V.M. GLUSHKOV INSTITUTE OF CYBERNETICS
OF NATIONAL ACADEMY OF SCIENCE OF UKRAINE

TERESHCHENKO ANDRIY

OPTIMIZATION OF PARALLEL MULTI-DIGIT ALGORITHMS

Senior developer, Simcorp Ukraine,
Front Office (Ukraine)

Ph.D. Physical and Mathematical sciences
(theoretical basis of informatics and
cybernetics)

Dyalog'12 Conference, 17 October, Elsinore

AGENDA

- Education, articles, conferences
- Definitions, terms
- Scope of using multi-digit operations (arithmetic)
- APL advantages
- Scalable Data-Parallel Computing Using GPUs
- Formula 80/20. Multi-digit multiplication based on FFT (Fast Fourier Transform)
- Multi-digit multiplication (example 1 of optimization). Standard and diagonal methods
- Multi-digit multiplication (example 2 of optimization). Karatsuba-Ofman method.
- Example of multiplication 4-digit numbers based on Karatsuba-Ofman method.
- Multi-digit cyclic convolution
- Multiplication algorithm based on multi-digit cyclic convolution
- Computation of convolution length of N=2n based on method Pitassi-Devisa
- Analysis of algorithm based on fast Haar transform
- Finding approach for Walsh transform calculation
- Computation scheme of convolution length of 8
- APL is a tool to find approach
- Improvement in computational scheme
- Final computation scheme of convolution length of 8
- APL describes parallel models
- Analysis of FFT algorithm
- References

EDUCATION, ARTICLES, CONFERENCES

1991-1995: Sumy state university, faculty – automation of manufacture, specialty – industrial electronics.
2004-2007: V.M. Glushkov Institute of Cybernetics
2010 Thesis “The fast proceeding algorithms of multi-digit arithmetic”.

Languages: APL, Assembler, FORTRAN, Pascal, Perl, C, FoxPro, Clipper
Assembler, C – fast execution
FoxPro, Clipper – database calls
Perl – text files (like HTML)
APL, FORTRAN, MathCad – describing mathematical models
APL (CUDA C, OpenCl library) – describing parallel models

Articles:
Control systems and machines: 2006 № 3, 4
Computer mathematic: 2006 № 3, 4; 2008 № 1; 2010 № 1
Artificial intellect: 2006 № 3; 2009 № 1; 2010; 2011; 2012 № 3
Problems of control and informatics: 2010 № 2

Conferences (with appearing):
Young scientists: 2005 (Kiev, KPI)
Artificial intellect (international): 2006, 2008, 2010, 2012 (Crymya, Kaciveli)
Optimization of calculations: 2007, 2009, 2011 (Crymya, Kaciveli)

DEFINITIONS, TERMS

Optimization – method that gives possibility to reduce complexity (the number of operations executed by one processor) in such way that original algorithm executed faster on a computer. Parallel model of computation is taken into account as well.

Multi-digit number is value that is allocated in more than one byte (16, 32, 64,128-bit word). Operands of single and double precision operations are considered as one-digit number. High precision numbers like:

1,23456789012345678901234567890123456789…E324…

is part of multi-digit numbers.

Arithmetic – multi-digit operations: multiplication, addition, subtraction, convolution, correlation, etc. Main focus is on convolution. There will be giving some theory to describe convolution operation as simple as possible.

Algorithm… The optimization was needed from the moment when the first algorithm was built.

SCOPE OF USING MULTI-DIGIT OPERATIONS (ARITHMETIC)

