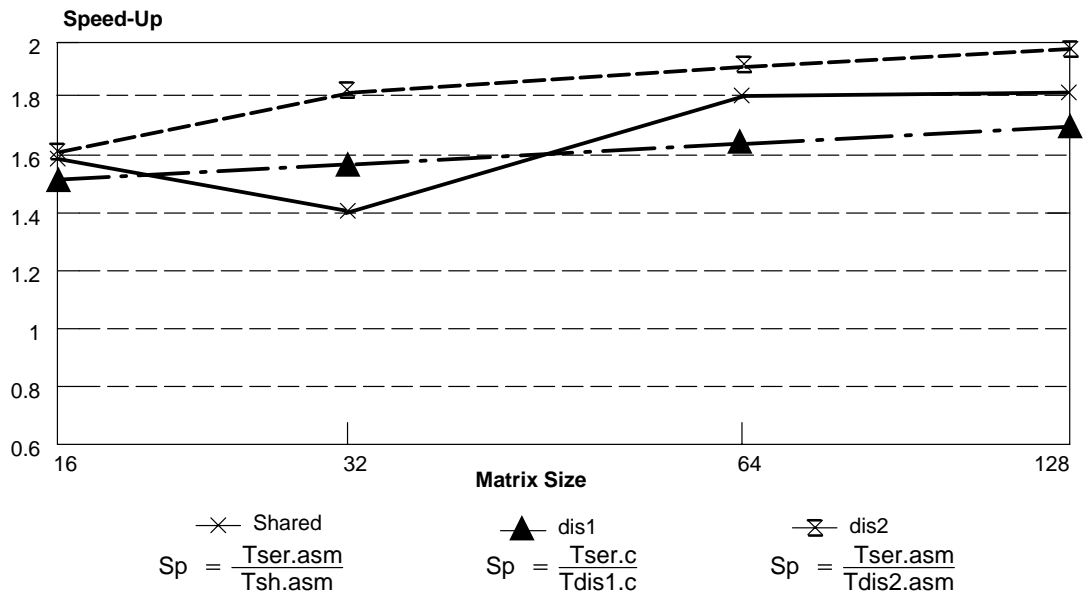
Table 1 and Table 2 show the TMS320C40 2-D FFT timing benchmarks. Data I/O is not considered, because it is host-computer dependent.

Figure 3. Double-Buffering Performance Analysis (Serial Program)



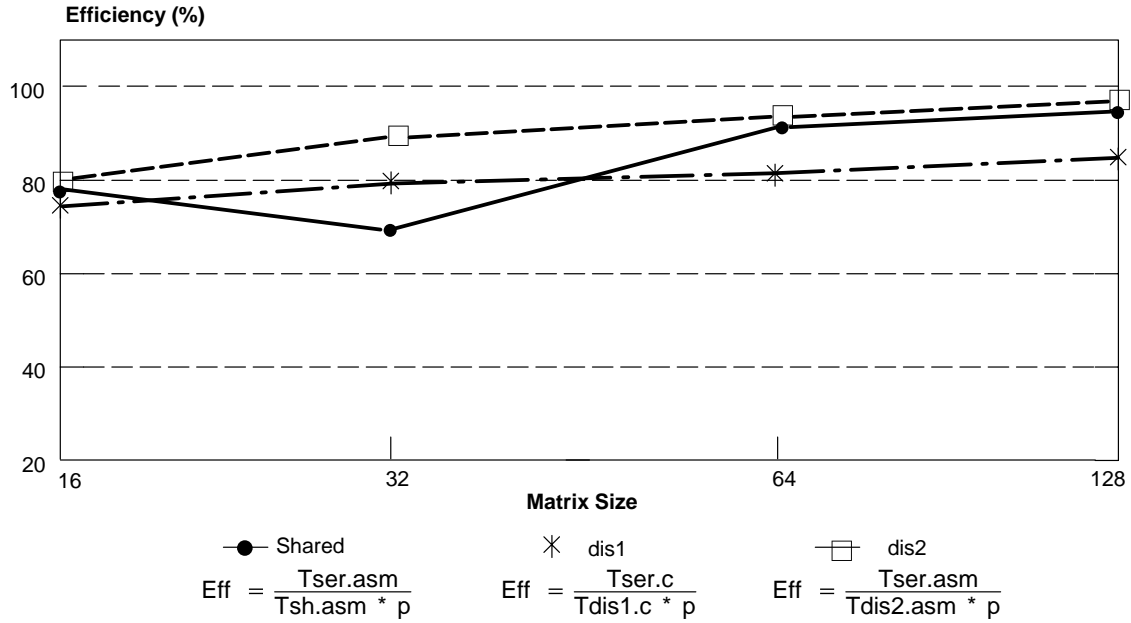
$$\text{Percentage of Improvement} = \frac{T_{\text{ser.asm}} - T_{\text{serb.asm}}}{T_{\text{ser.asm}}} \times 100\%$$

Figure 4. Speed-Up Vs. Matrix Size (p = 2) Over Single-Buffered Serial Program



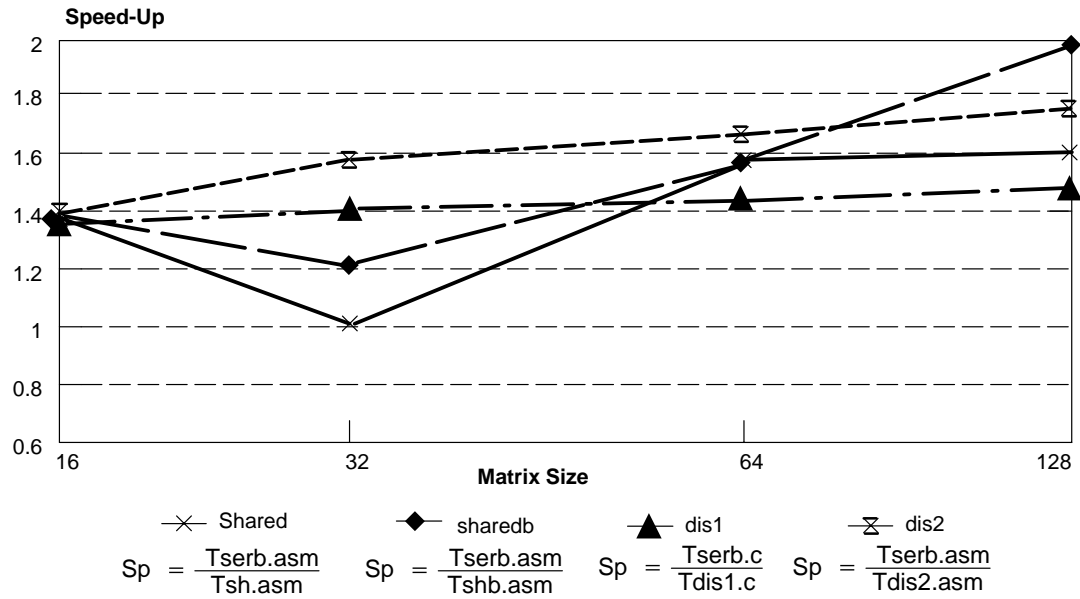
Note: Number of processors = p = 2

Figure 5. Efficiency Vs. Matrix Size (p = 2) Over Single-Buffered Serial Program



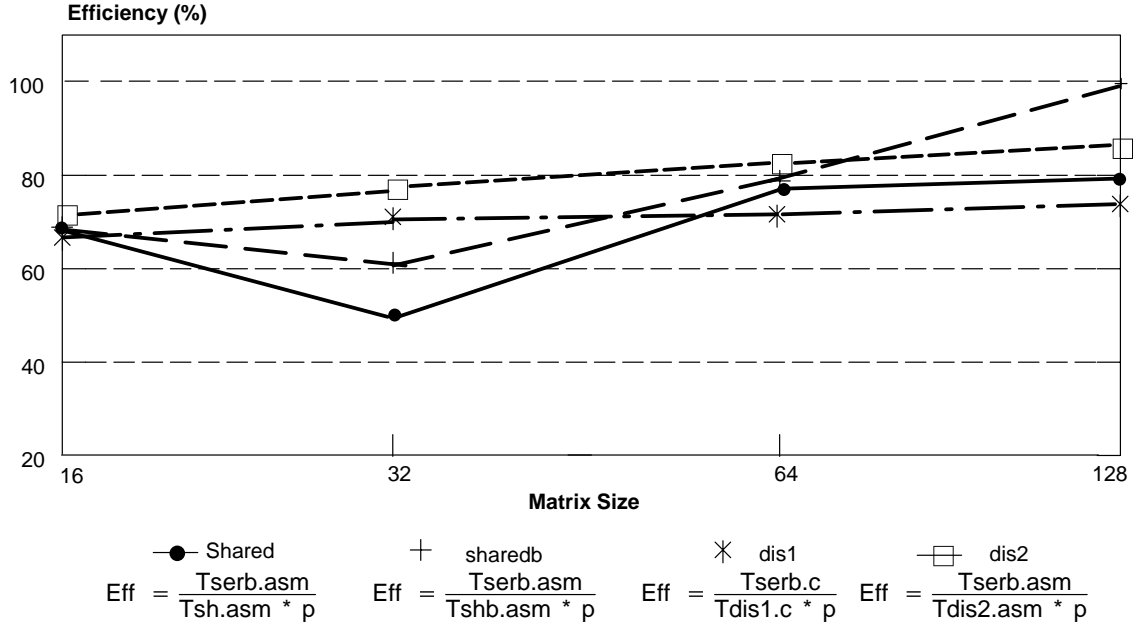
Note: Number of processors = p = 2

Figure 6. Speed-Up Vs. Matrix Size (p = 2) Over Double-Buffered Serial Program



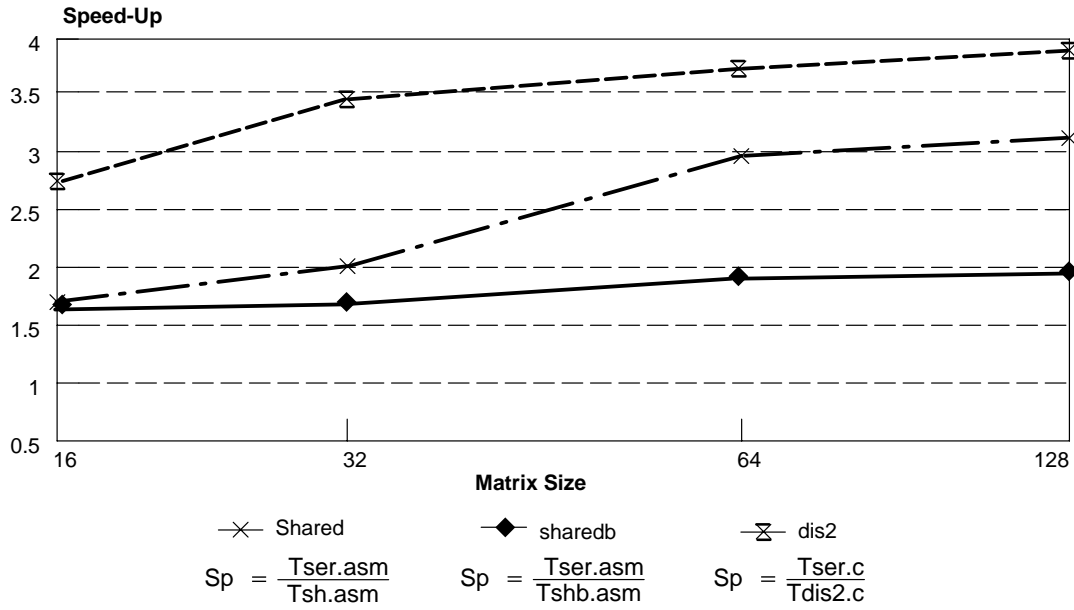
Note: Number of processors = p = 2

Figure 7. Efficiency Vs. Matrix Size (p = 2) Over Double-Buffered Serial Program



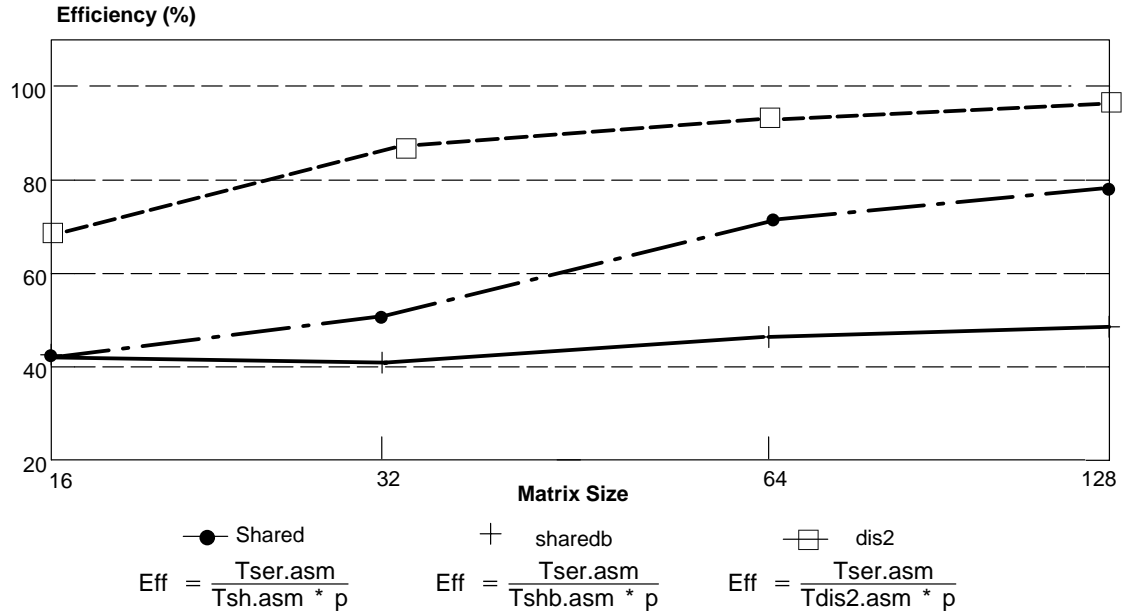
Note: Number of processors = p = 2

Figure 8. Speed-Up Vs. Matrix Size (p = 4) Over Single-Buffered Serial Program



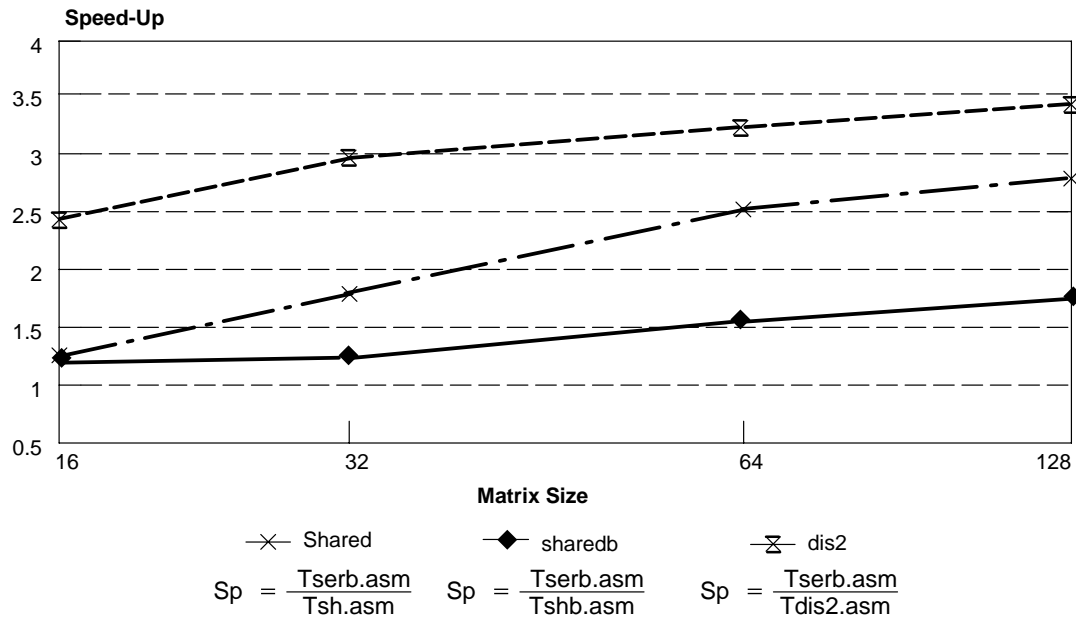
Note: Number of processors = p = 4

Figure 9. Efficiency Vs. Matrix Size (p = 4) Over Single-Buffered Serial Program



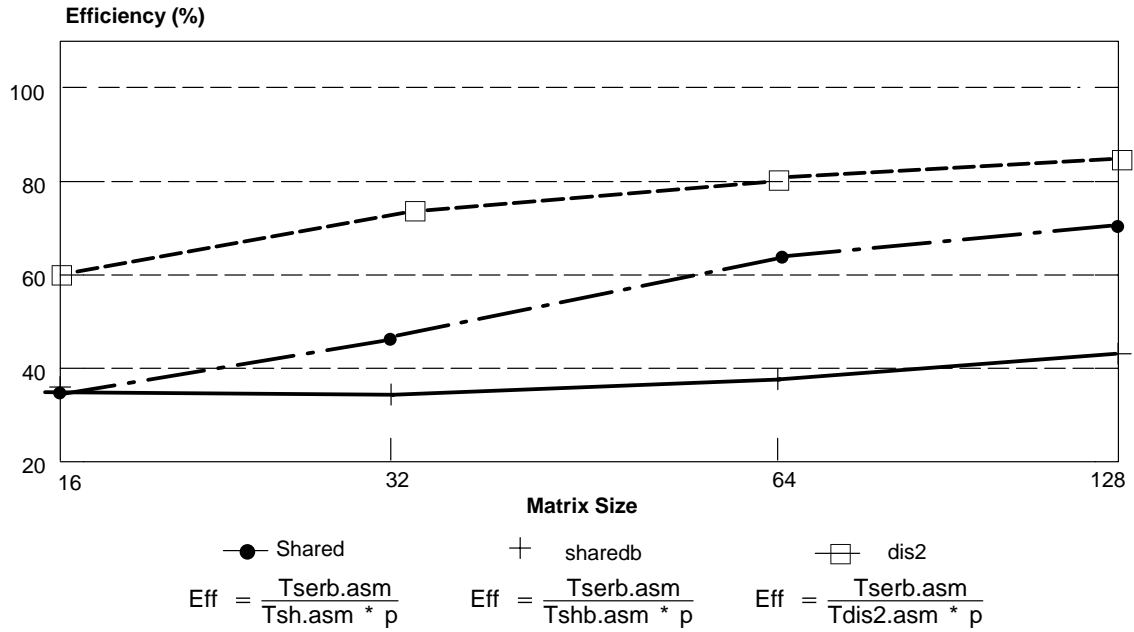
Note: Number of processors = p = 4

Figure 10. Speed-Up Vs. Matrix Size (p = 4) Over Double-Buffered Serial Program



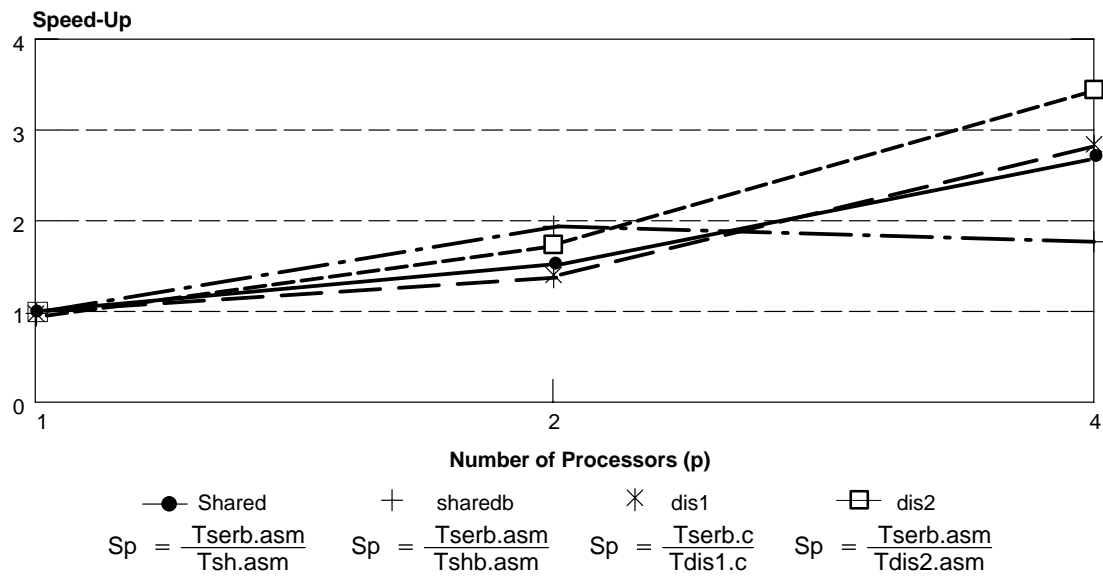
Note: Number of processors = p = 4

Figure 11. Efficiency Vs. Matrix Size (p = 4) Over Double-Buffered Serial Program



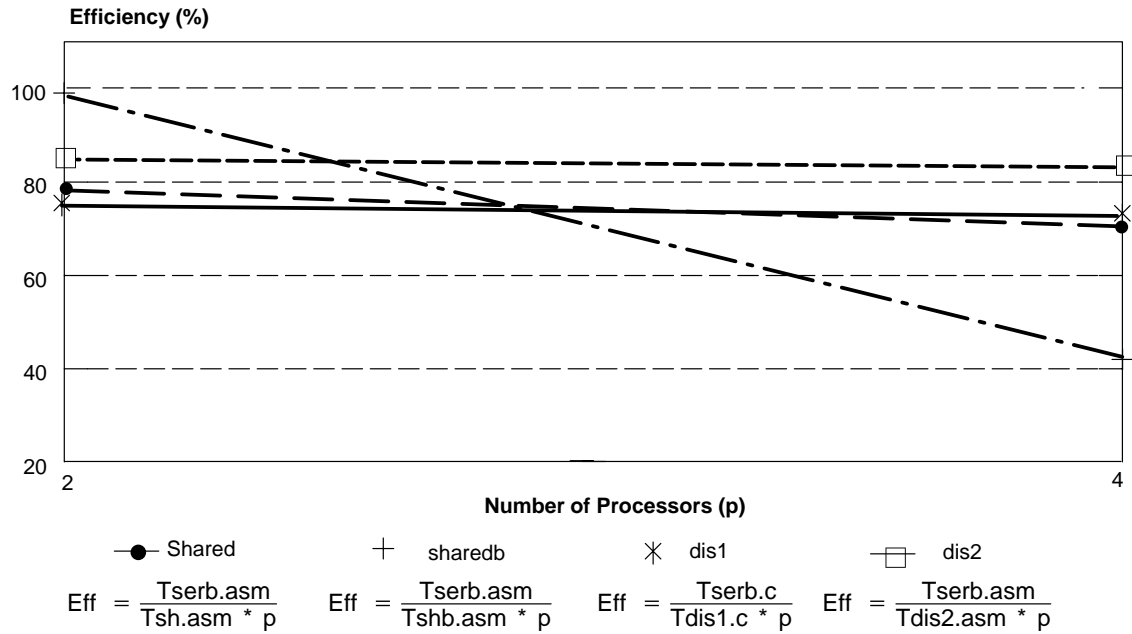
Note: Number of processors = p = 4

Figure 12. Speed-Up Vs. Number of Processors Over Double-Buffered Serial Program



Note: Matrix Size = 128

Figure 13. Efficiency Vs. Number of Processors Over Double-Buffered Serial Program



Note: Matrix Size = 128

Table 1. TMS320C40 2-D FFT Timing Benchmarks (in Milliseconds)

Program	Number of Processors	Matrix Size in Complex Numbers			
		16 × 16	32 × 32	64 × 64	128 × 128
SER.ASM	(p = 1)	0.706	3.115	13.812	61.138
SERB.ASM	(p = 1)	0.614	2.682	11.959	53.498
SER.C	(p = 1)	0.773	3.248	14.078	61.671
SERB.C	(p = 1)	0.671	2.794	12.183	53.944
SH.ASM	(p = 2)	0.442	2.217	7.737	33.564
SHB.ASM	(p = 2)	0.441	1.850	7.550	26.800
DIS2.ASM	(p = 2)	0.438	1.707	7.200	31.150
DIS1.C	(p = 2)	0.504	2.020	8.467	36.231
DIS2.C	(p = 2)	0.448	1.744	7.266	31.270
SH.ASM	(p = 4)	0.424	1.496	4.757	19.196
SHB.ASM	(p = 4)	0.421	1.880	7.692	31.104
DIS2.ASM	(p = 4)	0.255	0.902	3.693	15.750

Note: The data in this table was obtained with the complex FFT routine in Appendix D.

