Overview of Parallel Platforms for Common High Performance Computing

Tomas FRYZA, Jitka SVOBODOVA, Filip ADAMEC, Roman MARSALEK, Jan PROKOPEC

Dept. of Radio Electronics, Brno University of Technology, Purkynova 118, 612 00 Brno, Czech Republic

{fryza,marsaler,prokopec}@feec.vutbr.cz, {xsvobo61,xadame24}@stud.feec.vutbr.cz

Abstract. *The paper deals with various parallel platforms used for high performance computing in the signal processing domain. More precisely, the methods exploiting the multicore central processing units such as message passing interface and OpenMP are taken into account. The properties of the programming methods are experimentally proved in the application of a fast Fourier transform and a discrete cosine transform and they are compared with the possibilities of MATLAB's built-in functions and Texas Instruments digital signal processors with very long instruction word architectures. New FFT and DCT implementations were proposed and tested. The implementation phase was compared with CPU based computing methods and with possibilities of the Texas Instruments digital signal processing library on C6747 floating-point DSPs. The optimal combination of computing methods in the signal processing domain and new, fast routines' implementation is proposed as well.*

Keywords

Digital signal processing, fast Fourier transforms, discrete cosine transforms, parallel programming, high performance computing, message passing interface, OpenMP, MATLAB, digital signal processors, optimization.

1. Introduction

Nowadays, the high performance computing tools are used for extremely time consuming tasks in signal processing domain, optimization solvers, data mining, etc. These systems consist of multi-core central processing units (CPUs), graphical processing units (GPUs), field-programmable gate arrays (FPGAs), or digital signal processors (DSPs). Even laptops are commonly equipped with dual-core processors. Ordinarily, those processing units are merged together, so heterogeneous computing systems are established. A promising method for high performance computing is using units or systems with an extreme degree of parallelism. Such systems are GPUs (both CUDA and OpenCL programming), multicore DSPs (e.g. TMS320C66x series from Texas Instruments), or even heterogeneous systems. Nevertheless, this paper mainly concentrates on CPU-based methods. The GPUs approach will be examined and compared in the future.

From the software point of view, the computing or simulation environments can comprise a large scale of processing algorithms from many researchers' domains. Despite the fact that processor manufactures can still preserve the Moore's law [1], and the computing performance is steadily increasing, the way of effective programming persistently has an irreplaceable position in many research areas.

There are several approaches for effective parallel programming. The most used approach for distributed parallel computing for multicore CPUs is message passing interface (MPI). 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 in September, 2009. The MPI-3.0 version is available in a draft version. The MPI library contains functions written in C and Fortran languages and it is described in detail in literature, such as [2], [3], or [4].

A different approach represents OpenMP with shared memory space, where all the cores can access the whole memory space. OpenMP is an application programming interface for multi-platform parallel programming in C/C++ and Fortran. The actual version of the standard is OpenMP 3.1 from July 2011. The specification and detailed tutorials could be found in [5], [6], or [7].

There are several projects implementing main algorithms for digital signal processing. This paper deals with the possibility of effectively implementing of fast Fourier transform and discrete cosine transform. Libraries for fast computing the discrete Fourier transform, which commonly includes real and/or complex, multidimensional, and parallel transforms can be found in [8], [9], etc.

The paper presents the multiplatform approaches for optimal parallel computing in signal processing domain and it is divided into four main parts. In Section 2, a brief introduction to parallel and distributed computing in MATLAB is outlined. Section 3 outlines the basic structure and application of MPI inter-core communication. Section 4 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 5 and Section 6, followed by short a conclusion.

2. Parallel Computing in MATLAB

MATLAB's Parallel Computing Toolbox provides running the script in up to n threads on a local computer or running it on a cluster machine using a MATLAB Distributed Computing Server. The main task is called Job and it is divided into Tasks, which are assigned to the individual workers by a 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; see Fig. 1.

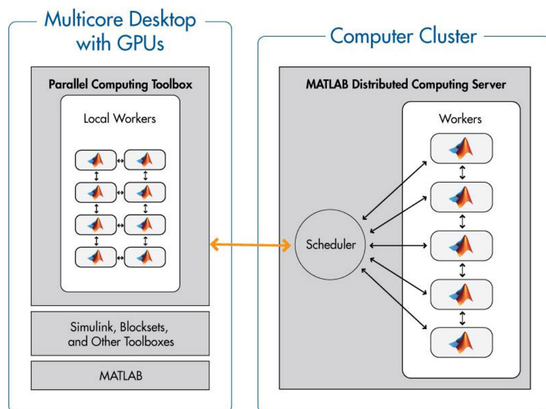


Fig. 1. Architecture of parallel and distributed computing in MATLAB [10].