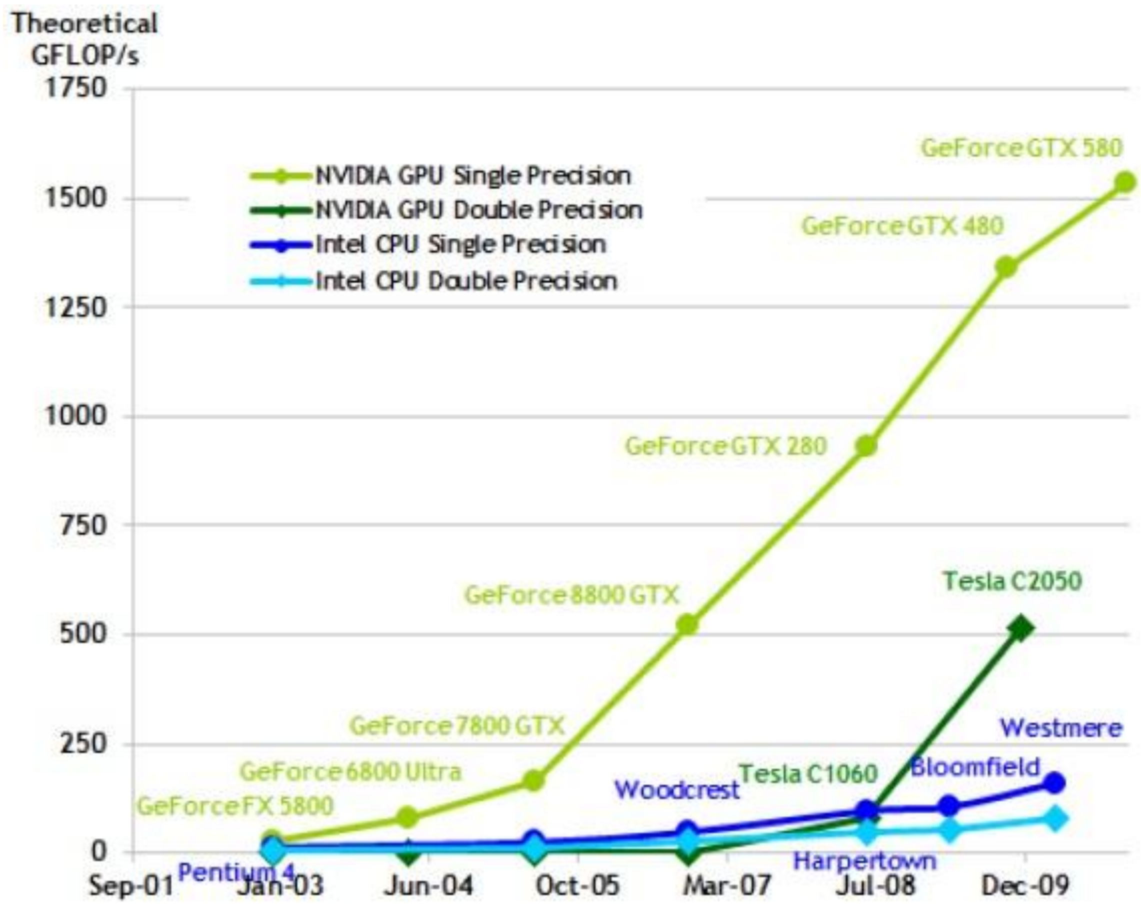
1. Two-key cryptography:
   - Encryption/decryption;
   - Generation EDS;
   - Verifying EDS;
   - Authentication;
   - Cryptographic protocols.
2. High precision computations:
   - Analysis of error of rounding.
3. Modeling of processes:
   - Physical, chemical (biochemical), aerodynamics, hydrodynamics, astronomic computations.

APL ADVANTAGES

- APL is cool
- Code is very close to mathematical formulas
- Reduces time to transform mathematical models to code and vice versa
- Shows complicated models in simple way
- Analysis complicated algorithms
- Describing parallel algorithms
- Pen and paper are not needed due to nature of APL