- 'C40 instruction cycle, Tcycle = 40 ns
- C compiler optimization level : o2

Table 2. TMS320C40 2-D FFT Timing Benchmarks (in Milliseconds)

Program	Number of Processors	Matrix Size in Complex Numbers		
		32×32	64×64	128×128
SER.ASM	($p = 1$)	2.080	9.418	40.390
SERB.ASM	($p = 1$)	1.791	8.146	35.340
SH.ASM	($p = 2$)	1.478	5.274	22.172
SHB.ASM	($p = 2$)	1.234	5.147	17.825
DIS2.ASM	($p = 2$)	1.140	4.910	20.586
DIS2.ASM	($p = 4$)	0.602	2.518	10.410

Note: This table gives expected values using the faster version complex Radix-2 DIT FFT routine in Example 12-44 of the *TMS320C4x User's Guide* (1993). Tcycle = 40 ns.

Conclusion

This report has presented shared- and distributed-memory 2-D FFT parallel implementations. High speed-up/efficiency has been attained. Parallelization of the 2-D FFT is important when dealing with large matrices. For small matrices, a serial implementation is more convenient.

A virtually unlimited number of parallel algorithms can be implemented in 'C40-based systems. Parallel implementations of 1-D FFT can be found in [1]. These require cube/mesh mapping techniques that can also be implemented in a parallel system, such as the PPDS.

References

- [1] K. Hwang and F. A. Briggs. *Computer Architecture and Parallel Processing*. New York: McGraw-Hill, 1984.
- [2] R. M. Piedra. *A Parallel Approach for Solving Matrix-Multiplication on the TMS320C4x DSP*. Dallas, Texas: Texas Instruments, Incorporated, 1991.
- [3] C. S. Burrus and T. W. Parks. *DFT/FFT and Convolution Algorithms*. New York: John Wiley and Sons, 1985.
- [4] P. Papamichalis. "An Implementation of FFT, DCT, and Other Transforms on the TMS320C30." *Digital Signal Processing Applications with the TMS320 Family, Volume 3*. Dallas, Texas: Texas Instruments, Incorporated, 1990, page 53.
- [5] D.C. Chen and R. H. Price. "A Real-Time TMS320C40 Based Parallel System for High Rate Digital Signal Processing." *Proceedings of ICASSP 91, USA, Volume 2*, page 1573, May 1991.
- [6] A. V. Oppenheim and R. W. Schaffer. *Digital Signal Processing*. Englewood Cliffs, New Jersey: Prentice-Hall, 1975.
- [7] S. G. Akl. *The Design and Analysis of Parallel Algorithms*. Englewood Cliffs, New Jersey: Prentice-Hall, 1989, page 171.
- [8] *TMS320C4x User's Guide*. Dallas, Texas: Texas Instruments, Incorporated, 1991.
- [9] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation, Numerical Methods*. Englewood Cliffs, New Jersey: Prentice-Hall, 1989.
- [10] S. Y. Kung. *VLSI Array Processors*. Englewood Cliffs, New Jersey: Prentice-Hall, 1988.
- [11] H. Kunieda and K. Itoh. "Parallel 2D-FFT Algorithm on Practical Multiprocessor Systems." *Proceedings of the 3rd Transputer/Occam International Conference*, May 1990.

Appendices

Appendix A: Serial Implementations of 2-D FFT

- A.1 SER.C: Single-Buffered Implementation (C Program)
- A.2 SER.ASM: Single-Buffered Implementation ('C40 Assembly Program)
- A.3 SERB.C: Double-Buffered Implementation (C Program)
- A.4 SERB.ASM: Double-Buffered Implementation ('C40 Assembly Program)

Appendix B: Parallel 2-D FFT (Shared-Memory Version)

- B.1 SH.C: Single-Buffered Implementation (C Program)
- B.2 SH.ASM: Single-Buffered Implementation ('C40 Assembly Program)
- B.3 SHB.C: Double-Buffered Implementation (C Program)
- B.4 SHB.ASM: Double-Buffered Implementation ('C40 Assembly Program)

Appendix C: Parallel 2-D FFT (Distributed-Memory Version)

- C.1 DIS1.C: Distributed-Memory Implementation (C Program) — DMA Used Only for Interprocessor Communication
- C.2 DIS2.C: Distributed-Memory Implementation (C Program) — DMA Used for Interprocessor Communication and Matrix Transposition
- C.3 DIS2.ASM: Distributed-Memory ('C40 Assembly Program) — DMA Used for Interprocessor Communication and Matrix Transposition

Appendix H: Mylib.lib Routines

- D.1 CFFT.ASM: Assembly Language FFT Routine
- D.2 CFFTC.ASM: Assembly Language FFT Routine (C-Callable)
- D.3 CMOVE.ASM: Complex-Vector Move Routine
- D.4 CMOVEB.ASM: Complex-Vector Bit-Reversed Move Routine
- D.5 SET_DMA.ASM: Routine to Set DMA Register Values
- D.6 EXCHANGE.ASM: Routine for Interprocessor Communication
- D.7 SYNCOUNT.ASM: Interprocessor Synchronization Routine

Appendix A: Serial Implementations of 2-D FFT

A.1. SER.C: Single-Buffered Implementation (C Program)

SER.C

```
/* *****  
SER.C : Serial 2-dimensional complex FFT (Single-buffered version)  
To run:  
cl30 -v40 -gs -mr -o2 ser.c  
asm30 -v40 -s sintab.asm  
asm30 -v40 -s input.asm  
lnk30 serc.cmd  
*****  
#define SIZE 4 /* FFT size (n) */  
#define LOGSIZE 2 /* log(FFT size) */  
#define BLOCK0 0x002ff800 /* on-chip RAM buffer */  
extern void cfft(), /* C-callable complex FFT */  
          cmove(), /* CPU complex move */  
          cmoveb(); /* CPU bit-reversed move */  
extern float MATRIX[SIZE][SIZE*2]; /* Input matrix */  
float *block0 = (float *)BLOCK0,  
      *MM[SIZE];  
int size2 = 2*SIZE,  
    i;  
int tcomp; /* for benchmarking */  
/* *****  
main()  
{  
  asm(" or 1800h,st"); /* cache enable */  
  for (i=0;i<SIZE;i++) MM[i]=MATRIX[i]; /* accessing assembly variables */  
  t0: time_start(0); /* start timer 0 for benchmark */  
  /* ***** Step 1: FFT by rows *****  
  for (i=0;i<SIZE;i++) {  
    cmove (&MM[i][0],block0,2,2,SIZE); /* move row(i) to on-chip mem. */  
    cfft(block0,SIZE,LOGSIZE); /* FFT on row (i) */  
    cmoveb(block0,&MM[i][0],SIZE,2,SIZE); /* move FFT{row(i)} to off-chip mem. */  
  }  
  /* ***** Step 2: FFT by columns *****  
  t1:  
  for (i=0;i<size2;i+=2) {  
    cmove (&MM[0][i],block0,size2,2,SIZE); /* move column (i) to on-chip mem. */  
    cfft(block0,SIZE,LOGSIZE); /* FFT on column (i) */  
    cmoveb (block0,&MM[0][i],SIZE,size2,SIZE); /* move FFT {col. (i) to off-chip mem */  
  }  
  tcomp= time_read(0);  
  t2: ;  
} /*main*/
```

SERC.CMD

```
-c                                     /* LINK USING C CONVENTIONS */
ser.obj
sintab.obj
input.obj
-stack 0x0040
-lrts40.lib                           /* GET RUN-TIME SUPPORT */
-lprts40r.lib
-lmylib.lib
-m serc.map
-o serc.out

MEMORY
{
    ROM:      org = 0x00             len = 0x1000
    RAM0:     org = 0x002fff800      len = 0x0400
    RAM1:     org = 0x002ffc00       len = 0x0400
    LM:       org = 0x40000000       len = 0x10000    /* LOCAL MEMORY */
    GM:       org = 0x80000000       len = 0x20000    /* GLOBAL MEMORY */
}

SECTIONS
{
    .text:    ( ) > RAM1             /* CODE */
    .cinit:   { } > RAM1             /* INITIALIZATION TABLES */
    .stack:   { } > RAM1             /* SYSTEM STACK */
    .bss:     { } > RAM1             /* GLOBAL & STATIC VARS */
    .data:    { } > RAM1             /* Sine tables */
    INPUT:    { } > LM               /* Input matrix */
}
```


INPUT.ASM

```
*****
*
* INPUT.ASM : input matrix 4 x 4 for serial/shared program
*
*****

.global  _MATRIX
.sect    "INPUT"

_MATRIX

.float 130.0,90.0    ;[0][0]:  output 2264.0 , 2288.0
.float 66.0,230.0    ;[0][1]:  output -56.0 , -532.0
.float 205.0,136.0   ;[0][2]:  output -50.0 , 378.0
.float 15.0,187.0     ;[0][3]:  output -182.0 , -22.0

.float 150.0,164.0    ;[1][0]:  output -401.0 , 227.0
.float 222.0,44.0     ;[1][1]:  output -353.0 , 1.0
.float 95.0,243.0     ;[1][2]:  output 423.0 , -373.0
.float 80.0,60.0      ;[1][3]:  output 167.0 , 229.0

.float 97.0,36.0      ;[2][0]:  output -68.0 , -26.0
.float 215.0,191.0    ;[2][1]:  output 106.0 , -176.0
.float 209.0,239.0    ;[2][2]:  output 418.0 , -636.0
.float 161.0,22.0     ;[2][3]:  output -616.0 , -266.0

.float 117.0,238.0    ;[3][0]:  output -131.0 , 83.0
.float 203.0,44.0     ;[3][1]:  output 175.0 , 319.0
.float 104.0,187.0    ;[3][2]:  output 225.0 , -133.0
.float 195.0,177.0    ;[3][3]:  output 159.0 , 79.0
.end
```

SINTAB.ASM

```
*****
*
*   SINTAB.ASM : Table with twiddle factors for a 4-point CFFT
*               and data input. File to be linked with the
*               source code for a 4-point, radix-2 FFT.
*
*****

        .global  SINE
        .global  N
        .global  M

N        .set     4
M        .set     2

        .data

SINE     .float   0.000000
COSINE   .float   1.000000
         .float   -0.000000
         .float   -1.000000
         .float   0.000000

        .end
```

A.2. SER.ASM: Single-Buffered Implementation ('C40 Assembly Program)

SER.ASM

```
*****
*
*   SER.ASM : TMS320C40 complex 2D-FFT serial program
*             (Single-buffered version)
*
*   Routines used:
*             cfft.asm (complex FFT)
*
*   Requirements: Matrix size = N  > 0
*
*   To run:
*
*       asm30 -v40 -gs ser.asm
*       asm30 -v40 -gs sintab.asm
*       asm30 -v40 -gs input.asm
*       lnk30 ser.cmd
*
*****

        .global      N           ; FFT size
        .global      _MATRIX     ; Matrix address
        .global      CFFT        ; Complex 1D-FFT subroutine
        .global      C2DFFT      ; Entry point for execution

_STACK  .usect       "STACK",10h ; Stack definition

        .text
FFTSIZE .word        N
MATR     .word        _MATRIX
STACK    .word        _STACK      ; Stack address
BLOCK0   .word        002FF800H   ; On-chip buffer (RAM block 0)
TIMER    .word        0100020H    ; Timer 0 address

C2DFFT
        LDP          FFTSIZE      ; Data page pointer initialization
        LDI          @STACK,SP    ; Stack pointer initialization

t0:      LDI          @TIMER,AR2    ; Optional: benchmarking (time_start)
        STIK         -1,*+AR2(8)
        LDI          961,R0
        STI          R0,*AR2

        OR           1800h,ST      ; Enabling cache
        LDI          @FFTSIZE,AR3  ; AR3 = N = FFT size
        SUBI         1,AR3,AR5    ; AR5 = row counter

        LDI          @MATR,AR7     ; AR7 = matrix pointer
        LDI          @BLOCK0,AR6   ; AR6 = on-chip buffer pointer

*****
*
*             FFT ON ROWS
*
*****

LOOPR
```

```
*****
* Move row X          *
* to on-chip memory   *
*****
```

```

SUBI3    2,AR3,RC          ; RC = N-2
LDI      AR7,AR0           ; Source address
RPTBD    LOOP1
LDI      AR6,AR1           ; Destination address
LDI      2,IR0
LDF      *+AR0(1),R0       ; R0 = X(I) Im

      LDF      *AR0++(IR0),R1      ; X(I) Re & points to X(I+1)
||      STF      R0, *+AR1(1)      ; Store X(I) Im
LOOP1    LDF      *+AR0(1),R0       ; R0 = X(I+1) Im
||      STF      R1,*AR1++(IR0)    ; Store X(I) Re

```

```
*****
* FFT on row X *
*****
```

```

LAJ      CFFT              ; Call 1D-FFT (complex)
LDF      *AR0,R1           ; Load X(N-1) Re
NOP
STF      R0,*+AR1(1)       ; Store X(N-1) Im
||      STF      R1,*AR1      ; Store X(N-1) Re

```

```
*****
* Move row X (bit-reversed) from      *
* on-chip memory to external memory *
*****
```

```

SUBI3    2,AR3,RC          ; Source address
LDI      AR6,AR0           ; Destination Address
RPTBD    LOOP2
LDI      AR3,IR0           ; Source offset for bit-reverse = N
LDI      2,IR1             ; Destination offset
LDF      *+AR0(1),R0

      LDF      *AR0++(IR0)B,R1
||      STF      R0,*+AR1(1)
LOOP2    LDF      *+AR0(1),R0
||      STF      R1,*AR1++(IR1)
      LDF      *AR0++(IR0)B,R1
||      STF      R0,*+AR1(1)

DBUD     AR5,LOOPR
STF      R1,*AR1++(IR1)
LSH3     1,AR3,R0
ADDI     R0,AR7

```

```
*****
*                      FFT ON COLUMNS
*****
```

```

t1:      SUBI      1,AR3,AR5 ; AR5 = column counter
          LDI      @MATR,AR7 ; AR7 = Matrix pointer

```

LOOPC

```
*****
* Move column X (X=AR7) *
* to on-chip memory      *
*****
```

```

SUBI3    2,AR3,RC          ; RC=N-2
LDI      AR7,AR0           ; Source address
LDI      AR6,AR1           ; Destination address
RPTBD    LOOP3
LSH3     1,AR3,IR1         ; Source offset = 2*N
LDI      2,IR0             ; Destination offset
LDF      *+AR0(1),R0       ; R0= X(I) Im

||
LDF      *AR0++(IR1),R1    ; X(I) Re & points to X(I+1)
STF      R0,*+AR1(1)       ; Store X(I) Im
LOOP3    LDF      *+AR0(1),R0 ; R0=X(I+1) Im
||
STF      R1,*AR1++(IR0)   ; Store X(I) Re
```

```
*****
* FFT on column X *
*****
```

```

LAJ      CFFT
LDF      *AR0,R1          ; Load X(N-1) Re
NOP
STF      R0,*+AR1(1)      ; Store X(N-1) Im
||
STF      R1,*AR1          ; Store X(N-1) Re
```

```
*****
* Move column X (bit-reversed) from *
* on-chip memory to external memory *
*****
```

```

SUBI3    2,AR3,RC          ; RC=N-2
LDI      AR6,AR0           ; Source address
LDI      AR7,AR1           ; Destination address
RPTBD    LOOP4
LDI      AR3,IR0           ; Source offset = IR0 = N (bit-reverse)
LSH      1,AR3,IR1         ; Destination offset (columns) = IR1 = 2N
LDF      *+AR0(1),R0

||
LDF      *AR0++(IR0)B,R1
STF      R0,*+AR1(1)
LOOP4    LDF      *+AR0(1),R0
||
STF      R1,*AR1++(IR1)

DBUD     AR5,LOOPC
||
LDF      *AR0++(IR0)B,R1
STF      R0,*+AR1(1)
STF      R1,*AR1++(IR1)
ADDI     2,AR7

LDI      @TIMER,AR2        ; Optional: benchmarking
LDI      *+AR2(4),R0       ; tcomp = R0

t2       B        t2
        .end
```

SER.CMD

```
input.obj
ser.obj
sintab.obj
-lmylib.lib
-m ser.map
-o ser.out
```

MEMORY

```
{
    ROM:      o = 0x00000000 l = 0x1000
    RAM0:     o = 0x002ff800 l = 0x400
    RAM1:     o = 0x002ffc00 l = 0x400
    LM:       o = 0x40000000 l = 0x10000
    GM:       o = 0x80000000 l = 0x20000
}
```

SECTIONS

```
{
    .text     :{} > RAM1
    .data     :{} > RAM1 /* SINE TABLE */
    STACK     :{} > RAM1 /* STACK */
    INPUT     :{} > LM /* INPUT MATRIX */
}
```

A.3. SERB.C: Double-Buffered Implementation (C Program)

SERB.C

```

/*****

SERB.C : Serial 2-dimensional complex FFT (Double-buffered version)
To run:
cl30 -v40 -g -s -mr -o2 serb.c
asm30 -v40 -s sintab.asm
asm30 -v40 -s input.asm
lnk30 serbc.cmd
Requirement: SIZE ≥ 4

*****/
#define SIZE          4                /* FFT size */
#define LOGSIZE       2                /* log(FFT size) */

#define BLOCK0        0x002ff800      /* on-chip buffer 1 */
#define BLOCK1        0x002ffc00      /* on-chip buffer 2 */
#define DMA0          0x001000a0      /* DMA0 address */
#define SWAP(x,y)      temp = x; x = y; y = temp;
#define WAIT_DMA(x)    while ((0x00c00000 & *x) != 0x00800000);

extern void            cfftc(),        /* C-callable complex FFT */
                    cmove(),          /* CPU complex move */
                    cmoveb(),         /* CPU bit-reversed move */
                    set_dma();        /* Set-up DMA registers */

extern float          MATRIX[SIZE][SIZE*2]; /* Input complex matrix */

/* DMA control register values */

int    ctrl0= 0x00c41004;              /* no autoinit.,dmaint,bit_rev */
int    ctrl1= 0x00c01008;              /* autoinit.,no dmaint,bitrev */
int    ctrl2= 0x00c00008;              /* autoinit, no dmaint */
int    ctrl3= 0x00c40004;              /* no autoinit.,dmain */
int    dma01[7], dma02[7], dma03[7], dma04[7];

float   *CPUbuffer = (float *)BLOCK0,  /* block for CPU FFT operations */
        *DMABuffer = (float *)BLOCK1,  /* block for DMA operations */
        *MM[SIZE], *temp;

volatile int    *dma0 = (int *)DMA0;
int             size2   = (SIZE*2), i, j;
int             tcomp;
/*****/
main()
{
asm(" or 1800h,st");
for (i=0;i<SIZE;i++) MM[i]=MATRIX[i];

t0: time_start(0);
```

```

/***** FFT on rows *****/

/*****
1. DMA:      - moves row 1 to on-chip RAM buffer 1      *
2. CPU:      - moves row 0 to on-chip RAM buffer 0      *
              - FFT on row 0                            *
*****/
set_dma(dma0,ctrl13,&MM[1][0],1,size2,DMAbuffer,1,1);
cmove(&MM[0][0],CPUbuffer,2,2,SIZE);
cfft(CPUbuffer,SIZE,LOGSIZE);

/*****
1. DMA:      - moves Re FFT(row 0) to off-chip RAM      *
              - moves Im FFT(row 0) to off-chip RAM      *
              - moves row 2 to on-chip RAM                *
2. CPU:      FFT on row 1                                *
*****/
WAIT_DMA (dma0);
set_dma(dma01,ctrl11,CPUbuffer,SIZE,SIZE,&MM[0][0],2,dma02);
set_dma(dma02,ctrl11,(CPUbuffer+1),SIZE,SIZE,&MM[0][1],2,dma03);
set_dma(dma03,ctrl13,&MM[2][0],1,size2,CPUbuffer,1,1);
*(dma0+3) = 0;      *(dma0+6) = (int) dma01; *dma0 =ctrl2; /* start DMA */
cfft(DMAbuffer,SIZE,LOGSIZE);

/*****
1. DMA:      - moves Re FFT(row i) to off-chip RAM      *
              - moves Im FFT(row i) to off-chip RAM      *
              - moves row (i+2) to on-chip RAM            *
2. CPU:      FFT on row (i+1)                            *
*****/
for (i=1;i<SIZE-2;i+=1) {
WAIT_DMA (dma0);

*(dma03+1) = (int)&MM[i+2][0];      *(dma03+4) = (int)DMAbuffer;
*(dma01+1) = (int)DMAbuffer;      *(dma01+4) = (int)&MM[i][0];
*(dma02+1) = (int)DMAbuffer+1;      *(dma02+4) = (int)&MM[i][1];
*(dma0+3) = 0;      *(dma0+6) = (int) dma01; *dma0 = ctrl2; /* start DMA */

cfft(CPUbuffer,SIZE,LOGSIZE);      /* work in current row */
SWAP (CPUbuffer,DMAbuffer);      /* switch buffers */
}

/*****
1. DMA:      - moves Re FFT(row(size-2)) to off-chip RAM *
              - moves Im FFT(row(size-2)) to off-chip RAM *
              - moves Re column 0 to on-chip RAM (except last row element) *
              - moves Im column 0 to on-chip RAM          *
2. CPU:      - FFT on last row: row (size-1)              *
              - transfer element [size-1][0] to corresponding position in *
                on-chip buffer 0                            *
*****/

WAIT_DMA (dma0);
*(dma01+1) = (int)DMAbuffer;      *(dma01+4) = (int)&MM[i][0];
*(dma02+1) = (int)DMAbuffer+1;      *(dma02+4) = (int)&MM[i][1];

set_dma(dma03,ctrl12,&MM[0][0],size2,(SIZE-1),DMAbuffer,2,dma04);
set_dma(dma04,ctrl13,&MM[0][1],size2,(SIZE-1),(DMAbuffer+1),2,2);

