# Multiplatform Approaches and Tools for Parallel Computing in Signal Processing Domain

Tomas Fryza, Jitka Svobodova, Roman Marsalek, Jan Prokopec
*Department of Radio Electronics*
*Brno University of Technology*
*Brno, Czech Republic*
*{fryza,marsaler,prokopec}@feec.vutbr.cz, xsvobo61@stud.feec.vutbr.cz*

***Abstract*—The paper deals with various approaches used for parallel computing in signal processing domain. More precisely, the methods exploiting the multicores Central Processing Units such as Message Passing Interface and OpenMP are taking into account. The properties of the programming methods are experimentally proved in application of two-dimensional Fast Fourier Transform and Discrete Cosine Transform and are compared with possibilities of MATLAB built-in functions and Digital Signal Processors with Very Long Instruction Word architecture. The optimal combination of computing methods in signal processing domain is proposed. Results in the paper prove the possibility of creation of a heterogeneous computing system compounded of CPU and DSP architectures.**

***Keywords-transform coding; parallel computing; MPI; OpenMP; DSP.***

## I. Introduction

There are several approaches for effective parallel programming. The most widely used approach for distributed parallel computing for multicore Central Processing Units (CPUs) is Message Passing Interface (MPI) [1]. The MPI specifies the communication between separate processes, and it was designed for high performance on both massively parallel machines and on workstation clusters. The present-day version of the standard is MPI-2.2 approved by the MPI Forum at September, 2009. MPI library contains functions written in C and Fortran languages and in detail it is described in literature, such as [1], [2], or [3].

A different approach represents OpenMP with shared memory space, where all the cores can access the whole memory space. The OpenMP is an application programming interface for multi-platform parallel programming in C/C++ and Fortran. The current version of the standard is OpenMP 3.1 from July 2011. The specification and detailed tutorials could be found in [4], [5], or [6].

In this paper, both efficiency and limitations of multi-core processing are discussed and the impact in signal processing domain is proved. There are several projects implementing the main algorithms for digital signal processing. This paper deals with possibility of effective implementation of fast Fourier transform and discrete cosine transform. Libraries for fast computing the discrete Fourier transform, which commonly include real and/or complex, multidimensional, and parallel transforms can be found in [7], [8], etc.

Aim of this paper is to provide the comparison of different parallel approaches for signal processing. Two of well-known and widely used signal processing algorithms are implemented using MPI, OpenMP, MATLAB and DSP, then the results are discussed and compared. These algorithms will form a part of the benchmarks set useful for students and researchers interested in radio electronics.

The rest of this paper is organized as follows. Section II presents the chosen algorithms for parallel implementation in both CPU and DSP processors. The considered experiments with implementation of digital signal processing algorithms and achieved results are described in Section III, followed by short conclusion and future plans.

## II. Evaluated Algorithms

In this section, two implementation of digital signal processing algorithms are outlined. The algorithms used for evaluation of parallel potentialities are Fast Fourier Transform and Discrete Cosine Transform.

### A. Fast Fourier Transform Algorithms

A Discrete Fourier Transform (DFT) complexity grows with the square of the data length ($N$). Therefore, since the original paper of Cooley and Tukey published in 1965 [9] a tremendous effort has been devoted to the Fast Fourier Transform (FFT) algorithm research. The complexity of the FFT is generally in order of $N \log_2 N$ operations.

Many algorithms for the FFT calculations have been proposed in the past. Their very detailed overview containing the mathematical derivations gives a book [10]. The methods can be basically classified to the Decimation In Time (DIT) or Decimation In Frequency (DIF) families. Further classification of the methods is according the used radix – from the basic radix-2 the algorithms of radix-4 or radix-8 can be derived. It is also possible to use the combinations called split-radix [11] or mixed-radix FFT. A derivation of the basic method – radix-2 DIF is based on the recursive decomposition of the DFT [10]

$$X(r) = \sum_{l=0}^{N-1} x(l)\omega_N^{rl} \tag{1}$$

of the $N$-point input sequence $x(l)$ into two parts of the same length [10]:

$$X(r) = \sum_{l=0}^{N/2-1} x(l)\omega_N^{rl} + \sum_{l=N/2}^{N-1} x(l)\omega_N^{rl}. \tag{2}$$