The application has to be divided into independent tasks which are then processed simultaneously. The `parfor` loop corresponds to the `for` loop, but there are some differences. When using the parfor loop, the matlabpool has to be open and closed after the parfor loop usage as shown below.

```
if matlabpool('size') == 0
   matlabpool('open', 8)
end
...
matlabpool close
```

In this expression, the number of threads is specified. If some file is processed or a function is called within the parfor loop, it has to be expressed here as 'FileDependencies'.

The main difference between the for and parfor loop is the use of sliced variables. These variables 'slice' the current submatrix from the original matrix and use them for the computation in individual workers. In this case, the sliced variable is in the cell 'data type'. Then the results are released from the workers after the parfor loop:

```
% all submatrices
parfor i=1:length(Y)
   % sliced variables preparation
   Y_fft1=cell(size(Y));

   % every row in current submatrix
   for j=1:N
      Y_fft1{i}(j,:)=fft(Y{i}(j,:));
   end         % fft of current row

   % every column in current submatrix
   for j=1:N
      Y_fft2{i}(:,j)=fft(Y_fft1{i}(:,j));
   end         % fft of current column
end

% releasing data from workers
Y_fft=Y_fft2;
```

The most convenient way to solve this particular task is by using the MATLAB functions as much as possible, because they are optimized to run fast and to use the proper amount of memory.

3. Message Passing Interface

The message passing interface is an application programming interface (API) for communication between separate processes, representing the most widely used approach for distributed parallel computing. The MPI standard defines interfaces to C and Fortran programming languages and although the interface is large (contains over 120 procedures), it stays concise. That is why, very often only six procedures are needed to establish, control, and finalize interprocess communications. Each process is identified by a rank which is assigned during the runtime. Therefore, processes can perform different tasks and handle different data based on their rank. The process with the lowest rank is called master, all others processes are called slaves.

A parallel program is launched as a set of independent, identical processes, where the same program code and instructions can reside in different computing nodes, or even in different computers. Concurrently, all variables and data structures are local to the process and processes can exchange data by sending and receiving messages. There are two main modes of communication between processes: sending and receiving messages between two processes, or sending and receiving messages between several processes simultaneously.

The main structure of any MPI application is listed in the following example. It can be seen, that only three routines from the MPI interface are used: one for initialization of a parallel section, the second is for process rank detection, and the third stands for finalization of a parallel section. Within both the master and slave sections, a common sequential routine for fast Fourier transform is executed. The benefit of this source code structure is to process different (and independent) input data at dissimilar nodes.

```
#include <mpi.h>

int main( int argc, char *argv[] )
{
   int rank ;

   // beginning of parallelism
   MPI_Init( &argc, &argv ) ;

   // assigned rank to each process
   MPI_Comm_rank( MPI_COMM_WORLD, &rank );

   if( rank == 0 ){ // this is master's code
      ...    // send different data to slaves
      fft( ... ) ;  // FFT routine
      ...    // receive coeffs. from slaves
   }
   else{     // slaves' code
      ...    // receive data from master
      fft( ... ) ;  // FFT routine
      ...    // send coeffs to master
   }

   // end of parallelism
   MPI_Finalize() ;
   return( 0 ) ;
}
```

4. Evaluated Algorithms

In this Section, two implementations of digital signal processing algorithms are outlined. The algorithms used for the evaluation of parallel potentialities are fast Fourier transform and discrete cosine transform.

4.1 Fast Fourier Transform Algorithms

The 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 [11] 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. A very detailed overview containing the mathematical derivations is given in book [12]. The methods can be basically classified as the decimation in time (DIT) or decimation in frequency (DIF) families. Further classification of the methods is according to 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 combinations called split-radix [13] or mixed-radix FFT. A derivation of one of the basic methods – radix-2 DIT is based on the recursive decomposition of the original DFT (note that the twiddle factors are defined as $\omega_N^r = e^{\jmath r\theta} = e^{\jmath r\frac{2\pi}{N}}$ where $\jmath = \sqrt{-1}$)

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

of the N-point input sequence $x(l)$ into two parts of the same length [12] corresponding to the odd and even components:

$$X(r) = \sum_{k=0}^{N/2-1} x(2k)\omega_N^{r2k} + \omega_N^r \sum_{k=0}^{N/2-1} x(2k+1)\omega_N^{r2k}. \tag{2}$$

Considering that $\omega_{\frac{N}{2}} = \omega_N^2$ the radix-2 DIT FFT of N-sample length sequence $x(l)$ can be computed with the use of two half-size FFT's of sequences $x(2k)$ (even samples) and $x(2k+1)$ (odd samples):

$$Y(r) = \sum_{k=0}^{N/2-1} x(2k)\omega_{\frac{N}{2}}^{rk} \tag{3}$$

and

$$Z(r) = \sum_{k=0}^{N/2-1} x(2k+1)\omega_{\frac{N}{2}}^{rk}. \tag{4}$$

The first $N/2$ output samples of the radix-2 DIT FFT can thus be expressed according to equation (2) as:

$$X(r) = Y(r) + \omega_N^r Z(r). \tag{5}$$

Similarly it can be simply shown that the second half of the output samples can be computed as:

$$X(r+N/2) = Y(r) - \omega_N^r Z(r). \tag{6}$$

An example of an 8-point long FFT calculated using the radix-2 DIT algorithm is shown in Fig. 2.