```



```

*(dma0+3) = 0;      *(dma0+6) = (int) dma01;  *dma0 = ctrl2; /* start DMA */

cfft(c(CPUbuffer,SIZE,LOGSIZE);

WAIT_DMA (dma0);
*(DMAbuffer+size2-2) = *(CPUbuffer);
*(DMAbuffer+size2-1) = *(CPUbuffer+1);

/***** FFT on columns *****/

/*****
1. DMA:  - moves Re FFT(row(size-1)) to off-chip RAM      *
          - moves Im FFT(row(size-1)) to off-chip RAM      *
          - moves Re column 1 to on-chip RAM (except last row element) *
          - moves Im column 1 to on-chip RAM                *
2. CPU:  - FFT on column 0                                  *
*****/

CPUbuffer= (float *) BLOCK0;      /*initialize buffer pointer */
DMAbuffer= (float *) BLOCK1;

*(dma01+1) = (int)DMAbuffer;      *(dma01+4) = (int)&MM[SIZE-1][0];
*(dma02+1) = (int)DMAbuffer+1;    *(dma02+4) = (int)&MM[SIZE-1][1];
*(dma03+1) = (int)&MM[0][2];      *(dma03+4) = (int)DMAbuffer;
*(dma04+1) = (int)&MM[0][3];      *(dma04+4) = (int)DMAbuffer+1;
*(dma03+3) =                      *(dma04+3) = SIZE;

*(dma0+3) = 0; *(dma0+6) = (int) dma01; *dma0 = ctrl2; /* start DMA */

cfft(c(CPUbuffer,SIZE,LOGSIZE);      /* work in column 0 */

/*****
1. DMA:  - moves Re FFT(column 0) to off-chip RAM      *
          - moves Im FFT(column 0) to off-chip RAM      *
          - moves Re column 2 to on-chip RAM            *
          - moves Im column 2 to on-chip RAM            *
2. CPU:  - FFT on column 1                              *
*****/
WAIT_DMA (dma0);
t1:

*(dma01+1) = (int)CPUbuffer;      *(dma01+4) = (int)&MM[0][0];
*(dma02+1) = (int)CPUbuffer+1;    *(dma02+4) = (int)&MM[0][1];
*(dma01+5) = *(dma02+5) = size2;
*(dma03+1) = (int)&MM[0][4];      *(dma03+4) = (int)CPUbuffer;
*(dma04+1) = (int)&MM[0][5];      *(dma04+4) = (int)CPUbuffer+1;

*(dma0+3) = 0;      *(dma0+6) = (int) dma01;*dma0 = ctrl2; /* start DMA */

cfft(c(DMAbuffer,SIZE,LOGSIZE);

```

```

/*****
1. DMA:  - moves Re FFT(column i) to off-chip RAM      *
          - moves Im FFT(column i) to off-chip RAM      *
          - moves Re column (i+2) to on-chip RAM        *
          - moves Im column (i+2) to on-chip RAM        *
2. CPU:  - FFT on column (i+1)                        *
*****/

for (i=2;i<size2-4;i+=2)  {
WAIT_DMA (dma0);

*(dma01+1) = (int)DMAbuffer;          *(dma01+4) = (int)&MM[0][i];
*(dma02+1) = (int)DMAbuffer+1;        *(dma02+4) = (int)&MM[0][i+1];
*(dma03+1) = (int)&MM[0][i+4];        *(dma03+4) = (int)DMAbuffer;
*(dma04+1) = (int)&MM[0][i+5];        *(dma04+4) = (int)DMAbuffer+1;

*(dma0+3) = 0; *(dma0+6) = (int) dma01; *dma0 = ctrl2; /* start DMA */

cfft(c(CPUbuffer,SIZE,LOGSIZE);      /* work in current column */
SWAP (CPUbuffer,DMAbuffer);
}

/*****
1. DMA:  - moves Re FFT(column (size-2)) to off-chip RAM      *
          - moves Im FFT(column (size-2)) to off-chip RAM      *
2. CPU:  - FFT on last column (size-1)                        *
          - moves FFT(last column) to off-chip RAM            *
*****/

WAIT_DMA (dma0);

*(dma01+1) = (int)DMAbuffer;          *(dma01+4) = (int)&MM[0][i];
*(dma02+1) = (int)DMAbuffer+1;        *(dma02+4) = (int)&MM[0][i+1];
*(dma02) = ctrl0;
*(dma0+3) = 0; *(dma0+6) = (int) dma01; *dma0 = ctrl2; /* start DMA */

cfft(c(CPUbuffer,SIZE,LOGSIZE);      /* fft on last column */
cmoveb (CPUbuffer,&MM[0][size2-2],SIZE,size2,SIZE);

WAIT_DMA (dma0);                      /* wait for DMA to finish */
tcomp= time_read(0);
t2;;
} /*main*/

```

SERBC.CMD

```
-c                      /* LINK USING C CONVENTIONS          */
serb.obj
sintab.obj
input.obj
-stack 0x0040
-lrts40.lib            /* GET RUN-TIME SUPPORT          */
-lprts40r.lib          /* PARALLEL RUN-TIME SUPPORT LIBRARY */
-lmylib.lib
-m serbc.map
-o serbc.out

MEMORY
{
    ROM:      org = 0x00      len = 0x1000
    BUF0:     org = 0x002ff800 len = 0x0200
    RAM0:     org = 0x002ffa00 len = 0x0200
    BUF1:     org = 0x002ffc00 len = 0x0200
    RAM1:     org = 0x002ffe00 len = 0x0200
    LM:       org = 0x40000000 len = 0x10000
    GM:       org = 0x80000000 len = 0x20000
}

SECTIONS
{
    INPUT:    {} > LM          /* INPUT MATRIX          */
    .text:    {} > LM          /* CODE                  */
    .cinit:    {} > RAM1       /* INITIALIZATION TABLES */
    .stack:    {} > RAM1       /* SYSTEM STACK          */
    .bss :     {} > RAM1       /* GLOBAL & STATIC VARS  */
    .data:     {} > RAM1       /* SINE TABLES          */
}
```

A.4. SERB.ASM: Double-Buffered Implementation ('C40 Assembly Program)

SERB.ASM

```
*****
*
*   SERB.ASM :      TMS320C40 complex 2D-FFT serial program
*                   (Double-buffered version)
*
*   Routines used:  cfft.asm (complex FFT)
*
*   Requirements:   matrix size = N >= 4
*
*   To run:
*       asm30 -v40 -s -g serb.asm
*       asm30 -v40 -s -g sintab.asm
*       asm30 -v40 -s -g input.asm
*       lnk30 serb.cmd
*
*****

        .global      N                ; FFT SIZE
        .global      _MATRIX          ; MATRIX ADDRESS
        .global      CFFT             ; 1D-FFT SUBROUTINE
        .global      C2DFFT           ; ENTRY POINT FOR EXECUTION

_STACK  .usect       "STACK", 10h    ; Stack definition

* DMA AUTOINITIALIZATION VALUES

        .sect        "DMA_AUTOINT" ' ; DMA autoinitialization values
DMA01   .space       6
        .word        DMA02
DMA02   .space       6
        .word        DMA03
DMA03   .space       6
        .word        DMA04
DMA04   .space       6

        .text
FFTSize .word        N
MATR     .word        _MATRIX
BLOCK0   .word        002FF800H      ; RAM BLOCK 0
BLOCK1   .word        002FFC00H      ; RAM BLOCK 1
STACK_A  .word        _STACK         ; STACK ADDRESS
DMA0     .word        001000A0H      ; ADDRESS OF DMA0
CTRL0    .word        00C41004H      ; NO AUTOINITIALIZATION, DMA INT., BITREV
CTRL1    .word        00C01008H      ; AUTOINITIALIZATION, NO DMA INT., BITREV
CTRL2    .word        00C00008H      ; AUTOINITIALIZATION, NO DMA INT.
CTRL3    .word        00C40004H      ; NO AUTOINITIALIZATION, DMA INT.
P04      .word        DMA04          ; POINTER TO REGISTER VALUES (DMA04)
P03      .word        DMA03          ; POINTER TO REGISTER VALUES (DMA03)
P02      .word        DMA02          ; POINTER TO REGISTER VALUES (DMA02)
P01      .word        DMA01          ; POINTER TO REGISTER VALUES (DMA01)
MASK     .word        02000000H      ; 1 IN DMAINT0
TIMER    .word        0100020H      ; TIMER 0 address

C2DFFT   LDP          FFTSize        ; LOAD DATA PAGE POINTER
        LDI          @STACK_A,SP    ; INITIALIZE THE STACK POINTER
```

```

t0:      LDI      @TIMER,AR2      ; OPTIONAL: BENCHMARKING (TIME_START)
          STIK     -1,*+AR2(8)
          LDI      961,R0
          STI      R0,*AR2
          OR       1800h,ST       ; ENABLE CACHE
          LDI      @FFTSIZE,AR3   ; AR3=N
          LDI      @MATR,AR7      ; POINTER TO MATRIX
          LDI      @BLOCK1,R7     ; POINTER TO DMA BUFFER
          LDI      @BLOCK0,AR6    ; POINTER TO FFT BUFFER

```

```

*****
*                               *
*          CPU MOVES ROW 0      *
*                               *
*****

```

```

          SUBI3     2,AR3,RC      ; RC=N-2
          LDI      AR7,AR0        ; SOURCE
          RPTBD     LOOP1
          LDI      AR6,AR1        ; DESTINATION
          LDI      2,IR1
          LDF       *+AR0(1),R0   ; R0= X0(I) IM

* LOOP
          LDF       *AR0++(IR1),R1 ; X0(I) RE & POINTS TO X0(I+1)
          ||      STF       R0,*+AR1(1) ; STORE X0(I) IM
LOOP1    LDF       *+AR0(1),R0    ; R0=X0(I+1) IM
          ||      STF       R1,*AR1++(IR1) ; STORE X0(I) RE

```

```

* STORE LAST VALUE
          LDI      @P02,AR2       ; POINTS DMA REGISTER
          LDF       *AR0++(IR1),R1 ; LOAD X0(N-1) RE
          ||      STF       R0,*+AR1(1) ; STORE X0(N-1) IM
          ||      STF       R1,*AR1    ; STORE X0(N-1) RE

```

```

*****
*
* SET PARAMETERS FOR DMA02, DMA03 THAT WILL ALWAYS BE FIXED
*
*****

```

```

          LDI      @P03,AR1       ; POINTS DMA REGISTER
          LDI      @P01,AR4
          LDI      @CTRL1,R0
          STI      R0,*AR2
          STI      AR3,*+AR2(2)   ; SOURCE INDEX
          STI      R0,*AR4
          STI      AR3,*+AR4(2)   ; SOURCE INDEX
          STI      AR3,*+AR2(3)   ; COUNTER
          STI      AR3,*+AR4(3)   ; COUNTER
          LSH3     1,AR3,R0       ; R0=2*N
          STIK     2H,*+AR2(5)    ; DESTINATION INDEX
          STIK     2H,*+AR4(5)    ; DESTINATION INDEX
          LDI      @CTRL3,R2
          STI      R2,*AR1
          LDI      @DMA0,AR2      ; POINTS DMA REGISTER
          STIK     1H,*+AR1(2)    ; SOURCE INDEX
          SUBI     3,AR3,AR5      ; AR5=N-3 : (N-2) DMA TRANSFERS
          STI      R0,*+AR1(3)    ; COUNTER
          STI      AR0,*+AR2(1)   ; SOURCE
          STIK     1H,*+AR1(5)    ; DESTINATION INDEX

```

```

*****
*          CPU : FFT ON ROW 0          *
*          DMA : BEGINS TO TRANSFER ROW1      *
*****
          STIK      1H,*+AR2(2)          ; SOURCE INDEX
          STI       R0,*+AR2(3)          ; COUNTER=2N
          LAJ       CFFT                  ; FFT ON ROW 0
          STI       R7,*+AR2(4)          ; DESTINATION
          STIK      1H,*+AR2(5)          ; DESTINATION INDEX
          STI       R2,*AR2              ; CONTROL3

*****
*          FFT ON ROWS          *
*****
          LDI       @P02,AR1
          LDI       @P01,AR0
          LSH3      1,AR3,R0
LOOP2    TSTB      @MASK,IIF
          BZAT      LOOP2
* DMA02: BIT-REVERSED TRANSFER OF LAST RESULT (Im)
          ADDI      1H,AR6,R1
          STI       R1,*+AR1(1)          ; SOURCE
          ADDI      1H,AR7,R1
          STI       R1,*+AR1(4)          ; DST

* DMA01: BIT-REVERSED TRANSFER OF LAST RESULT (Re)
          STI       AR6,*+AR0(1)          ; SOURCE
          STI       AR7,*+AR0(4)          ; DST

* DMA03: TRANSFER NEXT ROW
          LDI       @P03,AR1              ; DMA0
          ADDI      R0,AR7
          ADDI      AR7,R0
          STI       R0,*+AR1(1)          ; SOURCE: NEXT ROW
          AND       0H,IIF                ; CLEAR FLAG
          STI       AR6,*+AR1(4)          ; DESTINATION:
          LDI       @DMA0,AR1
          LDI       R7,R2                  ; EXCHANGE BUFFER POINTERS
          LDI       AR6,R7                ; R7: POINTER FOR NEXT DMA
          STIK      0,*+AR1(3)            ; TEMPORAL FIX
          STI       AR0,*+AR1(6)

* FFT ON CURRENT ROW
          LAJ       CFFT
          LDI       R2,AR6                ; AR6: POINTER FOR NEXT FFT
          LDI       @CTRL2,R0
          STI       R0,*AR1              ; START (DMA)

          DBUD      AR5,LOOP2
          LDI       @P02,AR1              ; DMA0
          LDI       @P01,AR0
          LSH3      1,AR3,R0

```

```

*****
* DMA:- TRANSFER BACK RESULT (ROW N-2). BIT-REVERSED *
* - TRANSFER FIRST COLUMN (EXCEPT LAST LOCATION) *
* * *
* CPU:FFT ON LAST ROW (ROW N-1) *
*****

LDI @P01,AR2 ; DMA0
LDI @P02,AR1 ; DMA02
LDI @P03,AR0 ; AR0 POINTS TO DMA03
LDI @P04,AR4 ; AR1 POINTS TO DMA04
B2 TSTB @MASK,IIF
BZAT B2
* DMA02: BIT-REVERSED TRANSFER OF LAST RESULT (Im)
ADDI 1H,AR6,R0
STI R0,*+AR1(1) ; SOURCE
ADDI 1H,AR7,R0
STI R0,*+AR1(4) ; DST

* DMA01: BIT-REVERSED TRANSFER OF LAST RESULT (ROW N-2: Re)
STI AR6,*+AR2(1) ; SOURCE
STI AR6,*+AR0(4) ; DESTINATION: BLOCK0(RE)
STI AR7,*+AR2(4) ; DST
STIK 2H,*+AR0(5) ; DESTINATION INDEX=2 (RE)
STIK 2H,*+AR4(5) ; DESTINATION INDEX=2 (IM)
* DMA03: TRANSFER COLUMN 0 (Re) EXCEPT LAST LOCATION
* DMA04: TRANSFER COLUMN 0 (Im) EXCEPT LAST LOCATION
LDI @CTRL2,R1
STI R1,*AR0
LDI @CTRL3,R0
STI R0,*AR4
LDI @MATR,R0 ; R0:ADDRESS OF FIRST COLUMN
STI R0,*+AR0(1) ; SOURCE: (RE)

ADDI 1,R0 ; POINTS TO IMAGINARY PART
STI R0,*+AR4(1) ; SOURCE: (IM)
ADDI 1,AR6,R1 ;
STI R1,*+AR4(4) ; DESTINATION: BLOCK0(IM)
LSH3 1,AR3,R1
STI R1,*+AR0(2) ; SOURCE INDEX=2*N
SUBI 1,AR3,R0 ; R0=N-1
STI R1,*+AR4(2) ;
AND 0H,IIF ; CLEAR FLAG
STI R0,*+AR0(3) ; COUNTER=N-1
ADDI R1,AR7 ; R1=2N
STI R0,*+AR4(3)

LDI @DMA0,AR0 ; GIVE THE START
LDI R7,R2 ; EXCHANGE BUFFER POINTERS
LDI AR6,R7 ; R7: POINTER FOR NEXT DMA
STIK 0,*+AR0(3)
STI AR2,*+AR0(6)

* FFT ON LAST ROW
LAJ CFFT
LDI R2,AR6 ; AR6: POINTER FOR NEXT FFT
LDI @CTRL2,R0
STI R0,*AR0 ; START (DMA)

```

```

*****
*      DMA: -TRANSFER BACK RESULT (LAST ROW)
*
*      -TRANSFER SECOND COLUMN (COLUMN 1)
*
*      CPU: FFT ON FIRST COLUMN
*
*****

        LDI      @P02,AR1      ; DMA02
        LDI      @P01,AR0      ; P01
        LDI      @P03,AR4      ; AR0 POINTS TO DMA03
        LDI      @P04,AR5      ; AR1 POINTS TO DMA04

B3      TSTB      @MASK,IIF
        BZAT      B3
* CPU MOVES LAST VALUE (1ST COLUMN)FROM AR6: BLOCK1 TO R7:
        LSH3      1,AR3,R2
        ADDI      R2,R7,AR2
        SUBI      2,AR2        ; AR2= BLOCK0+SIZE2-2
        ADDI      1H,AR6,R2
        LDF       *AR6,R0      ; RE
||      LDF       *+AR6(1),R1  ; IM
        STI       R2,*+AR1(1)  ; SOURCE
        STF       R0,*AR2
||      STF       R1,*+AR2(1)

* DMA02: BIT-REVERSED TRANSFER OF LAST RESULT (Im)
        ADDI      1H,AR7,R0
        STI       R0,*+AR1(4)  ; DST
* DMA01: BIT-REVERSED TRANSFER OF LAST RESULT (Re)
        STI       AR6,*+AR0(1)  ; SOURCE
        STI       AR7,*+AR0(4)  ; DST
* DMA03: TRANSFER COLUMN 1 (Re)
* DMA04: TRANSFER COLUMN 1 (Im)
        LDI      @MATR,AR7
        ADDI      2,AR7,R0      ; R0: POINTS TO COLUMN 1
        STI       R0,*+AR4(1)  ; SOURCE: (RE)
        ADDI      1,R0         ; POINTS TO IMAGINARY PART
        STI       R0,*+AR5(1)  ; SOURCE: (IM)
        AND       0H,IIF       ; CLEAR FLAG
        STI       AR6,*+AR4(4)  ; DESTINATION: BLOCK1(RE)
        ADDI      1,AR6,R1
        STI       R1,*+AR5(4)  ; DESTINATION: BLOCK0(IM)
        LDI      R7,R2
        STI       AR3,*+AR4(3)  ; COUNTER=N
        LDI      @DMA0,AR1     ; GIVE THE START
        LDI      AR6,R7        ; R7: BLOCK1
        STI       AR3,*+AR5(3)
        STIK      0,*+AR1(3)
        STI       AR0,*+AR1(6)

* FFT ON FIRST COLUMN

        LAJ       CFFT
        LDI      R2,AR6        ; AR6: POINTER FOR NEXT FFT
        LDI      @CTRL2,R0
        STI       R0,*AR1      ; START (DMA)

```



```

*****
*                      FFT ON COLUMNS                      *
*****

t1:    LDI        @P02,AR1
        LDI        @P01,AR0          ; P01
        SUBI       3,AR3,AR5        ; AR5=N-3: (N-2) DMA TRANSFERS
        LSH3       1,AR3,R1
        STI        R1,*+AR0(5)      ; DST INDEX
        STI        R1,*+AR1(5)      ; DST INDEX
        ADDI       1H,AR6,R0

B4      TSTB       @MASK,IIF
        BZAT       B4

* DMA02: BIT-REVERSED TRANSFER OF LAST RESULT (Im)
        STI        R0,*+AR1(1)      ; SOURCE
        ADDI       1H,AR7,R0
        STI        R0,*+AR1(4)      ; DST
* DMA01: BIT-REVERSED TRANSFER OF LAST RESULT (Re)
        STI        AR6,*+AR0(1)     ; SOURCE
        STI        AR7,*+AR0(4)     ; DST
* DMA03: TRANSFER NEXT COLUMN (Re)
* DMA04: TRANSFER NEXT COLUMN (Im)

        LDI        @P03,AR4          ; AR0 POINTS TO DMA03
        LDI        @P04,AR2          ; AR1 POINTS TO DMA04
        ADDI       2,AR7              ; R0: POINTS TO NEXT COLUMN
        ADDI       2,AR7,R0
        STI        R0,*+AR4(1)       ; SOURCE: (RE)
        AND        0H,IIF            ; CLEAR FLAG
        STI        AR6,*+AR4(4)       ; DESTINATION: BLOCK1(RE)
        ADDI       1,R0 ; POINTS TO IMAGINARY PART
        STI        R0,*+AR2(1)       ; SOURCE: (IM)
        ADDI       1,AR6,R1
        STI        R1,*+AR2(4)       ; DESTINATION: BLOCK0(IM)
        LDI        @DMA0,AR1         ; GIVE THE START
        LDI        R7,R2
        LDI        AR6,R7            ; R7: BLOCK1
        STIK       0,*+AR1(3)
        STI        AR0,*+AR1(6)

* FFT ON CURRENT COLUMN
        LAJ        CFFT
        LDI        R2,AR6            ; AR6: POINTER FOR NEXT FFT
        LDI        @CTRL2,R0
        STI        R0,*AR1           ; START (DMA)
        DBUD       AR5,B4
        LDI        @P02,AR1
        LDI        @P01,AR0
        ADDI       1H,AR6,R0

*****
* DMA: TRANSFER LAST FFT RESULT
* CPU:   FFT ON LAST COLUMN
*****

        LDI        @P02,AR1          ; DMA0
B5      TSTB       @MASK,IIF
        BZAT       B5

```

```

* DMA02/DMA01: BIT-REVERSED TRANSFER OF LAST RESULT
    ADDI    1H,AR6,R0
    STI     R0,*+AR1(1)      ; SOURCE
    ADDI    1H,AR7,R0
    STI     R0,*+AR1(4)      ; DST
    LDI     @P01,AR0         ; P01
    LDI     @CTRL0,R0
    STI     R0,*AR1
    STI     AR6,*+AR0(1)      ; SOURCE
    STI     AR7,*+AR0(4)      ; DST
    LDI     @DMA0,AR1        ; GIVE THE START
    ADDI    2,AR7
    AND     0,IIF
    STIK    0,*+AR1(3)
    STI     AR0,*+AR1(6)
    LDI     @CTRL2,R0
    STI     R0,*AR1          ; START (DMA)

* FFT ON CURRENT ROW
    LAJ     CFFT
    LDI     R7,R2
    LDI     AR6,R7
    LDI     R2,AR6
    SUBI3   2,AR3,RC         ; RC=N-2
    LDI     AR6,AR0          ; SOURCE
    LDI     AR7,AR1          ; DESTINATION
    RPTBD   B6
    LDI     AR3,IR0
    LSH3    1,AR3,IR1
    LDF     *+AR0(1),R0 ; R0= X(I) IM

* LOOP
    LDF     *AR0++(IR0)B,R1   ; X(I) RE & POINTS TO X(I+1)
    ||     STF     R0,*+AR1(1) ; STORE X(I) IM
B6        LDF     *+AR0(1),R0 ; R0=X(I+1) IM
    ||     STF     R1,*AR1++(IR1) ; STORE X(I) RE

* STORE LAST VALUE
B7        TSTB     @MASK,IIF
    BZ      B7
    LDF     *AR0++(IR0)B,R1   ; LOAD X(N-1) RE
    ||     STF     R0,*+AR1(1) ; STORE X(N-1) IM
    STF     R1,*AR1          ; STORE X(N-1) RE
    LDI     @TIMER,AR2        ; OPTIONAL: BENCHMARKING (TIME_READ)
    LDI     *+AR2(4),R0       ; TCOMP = R0

t2        B        t2
    .end

```

SERB.CMD

```
input.obj
serb.obj
sintab.obj
-lmylib.lib
-m serb.map
-o serb.out
```