After simple manipulations, it can be shown, that the radix-2 DIF FFT of $N$-sample length sequence $x(l)$ can be computed with the use of two half-size FFT's of sequences $y(l), z(l)$ [10]:

$$Y(k) = \sum_{l=0}^{N/2-1} y(l)\omega_{\frac{N}{2}}^{kl} \quad \text{and} \tag{3}$$

$$Z(k) = \sum_{l=0}^{N/2-1} z(l)\omega_{\frac{N}{2}}^{kl}, \tag{4}$$

with $Y(k) = X(2k), y(l) = x(l) + x\left(l + \frac{N}{2}\right)$, $Z(k) = X\left(2k+1\right), z(l) = \left(x(l) - x\left(l + \frac{N}{2}\right)\right)\omega_N^l$. Note that the twiddle factors $\omega_N^r$ are defined as

$$\omega_N^r = e^{jr\theta} = e^{jr\frac{2\pi}{N}}, \quad \text{where} \quad j = \sqrt{-1}. \tag{5}$$

An example of 8-point long FFT calculated using the radix-2 DIF algorithm is shown in Figure 1.

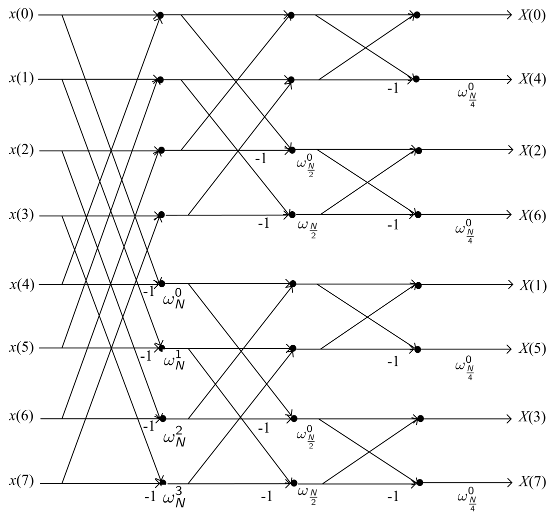


Figure 1. Radix-2 DIF graphical representation for 8-point data sequence.

## B. Discrete Cosine Transform

For vector with dimension of $N$, the forward one-dimensional discrete cosine transform (1-D DCT) is defined in the following way [12]

$$D(u) = \gamma(u) \cdot \sum_{x=0}^{N-1} f(x) \cdot \frac{\pi u(2x+1)}{2N} \tag{6}$$

where $D(u)$ represents 1-D DCT coefficient of a vector item $f(x)$ while $u = 0, \ldots, N-1$. The constant $\gamma(u)$ could be expressed as follows [12]

$$\gamma(u) = \begin{array}{lcl} \sqrt{1/N} & : & u = 0 \\ \sqrt{2/N} & : & u \neq 0. \end{array}$$

From the symmetry of DCT base function, the computation load of the DCT can be exploited. There are several known algorithms, such as Arai's [13], Chen's [14], Loeffler's [15], or Vetterli's [16]. For further implementation, the Arai's forward DCT approach was chosen. Let $N = 8$, then according to [13], [17], 5 multiplication and 29 addition operations have to be evaluated in order to calculate eight one-dimensional coefficients. Supposing color block with $8\times8$ elements, the 1-D transform has to be repeated 48 ($8\times3$ + $8\times3$) times to obtain 64 two-dimensional frequency coefficients. Therefore, for $8\times8$ color block, only 720 multiplications and 4 176 additions have to be calculated for transforming a single color block.

## III. Experiments

Algorithms were tested via two dimensional transformation of color frame(s) with QSXGA resolution, i.e., with dimensions of 2,560×2,048 pixels. Each pixel is coded in RGB color space by 24 bits. Tested frames were separated into small blocks of $N \times N$ pixels. Those blocks represent input signal for the two-dimensional FFT, or DCT coder. FFT uses complex input/output values, whereas DCT algorithm is adapted for real data only. The proposed implementation of both algorithms (according to Subsection II-A and II-B) uses the common dimension of transform base in signal processing domain, i.e., $N = 8$. Only in MATLAB environment, the built-in functions with dimensions from 8 to 2,048 were used.