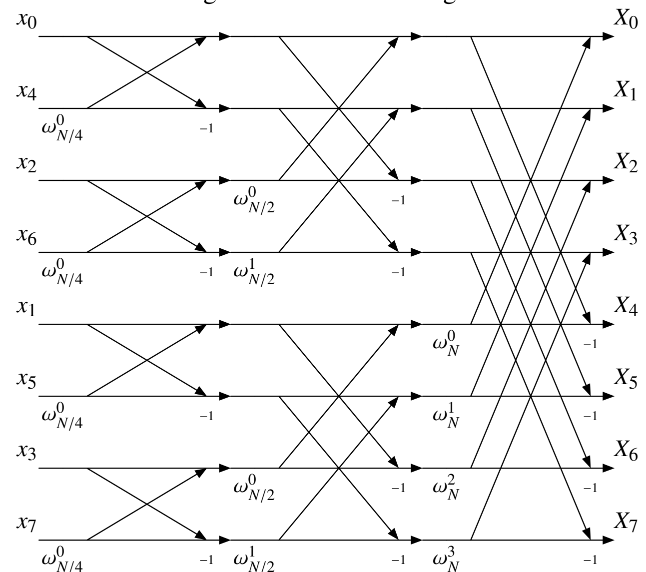


Fig. 2. Radix-2 DIT graphical representation for 8-point data sequence.

4.2 Discrete Cosine Transform

For vector with a dimension of N, the forward one-dimensional discrete cosine transform (1-D DCT) is defined in the following way [14]

$$D(r) = \gamma(r) \cdot \sum_{k=0}^{N-1} f(k) \cdot \cos\frac{\pi r(2k+1)}{2N} \tag{7}$$

where $D(r)$ represents the 1-D DCT coefficient of a vector item $f(k)$ while $r = 0, \ldots, N-1$. The constant $\gamma(r)$ could be expressed as follows

$$\gamma(r) = \begin{cases} \sqrt{1/N} & : \; r = 0, \\ \sqrt{2/N} & : \; r \neq 0. \end{cases}$$

According to the definition, the basic N-point DCT calculation requires N^2 multiplications and $N \cdot (N-1)$ additions. Supposing a color block with 8×8 elements, the 1-D transform has to be repeated 48 (8×3 + 8×3) times to obtain 64 two-dimensional frequency coefficients. According to equation (7), calculation of such a block requires 3 072 multiplications and 2 688 addition operations.

From the symmetry of the DCT base function, the computation load of the DCT can be exploited. There are several known algorithms, such as Arai's [15], Chen's [16], Loeffler's [17], or Vetterli's [18]. For further implementation, the Arai's forward DCT approach was chosen. Let $N = 8$, then according to [15], [19], 5 multiplication and 29 addition operations have to be evaluated in order to calculate eight one-dimensional coefficients. Therefore, for a 8×8 color block, only 720 multiplications and 4 176 additions have to be calculated.

5. Practical 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 and those blocks represent an input signal for the two-dimensional FFT, or DCT coder. FFT uses complex input/output values, whereas the DCT algorithm is adapted for real data only. The proposed implementation of both algorithms (according to subsection 4.1 and 4.2) uses the common dimension of a transform base in signal processing domain, i.e., $N = 8$. Only in the MATLAB environment were the built-in functions with dimensions from 8 to 2,048 used.

For evaluating the considered parallel computing methods, several test cases were performed. Mainly, the consuming time 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 calculations of two 1-D transforms.

All CPU based parallel computing tests were performed on the 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 at 800 MHz.

For the simulation results discussion, we also mention the size of the CPUs internal cache. Internal L1 cache is 256 kB per processor (64 kB for data and 64 kB for instruction), L2 cache is 2 MB (4×512 kB) per processor, L3 cache 6 MB per processor, TLB (Translation Lookaside Buffer) of 4 kB.