MEMORY

```
{
    ROM:      o = 0x00000000 l = 0x1000
    BUF0:     o = 0x002ff800 l = 0x200
    RAM0:     o = 0x002ffa00 l = 0x200
    BUF1:     o = 0x002ffc00 l = 0x200
    RAM1:     o = 0x002ffe00 l = 0x200
    LM:       o = 0x40000000 l = 0x10000
    GM:       o = 0x80000000 l = 0x20000
}
```

SECTIONS

```
{
    INPUT      :{} > LM
    .text      :{} > LM
    .data      :{} > RAM1      /* SINE TABLE */
    STACK      :{} > RAM1
    DMA_AUTOINI :{} > RAM1      /* AUTOINIT. VALUES */
}
```

Appendix B: Parallel 2-D FFT (Shared-Memory Version)

B.1. SH.C: Single-Buffered Implementation (C Program)

SH.C

```
/* **** */

SH.C :   Parallel 2-dimensional complex FFT
        (shared-memory single-buffered version)

Routines used:   cfftc.asm      (C-callable complex-fft)
                  cmove.asm     (CPU complex move)
                  cmoveb.asm    (CPU bit-reversed complex move)
                  syncount.asm  (synchronization routine via counter in-shared memory)

To run:
    cl30 -v40 -g -as -mr -o2 sh.c
    asm30 -v40 -s input.asm
    asm30 -v40 -s synch.asm
    asm30 -v40 -s sintab.asm
    lnk30 shc.cmd

Note: Before running, initialize my_node variable with the corresponding value, using
the 'C40 emulator or an assembly file.

/* **** */
#define SIZE      16                /* FFT size (n)                */
#define LOGSIZE   4                /* log (FFT size)              */
#define P         2                /* number of processors        */
#define Q         SIZE/P           /* rows/col. per processor     */
#define BLOCK0    0x002ff800       /* on-chip RAM buffer */

extern void cfftc(), cmove(), cmoveb(), syncount();
extern int colsynch;               /* column/row synchronization */
extern float MATRIX[SIZE][SIZE*2]; /* Input matrix                */

float *block0 = (float *)BLOCK0,
      *MM[SIZE];

int my_node ,                      /* node-id                      */
    *colsynch_p = & colsynch, /* row/column synchronization */
    size2 = 2*SIZE,
    q2 = 2*Q,
    i,l1,l2;
int tcomp;

/* **** */

main()
{
    asm(" OR 1800h,st");           /* cache enable                */

    for (i=0;i<SIZE;i++) MM[i]=MATRIX[i]; /* accessing assembly variables */

    t0: syncount(colsynch_p,P);     /* Optional: Common start      */
        time_start(0);             /* Optional: Benchmarking      */
}
```

```

/***** FFT on rows *****/

l1= Q*my_node;      l2 = l1+Q;      /* select row working set      */

for (i=l1; i<l2;i++) {
    cmove (&MM[i][0],block0,2,2,SIZE);
    cfft(c(block0,SIZE,LOGSIZE);
    cmoveb (block0,&MM[i][0],SIZE,2,SIZE);
}

t1: syncount(colsynch_p,2*P);      /* row/column synchronization      */

/***** FFT by columns *****/

l1 = l1<<1;      l2 = l2<<1;      /* select column working set: multiply by 2 */

for (i=l1;i<l2;i+=2) {
    cmove (&MM[0][i],block0,size2,2,SIZE);
    cfft(c(block0,SIZE,LOGSIZE);
    cmoveb (block0,&MM[0][i],SIZE,size2,SIZE);
}

tcomp = time_read(0);      /* Optional: Benchmarking (timer)      */

t2: ;
} /*main*/

```

SHC.CMD

```
sh.obj
input.obj
sintab.obj
synch.obj
-c
-stack 0x0100
-lrts40.lib
-lprts40r.lib
-lmylib.lib
-m shc.map
-o shc.out

/* SPECIFY THE SYSTEM MEMORY MAP */

MEMORY
{
    ROM:      org = 0x0      len = 0x1000
    RAM0:     org = 0x002ff800 len = 0x0400 /* on-chip RAM block 0 */
    RAM1:     org = 0x002ffc00 len = 0x0400 /* on-chip RAM block 1 */
    LM:       org = 0x40000000 len = 0x10000 /* LOCAL MEMORY */
    GM:       org = 0x80000000 len = 0x20000 /* GLOBAL MEMORY */
}

/* SPECIFY THE SECTIONS ALLOCATION INTO MEMORY */

SECTIONS
{
    .text:    {} > RAM1
    .cinit:   {} > RAM1 /* INITIALIZATION TABLES */
    .stack:   {} > RAM1 /* SYSTEM STACK */
    .bss:     {} > RAM1
    .data:    {} > RAM1 /* Sine table */
    INPUT:    {} > GM /* Input matrix */
    SYNCH:    {} > GM /* Synchronization */
}
```

SYNCH.ASM

```
*****
*
* SYNCH.ASM : File containing shared-memory location for
* interprocessor synchronization
*
*****

.global _colsynch
.sect "SYNCH"

_colsynch .int 0

.end
```

B.2. SH.ASM: Single-Buffered Implementation ('C40 Assembly Program)

SH.ASM

```
*****
*
*      SH.ASM :          TMS320C40 complex 2D-FFT serial program
*                        (Single-buffered version)
*
*      Routines used:    cfft.asm (radix-2 complex FFT)
*
*      Requirements:     Number of processors = P > 0
*                        rows/columns per processor = Q > 0
*
*      To run:
*      asm30 -v40 -g -s sh.asm
*      asm30 -v40 -g -s spinput.asm
*      asm30 -v40 -g -s input.asm
*      asm30 -v40 -g -s ssintab.asm
*      asm30 -v40 -g -s synch.asm
*      asm30 -v40 -g -s 0.asm
*      asm30 -v40 -g -s 1.asm
*      lnk30 sh.cmd 0.obj -o a0.out (program for processor 0)
*      lnk30 sh.cmd 1.obj -o a1.out (program for processor 1)
*
*****

        .global      N          ; FFT size
        .global      P          ; Number of processors
        .global      Q          ; Rows per processor
        .global      MYNODE
        .global      _MATRIX    ; Matrix address
        .global      _colsynch   ; Synchronization counter
        .global      CFFT       ; Complex 1D-FFT subroutine
        .global      C2DFFT     ; Entry point for execution
        .global      _syncount
_STACK   .usect      "STACK",10h ; Stack definition
        .text
FFTSIZE  .word       N
PROC     .word       P
NROWS    .word       Q
MYID     .word       MYNODE
MATR     .word       _MATRIX
SYNCH    .word       _colsynch
STACK    .word       _STACK     ; Stack address
BLOCK0   .word       002FF800H  ; On-chip buffer (RAM block 0)
TIMER    .word       0100020H   ; Timer 0 address (Benchmarking)

C2DFFT
        LDP         FFTSIZE     ; Data page pointer initialization
        LDI         @STACK,SP   ; Stack pointer initialization
        LDI         @SYNCH,AR2  ; Optional: Common start (benchmarking)
        LDI         @PROC,R2    ; wait until counter = P
```

```

t0:      CALL      _syncount
        LDI        @TIMER,AR2          ; Optional: benchmarking (time_start)
        STIK       -1,*+AR2(8)
        LDI        961,R0
        STI        R0,*AR2
        OR         1800h,ST           ; Enabling cache
        LDI        @FFTSIZE,AR3       ; AR3 = N = FFT size
        LDI        @MYID,R0
        LDI        @NROWS,AR5        ; AR5 = row counter = Q
        MPYI       AR5,R0             ; Q*MY_NODE
        MPYI       AR3,R0             ; R0 = N*Q*MYNODE
        LSH        1,R0               ; R0 = 2*N*Q*MYNODE
        LDI        @MATR,AR7
        ADDI       R0,AR7             ; AR7 = matrix pointer = &MATRIX[Q*MYNODE][0]
        SUBI       1,AR5              ; AR5 = Q-1
        LDI        @BLOCK0,AR6       ; AR6 = on-chip buffer pointer

*****
*                               FFT ON ROWS
*****

LOOPR
*****
* Move row X (X = AR7) *
* to on-chip memory *
*****

        SUBI3      2,AR3,RC           ; RC = N-2
        LDI        AR7,AR0            ; Source address
        RPTBD      LOOP1
        LDI        AR6,AR1            ; Destination address
        LDI        2,IR0              ; Destination offset
        LDF        *+AR0(1),R0        ; R0 = X(I) Im
        LDF        *AR0++(IR0),R1     ; X(I) Re & points to X(I+1)
        ||        STF        R0,*+AR1(1) ; Store X(I) Im
LOOP1   LDF        *+AR0(1),R0        ; R0 = X(I+1) Im
        ||        STF        R1,*AR1++(IR0) ; Store X(I) Re

*****
* FFT on row X *
*****

        LAJ        CFFT               ; Call 1D-FFT (complex)
        LDF        *AR0,R1            ; Load X(N-1) Re
        NOP
        STF        R0,*+AR1(1)        ; Store X(N-1) Im
        ||        STF        R1,*AR1   ; Store X(N-1) Re

*****
* Move row X (bit-reversed) from *
* on-chip memory to external memory *
*****

        SUBI3      2,AR3,RC
        LDI        AR6,AR0            ; Source address
        LDI        AR7,AR1            ; Destination Address
        RPTBD      LOOP2
        LDI        AR3,IR0            ; Source offset for bit-reverse = N
        LDI        2,IR1              ; Destination offset
        LDF        *+AR0(1),R0

```



```

        LDF      *AR0++(IR0)B,R1
||      STF      R0,*+AR1(1)
LOOP2   LDF      *+AR0(1),R0
||      STF      R1,*AR1++(IR1)
        LDF      *AR0++(IR0)B,R1
||      STF      R0,*+AR1(1)
        DBUD     AR5,LOOPR
        STF      R1,*AR1++(IR1)
        LSH3     1,AR3,R0
        ADDI     R0,AR7

*****
*                      FFT ON COLUMNS
*****

        LDI      @MYID,R0
        LDI      @NROWS,AR5           ; AR5 = column counter = Q
        MPYI     AR5,R0               ; Q*MY_NODE
        LSH      1,R0                ; 2*Q*MY_NODE (Complex numbers)

        LDI      @MATR,AR7
        ADDI     R0,AR7               ; AR7 = &MATRIX[0][2*Q*MYNODE]

        SUBI     1,AR5               ; AR5 = Q-1

        LDI      @SYNCH,AR2           ; Row/column synchronization
        LDI      @PROC,R2
        LSH      1,R2                ; Optional: not needed if no common start
                                         ; is required

t1:      CALL     _syncount

LOOPC
*****
* Move column X (X=AR7) *
* to on-chip memory      *
*****

        SUBI3    2,AR3,RC             ; RC=N-2
        LDI      AR7,AR0              ; Source address
        LDI      AR6,AR1              ; Destination address
        RPTBD    LOOP3
        LSH3     1,AR3,IR1            ; Source offset = 2*N
        LDI      2,IR0               ; Destination offset
        LDF      *+AR0(1),R0          ; R0= X(I) Im

        LDF      *AR0++(IR1),R1       ; X(I) Re & points to X(I+1)
||      STF      R0,*+AR1(1)          ; Store X(I) Im
LOOP3   LDF      *+AR0(1),R0          ; R0=X(I+1) Im
||      STF      R1,*AR1++(IR0)      ; Store X(I) Re

*****
* FFT on column X *
*****

        LAJ      CFFT
        LDF      *AR0,R1              ; Load X(N-1) Re
        NOP
        STF      R0,*+AR1(1)          ; Store X(N-1) Im
||      STF      R1,*AR1              ; Store X(N-1) Re

```

```

*****
* Move column X (bit-reversed) from *
* on-chip memory to external memory *
*****

                SUBI3      2,AR3,RC          ; RC=N-2
                LDI        AR6,AR0          ; Source address
                LDI        AR7,AR1          ; Destination address
                RPTBD      LOOP4
                LDI        AR3,IR0          ; Source offset = IR0 = N (bit-reverse)
                LSH        1,AR3,IR1        ; Destination offset (columns) = IR1 = 2N
                LDF        *+AR0(1),R0
                LDF        *AR0++(IR0)B,R1
||              STF        R0,*+AR1(1)
LOOP4          LDF        *+AR0(1),R0
||              STF        R1,*AR1++(IR1)
                DBUD       AR5,LOOPC
                LDF        *AR0++(IR0)B,R1
||              STF        R0,*+AR1(1)
                STF        R1,*AR1++(IR1)
                ADDI       2,AR7
                LDI        @TIMER,AR2       ; Optional: benchmarking (time_read)
                LDI        *+AR2(4),R0      ; tcomp = R0

t2             B          t2
                .end

```

SH.CMD

```

input.obj
sh.obj
spininput.obj
ssintab.obj
synch.obj
-m sh.map
-lmylib.lib
-osh.out

MEMORY
{
    ROM:          org = 0x00          len = 0x1000
    RAM0:         org = 0x002ff800 len = 0x0400 /* On-chip RAM block 0 */
    RAM1:         org = 0x002ffc00 len = 0x0400 /* On-chip RAM block 1 */
    LM:           org = 0x40000000 len = 0x10000 /* LOCAL MEMORY */
    GM:           org = 0x80000000 len = 0x20000 /* GLOBAL MEMORY */
}

/* SPECIFY THE SECTIONS ALLOCATION INTO MEMORY */

SECTIONS
{
    .text:        {}      >RAM1          /* CODE */
    .data:        {}      >RAM1          /* Sine tables */
    INPUT:        {}      >GM            /* Input matrix */
    STACK:        {}      >RAM1          /* SYSTEM STACK */
    SYNCH:        {}      >GM            /* synchronization */
}

```

SPINPUT.ASM

```
*****
*
* SPINPUT.ASM : input file for shared-memory program (Data on parallel system)
*
*****

        .global  N   ; FFT size
        .global  M   ; LOG2 FFT
        .global  P   ; Number of processors
        .global  Q   ; Rows per processor
N        .set     16
M        .set     4
P        .set     2
Q        .set     N/P

        .end
```

0.ASM

```
*****
* 0.ASM : File containing node-id for processor 0
*****
        .global MYNODE
MYNODE  .set 0
```

1.ASM

```
*****
* 1.ASM : File containing node-id for processor 1
*****
        .global MYNODE
MYNODE  .set 1
```

B.3. SHB.C: Double-Buffered Implementation (C Program)

SHB.C

```
/*
*****

SHB.C :   Parallel 2-dimensional complex FFT
         (shared-memory double-buffered version)

```

```
Routines used:   cfftc.asm      (C-callable radix-2 complex-fft)
                  cmove.asm     (CPU complex move)
                  cmoveb.asm    (CPU bit-reversed complex move)
                  set_dma.asm   (Routine to set DMA register values)
                  syncount.asm  (synchronization routine)

```

To run:

```
cl30 -v40 -g -as -mr -o2 shb.c
asm30 -v40 -s input.asm
asm30 -v40 -s synch.asm
asm30 -v40 -s sintab.asm
lnk30 shbc.cmd

```

Requirement: $Q \geq 4$

Note: Before running initialize the my_node variable to the corresponding value using the 'C40 emulator or an assembly file.

```
*****/
#define SIZE      16                /* FFT size */
#define LOGSIZE   4                /* log (FFT size) */
#define P         2                /* number of processors */
#define Q         SIZE/P           /* row/col. per processor */
#define BLOCK0    0x002ff800       /* on-chip buffer 0 */
#define BLOCK1    0x002ffc00       /* on-chip buffer 1 */
#define DMA0      0x001000a0       /* DMA0 address */
#define SWAP(x,y) temp = x; x = y; y = temp;
#define WAIT_DMA(x) while ((0x00c00000 & *x) != 0x00800000);

extern void cfftc(), set_dma(), cmove(), cmoveb(), syncount();
extern int colsynch; /* counter in GM */
extern float MATRIX[SIZE][SIZE*2]; /* input matrix */
int ctrl0= 0x00c41004, /* no autoinit.,dmaint,bit_rev */
    ctrl1= 0x00c01008, /* autoinit.,no dmaint,bit-rev */
    ctrl2= 0x00c00008, /* autoinit., no dmaint */
    ctrl3= 0x00c40004; /* no autoinit.,dmaint */
float *CPUbuffer  =(float *)BLOCK0, /* For CPU FFT operations */
      *DMABuffer  =(float *)BLOCK1, /* For DMA operations */
      *temp,
      *MM[SIZE];
volatile int *dma0 = (int *)DMA0;
int dma01[7], dma02[7], dma03[7], dma04[7]; /* DMA autoinit.values */
int my_node,
    base,
    *colsynch_p = &colsynch,
    size2 = (SIZE*2),
    q = Q,
    q2 = 2*Q,
    ii,i,j;
int tcomp;
```

```

/*****
main()
{
asm(" OR 1800h,ST");
for (i=0;i<SIZE;i++) MM[i]=MATRIX[i];    /* accessing assembly variables */

t0:

syncount(colsynch_p,P);    /* Optional: Common start */
time_start(0);    /* Optional: Benchmarking (timer) */
base = q*my_node;    /* point to 1st row allocated to each processor */

/***** FFT on rows *****/

/*****
1.DMA:    moves row (base+1) to on-chip RAM buffer 0    *
2.CPU:    - moves row (base+0) to on-chip RAM buffer 1    *
          - FFT on row (base+0) in buffer 1    *
*****/
ii =base+1;
set_dma(dma0,ctrl3,&MM[ii][0],1,size2,DMAbuffer,1,1);
cmove(&MM[base][0],CPUbuffer,2,2,SIZE);
cfft(c(CPUbuffer,SIZE,LOGSIZE);

/*****
1.DMA:    - moves Re FFT (row(base+0)) to off-chip RAM    *
          - moves Im FFT (row(base+0)) to off-chip RAM    *
          - moves row (base+2) to on-chip RAM    *
2.    CPU:    FFT on row (base+1)    *
*****/

WAIT_DMA(dma0);
set_dma(dma01,ctrl1,CPUbuffer,SIZE,SIZE,&MM[base][0],2,dma02);
set_dma(dma02,ctrl1,(CPUbuffer+1),SIZE,SIZE,&MM[base][1],2,dma03);
set_dma(dma03,ctrl3,&MM[base+2][0],1,size2,CPUbuffer,1,1);
*(dma0+3) = 0; *(dma0+6) = (int) dma01;    *dma0 =ctrl2; /* start DMA */

cfft(c(DMAbuffer,SIZE,LOGSIZE);

/*****
1.    DMA: - moves Re FFT row ii to off-chip RAM    *
          - moves Im FFT row ii to off-chip RAM    *
          - moves row(ii+2) to on-chip RAM    *
2.    CPU:    FFT on row(ii+1)    *
*****/

for (i=1;i<q-2;i++,ii++)    {
WAIT_DMA(dma0);

*(dma03+1) = (int)&MM[ii+2][0];    *(dma03+4) = (int)DMAbuffer;
*(dma01+1) = (int)DMAbuffer;    *(dma01+4) = (int)&MM[ii][0];
*(dma02+1) = (int)DMAbuffer+1;    *(dma02+4) = (int)&MM[ii][1];
*(dma0+3) = 0; *(dma0+6) = (int) dma01; *dma0 = ctrl2; /* start DMA */

cfft(c(CPUbuffer,SIZE,LOGSIZE);    /* work in current row */
SWAP (CPUbuffer,DMAbuffer);    /* switch buffers */
}

```

```

/*****
1.    DMA:  - moves Re FFT(row(base+q-2)) to off-chip RAM      *
          - moves Im FFT(row(base+q-2)) to off-chip RAM      *
2.    CPU:  - FFT on last row : row(base+q-1)                  *
          - moves FFT last row to off-chip RAM                *
*****/

WAIT_DMA(dma0);
*(dma01+1) = (int)DMAbuffer      ;   *(dma01+4) = (int)&MM[ii][0];
*(dma02+1) = (int)DMAbuffer+1    ;   *(dma02+4) = (int)&MM[ii][1];
*dma02     = (int)ctrl10;
*(dma03)    = 0                  ;   *(dma0+6) = (int) dma01; *dma0 = ctrl2; /* start DMA*/

cfft(CPUbuffer,SIZE,LOGSIZE); /* fft on last row */
cmov(b(CPUbuffer,&MM[ii+1][0],SIZE,2,SIZE);

/***** FFT on columns *****/

CPUbuffer= (float *) BLOCK0;
DMAbuffer= (float *) BLOCK1;
WAIT_DMA(dma0);
syncount(col      synch_p,2*P); /* row/column synchronization */

/*****
1.    DMA:  - moves Re column (base+1) to on-chip RAM      *
          - moves Im column (base+1) to on-chip RAM      *
2.    CPU:  - moves column (base+0) to on-chip RAM        *
          - FFT on column (base+0)                        *
*****/

ii = 2*base;
set_dma(dma03,ctrl2,&MM[0][ii+2],size2,SIZE,DMAbuffer,2,dma04);
set_dma(dma04,ctrl3,&MM[0][ii+3],size2,SIZE,(DMAbuffer+1),2,2);
*(dma0+3) = 0;      *(dma0+6) = (int) dma03; *dma0 = ctrl2;
cmov(&MM[0][ii],CPUbuffer,size2,2,SIZE);
cfft(CPUbuffer,SIZE,LOGSIZE);

/*****
1.    DMA:  - moves Re FFT column (ii) to off-chip RAM      *
          - moves Im FFT column (ii) to off-chip RAM      *
          - moves Re column (ii+2) to on-chip RAM          *
          - moves Im column (ii+2) to on-chip RAM          *
2.    CPU:  - FFT on column (ii+1)                          *
*****/

*(dma02+5) = *(dma01+5) = (int)size2; /* offset */
*dma02     = (int)ctrl11;

for (i=0;i<q2-4;i+=2,ii+=2) {

SWAP (CPUbuffer,DMAbuffer);
WAIT_DMA(dma0);

*(dma01+1) = (int)DMAbuffer;      *(dma01+4) = (int)&MM[0][ii];
*(dma02+1) = (int)DMAbuffer+1;    *(dma02+4) = (int)&MM[0][ii+1];
*(dma03+1) = (int)&MM[0][ii+4];    *(dma03+4) = (int)DMAbuffer;
*(dma04+1) = (int)&MM[0][ii+5];    *(dma04+4) = (int)DMAbuffer+1;

```

```

*(dma0+3) = 0; *(dma0+6) = (int) dma01; *dma0 = ctrl2;

cfftC(CPUbuffer,SIZE,LOGSIZE);          /* work in current column */
}

/*****
1.      DMA:      - moves Re FFT column (base+q-2) to off-chip RAM      *
                - moves Im FFT column (base+q-2) to off-chip RAM      *
2.      CPU:      - FFT on last column (base+q-1)                      *
                - moves FFT (last column) to off-chip RAM            *
*****/
WAIT_DMA(dma0);

*(dma01+1) = (int)CPUbuffer;      *(dma01+4) = (int)&MM[0][ii];
*(dma02+1) = (int)CPUbuffer+1;    *(dma02+4) = (int)&MM[0][ii+1];
*(dma02)   = ctrl0;

*(dma0+3) = 0;      *(dma0+6) = (int) dma01; *dma0 = ctrl2;

cfftC(DMAbuffer,SIZE,LOGSIZE);      /* fft on last column */
cmovEb(DMAbuffer,&MM[0][ii+2],SIZE,size2,SIZE);

WAIT_DMA(dma0);

tcomp= time_read(0);          /* Optional: Benchmarking (timer) */
t2: ;
} /*main*/