For the evaluation of considered parallel computing methods, the several test cases were performed. Mainly, the time consuming of two-dimensional FFT and DCT algorithms with MPI, OpenMP, MATLAB, and Texas Instruments DSP approaches were tested. Two-dimensional transforms were always divided to successive calculation of two 1-D transforms. In general, algorithms could use either fixed-point or floating-point number representation. The most famous open source FFT library FFTW [7] uses double precision floating-point representation in theirs functions, while DSP the library [8] from Texas Instruments (produces of present-day's most powerful DSPs) incorporates both, single and double precision routines. For basic confrontation with mentioned libraries, all data in our tests were represented in single precision floating-point format. Fixed-point releases would be implemented and optimized in the future.

All CPU based parallel computing tests were performed on HP BL465c G5 Blade Server with two quad-core Opteron processors and 32 GB of RAM. The core clock frequency is 2.7 GHz, synchronous DDRII memory was running on 800 MHz.

For the simulation results discussion, we also mention size of CPUs internal cache. Internal L1 cache 256 kB per processor (64 kB for data and 64 kB for instruction), L2 cache is 2 MB ($4\times512$ kB) per processor, L3 cache 6 MB per processor, TLB (Translation Lookaside Buffer) of 4 kB.

The DSP based computing test were performed on Texas Instruments evaluation board with 32-bit floating-point digital signal processor TMS320C6747, with VLIW (Very Long Instruction Word) architecture, and clock frequency $f_{CPU} = 300$ MHz.

## A. Implementation Results

Results from first test case are shown in Figure 2. For various QSXGA color frames, the length of MPI message buffer was altered. The buffer contains both the input picture data (from master to slaves communication), and transformed two-dimensional coefficients as well (from slaves to master communication). Average computation times were calculated from sixteen evaluations; 8 cores were used for all calculations. It can be seen, the first fall of the computation time for both transforms, which corresponds with hardware setting of blade server; concretely with TLB size. On the other hand, the second (wider) fall of the computation time corresponds with the L2 cache size. For further computing, the MPI message buffer size of 4 kB would be chosen.

From Figure 2 (a) and Figure 2 (b) it is obvious that the selected implementation of FFT algorithm is slower than implementation of DCT algorithm. For $N = 8$, the implemented FFT algorithm is approximately 1.5-times slower than DCT algorithm. The reason is that FFT needs complex data, while as DCT needs real input and output values. Therefore, thirty two QSXGA color frames could be transformed in 2.2 s by FFT, but only in 1.4 s by DCT method.

Second test case describes parallel implementation of FFT and DCT algorithms with help of OpenMP approach. For transformation of several QSXGA color frames, 1, 2, 4, or 8 cores were used. The number of transformed frames varied between 1 and 32 for FFT algorithm and between 1 and 128 for DCT algorithm. The computation times are shown in Figure 3. With dotted lines, the serial versions of implemented algorithms, as well as ideal curves for parallel versions are expressed. The ideal versions are computed as the portion of serial results. The dashed line in figures represents the results achieved by MPI approach as well.

It can be seen, for lower number of processed data, the OpenMP version is less effective than MPI version. In addition, while a single QSXGA color frame is being transformed, the computation time for serial version is lower that parallel version with 2 cores! Therefore, the beneficial using of simple OpenMP in signal processing domain could be bitrate, which is adequate to 64 QSXGA color frames.

## B. Non Standard Implementations

MATLAB's Parallel Computing Toolbox provides running the script in up to 8 threads on a local computer or running it on a cluster machine using MATLAB Distributed Computing Server [18]. The main task is called Job and it is divided into Tasks, which are assigned to the individual workers by scheduler. The default scheduler for MATLAB Distributed Computing Server, MathWorks Job Manager, supports the Platform LSF, Microsoft Compute Cluster Server and Altair PBS Pro. Other schedulers can be integrated by user.