The DSP based computing tests were performed on the Texas Instruments evaluation board OMAP-L137. The board incorporates the Texas Instrument OMAP-L137 processor [20], which integrates 300 MHz ARM9 processor and 300 MHz fixed/floating-point C674x DSP core with very long instruction word (VLIW) architecture, where eight simultaneous instructions can be dispatched. The instructions in a VLIW packet can be executed in parallel or serial based on a special status bit in the VLIW packet. The cache memories can be used as well as memory for data (L1D 32 kB) and instructions (L1P 32 kB). In such a case, the access to data or instructions are in full speed but read from main memory is not cached and access data from main memory can be slow.

The Texas Instruments DSP library (DSPLib) [9] for TMS320C67x DSP's contains optimized DSP functions for floating-point C67x series of DSP's. Among others, the library contains various versions of FFT algorithm [12] as radix-2, radix-4 and mixed radix FFT. It contains a function for adaptive filters, correlations, FIR and IIR filters, matrix operations as well. All functions in this library are hand optimized to maximize their speed on a target DSP core.

For radix-2 FFT decimation-in-time algorithm, the function `DSPF_sp_cfftr2_dit` from DSPLib must be used. The input complex array is stored in normal order. The twiddle factors are passed into a vector, which contains $N/2$ complex numbers. The result of the FFT is stored back to an input vector in bit-reversed order. The functions to generate twiddle factor and reverse order are not part of this library. Using this library function could be as follows and theoretical computation complexity of this function is from [9]: $cycles = 2 \cdot N \cdot \log 2(N) + 42$.

```
// generate coefficient table
gen_twiddle( w, N ) ;

// bit-reverse coefficient table
bit_rev( w, N>>1 ) ;

DSPF_sp_cfftr2_dit( x, w, N ) ;

// normal order FFT coefficients
bit_rev( data, N ) ;
```

New functions for VLIW DSP were optimized with an auxiliary tool for assembly code generation. This tool could provide optimal and synoptical reprogramming of intended algorithms. The main goal of the tool is to ensure the utilization of the majority of functional DSP's units in any stage of execution and to bring information to the programmer what general purpose registers are available and ready to use. The tool contains a list of known instructions [21], their brief description including a list of input arguments, possible functional unit(s), execution time and/or pipelining stages. In any time, the number of used functional units, i.e. the instruction

packet length, is calculated. With this tool, the programmer has absolute control of instruction packet length, free registers and instants when previous executions are finished. This list of instructions are not limited only to the used Texas Instruments DSPs, but can be easily extended to an arbitrary processor or microcontroller. The final code can be exported to development environment (such as Code Composer Studio) and assembled for target device.

6. Results

6.1 Message Passing Interface

Results from the first test case are shown in Fig. 3. 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). All data were represented in single precision floating-point format. Average computation times were calculated from sixteen evaluations; eight cores were used for all calculations. The first fall of the computation time for both transforms can be seen, which corresponds with hardware setting of the blade server; specifically the 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 Fig. 3 (a) and Fig. 3 (b) it is obvious, the selected implementation of the FFT algorithm is slower than the implementation of the DCT algorithm. For $N = 8$, the implemented FFT algorithm is approximately 1.5-times slower than the DCT algorithm. The reason is that the FFT needs complex data, whereas DCT needs real input and output values. Therefore, thirty two QSXGA color frames could be transformed in 2.2 s by the FFT, but only in 1.4 s by the DCT method.

6.2 OpenMP

The second test case describes a parallel implementation of FFT and DCT algorithms with the help of the OpenMP approach. For the transformation of several QSXGA color frames, 1, 2, 4, and 8 cores were used. The number of transformed frames varied between 1 and 32 for the FFT algorithm and between 1 and 128 for the DCT algorithm. The computation times are shown in Fig. 4. 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 a portion of serial results. The dashed line in the figures represents the results achieved by the MPI approach as well.