```

SHBC.CMD

```

shb.obj
input.obj
sintab.obj
synch.obj
-c
-stack 0x0100
-lrts40.lib
-lprts40r.lib
-lmylib.lib
-m shbc.map
-o shbc.out

/* SPECIFY THE SYSTEM MEMORY MAP */

MEMORY
{
    ROM:      org = 0x0          len = 0x1000
    BUF0:     org = 0x002ff800 len = 0x0200    /* buffer in onchip RAM block0 */
    BUF1:     org = 0x002ffc00 len = 0x0200    /* buffer in onchip RMA block1 */
    RAM0:     org = 0x002ffa00 len = 0x0200    /* on-chip RAM block 0 */
    RAM1:     org = 0x002ffe00 len = 0x0200    /* on-chip RAM block 1 */
    LM:       org = 0x40000000 len = 0x10000    /* local memory */
    GM:       org = 0x80000000 len = 0x20000    /* global memory */
}

/* SPECIFY THE SECTIONS ALLOCATION INTO MEMORY */

```

```

SECTIONS
{
    .text:    {} >    LM
    .cinit:   {} >    RAM1    /* initialization tables    */
    .stack:   {} >    RAM1    /* system stack            */
    .bss :    {} >    RAM1
    .data:    {} >    RAM1    /* Sine table              */
    INPUT:    {} >    GM      /* Input matrix            */
    SYNCH:    {} >    GM      /* Synchronization counter */
}

```


B.4. SHB.ASM: Double-Buffered Implementation ('C40 Assembly Program)

SHB.ASM

```
*****
*
*      SHB.ASM :      TMS320C40 complex 2D-FFT shared-memory program
*                      (Double-buffered version)
*
*      Routines used: cfft.asm (radix-2 complex FFT)
*
*      Requirements:  Number of processors = P > 0
*                      Rows/columns per processor = Q >= 4
*
*
*      To run:
*      asm30 -v40 -g -s shb.asm
*      asm30 -v40 -g -s spinput.asm
*      asm30 -v40 -g -s input.asm
*      asm30 -v40 -g -s sintab.asm
*      asm30 -v40 -g -s synch.asm
*      asm30 -v40 -g -s 0.asm
*      asm30 -v40 -g -s 1.asm
*      lnk30 shb.cmd 0.obj -o a0.out (program for processor 0)
*      lnk30 shb.cmd 1.obj -o a1.out (program for processor 1)
*
*****

        .global      N                ; FFT SIZE
        .global      P                ; NUMBER OF PROCESSORS
        .global      Q                ; ROWS/COLUMNS PER PROCESSOR
        .global      MYNODE           ; PROCESSOR ID
        .global      _MATRIX          ; MATRIX ADDRESS
        .global      _colsynch        ; SYNCHRONIZATION COUNTER
        .global      CFFT             ; COMPLEX 1D-FFT SUBROUTINE
        .global      C2DFFT          ; ENTRY POINT FOR EXECUTION
        .global      _syncount        ; SYNCHRONIZATION ROUTINE

_stack   .usect      "STACK", 10h      ; STACK DEFINITION

* DMA AUTOINITIALIZATION VALUES

        .sect        "DMA_AUTOINI"    ; DMA AUTOINITIALIZATION VALUES
DMA01    .space      6
        .word        DMA02
DMA02    .space      6
        .word        DMA03
DMA03    .space      6
        .word        DMA04
DMA04    .space      6      .text
FFTSIZE  .word        N
PROC     .word        P
NROWS    .word        Q
MYID     .word        MYNODE
MATR     .word        _MATRIX
SYNCH    .word        _colsynch
STACK    .word        _stack          ; STACK ADDRESS
BLOCK0   .word        002FF800H      ; RAM BLOCK 0
BLOCK1   .word        002FFC00H      ; RAM BLOCK 1
```

```

DMA0      .word      001000A0H      ; ADDRESS OF DMA0
CTRL0     .word      00C41004H      ; NO AUTOINITIALIZATION, DMA INT., BITREV
CTRL1     .word      00C01008H      ; AUTOINITIALIZATION, NO DMA INT., BITREV
CTRL2     .word      00C00008H      ; AUTOINITIALIZATION, NO DMA INT.
CTRL3     .word      00C40004H      ; NO AUTOINITIALIZATION, DMA INT.
P04       .word      DMA04          ; POINTER TO REGISTER VALUES (DMA04)
P03       .word      DMA03          ; POINTER TO REGISTER VALUES (DMA03)
P02       .word      DMA02          ; POINTER TO REGISTER VALUES (DMA02)
P01       .word      DMA01          ; POINTER TO REGISTER VALUES (DMA01)
MASK      .word      02000000H      ; 1 IN DMAINT0
TIMER     .word      0100020H      ; TIMER 0 ADDRESS (BENCHMARKING)

```

```

C2DFFT    LDP        FFTSIZE        ; LOAD DATA PAGE POINTER
          LDI        @STACK,SP      ; INITIALIZE THE STACK POINTER
          LDI        @SYNCH,AR2     ; OPTIONAL: COMMON START (BENCHMARKING)
          LDI        @PROC,R2       ; WAIT UNTIL COUNTER = P

```

```

t0:        CALL      _syncount
          LDI        @TIMER,AR2     ; OPTIONAL: BENCHMARKING (TIME_START)
          STIK      -1,*+AR2(8)
          LDI        961,R0
          STI       R0,*AR2
          OR        1800h,ST        ; ENABLE CACHE
          LDI        @FFTSIZE,AR3   ; AR3 = N = FFT SIZE
          LDI        @MYID,R0
          LDI        @NROWS,AR5     ; AR5 = Q = ROW COUNTER
          MPYI      AR5,R0          ; R0 = Q*MYNODE
          MPYI      AR3,R0          ; R0 = N*Q*MYNODE
          LSH       1,R0            ; R0 = 2*N*Q*MYNODE
          LDI        @MATR,AR7
          ADDI      R0,AR7          ; MATRIX POINTER = &MATRIX[2*Q*MYNODE][0]
          LDI        @BLOCK1,R7     ; POINTER TO DMA BUFFER
          LDI        @BLOCK0,AR6    ; POINTER TO FFT BUFFER

```

```

*****
*                CPU MOVES ROW 0                *
*****

```

```

          SUBI3     2,AR3,RC        ; RC=N-2
          LDI       AR7,AR0         ; SOURCE
          RPTBD
          LDI       AR6,AR1         ; DESTINATION
          LDI       2,IR1
          LDF       *+AR0(1),R0     ; R0= X0(I) IM

```

```

* LOOP
          LDF       *AR0++(IR1),R1   ; X0(I) RE & POINTS TO X0(I+1)
          STF       R0,*+AR1(1)     ; STORE X0(I) IM
          LDF       *+AR0(1),R0     ; R0=X0(I+1) IM
          STF       R1,*AR1++(IR1)  ; STORE X0(I) RE

```

```

* STORE LAST VALUE

```

```

          LDI       @P02,AR2        ; POINTS DMA REGISTER
          LDF       *AR0++(IR1),R1   ; LOAD X0(N-1) RE
          STF       R0,*+AR1(1)     ; STORE X0(N-1) IM
          STF       R1,*AR1         ; STORE X0(N-1) RE

```

```

*****
*
* SET PARAMETERS FOR DMA AUTOINITIALIZATION VALUES THAT ARE FIXED
*
*****
        LDI        @P03,AR1            ; POINTS DMA REGISTER
        LDI        @P01,AR4
        LDI        @CTRL1,R0
        STI        R0,*AR2
        STI        AR3,*+AR2(2)        ; SOURCE INDEX
        STI        R0,*AR4
        STI        AR3,*+AR4(2)        ; SOURCE INDEX
        STI        AR3,*+AR2(3)        ; COUNTER
        STI        AR3,*+AR4(3)        ; COUNTER
        LSH3       1,AR3,R0            ; R0=2*N
        STIK       2H,*+AR2(5)        ; DESTINATION INDEX
        STIK       2H,*+AR4(5)        ; DESTINATION INDEX
        LDI        @CTRL3,R2
        STI        R2,*AR1
        LDI        @DMA0,AR2          ; POINTS DMA REGISTER
        STIK       1H,*+AR1(2)        ; SOURCE INDEX
        SUBI       3,AR5              ; AR5=Q-3 : (Q-2) DMA TRANSFERS
        STI        R0,*+AR1(3)        ; COUNTER
        STI        AR0,*+AR2(1)        ; SOURCE
        STIK       1H,*+AR1(5)        ; DESTINATION INDEX

*****
*
*          CPU : FFT ON ROW 0
*          DMA : BEGINS TO TRANSFER ROW1
*
*****

        STIK       1H,*+AR2(2)        ; SOURCE INDEX
        STI        R0,*+AR2(3)        ; COUNTER=2N
        LAJ        CFFT              ; FFT ON ROW 0
        STI        R7,*+AR2(4)        ; DESTINATION
        STIK       1H,*+AR2(5)        ; DESTINATION INDEX
        STI        R2,*AR2

*****
*
*          FFT ON ROWS
*
*****

        LDI        @P02,AR1
        LDI        @P01,AR0
        LSH3       1,AR3,R0
LOOP2   TSTB       @MASK,IIF
        BZAT       LOOP2

* DMA02: BIT-REVERSED TRANSFER OF LAST RESULT (Im)
        ADDI       1H,AR6,R1
        STI        R1,*+AR1(1)        ; SOURCE
        ADDI       1H,AR7,R1
        STI        R1,*+AR1(4)        ; DST

* DMA01: BIT-REVERSED TRANSFER OF LAST RESULT (Re)
        STI        AR6,*+AR0(1)        ; SOURCE
        STI        AR7,*+AR0(4)        ; DST

```

```

* DMA03: TRANSFER NEXT ROW
    LDI    @P03,AR1        ; DMA0
    ADDI   R0,AR7
    ADDI   AR7,R0
    STI    R0,*+AR1(1)    ; SOURCE: NEXT ROW
    AND    0H,IIF         ; CLEAR FLAG
    STI    AR6,*+AR1(4)   ; DESTINATION:
    LDI    @DMA0,AR1
    LDI    R7,R2          ; EXCHANGE BUFFER POINTERS
    LDI    AR6,R7         ; R7: POINTER FOR NEXT DMA
    STIK   0,*+AR1(3)     ; TEMPORAL FIX
    STI    AR0,*+AR1(6)

* FFT ON CURRENT ROW
    LAJ    CFFT
    LDI    R2,AR6         ; AR6: POINTER FOR NEXT FFT
    LDI    @CTRL2,R0
    STI    R0,*AR1        ; START (DMA)
    DBUD   AR5,LOOP2
    LDI    @P02,AR1       ; DMA0
    LDI    @P01,AR0
    LSH3   1,AR3,R0

*****
* DMA: - TRANSFERS BACK RESULT (ROW N-2). BIT-REVERSED
*
* CPU: - FFT ON LAST ROW (ROW N-1)
* - TRANSFERS BACK FFT OF LAST ROW (ROW N-1)
*****

    LDI    @P01,AR2       ; DMA01
    LDI    @P02,AR1       ; DMA02
B2    TSTB  @MASK,IIF
    BZAT   B2

* DMA02: BIT-REVERSED TRANSFER OF LAST RESULT (Im)
    ADDI   1H,AR6,R0
    STI    R0,*+AR1(1)    ; SOURCE
    ADDI   1H,AR7,R0
    STI    R0,*+AR1(4)    ; DST
    LDI    @CTRL0,R0
    STI    R0,*AR1        ; CONTROL REGISTER

* DMA01: BIT-REVERSED TRANSFER OF LAST RESULT (ROW N-2: Re)
    STI    AR6,*+AR2(1)   ; SOURCE
    STI    AR7,*+AR2(4)   ; DST
    AND    0H,IIF         ; CLEAR FLAG
    LSH    1,AR3,R0
    ADDI   R0,AR7         ; AR7 POINTS TO MATRIX (LAST ROW)
    LDI    @DMA0,AR0      ; GIVE THE START
    LDI    R7,R2          ; EXCHANGE BUFFER POINTERS
    LDI    AR6,R7         ; R7: POINTER FOR NEXT DMA
    STIK   0,*+AR0(3)     ; COUNTER = 0
    STI    AR2,*+AR0(6)   ; LINK POINTER = DMA01

* FFT ON LAST ROW
    LAJ    CFFT
    LDI    R2,AR6         ; AR6: POINTER FOR NEXT FFT
    LDI    @CTRL2,R0
    STI    R0,*AR0        ; START (DMA)

```

* CPU TRANSFERS FFT OF LAST ROW TO OFF-CHIP RAM

```

        SUBI3      2,AR3,RC          ; RC=N-2
        LDI        AR6,AR0           ; SOURCE
        LDI        AR7,AR1           ; DESTINATION
        RPTBD      B66
        LDI        AR3,IR0
        LDI        2,IR1
        LDF        *+AR0(1),R0       ; R0= X(I) IM

* LOOP
        LDF        *AR0++(IR0)B,R1   ; X(I) RE & POINTS TO X(I+1)
||      STF        R0, *+AR1(1)      ; STORE X(I) IM
B66     LDF        *+AR0(1),R0       ; R0=X(I+1) IM
||      STF        R1,*AR1++(IR1)    ; STORE X(I) RE

* STORE LAST VALUE
        LDF        *AR0++(IR0)B,R1   ; LOAD X(N-1) RE
||      STF        R0,*+AR1(1)      ; STORE X(N-1) IM
        STF        R1,*AR1           ; STORE X(N-1) RE

```

```

*****
*                               FFT ON COLUMNS                               *
*****

```

```

WAIT    TSTB      @MASK,IIF
        BZAT      WAIT
        LDI       @SYNCH,AR2         ; ROW/COLUMN SYNCHRONIZATION
        LDI       @PROC,R2
        NOP
        LSH       1,R2               ; OPTIONAL: NOT NEEDED IF COMMON START
                                         ; IS NOT REQUIRED
        AND       0,IIF
t1:     CALL      _syncount

```

```

*****
* SET AUTOINITIALIZATION VALUES
*****

```

```

        LDI       @P01,AR0           ; DMA01
        LDI       @P02,AR1           ; DMA02
        LDI       @P03,AR2           ; DMA03
        LDI       @P04,AR4           ; DMA04 LDI 5,IR0
        LDI       @CTRL1,R4          ; SET DMA02 AND DMA01 NEW VALUES
        LSH       1,AR3,R0
        STI       R4,*AR1
        STI       R0,*+AR0(IR0)
||      STI       R0,*+AR1(IR0)      ; SET DMA03 AND DMA04 VALUES
        STI       R0,*+AR2(2)        ; (SRC INDEX)
        STI       R0,*+AR4(2)
        LDI       @CTRL2,R0
        LDI       @CTRL3,R1
        STI       R0,*AR2             ; (CTRL)
||      STI       R1,*AR4
        STI       AR3,*+AR2(3)        ; (COUNTER)
        STI       AR3,*+AR4(3)
        STIK      2,*+AR2(IR0)        ; (DST INDEX)
        STIK      2,*+AR4(IR0)
        STI       AR4,*+AR2(6)        ; (LINK POINTER)

```

```

*****
* DMA : - TRANSFER COLUMN 1 TO ON-CHIP RAM (BLOCK1)
*****
      LDI      @BLOCK0,AR6
      LDI      @BLOCK1,R7
      LDI      @MYID,R0
      LDI      @NROWS,AR5          ; AR5 = Q = COLUMN COUNTER
      MPYI     AR5,R0              ; R0 = Q*MYNODE
      LSH      1,R0               ; R0 = 2*Q*MYNODE

      LDI      @MATR,AR7
      ADDI     R0,AR7              ; MATRIX POINTER = &MATRIX[2*Q*MYNODE][0]
      LDI      @BLOCK1,R7          ; POINTER TO DMA BUFFER
      LDI      @BLOCK0,AR6         ; POINTER TO FFT BUFFER

      ADDI     2,AR7,R0
      ADDI     1,R0,R1
      STI      R0,*+AR2(1)         ; SRC ADDRESS (Re PART)
      STI      R1,*+AR4(1)         ; SRC ADDRESS (Im PART)
      STI      R7,*+AR2(4)         ; DST ADDRESS (DMA_BUFFER)
      ADDI     1,R7,R0
      STI      R0,*+AR4(4)         ; DST ADDRESS (DMA_BUFFER+1)

      LDI      @DMA0,AR0           ; DMA START
      STI      AR2,*+AR0(6)
      STIK     0,*+AR0(3)
      LDI      @CTRL2,R4
      STI      R4,*AR0

*****
* CPU : - TRANSFER COLUMN 0 TO ON-CHIP RAM (BLOCK0)
*       - FFT ON COLUMN 0
*****
      SUBI3    2,AR3,RC           ; RC=N-2
      LDI      AR7,AR0            ; SOURCE ADDRESS
      LDI      AR6,AR1            ; DESTINATION ADDRESS
      RPTBD    LOOP3
      LSH3     1,AR3,IR1          ; SOURCE OFFSET = 2*N
      LDI      2,IR0              ; DESTINATION OFFSET
      LDF      *+AR0(1),R0        ; R0= X(I) IM

      LDF      *AR0++(IR1),R1     ; X(I) RE & POINTS TO X(I+1)
      ||      STF      R0,*+AR1(1) ; STORE X(I) IM
      LOOP3   LDF      *+AR0(1),R0 ; R0=X(I+1) IM
      ||      STF      R1,*AR1++(IR0) ; STORE X(I) RE

      LAJ      CFFT
      LDF      *AR0,R1            ; LOAD X(N-1) RE
      NOP
      STF      R0,*+AR1(1)        ; STORE X(N-1) IM
      ||      STF      R1,*AR1     ; STORE X(N-1) RE

*****
* DMA: - MOVES FFT COLUMN (I) (Re PART) TO OFF-CHIP RAM
*       - MOVES FFT COLUMN (I) (Im PART) TO OFF-CHIP RAM
*       - MOVES COLUMN (I+2) (Re PART) TO ON-CHIP RAM
*       - MOVES COLUMN (I+2) (Im PART) TO ON-CHIP RAM
*
* CPU: - FFT ON COLUMN (I+1)
*****

```

```

        LDI        @P02,AR1        ; DMA0
        LDI        @P01,AR0        ; P01
        SUBI       3,AR5           ; AR5=Q-3: (Q-2) DMA TRANSFERS
        ADDI       1H,AR6,R0
B4      TSTB       @MASK,IIF
        BZAT      B4

```

```

* DMA02: BIT-REVERSED TRANSFER OF LAST RESULT (Im)
        STI        R0,*+AR1(1)    ; SOURCE
        ADDI       1H,AR7,R0
        STI        R0,*+AR1(4)    ; DST

```

```

* DMA01: BIT-REVERSED TRANSFER OF COLUMN (Re)
        STI        AR6,*+AR0(1)   ; SOURCE
        STI        AR7,*+AR0(4)   ; DST

```

```

* DMA03: TRANSFER NEXT COLUMN (Re)
* DMA04: TRANSFER NEXT COLUMN (Im)

```

```

        LDI        @P03,AR4        ; AR0 POINTS TO DMA03
        LDI        @P04,AR2        ; AR1 POINTS TO DMA04
        ADDI       2,AR7           ; R0: POINTS TO NEXT COLUMN
        ADDI       2,AR7,R0
        STI        R0,*+AR4(1)    ; SOURCE: (RE)
        AND        0H,IIF         ; CLEAR FLAG
        STI        AR6,*+AR4(4)   ; DESTINATION: BLOCK1(RE)
        ADDI       1,R0           ; POINTS TO IMAGINARY PART
        STI        R0,*+AR2(1)    ; SOURCE: (IM)
        ADDI       1,AR6,R1
        STI        R1,*+AR2(4)    ; DESTINATION: BLOCK0(IM)
        LDI        @DMA0,AR1      ; GIVE THE START
        LDI        R7,R2
        LDI        AR6,R7         ; R7: BLOCK1
        STIK       0,*+AR1(3)
        STI        AR0,*+AR1(6)

```

```

* FFT ON CURRENT COLUMN
        LAJ        CFFT
        LDI        R2,AR6         ; AR6: POINTER FOR NEXT FFT
        LDI        @CTRL2,R0
        STI        R0,*AR1        ; START (DMA)
        DBUD       AR5,B4
        LDI        @P02,AR1      ; DMA0
        LDI        @P01,AR0      ; DMA0
        ADDI       1H,AR6,R0

```

```

*****
* DMA:   TRANSFER LAST FFT RESULT
* CPU:   FFT ON LAST COLUMN
*****

```

```

        LDI        @P02,AR1        ; DMA0
B5      TSTB       @MASK,IIF
        BZAT      B5

```

```

* DMA02: BIT-REVERSED TRANSFER OF LAST RESULT (Im)
    ADDI    1H,AR6,R0
    STI     R0,*+AR1(1)      ; SOURCE
    ADDI    1H,AR7,R0
    STI     R0,*+AR1(4)      ; DST
    LDI     @P01,AR0         ; P01
    LDI     @CTRL0,R0
    STI     R0,*AR1
    STI     AR6,*+AR0(1)     ; SOURCE
    STI     AR7,*+AR0(4)     ; DST
    LDI     @DMA0,AR1        ; GIVE THE START
    ADDI    2,AR7
    AND     0,IIF
    STIK    0,*+AR1(3)
    STI     AR0,*+AR1(6)
    LDI     @CTRL2,R0
    STI     R0,*AR1          ; START (DMA)

* FFT ON LAST COLUMN
    LAJ     CFFT
    LDI     R7,R2
    LDI     AR6,R7
    LDI     R2,AR6
    SUBI3   2,AR3,RC         ; RC=N-2
    LDI     AR6,AR0          ; SOURCE
    LDI     AR7,AR1          ; DESTINATION
    RPTBD   B6
    LD      AR3,IR0
    LSH3    1,AR3,IR1
    LDF     *+AR0(1),R0      ; R0= X(I) IM

* LOOP
    LDF     *AR0++(IR0)B,R1   ; X(I) RE & POINTS TO X(I+1)
    ||     STF     R0,*+AR1(1) ; STORE X(I) IM
B6        LDF     *+AR0(1),R0 ; R0=X(I+1) IM
    ||     STF     R1,*AR1++(IR1) ; STORE X(I) RE

* STORE LAST VALUE
B7        TSTB    @MASK,IIF
    BZ     B7
    LDF     *AR0++(IR0)B,R1   ; LOAD X(N-1) RE
    ||     STF     R0,*+AR1(1) ; STORE X(N-1) IM
    STF     R1,*AR1          ; STORE X(N-1) RE

    LDI     @TIMER,AR2        ; OPTIONAL: BENCHMARKING (TIME_READ)
    LDI     *+AR2(4),R0       ; TCOMP = R0

t2        B       t2
        .end

```


SHB.CMD

```
input.obj
shb.obj
spinput.obj
sintab.obj
synch.obj
-m shb.map
-lmylib.lib
```

```
/* SPECIFY THE SECTIONS ALLOCATION INTO MEMORY */
```

```
MEMORY
```

```
{
    ROM:      o = 0x00000000 l = 0x1000
    BUF0:     o = 0x002ff800 l = 0x200
    RAM0:     o = 0x002ffa00 l = 0x200
    BUF1:     o = 0x002ffc00 l = 0x200
    RAM1:     o = 0x002ffe00 l = 0x200
    LM:       o = 0x40000000 l = 0x10000
    GM:       o = 0x80000000 l = 0x20000
}
```

```
SECTIONS
```

```
{
    INPUT      :{} > GM      /* Input data          */
    .text      :{} > LM
    .data      :{} > RAM1    /* Sine table      */
    STACK      :{} > RAM1
    DMA_AUTOINI :{} > RAM1    /* DMA autoinit. values */
    SYNCH      :{} > GM      /* Synchronization */
}
```

Appendix C: Parallel 2-D FFT (Distributed-Memory Version)

C.1. DIS1.C: Distributed-Memory Implementation (C Program) — DMA Used Only for Interprocessor Communication

DIS1.C

```
/******
```

```
DIS1.C : Parallel 2-dimensional complex FFT (Distributed-memory)
        - single-buffered version
        - DMA is used only for interprocessor communication
```

Requirements: $P > 0$ $Q > 0$

To run:

```
cl30 -v40 -g -mr -o2 dis1.c
asm30 -v40 sintab.asm
asm30 -v40 input0.asm
asm30 -v40 input1.asm
lnk30 input0.obj dis1.obj disc.cmd -o a0.out
lnk30 input1.obj dis1.obj disc.cmd -o a1.out
```

Notes: 1) Before running, initialize the `my_node` variable to the corresponding value using the 'C40 emulator or an assembly file.