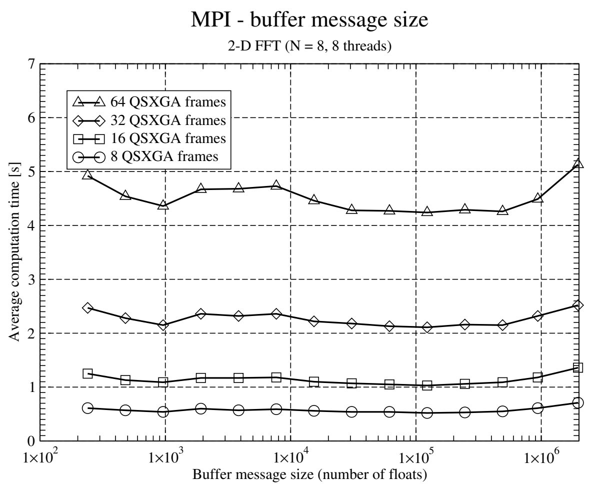
Third test case was performed in MATLAB environment. The MATLAB built-in functions `fft` and `dct` are called in all the individual workers. The computational time measurement starts before the parfor loop and ends after the variables' final reshape after the parfor loop. The results for the FFT and DCT computation from 1 to 8 threads for the blocks of vectors with the length of 8, 16, 32, 64, 128, 256, 512, 1024 and 2048 are depicted in Figure 4.

The application has to be divided into independent tasks which are then processed simultaneously. The most convenient way to solve this particular task uses the MATLAB functions as much as possible, because they are optimized to run fast and to use proper amount of memory. The *fft* and *dct* task is specific because of the use of both MATLAB functions and the parallel expressions. While the length of the array for the *fft* and *dct* function increases, the number of parallel loops decreases because of the total size of the matrix being processed. Thus, the computational time does not decrease constantly with increasing number of threads. This issue can be solved by using simpler algorithm, not the one which is based on two contradictory parts. This problem is to be solved and the new algorithm will be included in the benchmark dataset.

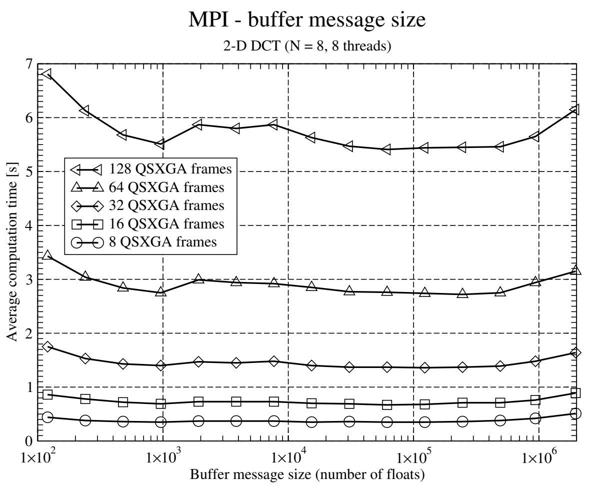
Table I
COMPUTATION TIME FOR TWO-DIMENSIONAL TMS320C6747 IMPLEMENTATION WITH VARYING PROGRAMMING APPROACHES ($f_{CPU} = 300$ MHZ, 1 QSXGA COLOR FRAME: $2{,}560\times2{,}048$ PIXELS)

| Algorithm | Programming language | Computation time [s] |
|---|---|---|
| 2-D FFT | C code | 7.08 |
| 2-D FFT | Linear assembly | 1.65 |
| 2-D DCT | C code | 3.05 |
| 2-D DCT | Linear assembly | 1.02 |

Last considered test case was performed by digital signal processor TMS320C6747, controlled by clock frequency of 300 MHz (9-times lower than CPU based tests). Although, the evaluation board contains only a single core DSP, the VLIW architecture meets the parallel approach. Selected