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

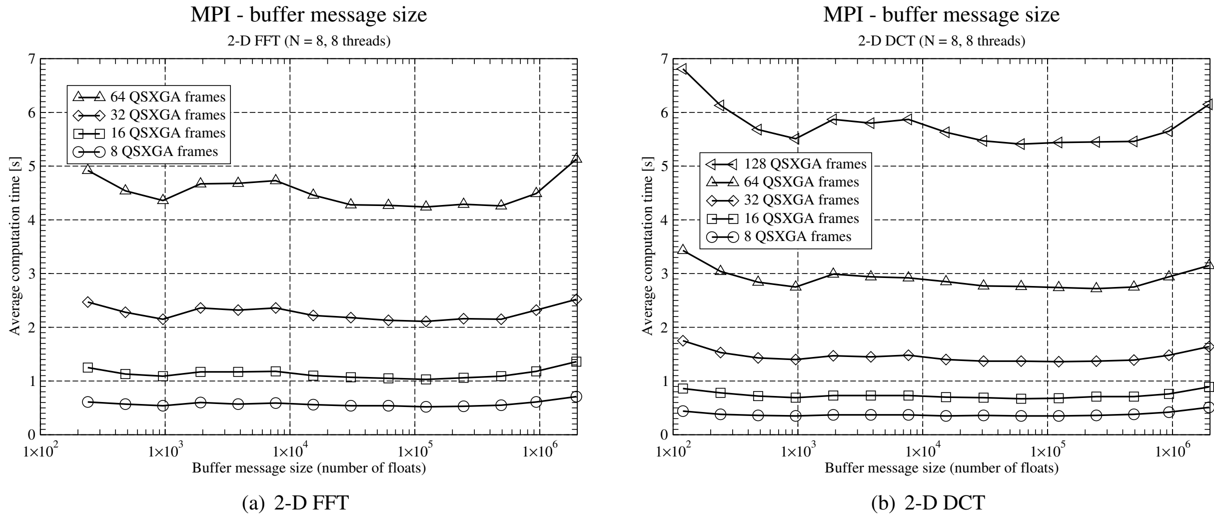
6.3 MATLAB Environment

The third test case was performed in a MATLAB environment. The MATLAB built-in functions `fft` and `dct` were 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 one to eight threads for the blocks of vectors with the lengths of 8, 16, 32, 64, 128, 256, 512, 1,024 and 2,048 are depicted in Fig. 5. It is obvious, that the parallel computing is most advantageous for the vector length of 8, because there is the highest number of blocks to be computed and the individual vectors are quite short. So the time to compute the corresponding FFT's and DCT's is short, but the number of runs, dependent on the amount of data to be computed, is high. For the other vector lengths, the time increases when computing by 8 threads. It is caused by redundant communication between threads which should be eliminated in newer releases of the Parallel Computing Toolbox. For the lengths of 512 and more, the parallel approach is unnecessary, because there are fewer loop runs and the vectors are long, so the FFT and DCT functions take a long time to be computed themselves and the parallelization of these calculations is not very effective. The FFT and DCT functions in MATLAB are optimized, so their computational time itself is as short as possible. A single QSXGA color frame could be transformed approximately in 5 s by both FFT and DCT functions.

6.4 Digital Signal Processors

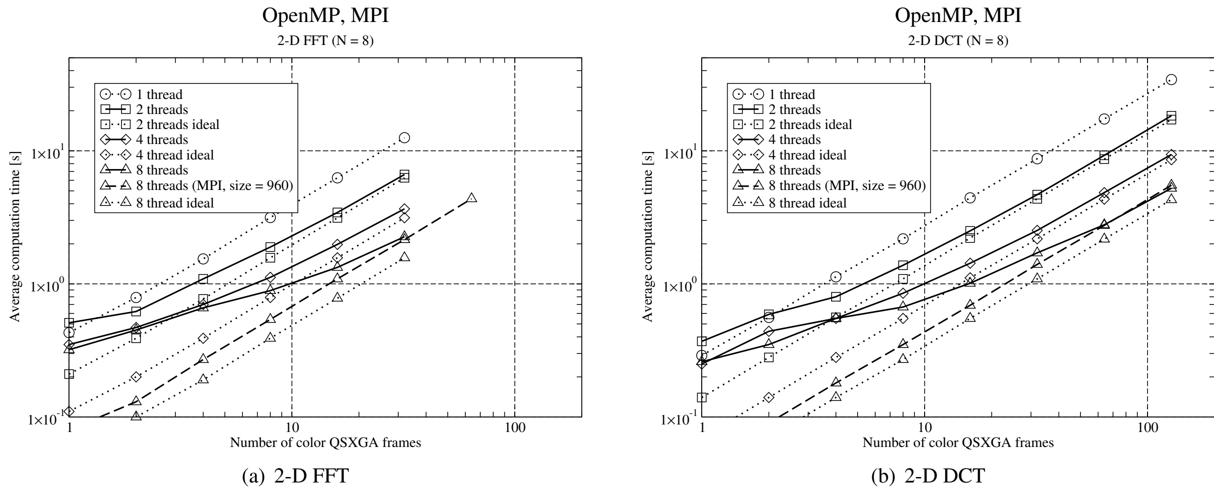
The last test considered was performed using the digital signal processor TMS320C6747, controlled using a clock frequency of 300 MHz (9-times slower than the CPU based tests). Although the evaluation board contains only a single core DSP, the VLIW architecture meets the parallel approach. Selected algorithms were implemented in C language, in linear assembly language, and in assembly language generated by a suggested generator tool. Development tool Code Composer Studio v.3.3 from Texas Instruments was used. Higher level codes were optimized using CCS internal tools as well.