 2) Output: columnwise

```
*****/
#define SIZE      4                /* FFT size                */
#define LOGSIZE   2                /* log (FFT size)         */
#define P         2                /* number of processors   */
#define Q         SIZE/P           /* rows/cols. per processor */
#define BLOCK0    0x002ff800       /* on-chip buffer 0       */
#define DMA0      0x001000a0       /* DMA0 address           */
#define SWAP(x,y) temp = *x; *x = *y ; *y = temp;
#define WAIT_DMA(x) while ((0x03c00000 & *x)!=0x02800000)

extern void cfft(),                /* C-callable complex FFT */
          cmove(),                 /* CPU complex move       */
          cmoveb(),                /* CPU bit-reversed move  */
          exchange(),              /* set DMA in split mode  */
          set_dma();               /* set DMA register values */
extern float MATRIX[Q][SIZE*2];   /* input matrix           */

float *block0 = (float *)BLOCK0,
      *MM[Q], *ptr, temp;
int *dma0 = (int *)DMA0;
int my_node,
    q      = Q,
    q2     = Q*2,
    i,j,ii,i2,k1;
```

```

#if (P == 4)
    int port [P][P]= { 0,0,4,3,
                       3,0,0,4,
                       1,3,0,0,
                       0,1,3,0}; /*Connectivity matrix: processor i
                                   is connected to processor j through
                                   port[i][j]: system specific PPDS */
#else
    int port[P][P] = { 0,0,3,0}; /* when P=2 */
#endif

struct NODE {
    int id; /* dst_node ID*q2 */
    int port; /* port number to which is connected */
    int *dma; /* dma address attached to that port */
} dnode[P+1];

int tcomp;

/*****
main()
{
asm(" or 1800h,st");

for (i=0;i<Q;i++) MM[i]=MATRIX[i]; /* accessing assembly variables */

/***** FFT on rows *****/
t0:
    time_start(0); /* benchmarking (C40 timer) */
    cmove (&MM[0][0],block0,2,2,SIZE); /* move row 0 to on-chip RAM */
    cfftc (block0,SIZE,LOGSIZE); /* FFT on row 0 */
    cmoveb (block0,&MM[0][0],SIZE,2,SIZE); /* move back FFT(row 0) */
    for (j=1;j<P;j++) { /* interprocessor comm. */
        i= (my_node ^ j); /* destination node */
        dnode[j].id = (i * q2); /* destination node * q2 */
        dnode[j].port = port[my_node][i]; /* port to be used */
        dnode[j].dma = dma0 + (dnode[j].port <<4);
        exchange (dnode[j].dma,dnode[j].port,&MM[0][dnode[j].id],q2);
    }

    for (i=1;i<q;++i) { /* loop over other rows */
        cmove (&MM[i][0],block0,2,2,SIZE); /* move row i to on-chip RAM */
        cfftc (block0,SIZE,LOGSIZE); /* FFT on row i */
        cmoveb (block0,&MM[i][0],SIZE,2,SIZE); /* move back FFT (row i) */
        for (j=1;j<P;j++) { /* interprocessor comm. */
            WAIT_DMA(dnode[j].dma); /* wait for DMA to finish */
            exchange (dnode[j].dma,dnode[j].port,&MM[i][dnode[j].id],q2);
        }
    }
}

```

```

/***** FFT on columns *****/
for      (j=1;j<P;j++)    WAIT_DMA(dnode[j].dma); /* wait for DMAs to finish*/

t1:
for  (i=0;i<(q-1);i++) {
    ptr = &MM[i][i*2]; k1 = 2*(q-i);
    for  (j=0;j<P;j++,ptr +=q2)
        for  (ii=2;ii<k1;ii+=2) {
            SWAP((ptr+ii),(ptr+ii*SIZE)); /* exchange Re parts */
            SWAP((ptr+ii+1),(ptr+ii*SIZE+1)); /* exchange Im parts */
        }
    cmove (&MM[i][0],block0,2,2,SIZE); /* FFT on column (i-1) */
    cfft (&block0,SIZE,LOGSIZE);
    cmoveb (block0,&MM[i][0],SIZE,2,SIZE);

} /*for*/
    cmove (&MM[q-1][0],block0,2,2,SIZE); /* FFT on last column */
    cfft (&block0,SIZE,LOGSIZE);
    cmoveb (block0,&MM[q-1][0],SIZE,2,SIZE);

tcomp =time_read(0); /* benchmarking */
t2: ;
} /*main*/

```

DISC.CMD

```

sintab.obj
-c
-stack 0x0040
-lrts40.lib
-lprts40r.lib
-lmylib.lib
-m disc.map

/* SPECIFY THE SYSTEM MEMORY MAP */

MEMORY
{
    ROM:      org = 0x00      len = 0x0800
    BUF0:     org = 0x002ff800 len = 0x0400      /* on-chip RAM block 0 */
    RAM1:     org = 0x002ffc00 len = 0x0400      /* on-chip RAM block 1 */
    LM:       org = 0x40000000 len = 0x10000     /* LOCAL MEMORY */
    GM:       org = 0x80000000 len = 0x20000     /* GLOBAL MEMORY */
}

/* SPECIFY THE SECTIONS ALLOCATION INTO MEMORY */

SECTIONS
{
    INPUT:    {} > LM          /* INPUT MATRIX */
    .text:    {} > LM
    .cinit:   {} > RAM1        /* INITIALIZATION TABLES */
    .stack:   {} > RAM1        /* SYSTEM STACK */
    .bss :    {} > RAM1
    .data:    {} > RAM1        /* SINE TABLE */
}

```

INPUT0.ASM

```
*****
*
* INPUT0.ASM : input matrix 2 x 4 (Distributed-memory program) for processor 0
*              -number of processors in the system: 2
*
*****

.global  _MATRIX
.sect    "INPUT"

_MATRIX

.float 130.0,90.0      ;[0][0]
.float 66.0,230.0      ;[0][1]
.float 205.0,136.0     ;[0][2]
.float 15.0,187.0      ;[0][3]
.float 150.0,164.0     ;[1][0]
.float 222.0,44.0      ;[1][1]
.float 95.0,243.0      ;[1][2]
.float 80.0,60.0       ;[1][3]
.end
```

INPUT1.ASM

```
*****
*
* INPUT1.ASM : input matrix 2 x 4 (Distributed-memory program) for processor 1
*              -number of processors in the system: 2
*
*****

.global  _MATRIX
.sect    "INPUT"

_MATRIX

.float 97.0,36.0       ;[2][0]
.float 215.0,191.0     ;[2][1]
.float 209.0,239.0     ;[2][2]
.float 161.0,22.0      ;[2][3]
.float 117.0,238.0     ;[3][0]
.float 203.0,44.0      ;[3][1]
.float 104.0,187.0     ;[3][2]
.float 195.0,177.0     ;[3][3]
.end
```

C.2. DIS2.C: Distributed-Memory Implementation (C Program) — DMA Used for Interprocessor Communication and Matrix Transposition

DIS2.C

```

/*****

DIS2.C :          Parallel 2-dimensional complex FFT (Distributed-memory)
                - single-buffered version
                - DMA is used for interprocessor communication and matrix
                  transposition
Requirements:    P > 0 ; Q > 0
To run:
    cl30 -v40 -g -mr -o2 dis2.c
    asm30 -v40 sintab.asm
    asm30 -v40 input0.asm
    asm30 -v40 input1.asm
    lnk30 input0.obj dis2.obj disc.cmd -o a0.out
    lnk30 input1.obj dis2.obj disc.cmd -o a1.out
Notes:  1) Before running, initialize the my_node variable to the corresponding value
        using the 'C40 emulator or an assembly file.

        2) Output: columnwise

*****/

```

```

*****/
#define SIZE      4          /* FFT size */
#define LOGSIZE   2          /* log (FFT size) */
#define P         2          /* number of processors */
#define Q         SIZE/P     /* rows/cols. per processor */

#define BLOCK0    0x002ff800 /* on-chip buffer 0 */
#define DMA0      0x001000a0 /* DMA0 address */
#define SWAP(x,y) temp = *x; *x = *y ; *y = temp;
#define WAIT_DMA(x) while ((0x03c00000 & *x)!=0x02800000)

extern void      cfft(),      /* C-callable complex FFT */
               cmove(),      /* CPU complex move */
               cmoveb(),     /* CPU bit-reversed move */
               exchange(),   /* set DMA in split mode */
               set_dma();    /* set DMA register values */

extern float     MATRIX[Q][SIZE*2]; /* input matrix */
int MEM[35*P]; /* autoinitialization values:
               5 set of different values
               per processor (5*7) */

float *block0 = (float *)BLOCK0,
      array[Q*2], *MM[Q], *ptr, temp;

int *dma0 = (int *)DMA0,
    *dma[P],
    *mem = MEM,
    ctrl2 = 0x00c00008, /* autoinit,TCC=0,DMA low pr. */
    ctrl3 = 0x00c40004, /* no autoinit,TCC=1,DMA low pr. */
    mask = 0x02000000,
    *mp;

```

```

int      my_node,
        size2 = SIZE*2,
        q = Q,
        q2 = Q*2,
        i,ii,j,i2,k1,k2;
#if
int      port[P][P] = { 0,0,4,3,
                        3,0,0,4,
                        1,3,0,0,
                        0,1,3,0 }; /* Connectivity matrix: processor i
                                   is connected to processor j through
                                   port[i][j]:system specific(PPDS) */

#else

int      port[P][P] = { 0,0,3,0}; /* when P = 2 */
#endif

struct NODE {
    int id; /* will keep (destination node ID* q2) */
    int port; /* port number to which dst node is connected */
    int *dma; /* dma address to be used with port */
} dnode[P+1];

int      tcomp;

/*****
main()
{
asm(" or 1800h,st"); /* cache enable */

for (i=0;i<Q;i++) MM[i]=MATRIX[i]; /* accessing assembly vars */

/***** FFT on rows *****/
t0:
time_start(0); /* benchmarking */
cmovb (&MM[0][0],block0,2,2,SIZE); /* move row 0 to on-chip RAM */
cfftcb (block0,SIZE,LOGSIZE); /* FFT on row 0 */
cmovb (block0,&MM[0][0],SIZE,2,SIZE); /* move back FFT(row 0) */
for (j=1;j<P;j++) { /* interprocessor comm. */
    i = (my_node ^ j); /* destination node */
    dnode[j].id = (i * q2); /* destination node * q2 */
    dnode[j].port = port[my_node][i]; /* port to be used */
    dnode[j].dma = dma0 + (dnode[j].port <<4); /* dma to be used */
    exchange (dnode[j].dma,dnode[j].port,&MM[0][dnode[j].id],q2);
}

for (i=1;i<q;i++) { /* loop over other rows */
    cmovb (&MM[i][0],block0,2,2,SIZE); /* move row i to on-chip RAM */
    cfftcb (block0,SIZE,LOGSIZE); /* FFT on row i */
    cmovb (block0,&MM[i][0],SIZE,2,SIZE); /* move back FFT (row i) */
    for (j=1;j<P;j++) { /* interprocessor comm. */
        WAIT_DMA(dnode[j].dma); /* wait for DMA to finish */
        exchange (dnode[j].dma,dnode[j].port,&MM[i][dnode[j].id],q2);
    }
}
}

```

```

/***** FFT on columns *****/

for      (j=1;j<P;j++) WAIT_DMA(dnode[j].dma)    /* wait for DMAs in split mode to
                                                    finish */

t1: if (q==1) goto lastcol;                      /* no need for transposition */

ptr=&MM[0][0];

for      (j=0;j<P;j++,ptr +=q2)                  /* transposition of row/col 0 */
for      (ii=2;ii<q2;ii+=2) {
    SWAP((ptr+ii),(ptr+ii*SIZE));                  /* Re part */
    SWAP((ptr+ii+1),(ptr+ii*SIZE+1));              /* Im part */
}

/* DMA0 transposes column/row 1 :
[1]: row(Re,Im) -> array(Re,Im)
[2.a]: col(Re) -> row(Re)    [2.b]: col(Im) -> row(Im)
[3.a]: array(Re) -> col(Re)  [3.b]: array(Im) -> col(Im) */

mp = mem;          ptr = &MM[1][2];

for      (j=0;j<(P-1);j++) {
    set_dma(mp,ctrl2,ptr,1,(q2-2),array,1,(mp+7));
    set_dma((mp+7),ctrl2,ptr,size2,(q-1),ptr,2,(mp+14));
    set_dma((mp+14),ctrl2,(ptr+1),size2,(q-1),(ptr+1),2,(mp+21));
    set_dma((mp+21),ctrl2,array,2,(q-1),ptr,size2,(mp+28));
    set_dma((mp+28),ctrl2,(array+1),2,(q-1),(ptr+1),size2,(mp+35));
    mp += 35; ptr += q2;
}

set_dma(mp,ctrl2,ptr,1,(q2-2),array,1,(mp+7));
set_dma((mp+7),ctrl2,ptr,size2,(q-1),ptr,2,(mp+14));
set_dma((mp+14),ctrl2,(ptr+1),size2,(q-1),(ptr+1),2,(mp+21));
set_dma((mp+21),ctrl2,array,2,(q-1),ptr,size2,(mp+28));
set_dma((mp+28),ctrl3,(array+1),2,(q-1),(ptr+1),size2,0);
*(dma0+3) = 0; *(dma0+6) =(int)mem; *dma0=ctrl2;

cmove (&MM[0][0],block0,2,2,SIZE);                /* move column 0 to on-chip */
cfftcb(block0,SIZE,LOGSIZE);                        /* FFT on column 0 */
cmoveb (block0,&MM[0][0],SIZE,2,SIZE);              /* move FFT column 0 off-chip */

for      (i=2;i<q-1;i++) {
    i2=2*i;
                                /* Check IIF register to see if DMA0
                                (unified mode) has finished */
    asm("WAIT:  TSTB @_mask,iif");
    asm(" BZAT WAIT");
    asm(" ANDN @_mask,iif");
}

```



```

mp      = mem;      ptr = &MM[i][i2]; k1 = (q-i); k2 =q2-i2;
if      (k1>1) {
for (j=0;j<P;j++) {      /* DMA transposes row/column i */
    *(mp+1) = *(mp+8) = *(mp+11) = *(mp+25) = (int)ptr;
    *(mp+15) = *(mp+18) = *(mp+32) = (int)(ptr+1) ;
    /* counter */
    *(mp+10) = *(mp+17) = *(mp+24) = *(mp+31) =k1;
    *(mp+3) = k2;
    mp += 35; ptr += q2; /* points to next submatrix */
}
    *(dma0+3) = 0; *(dma0+6) = (int)mem; *dma0=ctrl2;
}/* if */

cmove (&MM[i-1][0],block0,2,2,SIZE);
cfft (block0,SIZE,LOGSIZE); /* FFT on column (i-1) */
cmovb (block0,&MM[i-1][0],SIZE,2,SIZE);
}/* for */

lastcol:

asm("WAIT2: TSTB @_mask,iif");
asm(" BZAT WAIT2");
asm(" ANDN @_mask,iif");

cmove (&MM[q-2][0],block0,2,2,SIZE);
cfft (block0,SIZE,LOGSIZE); /* FFT on column (q-2) */
cmovb (block0,&MM[q-2][0],SIZE,2,SIZE);
cmove (&MM[q-1][0],block0,2,2,SIZE);
cfft (block0,SIZE,LOGSIZE); /* FFT on last column */
cmovb (block0,&MM[q-1][0],SIZE,2,SIZE);

tcomp =time_read(0); /* Optional: Benchmarking */

t2: ;

}/* main */

```

C.3. DIS2.ASM: Distributed-Memory ('C40 Assembly Program) — DMA Used for Interprocessor Communication and Matrix Transposition

DIS2.ASM

```
*****
*
*   DIS2.ASM :   TMS320C40 Parallel   2-dimensional complex FFT
*               - distributed-memory single-buffered version
*               - DMAs are used for interprocessor communication
*               and for matrix transposition
*
*   Routines used: cfft.asm (complex FFT)
*
*   Requirements :      Number of processors = P > 1
*                      Rows/columns per processor = Q >= 4
*
*   To run:
*
*   asm30 -v40 -g -s dis2.asm
*   asm30 -v40 -g -s dpinput.asm
*   asm30 -v40 -g -s ssintab.asm
*   asm30 -v40 -g -s 0.asm
*   asm30 -v40 -g -s 1.asm
*   asm30 -v40 -g -s input0.asm
*   asm30 -v40 -g -s input1.asm
*   lnk30 dis.cmd 0.obj input0.obj -o a0.out
*   lnk30 dis.cmd 1.obj input1.obj -o a1.out
*
*****/

.global N                ; fft size
.global P                ; number of processors
.global Q                ; rows/columns per processor
.global MYNODE           ; processor ID
.global _PORT            ; port matrix address
.global _MATRIX          ; input matrix address
.global _DMAMEM          ; memory address for autoinitialization values
.global _DMALIST         ; space reserved to store addresses of the
                        ; DMAs used for interprocessor communication
.global _CTRLIST         ; space for control register values
                        ; for DMAs used for interprocessor comm.
.global _DSTQLIST        ; space reserved to store (dst_node*q) values
                        ; to determine the source address for each
                        ; DMA interprocessor communication
.global _ARRAY           ; buffer to be used in matrix transposition
                        ; using DMA
.global CFFT             ; 1d-fft subroutine
.global C2DFFT           ; entry point for execution

_STACK .usect "STACK",10h
.text
```

```

FFTSIZE .word N
PROC .word P
NROWS .word Q
MYID .word MYNODE
PORT .word _PORT
MATR .word MATRIX
BLOCK0 .word 002FF800H ; ram block 0
STACK .word _STACK ; stack address
DMA0 .word 001000a0H ; DMA0 address
DMALIST .word _DMALIST
CTRLIST .word _CTRLIST
DSTQLIST .word _DSTQLIST
DMAMEM .word _DMAMEM ; pointer to autoinit. values in memory
SMASK .word 02000000H ; to check if DMA unified mode has finished
; using the IIF register
DMAMASK .word 03C00000H ; to check if DMA in split mode has finished
; using the start fields in the DMAs control
; register

SPLITD .word 02800000H
CTRL2 .word 00C00008H ; control register word: autoinit. , TCC = 0
CTRL3 .word 00C40004H ; control register word: no autoinit., TCC = 1
CONTROL .word 03C040D4H ; control register word: split mode for
; interprocessor communication

ARRAY .word _ARRAY
PORT0 .word 00000000H ; this values help to set DMA control registers
PORT1 .word 00008000H ; with the corresponding port values for the
PORT2 .word 00010000H ; port field
PORT3 .word 00018000H
PORT4 .word 00020000H
PORT5 .word 00028000H
PORTS .word PORT0
ENABLE .word 24924955H ; enable port interrupts to DMAs
TIMER .word 0100020h ; Timer 0 address (benchmarking)

C2DFFT LDP FFTSIZE ; load data page pointer
LDI 2,IR0 ; destination offset
LDI 2,IR1 ; source offset
LDI @STACK,SP ; initialize the stack pointer

t0:
LDI @TIMER,AR2 ; Optional: benchmarking : timer start
STIK -1,*+AR2(8)
LDI 961,R0
STI R0,*AR2
OR 9800h,ST ; cache enable and set condition flag =1
; (to enable any primary register to modify
; condition flags)
LDI @FFTSIZE,AR3 ; ar3 = n = matrix size
LDI @NROWS,R7 ; r7 = n/p = q = rows/columns per processor
LDI @MATR,AR7 ; initialize matrix pointer
LDI @BLOCK0,AR6 ; ar6: pointer to the on-chip RAM block that
; contains the input data for FFT computation

```

```

*****
*          FFT ON ROWS
*****

*****
*   CPU MOVES ROW 0   *
*   TO ON-CHIP RAM   *
*****

          SUBI3      2,AR3,RC          ; rc = n-2
          RPTBD      LOOP0
          LDI         AR7,AR0          ; source address = row 0 = & x(0)
          LDI         AR6,AR1          ; destination address
          LDF         *+AR0(1),R0      ; R0 = x(i) Im

          LDF         *AR0++(IR1),R1   ; x(i) Re & points to x(i+1)
||      STF         R0, *+AR1(1)      ; store x(i) Im
LOOP0   LDF         *+AR0(1),R0      ; R0 = x(i+1) Im
||      STF         R1,*AR1++(IR0)    ; store x(i) Re

*****
*   FFT ON ROW 0     *
*****

          LAJ         CFFT              ; call 1d-fft routine (complex FFT)
          LDF         *AR0,R1          ; LOAD X(N-1) RE
          STF         R0,*+AR1(1)      ; STORE X(N-1) IM
          STF         R1,*AR1          ; STORE X(N-1) RE

*****
*   CPU MOVES ROW 0   *
*   (BIT-REVERSED) TO *
*   EXTERNAL MEMORY   *
*****

          LDI         AR6,AR0          ; SOURCE
          LDI         AR7,AR1          ; DESTINATION
          SUBI3      2,AR3,RC
          RPTBD      LOOP1
          LDI         AR3,IR0          ; SOURCE OFFSET FOR BIT-REVERSE = N
          LDI         2,IR1           ; DESTINATION OFFSET
          LDF         *+AR0(1),R0
          LDF         *AR0++(IR0)B,R1
||      STF         R0,*+AR1(1)
LOOP1   LDF         *+AR0(1),R0
||      STF         R1,*AR1++(IR1)
||      LDF         *AR0++(IR0)B,R1
||      STF         R0,*+AR1(1)
||      STF         R1,*AR1++(IR1)

```

```

*****
*   INTERPROCESSOR   *
*   COMMUNICATION    *
*   (DMA)            *
*****

        LDI        @DMALIST,AR4    ; keeps dma addresses
        LDI        @CTRLIST,AR5    ; keeps dma control registers
                                      ; according to port attached
        LDI        @DSTQLIST,AR6    ; keeps pointer equal to (dest_node* q2)
        LDI        @MYID,R4        ; my node-id
        LDI        @PROC,R3        ; number of processors
        LDI        @NROWS,R7       ; q = (N/P)
        LSH        1,R7,R2        ; r2= 2*q
        MPYI       R3,R4,AR0       ; P*mynode
        ADDI       @PORT,AR0       ; &port[mynode][0]
        LDI        @ENABLE,DIE     ; enable port interrupts to all DMAs
        SUBI       1,R3,IR0        ; j = loop counter = (P-1)

LOOP2
        XOR        IR0,R4,IR1      ; destination node = mynode ^ j
        MPYI       R2,IR1,R0       ; destination node * q2
        STI        R0,*+AR6(IR0)   ; DSTQLIST update
        ADDI       AR7,R0          ; pointer to matrix location to transfer
        LDI        *+AR0(IR1),AR2   ; port[mynode][dest_node]
        MPYI       16,AR2,AR1      ;
        ADDI       @DMA0,AR1       ; DMA address
        STI        AR1,*+AR4(IR0)   ; DMALIST update
        STI        R0,*+AR1(1)     ; src primary channel
        STI        R0,*+AR1(4)     ; dst secondary channel
        STI        R2,*+AR1(3)     ; counter primary channel
        STI        R2,*+AR1(7)     ; counter secondary channel
        ADDI       @PORTS,AR2      ;
        LDI        @CONTROL,R0     ;
        OR         *AR2,R0         ; DMA control register
        STI        R0,*+AR5(IR0)   ; CTRLIST update
        SUBI       1,IR0
        BNZD       LOOP2
        STIK       1,*+AR1(2)      ; src index primary channel
        STIK       1,*+AR1(5)      ; src index secondary channel
        STI        R0,*AR1         ; DMA start

*****
*
*   (Q-1) ROWS
*
*****

*****
*   CPU MOVES ROW I   *
*   TO ON-CHIP RAM    *
*****

        LSH3       1,AR3,R0
        LDI        @BLOCK0,AR6
        ADDI       R0,AR7          ; AR7 POINTS TO ROW 1
        SUBI       2,R7,AR5        ; AR5 = Q-2
        LDI        AR7,AR0         ; SOURCE
        LDI        2,IR1
        SUBI3      2,AR3,RC        ; RC = N-2