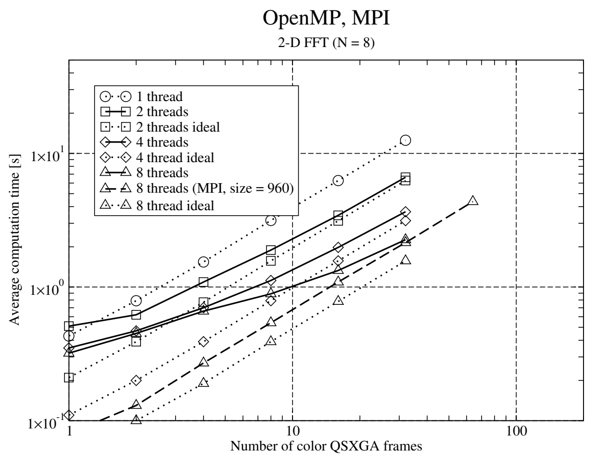


(a) 2-D FFT

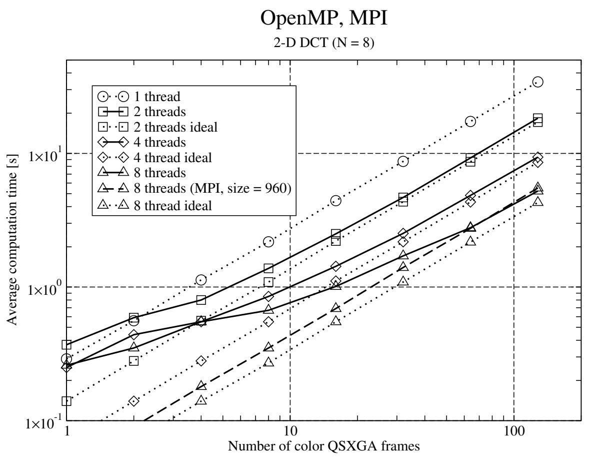


(b) 2-D DCT

Figure 2. Average computation time for two-dimensional MPI implementation with varying buffer message size ($N = 8$, $f_{CPU} = 2.7$ GHz, 8 threads, QSXGA color frames: 2,560×2,048 pixels).



(a) 2-D FFT



(b) 2-D DCT

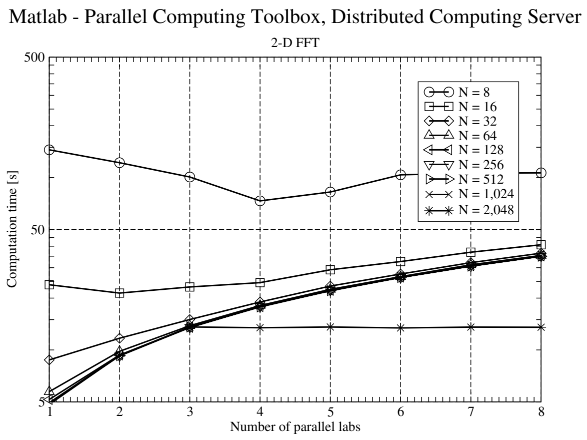
Figure 3. Average computation time for two-dimensional OpenMP implementation with varying transformed frames and threads number ($N = 8$, $f_{CPU} = 2.7$ GHz, QSXGA color frames: 2,560×2,048 pixels).

algorithms were implemented in C language and in linear assembly language. Development tool Code Composer Studio v.3.3 from Texas Instruments was used. All codes were optimized by CCS internal tools as well. The achieved results are shown in Table I.

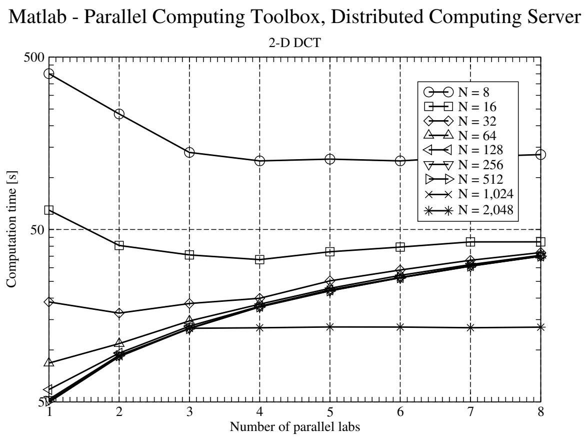
It can be seen that the general abstraction brought by C code is not useful in this case. The low-level programming of both FFT and DCT algorithms represents outstanding contribution in signal processing. A single QSXGA frame could be transformed in 1.65 s by FFT, and in 1.02 s by DCT method.