This new auxiliary tool was used for developing an efficient implementation of FFT decimation-in-time algorithm with length of $N = 4$, 8, and 16 and DCT algorithm with length of $N = 8$. All codes were furthermore hand optimized and FFT routines were also compared with official Texas Instruments function `DSPF_sp_cfftr2_dit` from DSP library [9]. The comparison was done using the number of needed CPU cycles, which were enumerated with the help



(a) 2-D FFT (b) 2-D DCT

Fig. 3. Average computation time for two-dimensional MPI implementations with varying buffer message size ($N = 8$, $f_{CPU} = 2.7$ GHz, 8 threads, QSXGA color frames: 2,560×2,048 pixels).



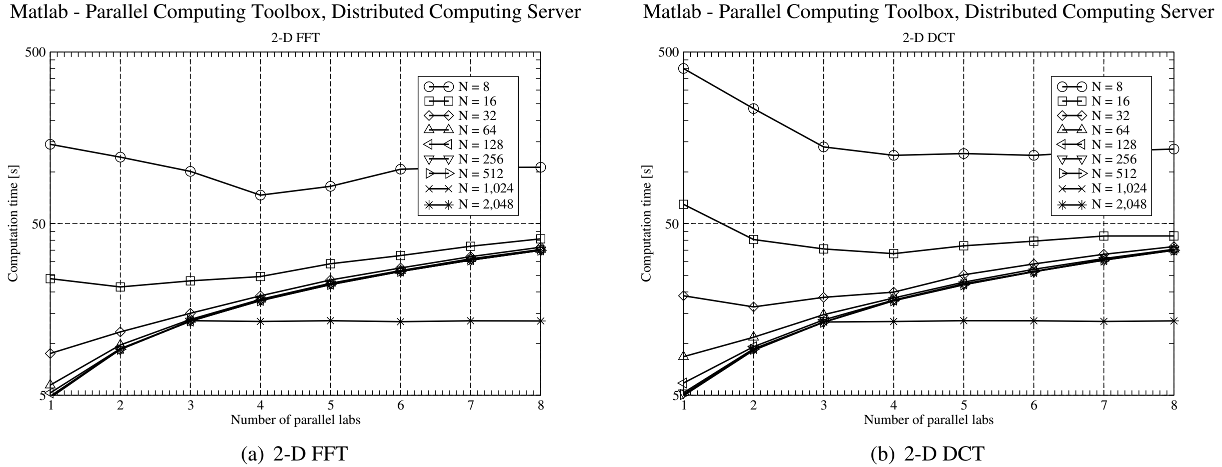
(a) 2-D FFT (b) 2-D DCT

Fig. 4. Average computation time for two-dimensional OpenMP implementations with varying transformed frames and threads number ($N = 8$, $f_{CPU} = 2.7$ GHz, QSXGA color frames: 2,560×2,048 pixels).

Algorithm	Programming language/tool	Cycles theoretical / measured	Remark
FFT, $N = 4$	Assembly code generator	– / 24	2×, normal order output
FFT, $N = 8$	C code	– / 540	`-o3` optimization
FFT, $N = 8$	Linear assembly	– / 126	`-o3` optimization
FFT, $N = 8$	Assembly code generator	– / 42	2×, normal order output
FFT, $N = 16$	Assembly code generator	– / 116	2×, normal order output
FFT, $N = 32$	Assembly code generator	350 estimated	2×, normal order output
DSPLib, $N = 32$	Assembly language	362 / 512	1×, bit-reverse order output
DSPLib, $N = 64$	Assembly language	810 / 1 016	1×, bit-reverse order output
DSPLib, $N = 128$	Assembly language	1 834 / 2 143	1×, bit-reverse order output
DSPLib, $N = 256$	Assembly language	4 138 / 4 658	1×, bit-reverse order output

Tab. 1. Velocity of complex single precision FFT algorithm implementations (TMS320C6747 floating-point DSP, $f_{DSP} = 300$ MHz).

of CCS v.3.3 profiling tool. The achieved results can be seen in Tab. 1. The optimization process was based on the idea that as many functional units as possible are executing a single instruction in every CPU cycle. Considering a limited number of assigned instructions for L, S, M, and D function units, a proposed parallel algorithm must be assembled with respect to the number of needed cycles, pipeline stage, and available general purpose registers.



(a) 2-D FFT (b) 2-D DCT

Fig. 5. Computation time for two-dimensional MATLAB implementations with varying parallel lab number (f_{CPU} = 2.7 GHz, 1 QSXGA color frame: 2,560×2,048 pixels).