```

```

LOOPR   RPTBD   LOOP3
        LDI     AR6,AR1           ; DESTINATION
        LDI     2,IR0            ; DESTINATION OFFSET
        LDF     *+AR0(1),R0      ; R0 = X(I) IM
        LDF     *AR0++(IR1),R1   ; X(I) RE & POINTS TO X(I+1)
||      STF     R0, *+AR1(1)      ; STORE X(I) IM
LOOP3   LDF     *+AR0(1),R0      ; R0 = X(I+1) IM
||      STF     R1,*AR1++(IR0)    ; STORE X(I) RE

*****
*       FFT ON ROW I       *
*****

        LAJ     CFFT             ; CALL 1D-FFT (COMPLEX)
        LDF     *AR0,R1          ; LOAD X(N-1) RE
        STF     R0,*+AR1(1)      ; STORE X(N-1) IM
        STF     R1,*AR1          ; STORE X(N-1) RE

*****
*       CPU MOVES ROW I   *
*       (BIT-REVERSED) TO *
*       EXTERNAL MEMORY   *
*****

        LDI     AR6,AR0          ; SOURCE
        LDI     AR7,AR1          ; DESTINATION
        SUBI3   2,AR3,RC
        RPTBD   LOOP4
        LDI     AR3,IR0          ; SOURCE OFFSET FOR BIT-REVERSE = N
        LDI     2,IR1            ; DESTINATION OFFSET
        LDF     *+AR0(1),R0
        LDF     *AR0++(IR0)B,R1
||      STF     R0,*+AR1(1)
LOOP4   LDF     *+AR0(1),R0
||      STF     R1,*AR1++(IR1)
        LDF     *AR0++(IR0)B,R1
||      STF     R0,*+AR1(1)
        STF     R1,*AR1++(IR1)

*****
*       WAIT FOR DMAS     *
*       TO FINISH         *
*****

        * DMAS DONE
        LDI     @DMALIST,AR4      ; POINTS TO DMALIST
        LDI     @PROC,R3         ; R3 = NUM OF PROCESSORS
        ADDI    R3,AR4
        SUBI    2,R3,RC
        RPTBD   LLP
        LDI     @SPLITD,R0
        LDI     @DMAMASK,R1
        SUBI    1,AR4,AR0        ; AR0 = POINTS TO DMA[0]
AGAINP  LDI     *AR0--(1),AR2
        AND     *AR2,R1,R4
        XOR     R0,R4            ; =0 IF DMA FINISH
LLP     BNZ     AGAINP

```

```

*****
*   INTERPROCESSOR   *
*   COMMUNICATION    *
*   (DMA)            *
*****

        LDI        @DMALIST,AR0
        LDI        @CTRLIST,AR1
        LDI        @DSTQLIST,AR2
        LDI        @NROWS,R2
        LSH        1,R2          ; R2 = Q2
        SUBI       1,R3,IR0

LOOP5   LDI        *+AR0(IR0),AR4 ; DMA ADDRESS
        LDI        *+AR2(IR0),R6  ; (DSTNODE*Q2)
        ADDI       AR7,R6         ; POINTS TO MATRIX LOCATION TO TRANSFER
        STI        R6,*+AR4(1)    ; SOURCE PRIMARY CHANNEL
        STI        R6,*+AR4(4)    ; SOURCE SECONDARY CHANNEL
        LDI        *+AR1(IR0),R0
        SUBI       1,IR0
        BNZD       LOOP5
        STI        R2,*+AR4(3)    ; PRIMARY COUNTER = Q2
        STI        R2,*+AR4(7)    ; SECONDARY COUNTER = Q2
        STI        R0,*AR4
        LSH3       1,AR3,R0
        ADDI       R0,AR7
        DBUD       AR5,LOOPR
        LDI        AR7,AR0        ; SOURCE
        LDI        2,IR1         ; SOURCE OFFSET
        SUBI3      2,AR3,RC

*****
*                               *
*   FFT ON COLUMNS            *
*                               *
*****

*****
*   WAIT FOR DMAS             *
*   TO FINISH                 *
*****

        LDI        @DMALIST,AR4   ; POINTS TO DMALIST
        LDI        @PROC,R3       ; R3 = NUM OF PROCESSORS
        ADDI       R3,AR4
        SUBI       2,R3,RC
        RPTBD      LLN
        LDI        @SPLITD,R0
        LDI        @DMAMASK,R1
        SUBI       1,AR4,AR0      ; AR0 = POINTS TO DMA[0]
        LDI        *AR0--(1),AR2
AGAINN  AND        *AR2,R1,R4
        XOR        R0,R4          ; =0 IF DMA FINISH
        LLN        BNZ          AGAINN

```

```
*****
* CPU TRANSPOSITION *
*   ROW 0           *
*****
```

t1:

```

      SUBI      1,R3,AR5
      LDI       @MATR,AR7      ; INITIALIZE POINTER TO COL 0
      LSH3      1,AR3,IR0      ; IR0 = 2N
      MPYI      AR5,R2,R4      ; R4 = (P-1)*Q2
      ADDI      AR7,R4         ; R4 = PTR
      ADDI      R2,R4
      SUBI3     2,R7,RC        ; RC = Q-2

LOOP10  RPTBD    LOOP11
      SUBI      R2,R4
      ADDI      2,R4,AR0      ; AR0 = PTR + 2
      ADDI      IR0,R4,AR1    ; AR1 = PTR + 2*SIZE
      LDF       *+AR0(1),R0   ; R0 =IM
      || LDF     *+AR1(1),R6   ; R6 =IM
      STF       R0, *+AR1(1)
      || STF     R6, *+AR0(1)
      LDF       *AR0,R0
      || LDF     *AR1,R6
LOOP11  STF     R0, *AR1++(IR0)
      || STF     R6, *AR0++(IR1)
      DBUD      AR5,LOOP10
      SUBI3     2,R7,RC
      LDI       @CTRL2,R3
      LDI       @DMAMEM,AR4   ; AR4 = MP

```

```
*****
* DMA TRANSPOSITION *
*   ROW 1           *
*****
```

```

      SUBI      1,R7,R4      ; R4 = Q-1
      LDI       @DMA0,AR0
      LSH3      1,AR3,AR2    ; AR2 = 2N
      ADDI      AR7,AR2,R6    ; R6 = M[1]
      ADDI      2, R6        ; R6 = M[1][2] = FI
      ADDI      1,R6,R10     ; FI+1
      STI       AR4,*+AR0(6)
      STIK      0,*+AR0(3)
      LDI       @PROC,RC
      SUBI      2,RC ; LOOP (P-1) TIMES
      RPTBD     TROW1
      SUBI      2,R2,R8      ; R8 = Q2-2
      LDI       @ARRAY,R5
      ADDI      1,R5,R9      ; ARRAY-1

```

* MP

```

      STI       R6,*+AR4(1)   ; SOURCE
      STIK      1,*+AR4(2)   ; SRC INDEX
      STI       R8,*+AR4(3)   ; COUNTER
      STI       R5,*+AR4(4)   ; ARRAY
      STIK      1,*+AR4(5)   ; DST INDEX
      STI       R3,*+AR4++(7) ; CTRL
      STI       AR4,*-AR4(1)  ; LINK POINTER

```



```

* MP+7
    STI      R6, *+AR4(1)      ; SOURCE
    STI      AR2, *+AR4(2)     ; SRC INDEX
    STI      R4, *+AR4(3)     ; COUNTER
    STI      R6, *+AR4(4)     ; DST
    STIK     2, *+AR4(5)      ; DST INDEX
    STI      R3, *AR4++(7)     ; CTRL
    STI      AR4, *-AR4(1)    ; LINK POINTER

* MP+14
    STI      R10, *+AR4(1)     ; SOURCE
    STI      AR2, *+AR4(2)     ; SRC INDEX
    STI      R4, *+AR4(3)     ; COUNTER
    STI      R10, *+AR4(4)    ; DST
    STIK     2, *+AR4(5)      ; DST INDEX
    STI      R3, *AR4++(7)     ; CTRL
    STI      AR4, *-AR4(1)    ; LINK POINTER

* MP+21
    STI      R5, *+AR4(1)     ; SOURCE
    STIK     2, *+AR4(2)      ; SRC INDEX
    STI      R4, *+AR4(3)     ; COUNTER
    STI      R6, *+AR4(4)     ; DST
    STI      AR2, *+AR4(5)     ; DST INDEX
    STI      R3, *AR4++(7)     ; CTRL
    STI      AR4, *-AR4(1)    ; LINK POINTER

* MP+28
    STI      R9, *+AR4(1)     ; SOURCE
    STIK     2, *+AR4(2)      ; SRC INDEX
    STI      R4, *+AR4(3)     ; COUNTER
    STI      R10, *+AR4(4)    ; DST
    STI      AR2, *+AR4(5)     ; DST INDEX
    STI      R3, *AR4++(7)     ; CTRL
    STI      AR4, *-AR4(1)    ; LINK POINTER

TROW1    ADDI      R2, R6
          ADDI      1, R6, R10      ; FI+1

* MP
    STI      R6, *+AR4(1)     ; SOURCE
    STIK     1, *+AR4(2)      ; SRC INDEX
    STI      R8, *+AR4(3)     ; COUNTER
    STI      R5, *+AR4(4)     ; ARRAY
    STIK     1, *+AR4(5)      ; DST INDEX
    STI      R3, *AR4++(7)     ; CTRL
    STI      AR4, *-AR4(1)    ; LINK POINTER

* MP+7
    STI      R6, *+AR4(1)     ; SOURCE
    STI      AR2, *+AR4(2)     ; SRC INDEX
    STI      R4, *+AR4(3)     ; COUNTER
    STI      R6, *+AR4(4)     ; DST
    STIK     2, *+AR4(5)      ; DST INDEX
    STI      R3, *AR4++(7)     ; CTRL
    STI      AR4, *-AR4(1)    ; LINK POINTER

```

```

* MP+14
    STI      R10,*+AR4(1)      ; SOURCE
    STI      AR2,*+AR4(2)      ; SRC INDEX
    STI      R4,*+AR4(3)      ; COUNTER
    STI      R10,*+AR4(4)      ; DST
    STIK     2,*+AR4(5)        ; DST INDEX
    STI      R3,*AR4++(7)      ; CTRL
    STI      AR4,*-AR4(1)      ; LINK POINTER

* MP+21
    STI      R5,*+AR4(1)      ; SOURCE
    STIK     2,*+AR4(2)        ; SRC INDEX
    STI      R4,*+AR4(3)      ; COUNTER
    STI      R6,*+AR4(4)      ; DST
    STI      AR2,*+AR4(5)      ; DST INDEX
    STI      R3,*AR4++(7)      ; CTRL
    STI      AR4,*-AR4(1)      ; LINK POINTER

* MP+28
    STI      R9,*+AR4(1)      ; SOURCE
    STIK     2,*+AR4(2)        ; SRC INDEX
    STI      R4,*+AR4(3)      ; COUNTER
    STI      R10,*+AR4(4)      ; DST
    STI      AR2,*+AR4(5)      ; DST INDEX
    LDI      @CTRL3,R0
    STI      R0,*AR4++(7)      ; CTRL
    STI      R3,*AR0           ; START DMA

*****
*   CPU MOVES COL 0   *
*   TO ON-CHIP RAM   *
*****

COLUMN0:
    LDI      AR7,AR0           ; SOURCE : AR7 : POINTS TO COL 0
    LDI      AR6,AR1           ; DESTINATION
    SUBI3    2,AR3,RC          ; RC=N-2
    RPTBD    LOOP8
    LDI      2,IR1             ; SOURCE OFFSET
    LDI      2,IR0             ; DESTINATION OFFSET
    LDF      *+AR0(1),R0        ; R0= X(I) IM
    LDF      *AR0++(IR1),R1     ; X(I) RE & POINTS TO X(I+1)
    ||      STF      R0,*+AR1(1) ; STORE X(I) IM
LOOP8      LDF      *+AR0(1),R0 ; R0=X(I+1) IM
    ||      STF      R1,*AR1++(IR0) ; STORE X(I) RE

*****
*   FFT ON COL 0     *
*****

    LAJ      CFFT
    LDF      *AR0,R1           ; LOAD X(N-1) RE
    STF      R0,*+AR1(1)       ; STORE X(N-1) IM
    STF      R1,*AR1           ; STORE X(N-1) RE

```

```

*****
*   FFT MOVES COL. 0   *
*   (BIT-REVERSED) TO *
*   EXTERNAL MEMORY   *
*****

        LDI      AR6,AR0          ; SOURCE = BLOCK0
        LDI      AR7,AR1          ; DESTINATION = MATRIX
        SUBI3    2,AR3,RC         ; RC=N-2
        RPTBD    LOOP9
        LDI      AR3,IR0          ; SOURCE OFFSET = IR0 = N (BIT-REVERSE)
        LDI      2,IR1           ; DESTINATION OFFSET (COLUMNS) = IR1 = 2N
        LDF      *+AR0(1),R0
        LDF      *AR0++(IR0)B,R1
||      STF      R0,*+AR1(1)

LOOP9   LDF      *+AR0(1),R0
||      STF      R1,*AR1++(IR1)
        LDF      *AR0++(IR0)B,R1
||      STF      R0,*+AR1(1)
        STF      R1,*AR1++(IR1)
        LSH3     1,AR3,R0
        ADDI     R0,AR7          ; AR7 POINTS TO COL 1

*****
*   (Q-2) COLUMNS
*****

        CMPI     2,R7            ; if Q=2 goto last column
        BZ       LASTCOL

*****
*   WAIT FOR DMAS     *
*   TO FINISH         *
*****

        LDI      2,AR5          ; AR5 = I

WAIT     TSTB     @SMASK,IIF
        BZAT     WAIT
        ANDN     @SMASK,IIF
        LDI      @DMA0,AR0
        LSH3     1,R7,R2        ; R2 = Q2
        ADDI     R0,AR7,R6      ; R6 = &M[2]
        LSH3     1,AR5,R0       ; R0 = I2
        ADDI     R0,R6          ; R6 = PTR
        LDI      @DMAMEM,AR4
        STI      AR4,*+AR0(6)
        STIK     0,*+AR0(3)

```

```
*****
*   TRANSPOSE ALL P   *
*   SECTIONS OF ROW I *
*****
```

```

LDI      @PROC,R0
SUBI     1,R0,RC          ; LOOP P TIMES
RPTBD    TRANSP
SUBI     AR5,R7,R4        ; R4 = Q-I
LSH3     1,R4,R3          ; R3 = Q2-I2
ADDI     1,R6,R10         ; R10= PTR+1
STI      R6,*+AR4(1)      ; PTR
STI      R6,*+AR4(8)
STI      R6,*+AR4(11)
STI      R6,*+AR4(25)
STI      R10,*+AR4(15)    ; PTR+1
STI      R10,*+AR4(18)
STI      R10,*+AR4(32)
STI      R4,*+AR4(10)     ; Q-I
STI      R4,*+AR4(17)
STI      R4,*+AR4(24)
STI      R4,*+AR4(31)
STI      R3,*+AR4(3)      ; Q2-I2
ADDI     35,AR4           ; MP+=35
ADDI     R2,R6            ; FI+=Q2

TRANSP   ADDI     1,R6,R10
LDI      @CTRL2,R0
STI      R0,*AR0          ; START DMA

```

```
*****
*   CPU MOVES COL I   *
*   TO ON-CHIP RAM    *
*****
```

```

SUBI3    2,AR3,RC          ; RC=N-2
LDI      AR7,AR0           ; SOURCE
LDI      AR6,AR1           ; DESTINATION
RPTBD    LOOP18
LDI      2,IR1             ; SOURCE OFFSET = 2N
LDI      2,IR0             ; DESTINATION OFFSET
LDF      *+AR0(1),R0       ; R0= X(I) IM
LDF      *AR0++(IR1),R1    ; X(I) RE & POINTS TO X(I+1)
||      STF      R0,*+AR1(1) ; STORE X(I) IM

LOOP18   LDF      *+AR0(1),R0 ; R0=X(I+1) IM
||      STF      R1,*AR1++(IR0) ; STORE X(I) RE

```

```
*****
*   FFT ON COL I      *
*****
```

```

LAJ      CFFT
LDF      *AR0,R1           ; LOAD X(N-1) RE
STF      R0,*+AR1(1)      ; STORE X(N-1) IM
STF      R1,*AR1          ; STORE X(N-1) RE

```

```

*****
*   FFT MOVES COL. I   *
* (BIT-REVERSED) TO   *
* EXTERNAL MEMORY     *
*****

        LDI      AR6,AR0          ; SOURCE = BLOCK0
        LDI      AR7,AR1          ; DESTINATION = MATRIX
        SUBI3    2,AR3,RC         ; RC=N-2
        RPTBD    LOOP19
        LDI      AR3,IR0          ; SOURCE OFFSET = IR0 = N   (BIT-REVERSE)
        LDI      2,IR1            ; DESTINATION OFFSET
        LDF      *+AR0(1),R0
        LDF      *AR0++(IR0)B,R1
||      STF      R0,*+AR1(1)

LOOP19  LDF      *+AR0(1),R0
||      STF      R1,*AR1++(IR1)
        LSH3     1,AR3,R0         ; R0 = 2*N
        ADDI     1,AR5
        CMPI     R7,AR5
        BND      WAIT
        LDF      *AR0++(IR0)B,R1
||      STF      R0,*+AR1(1)
        STF      R1,*AR1++(IR1)
        ADDI     R0,AR7

*****
*                               LAST COLUMN                               *
*****

WAIT2   TSTB     @SMASK,IIF
        BZAT     WAIT2
        ANDN     @SMASK,IIF

LASTCOL:

*****
* CPU MOVES COL (N-1) *
*   TO ON-CHIP RAM   *
*****

        LDI      AR7,AR0          ; SOURCE
        LDI      AR6,AR1          ; DESTINATION
        SUBI3    2,AR3,RC         ; RC=N-2
        RPTBD    LOOP28
        LDI      2,IR1            ; SOURCE OFFSET
        LDI      2,IR0            ; DESTINATION OFFSET
        LDF      *+AR0(1),R0       ; R0= X(I) IM
        LDF      *AR0++(IR1),R1    ; X(I) RE & POINTS TO X(I+1)
||      STF      R0,*+AR1(1)       ; STORE X(I) IM

LOOP28  LDF      *+AR0(1),R0       ; R0=X(I+1) IM
||      STF      R1,*AR1++(IR0)    ; STORE X(I) RE

```

```

*****
*   FFT ON COL (N-1)   *
*****

        LAJ      CFFT
        LDF      *AR0,R1          ; LOAD X(N-1) RE
        STF      R0,*+AR1(1)      ; STORE X(N-1) IM
        STF      R1,*AR1          ; STORE X(N-1) RE

*****
* FFT MOVES COL.(N-1)*
* (BIT-REVERSED) TO *
* EXTERNAL MEMORY *
*****

        LDI      AR6,AR0          ; SOURCE = BLOCK0
        LDI      AR7,AR1          ; DESTINATION = MATRIX
        SUBI3    2,AR3,RC        ; RC=N-2
        RPTBD    LOOP29
        LDI      AR3,IR0          ; SOURCE OFFSET = IR0 = N (BIT-REVERSE)
        LDI      2,IR1           ; DESTINATION OFFSET
        LDF      *+AR0(1),R0
        LDF      *AR0++(IR0)B,R1
||      STF      R0,*+AR1(1)

LOOP29  LDF      *+AR0(1),R0
||      STF      R1,*AR1++(IR1)
        LDF      *AR0++(IR0)B,R1
||      STF      R0,*+AR1(1)
        STF      R1,*AR1++(IR1)
        LDI      @TIMER,AR2      ; OPTIONAL: BENCHMARKING (TIME_READ)
        LDI      *+AR2(4),R0     ; TCOMP = R0

t2      BU       t2

        .end

```

DPINPUT.ASM

```
*****
*
* DINPUT.ASM :      Input file for distributed-memory program with parallel
*                  system information
*
*****

        .global    N            ; FFT size
        .global    M            ; LOG2 FFT
        .global    P            ; Number of processors
        .global    Q            ; Rows per processor
        .global    _PORT
        .global    _ARRAY       ; buffer to be used in matrix transposition
        .global    _DMAMEM      ; memory address for autoinitialization values
        .global    _DMALIST
        .global    _CTRLIST
        .global    _DSTQLIST

N        .set      16            ; FFT size
M        .set      4            ; LOG FFT
P        .set      2            ; number of processors
Q        .set      N/P          ; rows/columns per processor

        .text

*_PORT   .int       0,0,4,3      ; connectivity matrix: processor i is
*        .int       3,0,0,4      ; connected to processor j through port
*        .int       1,3,0,0      ; PORT[i][j]      (P = 4)
*        .int       0,1,3,0

_PORT    .int       0,0,3,0      ; P = 2

_DMAMEM  .space     35*P
_DMALIST .space     P
_CTRLIST .space     P
_DSTQLIST .space    P
_ARRAY   .space     2*Q
        .end
```

SSINTAB.ASM

```
*****
*
*   SSINTAB.ASM:   Table with twiddle factors for a 16-point CFFT
*   and data input. File to be linked with the
*   source code for a 16-point, radix-2 FFT.
*
*****

        .global   SINE
        .data
SINE    .float     0.000000
        .float     0.382683
        .float     0.707107
        .float     0.923880
COSINE  .float     1.000000
        .float     0.923880
        .float     0.707107
        .float     0.382683
        .float     -0.000000
        .float     -0.382684
        .float     -0.707107
        .float     -0.923880
        .float     -1.000000
        .float     -0.923880
        .float     -0.707107
        .float     -0.382683
        .float     -0.000000
        .float     -0.382684
        .float     -0.707107
        .float     -0.923880
        .end
```

DIS.CMD

```
dis2.obj
ssintab.obj
dpsinput.obj
-lmylib.lib
-m dis.map

MEMORY
{
    ROM:      o = 0x00000000 l = 0x1000
    RAM0:     o = 0x002ff800 l = 0x400
    RAM1:     o = 0x002ffc00 l = 0x400
    LM:       o = 0x40000000 l = 0x10000
    GM:       o = 0x80000000 l = 0x20000
}

SECTIONS
{
    INPUT    :      {}      >  LM
    .text    :      {}      >  LM
    .data    :      {}      >  RAM1
    STACK    :      {}      >  RAM1
}
```