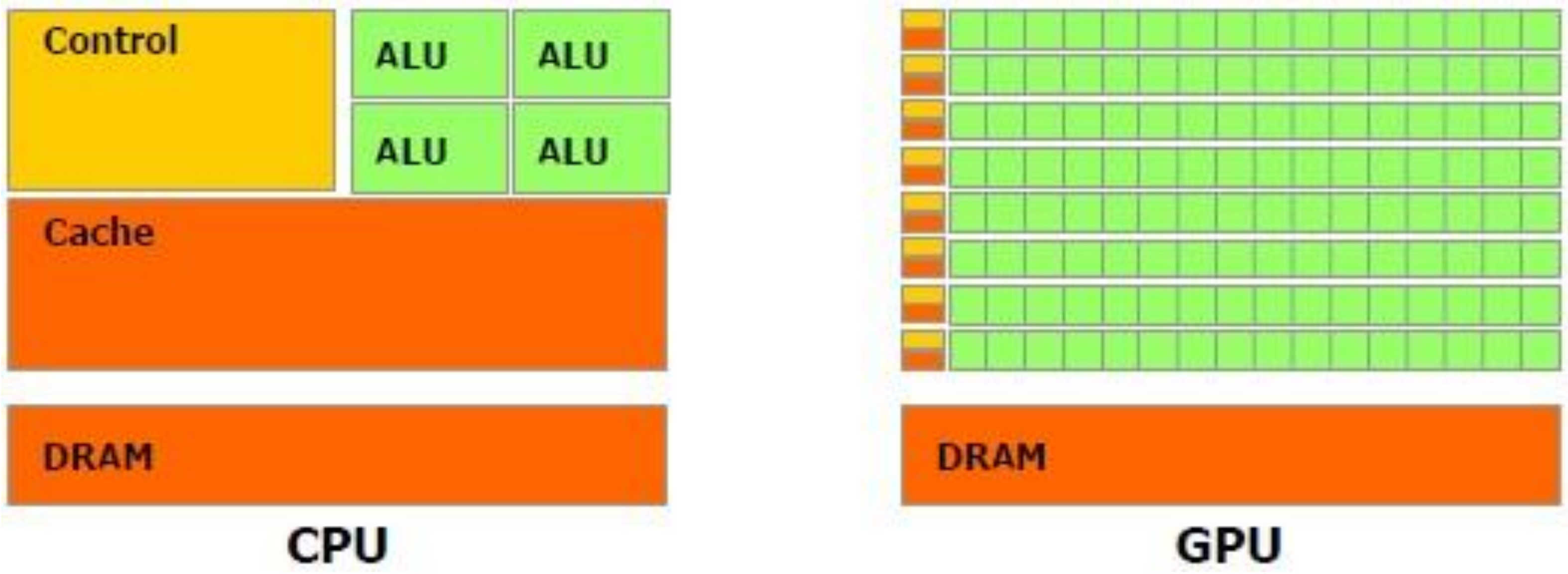
SCALABLE DATA-PARALLEL COMPUTING USING GPUs

Driven by the insatiable market demand for real-time, high-definition graphics, the programmable graphics processing unit (GPU) has evolved into a highly parallel, multithreaded, many-core processor with tremendous computational horsepower and very high memory bandwidth.



The floating-point operations per second for CPU and GPU

GPU is especially well-suited to address problems that can be expressed as data-parallel computations to speed up processing large data sets (SIMD - Single Instruction, Multiple Data). The effort in general-purpose computing on the GPU (GPGPU) has positioned the GPU as a compelling alternative to traditional microprocessors in high-performance computer systems of the future.



The GPU devotes more transistors to data processing

Amdahl's law specifies the maximum speed-up $S = \frac{1}{(1-P) + P/N}$, where P is the fraction of the total serial execution time taken by the portion of code that can be parallelized and N is the number of processors over which the parallel portion of the code runs.