In contrary to the `DSPF_sp_cfftr2_dit` function, the proposed solution can store the output frequency coefficients directly in normal order and not in bit-reverse order. The output coefficients normal-order addressing mode could be achieved using a specific storing routine which was included to the proposed function. With the help of a pointer with pre/post-incrementing/decrementing index, arbitrary coefficients order could be suggested.

Furthermore, proposed functions can operate only in one DSP data path. The important impact of this strategy is the opportunity to execute two independent data streams simultaneously: one in data path A, and the second in data path B. Due to this, the objective algorithms' velocity of proposed functions is doubled. Suppose a DSP core with clock frequency of f_{DSP} = 300 MHz. With the proposed FFT function more than 11.5 million executions of FFT $N = 8$ can be performed every second. According to functional units' support, the average workload for the proposed FFT DIT $N = 8$ algorithm is 50 %. The achieved computing performance of the fast DCT algorithm can be seen in Tab. 2 and the application of both FFT and DCT transforms in QSXGA color frame encoding is shown in Tab. 3.

It is obvious, that the general abstraction brought by the C code is not effective. The low-level programming of both FFT and DCT algorithms represents an outstanding contribution in signal processing. A single QSXGA frame could be transformed in 0.28 s by FFT, and in 0.24 s by the DCT method.

7. Conclusion

The paper focused on multiplatform approaches for effective parallel computing. The outline of currently used methods for parallel computing on a CPU was performed as well. The MPI, OpenMP, and MATLAB approaches were taken into account. The goal of the paper was also to present the possibility to create an interconnection between CPU based methods and VLIW architecture DSP evaluation boards. The computing performance of the parallel methods was tested by two transforms, commonly used in signal processing domain. The two-dimensional FFT and DCT were chosen. The optimal conditions for MPI, OpenMP and MATLAB approaches were tested. It was proved, using the more sophisticated MPI approach, that better computation times can be achieved. The dependency between hardware parameters (mainly the cache size) and processing time was also demonstrated. Besides the CPU approach, the implementations were optimized for a digital signal processor with very long instruction word architecture as well. To achieve an optimal workload of the DSPs involved, a low level programming approach had to be applied. The assembly code of the proposed functions were generated by a new auxiliary tool, which facilitates the exploitation of general purpose registers and parallel functional units. New routines were tested on fixed/floating-point processor C6747 with development board OMAP-L137. The achieved results were compared with Texas Instruments function from floating point DSP library. With the help of C6747, the proposed routines double the performance of previous functions. Future work would be focused mainly on the implementation of digital signal processing algorithms to graphical processing units as well as to compare with other CPUs, such as Intel quad-core Xeon e5640.

Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 230126. Research published in this paper was also financially supported by the project CZ.1.07/2.3.00/20.0007 WICOMT of the operational program Education for compet-

Algorithm	Programming language/tool	Cycles	Remark
DCT, $N = 8$	C code	233	`-o3` optimization
DCT, $N = 8$	Linear assembly	78	`-o3` optimization
DCT, $N = 8$	Assembly code generator	37	2×, normal order output

Tab. 2. Velocity of real single precision DCT algorithm implementations (TMS320C6747 floating-point DSP, $f_{DSP} = 300$ MHz).

Algorithm	Programming language	Computation time [s]
2-D FFT, $N = 8$	C code	7.08
2-D FFT, $N = 8$	Linear assembly	1.65
2-D FFT, $N = 8$	Assembly code generator	0.28
2-D DCT, $N = 8$	C code	3.05
2-D DCT, $N = 8$	Linear assembly	1.02
2-D DCT, $N = 8$	Assembly code generator	0.24

Tab. 3. Computation time for two-dimensional FFT and DCT implementations with varying programming approaches (TMS320C6747 floating-point DSP, $f_{DSP} = 300$ MHz, 1 QSXGA color frame: 2,560×2,048 pixels).

itiveness 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] Intel. *Intel 22nm Technology*. [Online] Cited 2012-03-14. Available at: http://www.intel.com/content/www/us/en/silicon-innovations/intel-22nm-technology.html.

[2] MPI Forum. *Message Passing Interface Forum*. [Online]. Cited 2012-03-14. Available at: http://www.mpi-forum.org/.

[3] Message Parsing Interface Forum. MPI: A Message-Passing Interface standard. *The International Journal of Supercomputer Applications and High Performance Computing*, 1994, vol 8.

[4] SNIR, M., OTTO, S., LEDERMAN, S. H., WALKER, D., DONGARRA, J. *MPI – The Complete Reference.* Cambridge (MA, USA): MIT Press, 1998.

[5] *OpenMP*. [Online]. Cited 2012-03-14. Available at: http://openmp.org/wp/.

[6] CHAPMAN, B., JOST, G., VAN DER PAR, R. *Using OpenMP – Portable Shared Memory Parallel Programming.* Cambridge (MA, USA): MIT Press, 2007.