Appendix D: Mylib.lib Routines

D.1. CFFT.ASM: Assembly Language FFT Routine

CFFT.ASM

```
*****
*
*      CFFT.ASM : TMS320C40 COMPLEX, RADIX-2, DIF FFT
*
*      GENERIC PROGRAM FOR A LOOPED-CODE RADIX-2 FFT COMPUTATION IN 320C40
*
*      THE PROGRAM IS TAKEN FROM THE BURRUS AND PARKS BOOK, P. 111.
*      THE (COMPLEX) DATA RESIDE IN INTERNAL MEMORY.  THE COMPUTATION
*      IS DONE IN-PLACE, BUT THE RESULT IS MOVED TO ANOTHER MEMORY
*      SECTION TO DEMONSTRATE THE BIT-REVERSED ADDRESSING.
*
*      THE TWIDDLE FACTORS ARE SUPPLIED IN A TABLE PUT IN A .DATA SECTION.
*      THIS DATA IS INCLUDED IN A SEPARATE FILE TO PRESERVE THE GENERIC
*      NATURE OF THE PROGRAM.  FOR THE SAME PURPOSE, THE SIZE OF THE FFT
*      N AND LOG2(N) ARE DEFINED IN A .GLOBL DIRECTIVE AND SPECIFIED
*      DURING LINKING.
*
*      INPUT PARAMETERS:
*      AR6: INPUT ADDRESS (BLOCK 0/1 ON-CHIP)
*      R11: RETURN ADDRESS
*
*      REGISTERS MODIFIED:  R0,R1,R2,R3,R4,R5,R6,R8,R9,R10
*                          AR0,AR1,AR2,AR4,AR5
*                          IR0,IR1
*                          RC
*
*****

        .globl    CFFT          ; Entry point for execution
        .globl    N              ; FFT size
        .globl    M              ; LOG2(N)
        .globl    SINE          ; Address of sine table
        .text
*      INITIALIZE

FFTSIZ  .word     N
LOGFFT  .word     M
SINTAB  .word     SINE

CFFT    PUSH      DP
        PUSH      AR5
        LDP       FFTSIZ
        LDI       1,R8          ; Initialize repeat counter of first loop
        LDI       1,AR5         ; Initialize IE index (AR5=IE)
        LDI       @FFTSIZ,R10   ; R10=N
        LSH3      -2,R10,IR1     ; IR1=N/4, pointer for SIN/COS table
        LDI       @LOGFFT,R9    ; R9 holds the remain stage number
        LSH3      1,R10,IR0      ; IR0=2*N (because of real/imag)
        LSH       1,R10
        SUBI3     1,R8,RC        ; RC should be one less than desired #

*      OUTER LOOP
LOOP:   RPTBD     BLK1           ; Setup for first loop
        LSH      -1,R10         ; N2=N2/2
        LDI      AR6,AR0        ; AR0 points to X(I)
        ADDI     R10,AR0,AR2    ; AR2 points to X(L)
```

```

*      FIRST LOOP
      ADDF      *AR0,*AR2,R0      ; R0=X(I)+X(L)
      SUBF      *AR2++,*AR0++,R1  ; R1=X(I)-X(L)
      ADDF      *AR2,*AR0,R2      ; R2=Y(I)+Y(L)
      SUBF      *AR2,*AR0,R3      ; R3=Y(I)-Y(L)
      STF       R2,*AR0- -        ; Y(I)=R2 and...
||      STF       R3,*AR2- -      ; Y(L)=R3
BLK1    STF       R0,*AR0++(IR0)  ; X(I)=R0 and...
||      STF       R1,*AR2++(IR0)  ; X(L)=R1 and AR0,2 = AR0,2 + 2*n

*      IF THIS IS THE LAST STAGE, YOU ARE DONE
      SUBI      1,R9
      BZD       END

*      MAIN INNER LOOP
      LDI       2,AR1             ; Init loop counter for inner loop
      LDI       @SINTAB,AR4       ; Initialize IA index (AR4=IA)
      ADDI      AR5,AR4           ; IA=IA+IE; AR4 points to cosine
      ADDI      AR6,AR1,AR0       ; (X(I),Y(I)) pointer
      SUBI      1,R8,RC           ; RC should be one less than desired #

INLOP:   RPTBD    BLK2             ; Setup for second loop
      ADDI      R10,AR0,AR2       ; (X(L),Y(L)) pointer
      ADDI      2,AR1
      LDF       *AR4,R6           ; R6=SIN

*      SECOND LOOP
      SUBF      *AR2,*AR0,R2      ; R2=X(I)-X(L)
      SUBF      *+AR2,*+AR0,R1    ; R1=Y(I)-Y(L)
      MPYF      R2,R6,R0          ; R0=R2*SIN
||      ADDF      *+AR2,*+AR0,R3

*      MPYF      R1,*+AR4(IR1),R3 ; R3=Y(I)+Y(L)
||      STF       R3,*+AR0        ; Y(I)=Y(I)+Y(L)
      SUBF      R0,R3,R4          ; R4=R1*COS-R2*SIN
      MPYF      R1,R6,R0          ; R0=R1*SIN and...
||      ADDF      *AR2,*AR0,R3    ; R3=X(I)+X(L)
      MPYF      R2,*+AR4(IR1),R3 ; R3 = R2 * COS and..
||      STF       R3,*AR0++(IR0)

*      ADDF      R0,R3,R5          ; X(I)=X(I)+X(L) and AR0=AR0+2*N1
BLK2    STF      R5,*AR2++(IR0)    ; X(L)=R2*COS+R1*SIN
||      STF      R4,*+AR2          ; Y(L)=R1*COS-R2*SIN
      CMPI      R10,AR1
      BNEAF     INLOP             ; Loop back to the inner loop
      ADDI      AR5,AR4           ; IA=IA+IE; AR4 points to cosine
      ADDI      AR6,AR1,AR0       ; (X(I),Y(I)) pointer
      SUBI      1,R8,RC
      LSH       1,R8             ; Increment loop counter
      BRD       LOOP             ; Next FFT stage (delayed)
      LSH       1,AR5            ; IE=2*IE
      LDI       R10,IR0          ; N1=N2
      SUBI3     1,R8,RC

END      BUD       R11
      POP       AR5
      POP       DP
      NOP
      .end

```

D.2. CFFTC.ASM: Assembly Language FFT Routine (C-Callable)

CFFTC.ASM

```
*****
*
*      CFFTC.ASM : Complex radix-2 DIF 1-D FFT routine (C-callable)
*
*      Generic program for a lopped-code radix-2 FFT computation using the
*      TMS320C4x family. The computation is done in-place and the result
*      is bit-reversed. The program is taken from the Burrus and Parks
*      book, p. 111.
*
*      The twiddle factors are supplied in a table put in a .data section.
*      This data is included in a separate file to preserve the generic
*      nature of the program. For the same purpose, the size of the FFT
*      N and log2(N) are defined in a .globl directive and specified
*      during linking.
*
*      Calling conventions:
*
*      cfftc((float *)input,int fft_size,int logfft)
*              ar2 r2 r3
*
*      where      input      :   Complex vector address
*                   fft_size  :   Complex FFT size
*                   logfft    :   logarithm (base 2) of FFT size
*
*      Registers modified: R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10
*                          AR0,AR1,AR6,AR4,AR5
*                          IR0,IR1
*                          RC,DP
*
*****

        .globl    SINE                ; Address of sine/cosine table
        .globl    _cfftc              ; Entry point for execution
        .text
SINTAB  .word     SINE

_cfftc:
        LDI       SP,AR0
        PUSH      DP
        PUSH      R4                  ; Save dedicated registers
        PUSH      R5
        PUSH      R6
        PUSHF     R6                  ; upper 32 bits
        PUSH      AR4
        PUSH      AR5
        PUSH      AR6
        PUSH      R8
        .if       .REGPARM == 0
        LDI       *-AR0(1),AR2        ; points to X(I): INPUT
        LDI       *-AR0(2),R10        ; R10=N
        LDI       *-AR0(3),R9        ; R9 holds the remain stage number
        .else
        LDI       R2,R10
        LDI       R3,R9
        .endif
```

```

LDP      SINTAB
LDI      1,R8                ; Initialize repeat counter of first loop
LSH3     1,R10,IR0           ; IR0=2*N1 (because of real/imag)
LSH3     -2,R10,IR1          ; IR1=N/4, pointer for SIN/COS table
LDI      1,AR5               ; Initialize IE index (AR5=IE)
LSH      1,R10
SUBI3    1,R8,RC              ; RC should be one less than desired #

* Outer loop

LOOP:
RPTBD    BLK1                ; Setup for first loop
LSH      -1,R10              ; N2=N2/2
LDI      AR2,AR0              ; AR0 points to X(I)
ADDI     R10,AR0,AR6          ; AR6 points to X(L)

* First loop
ADDF     *AR0,*AR6,R0         ; R0=X(I)+X(L)
SUBF     *AR6++,*AR0++,R1     ; R1=X(I)-X(L)
ADDF     *AR6,*AR0,R2         ; R2=Y(I)+Y(L)
SUBF     *AR6,*AR0,R3         ; R3=Y(I)-Y(L)
STF      R2,*AR0--           ; Y(I)=R2 and...
||      STF      R3,*AR6--    ; Y(L)=R3
BLK1    STF      R0,*AR0++(IR0) ; X(I)=R0 and...
||      STF      R1,*AR6++(IR0) ; X(L)=R1 and AR0,2 = AR0,2 + 2*n

* If this is the last stage, you are done

SUBI     1,R9
BZD      END

* main inner loop
LDI      2,AR1                ; Init loop counter for inner loop
LDI      @SINTAB,AR4          ; Initialize IA index (AR4=IA)
ADDI     AR5,AR4              ; IA=IA+IE; AR4 points to cosine
ADDI     AR2,AR1,AR0          ; (X(I),Y(I)) pointer
SUBI     1,R8,RC              ; RC should be one less than desired #

INLOP:
RPTBD    BLK2                ; Setup for second loop
ADDI     R10,AR0,AR6          ; (X(L),Y(L)) pointer
ADDI     2,AR1
LDF      *AR4,R6              ; R6=SIN

* Second loop
SUBF     *AR6,*AR0,R2         ; R2=X(I)-X(L)
SUBF     *+AR6,*+AR0,R1       ; R1=Y(I)-Y(L)
MPYF     R2,R6,R0             ; R0=R2*SIN and...
||      ADDF     *+AR6,*+AR0,R3 ; R3=Y(I)+Y(L)
MPYF     R1,*+AR4(IR1),R3     ; R3 = R1 * COS and ...
||      STF      R3,*+AR0       ; Y(I)=Y(I)+Y(L)
SUBF     R0,R3,R4             ; R4=R1*COS-R2*SIN
MPYF     R1,R6,R0             ; R0=R1*SIN and...
||      ADDF     *AR6,*AR0,R3   ; R3=X(I)+X(L)
MPYF     R2,*+AR4(IR1),R3     ; R3 = R2 * COS and...
||      STF      R3,*AR0++(IR0) ; X(I)=X(I)+X(L) and AR0=AR0+2*N1
ADDF     R0,R3,R5             ; R5=R2*COS+R1*SIN

```

```

BLK2      STF      R5,*AR6++(IR0)      ; X(L)=R2*COS+R1*SIN
||        STF      R4,*+AR6           ; Y(L)=R1*COS-R2*SIN
          CMPI     R10,AR1
          BNEAF    INLOP               ; Loop back to the inner loop
          ADDI     AR5,AR4             ; IA=IA+IE; AR4 points to cosine
          ADDI     AR2,AR1,AR0        ; (X(I),Y(I)) pointer
          SUBI     1,R8,RC
          LSH      1,R8               ; Increment loop counter for next time
          BRD      LOOP               ; Next FFT stage (delayed)
          LSH      1,AR5              ; IE=2*IE
          LDI      R10,IR0            ; N1=N2
          SUBI3    1,R8,RC
END        POP      R8
          POP      AR6
          POP      AR5                ; Restore the register values and return
          POP      AR4
          POPF     R6
          POP      R6
          POP      R5
          POP      R4
          POP      DP
          RETS
          .end

```

D.3. CMOVE.ASM: Complex-Vector Move Routine

CMOVE.ASM

```
*****
*
*      CMOVE.ASM : TMS320C40 C-callable routine to move a complex float
*                  vector pointed by src, to an address pointed by dst.
*
*      Calling conventions:
*
*      void cmove ((float *)src,(float *)dst,int src_displ,int dst_displ,int length)
*                  ar2 r2 r3 rc rs
*
*      where      src          : Vector Source Address
*                  dst          : Vector Destination Address
*                  src_displ    : Source offset (real)
*                  dst_displ    : Destination offset (real)
*                  length       : Vector length (complex)
*
*****

        .global  _cmove

_cmove:
        .if      .REGPARM == 0
        LDI      SP,AR0
        LDI      *-AR0(1),AR2      ; Source address
        LDI      *-AR0(4),IR1      ; Destination index (real)
        LDI      *-AR0(5),RC       ; Complex length
        SUBI     2,RC              ; RC=length-2
        RPTBD    CMOVE
        LDI      *-AR0(2),AR1      ; Destination address
        LDI      *-AR0(3),IR0      ; Source index (real)
        LDF      *+AR2(1),R0

        .else
        LDI      RC,IR1            ; destination index (real)
        SUBI     2,RS,RC           ; complex length -2
        RPTBD    CMOVE
        LDI      R2,AR1            ; source address
        LDI      R3,IR0            ; source index (real)
        LDF      *+AR2(1),R0
        .endif

*   loop
        LDF      *AR2++(IR0),R1
||      STF      R0,*+AR1(1)

CMOVE   LDF      *+AR2(1),R0
||      STF      R1,*AR1++(IR1)
        POP      AR0
        BUD      AR0
        LDF      *AR2++(IR0),R1
||      STF      R0,*+AR1(1)
        STF      R1,*AR1
        NOP
        .end
```

D.4. CMOVEB.ASM: Complex-Vector Bit-Reversed Move Routine

CMOVEB.ASM

```
*****
*
*   CMOVEB.ASM : TMS320C40 C-callable routine to bit-reversed move a complex
*                 float vector pointed by src, to an address pointed by dst.
*
*   Calling conventions:
*
*   void cmoveb ((float *)src,(float *)dst, int src_displ,int dst_displ,int length)*
*               ar2 r2 r3 rc rs
*
*   where      src          : Vector Source Address
*               dst          : Vector Destination Address
*               src_displ    : Source offset (real)
*               dst_displ    : Destination offset (real)
*               length       : Vector length (complex)
*
*****

.global      _cmoveb

_cmoveb:
    .if      .REGPARM == 0
        LDI      SP,AR0
        LDI      *-AR0(1),AR2          ; Source address
        LDI      *-AR0(4),IR1          ; Destination index (real)
        LDI      *-AR0(5),RC           ; Complex length
        SUBI     2,RC                  ; RC=length-2
        RPTBD    CMOVEB
        LDI      *-AR0(2),AR1          ; Destination address
        LDI      *-AR0(3),IR0          ; Source index (real)
        LDF      *+AR2(1),R0

    .else
        LDI      RC,IR1                ; destination index (real)
        SUBI     2,RS,RC               ; complex length -2
        RPTBD    CMOVEB
        LDI      R2,AR1                ; source address
        LDI      R3,IR0                ; source index (real)
        LDF      *+AR2(1),R0
    .endif

*   loop
    LDF      *AR2++(IR0)B,R1
||   STF      R0,*+AR1(1)
CMOVEB     LDF      *+AR2(1),R0
||   STF      R1,*AR1++(IR1)

        POP      AR0
        BUD      AR0
        LDF      *AR2++(IR0)B,R1
||   STF      R0,*+AR1(1)
        STF      R1,*AR1
        NOP
    .end
```

D.5. SET_DMA.ASM: Routine to Set DMA Register Values

```
*****
*
*   SET_DMA.ASM : TMS320C30/'C40 C-callable routine to set DMA register values
*
*   Calling conventions:
*
*   void set_dma  ((int *)dma, int ctrl, (float *)src, int src_index,
*                 int counter, (float *)dst, int dst_index, (int *)dma_link)
*
*   where      dma      :   DMA register address      :   ar2
*                ctrl    :   Control Register         :   r2
*                src     :   Source Address            :   r3
*                src_index :   Source Address Index     :   rc
*                counter  :   Transfer Count           :   rs
*                dst     :   Destination Address       :   re
*                dst_index :   Destination Address Index :   stack
*                dma_link :   Link Pointer             :   stack
*
*****

.global  _set_dma
.text

_set_dma:

    LDI    SP,AR0          ; Points to top of stack
    .if    .REGPARM == 0
    LDI    *-AR0(1),AR2    ; AR2 points to DMA registers
    LDI    *-AR0(2),R2     ; Control register
    LDI    *-AR0(3),R3     ; Source
    LDI    *-AR0(4),RC     ; Source index
    LDI    *-AR0(5),RS     ; Transfer counter
    LDI    *-AR0(6),RE     ; Destination address
    LDI    *-AR0(7),R0     ; Destination index
    LDI    *-AR0(8),R1     ; Link pointer
    .else
    LDI    *-AR0(1),R0     ; Destination index
    LDI    *-AR0(2),R1     ; Link pointer
    .endif

    STI    R3,*+AR2(1)     ; source address
    STI    RC,*+AR2(2)     ; source index
    STI    RS,*+AR2(3)     ; counter
    STI    RE,*+AR2(4)     ; destination address
    POP    AR0
    BUD    AR0
    STI    R0,*+AR2(5)     ; destination index
    STI    R1,*+AR2(6)     ; link pointer
    STI    R2,*AR2         ; control
    .end
```


D.6. EXCHANGE.ASM: Routine for Interprocessor Communication

EXCHANGE.ASM

```
*****
*
*      EXCHANGE.ASM : TMS320C40 C-callable routine to exchange
*                    two floating point vectors pointed by "address" in
*                    each processor memory. This routine uses
*                    DMA in split mode with source/destination
*                    synchronization given by OCRDY/ICRDY respectively.
*
*      Calling conventions:
*
*      void exchange ((int *) dma, int comport, (float *)address, int length)
*
*      where      dma      :   DMA address              :   ar2
*                   comport :   Comport number to be used :   r2
*                   address :   Floating-point vector address :   r3
*                   length  :   Vector length              :   rc
*
*****
*      This routine requires that the communicating 'C4xs enter to the routine at
*      approximately the same time. This can be guaranteed by using a system with a
*      common reset or by using a system with a common reset or by using the PDM
*      (part of the 'C4x emulator) when you start running the 2D-FFT application.
*      For systems without this capability, use exch2.asm instead of this routine.
*****

        .global      _exchange

        .text
CONTROL .word        03C040D4H      ; DMA interrupt, R/W sync, split mode,
                                   ; CPU higher priority

PORT0   .word        00000000H
PORT1   .word        00008000H
PORT2   .word        00010000H
PORT3   .word        00018000H
PORT4   .word        00020000H
PORT5   .word        00028000H
PORTS   .word        PORT0
ENABLE  .word        24924955H      ; Enable interrupts to DMAS

_exchange:
        LDI          SP,AR0        ; Points to top of stack
        PUSH         DP

        .if          .REGPARM == 0

        LDI          *-AR0(1),AR2  ; DMA address
        LDI          *-AR0(2),R2   ; comport address
        LDI          *-AR0(3),R3   ; Memory address
        LDI          *-AR0(4),RC   ; Vector length
        .endif
        LDI          R2,AR1
        LDP          CONTROL
        STI          R3,*+AR2(1)   ; Source primary channel
        STI          RC,*+AR2(3)   ; Primary channel counter
        STI          R3,*+AR2(4)   ; Source secondary channel
        LDI          @CONTROL,R3
```

```

STIK      1H,*+AR2(2)      ; Primary source index
STIK      1H,*+AR2(5)      ; Secondary source index
ADDI      @PORTS,AR1       ; Pointing to port to be used

STI        RC,*+AR2(7)     ; Secondary channel counter
OR        *AR1,R3          ; Selecting port in DMA control reg.
STI        R3,*AR2
LDI        @ENABLE,DIE
POP        DP
RETS

.end

```

EXCHANGE2.ASM

```

*****
*
*  EXCHANGE2.ASM :  TMS320C40 C-callable routine to exchange
*                  two floating point vectors pointed by "address" in
*                  each processor memory. This routine uses
*                  DMA in split mode with source/destination
*                  synchronization given by OCRDY/ICRDY respectively.
*
*  Calling conventions:
*
*  void exchange ((int *) dma, int comport, (float *)address, int lenght)
*
*  where          dma          : DMA address                      : ar2
*                   comport     : Comport number to be used       : r2
*                   address     : Floating-point vector address    : r3
*                   lenght      : Vector lenght (complex)         : rc
*
*****
* This routine can be used in multiprocessing systems without a common start
*****
.global _exchange

.text
CONTROL .word 03C040D4H ; DMA interrupt, R/W synch, split mode,
                      ; CPU higher priority
PORT0 .word 00000000H
PORT1 .word 00008000H
PORT2 .word 00010000H
PORT3 .word 00018000H
PORT4 .word 00020000H
PORT5 .word 00028000H
PORTS .word PORT0
ENABLE .word 24924955H ; Enable interrupts to DMA'S
PORTADR .word 100040h ; RMP: 8/13/93 :modified for async parallel
                      ; systems

_exchange:
    PUSH    DP

    .if     .REGPARM == 0
    LDI     SP,AR0      ; Points to top of stack
    LDI     *-AR0(1),AR2 ; DMA address
    LDI     *-AR0(2),R2  ; comport address
    LDI     *-AR0(3),R3  ; Memory adress

```

```

        LDI        *-AR0(4),RC    ; Vector lenght
    .endif
    LDP        CONTROL

*** RMP: 8/13/93 :modified for async parallel systems
    MPYI        10h,R2,R0    ; This instructions synchronize the
    ADDI        @PORTADR,R0    ; processors at both end of the comm ports
    ADDI        1,R0,AR0    ; in systems where a common processor
    ADDI        2,R0,AR1    ; start is not offered. This is done by
    STI        R0,*AR1    ; sending/recieving a dummy word.
    LDI        *AR0,R0
***

    LDI        R2,AR1
    STI        R3,*+AR2(1)    ; Source primary channel
    STI        RC,*+AR2(3)    ; Primary channel counter
    STI        R3,*+AR2(4)    ; Source secondary channel
    LDI        @CONTROL,R3
    STIK        1H,*+AR2(2)    ; Primary source index
    STIK        1H,*+AR2(5)    ; Secondary source index
    ADDI        @PORTS,AR1    ; Pointing to port to be used

    STI        RC,*+AR2(7)    ; Secondary channel counter
    OR        *AR1,R3    ; Selecting port in DMA control reg.
    STI        R3,*AR2
    LDI        @ENABLE,DIE
    POP        DP
    RETS
    .end

```

D.7. SYNCOUNT.ASM: Interprocessor Synchronization Routine

SYNCOUNT.ASM

```
*****
*
*      syncount.asm      : assembly language synchronization routine to provide
*                        a global start for all the processors. Rotating priority
*                        for shared-memory access should be selected. The
*                        processors start with a cycle difference of maximum 3
*                        instruction cycles, which for practical purposes is
*                        acceptable. This routine is C-callable and uses register
*                        for parameter passing.
*
*      Calling conventions: void syncount((int *)counter,int value)
*                               ar2                ,r2
*
*****

      .global _syncount
      .text

_syncount:
      .if      .REGPARM == 0
      LDI      SP,AR1
      LDI      *-AR1(1),AR2
      LDI      *-AR1(2),R2
      .endif
      LDII     *AR2,R1
      ADDI     1,R1
      CMPI     R1,R2
      STII     R1,*AR2
      BZ       L1

AGAIN    LDI      *AR2,R1
         CMPI     R1,R2
         BNZ      AGAIN
L1       RETS
      .end
```


IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.