## IV. Conclusion and Future Work

The paper was focused on the implementation of two transforms, commonly used in signal processing domain. The two-dimensional FFT and DCT were chosen. The outline of currently used methods for parallel computing on CPU was performed as well. The MPI, OpenMP, and MATLAB approach were taken into account. The goal of the paper was also to present a possibility to create an interconnection between CPU based methods and VLIW architecture DSP evaluation boards. The future work would be focused mainly to implementation of digital signal pro-



(a) 2-D FFT



(b) 2-D DCT

Figure 4. Computation time for two-dimensional MATLAB implementation with varying parallel lab number ($f_{CPU} = 2.7$ GHz, 1 QSXGA color frame: 2,560×2,048 pixels).

cessing algorithms to Graphical Processing Units (GPUs) as well as to comparison with other CPUs, such as Intel quad-core Xeon e5640.

## Acknowledgment

Research published in this submission was financially supported by the project CZ.1.07/2.3.00/20.0007 WICOMT of the operational program Education for competitiveness and by the Brno University of Technology Internal Grant Agency under project no. FEKT-S-11-12 (MOBYS) and the described research was performed in laboratories supported by the SIX project; the registration number CZ.1.05/2.1.00/03.0072, the operational program Research and Development for Innovation.

## References

[1] MPI Forum. *Message Passing Interface Forum* (2012-03-11). [online]. Available: http://www.mpi-forum.org/.

[2] A Message-Passing Interface standard. *The International Journal of Supercomputer Applications and High Performance Computing*, 8, 1994.

[3] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. *MPI: The Complete Reference.* The MIT Press, 1998.

[4] *OpenMP* (2012-03-11). [online]. Available: http://openmp.org/wp/.

[5] Barbara Chapman, Gabriele Jost, and Ruud van der Pas. *Using OpenMP – Portable Shared Memory Parallel Programming.* The MIT Press, 2007.

[6] Blaise Barney. *OpenMP* (2012-03-11). [online]. Available: https://computing.llnl.gov/tutorials/openMP/.

[7] *FFTW Home Page* (2012-03-11). [online]. Available: http://www.fftw.org/.

[8] Texas Instruments. *TMS320C67x DSP Library* (2012-03-11). [online]. Available: http://www.ti.com/tool/sprc121.

[9] James William Cooley and John Wilder Tukey. An algorithm for the machine calculation of complex Fourier series, *Mathematics of Computation*. 19, 297–301, 1965.

[10] Eleanor Chin-hwa Chu and Alan George. *Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms (Computational Mathematics)*, 1st ed. CRC Press, 1999.

[11] Pierre Duhamel and Henk Hollmann. Split radix FFT algorithm, *Electronics Letters*, vol.20, no.1, pp. 14–16, 1984.

[12] Kamisetty Ramamohan Rao and Patrick Yip. *Discrete Cosine Transform. Algorithms, Advantages, Applications.* San Diego: Academic Press, Inc., 1990.

[13] Yukihiro Arai, Takeshi Agui, and Masayuki Nakajima. A Fast DCT-SQ Scheme for Images. *IEICE Transactions (1976–1990)*, 1988, vol. E71-E, no. 11, pp. 1095–1097.

[14] Wen-Hsiung Chen, Harrison Smith, and Sam Fralick. A fast computational algorithm for the discrete cosine transform. *IEEE, Transactions Commun*, 1977, pp. 1004–1009.

[15] Christoph Loeffler, Adriaan Ligtenberg, and George Moschytz. Practical fast 1-D DCT algorithms with 11 multiplications. *Proc. IEEE ICASSP*, 1989, pp. 988–991.

[16] Martin Vetterli. Fast 2-D discrete cosine transform. In *Proc. ICASSP*, 1985, pp. 1538–1541.

[17] Rafael Gonzalez and Paul Wintz. *Digital Image Processing.* Boston: Addison Wesley Publishing Company, 1987.

[18] MathWorks. *MATLAB and Simulink for Technical Computing* (2012-03-11). [online]. Available: http://www.mathworks.com/.