[7] BLAISE, B. *OpenMP*. [Online]. Cited 2012-03-14. Available at: https:// computing.llnl.gov/tutorials/openMP/.

[8] *FFTW Home Page*. [Online]. Cited 2012-03-14. Available at: http://www.fftw.org/.

[9] Texas Instruments. *TMS320C67x DSP Library*. [Online]. Cited 2012-03-14. Available at: http://www.ti.com/tool/sprc121.

[10] MathWorks. *MATLAB and Simulink for Technical Computing*. [Online]. Cited 2012-03-14. Available at: http://www.mathworks.com/.

[11] COOLEY, J. W., TUKEY, J. W. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 1965, vol. 19, no. 90, p. 297 - 301.

[12] CHU, E., GEORGE, A. *Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms (Computational Mathematics).* Boca Raton (USA): CRC Press, 1999.

[13] DUHAMEL, P., HOLLMANN, H., Split radix FFT algorithm. *Electronics Letters*, 1984, vol. 20, no. 1, p. 14 - 16.

[14] RAO, K. R., YIP, P. *Discrete Cosine Transform. Algorithms, Advantages, Applications.* San Diego (USA): Academic Press, 1990.

[15] ARAI, Y., AGUI, T., NAKAJIMA, M. A fast DCT-SQ scheme for images. *IEICE Transactions (1976–1990)*, 1988, vol. E71-E, no. 11, p. 1095 - 1097.

[16] CHEN, W.-H., SMITH, C. H., FRALICK, S. C. A fast computational algorithm for the discrete cosine transform. *IEEE Transactions on Communications*, 1977, vol. 25, no. 9, p. 1004 - 1009.

[17] LOEFFLER, C., LIGHTENBERG, A., MOSCHYTZ, G. Practical fast 1-D DCT algorithms with 11 multiplications. In *International Conference on Acoustics, Speech and Signal Processing, ICASSP 1989.* Glasgow (UK), 1989, vol. 2, p. 988 - 991.

[18] VETTERLI, M. Fast 2-D discrete cosine transform. In *International Conference on Acoustics, Speech and Signal Processing, ICASSP 1985*. Tampa (USA), 1985, p. 1538 - 1541.

[19] GONZALEZ, R. C., WINTZ, P. *Digital Image Processing.* Boston (USA): Addison Wesley, 1987.

[20] Texas Instruments. *OMAP-L137, C6-Integra DSP+ARM Processor*. [Online]. Cited 2012-03-14. Available at: http://www.ti.com/product/omap-l137.

[21] Texas Instruments. *TMS320C674x DSP CPU and Instruction Set*. [Online]. Cited 2012-03-14. Available at: http://www.ti.com/litv/pdf/sprufe8b.

About Authors...

Tomas FRYZA was born in 1977 in Novy Jicin, Czech Republic. He received his M.Sc. and Ph.D. degrees in Electrical Engineering from the Faculty of Electrical Engineering and Communication, Brno University of Technology in 2002 and in 2006, respectively. At present he is an associate professor at the Department of Radio Electronics, Brno University of Technology. His research interests include digital and microprocessor techniques, parallel programming, source code optimization, and digital signal processing. He has been an IEEE member since 2003.

Jitka SVOBODOVA was born in 1984 in Olomouc, Czech Republic. She received her M.Sc. degree in Electrical Engineering from the Faculty of Electrical Engineering and Communication, Brno University of Technology in 2009. She is interested in parallel computing, neural networks and optimization.

Filip ADAMEC was born in 1983 in Celadna, Czech Republic. He received his M.Sc. degree in Electrical Engineering from the Faculty of Electrical Engineering and Communication, Brno University of Technology in 2008. At present he is a Ph.D. student at the Department of Radio Electronics, Brno University of Technology. His research interests include digital and microprocessor techniques, digital hardware design and programming.

Roman MARSALEK was born in 1976 in Brno, Czech Republic. He received his M.Sc. in Control and Measurements in 1999 from Brno University of Technology and the Ph.D. equivalent degree (Doctorat) in Electronics and Signal Processing from Université de Marne la Vallée, France in 2003. After his habilitation in 2008, he is currently working as an associate professor at Brno University of Technology. His research is oriented to signal processing applied to digital communications, the power amplifier linearization and multicarrier system communications.

Jan PROKOPEC was born in 1978 in Rychnov nad Kneznou, Czech Republic. He graduated at Brno University of Technology in 2001, where he also received the Ph.D. in 2006. He is currently an assistant professor at Department of Radio Electronics, Brno University of Technology. His research interests are mobile communication systems and parallel programming. He has been an IEEE member since 2003.