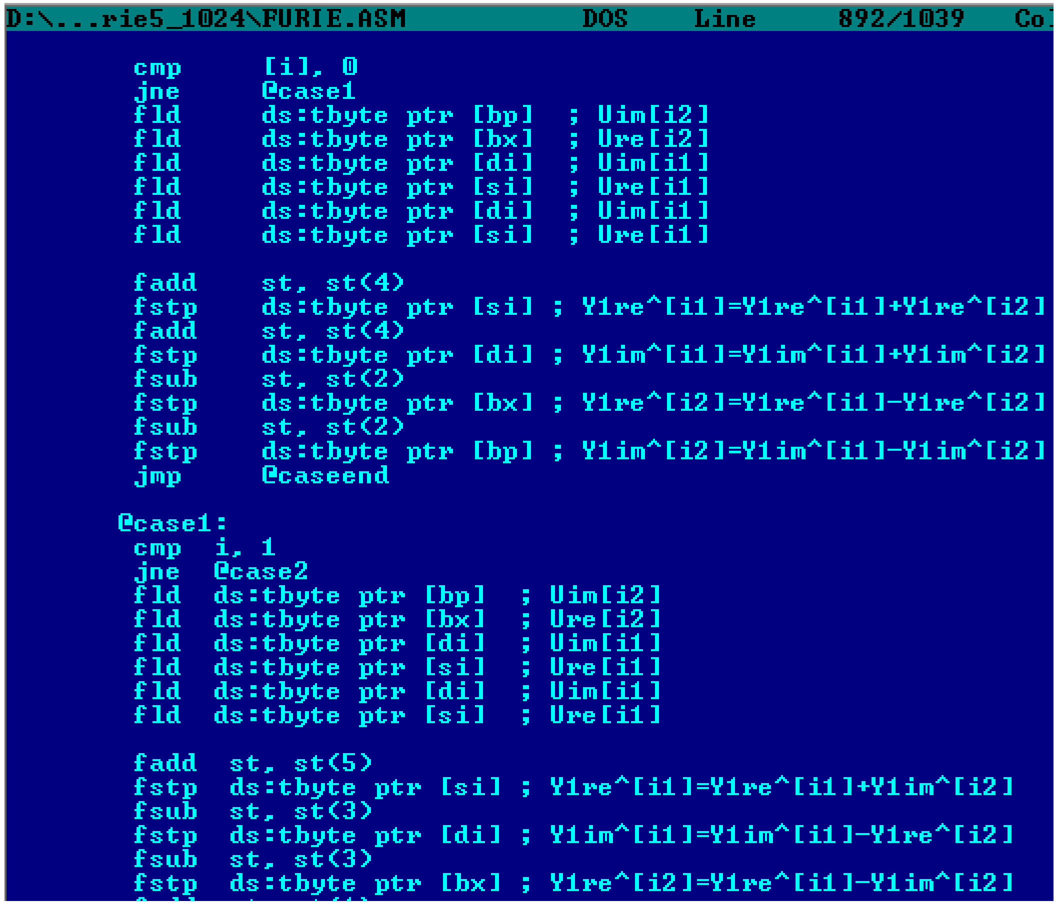
Formula 80/20
Multi-digit multiplication based on FFT (Fast Fourier Transform)

Language	Time for mathematical model	Time for development
Assembler	20%	80%
Pascal	20%	80%
APL	80%	20%

Turbo assembler v.2.71
Furie.asm
There are more than 1000 lines
(28 pages)

Turbo Pascal v.7.1
m_new2l9.pas
There are more than 500 lines
(14 pages)

Dyalog APL/W v.10.1.1
FFTMainF2
There are more than 250 lines
(7 pages)



```
D:\...rie5_1024\FURIE.ASM          DOS     Line      892/1039    Co

        cmp     [i], 0
        jne     @case1
        fld     ds:tbyte ptr [bp]  ; Uim[i2]
        fld     ds:tbyte ptr [bx]  ; Ure[i2]
        fld     ds:tbyte ptr [di]  ; Uim[i1]
        fld     ds:tbyte ptr [si]  ; Ure[i1]
        fld     ds:tbyte ptr [di]  ; Uim[i1]
        fld     ds:tbyte ptr [si]  ; Ure[i1]

        fadd    st, st(4)
        fstp    ds:tbyte ptr [si] ; Y1re^[i1]=Y1re^[i1]+Y1re^[i2]
        fadd    st, st(4)
        fstp    ds:tbyte ptr [di] ; Y1im^[i1]=Y1im^[i1]+Y1im^[i2]
        fsub    st, st(2)
        fstp    ds:tbyte ptr [bx] ; Y1re^[i2]=Y1re^[i1]-Y1re^[i2]
        fsub    st, st(2)
        fstp    ds:tbyte ptr [bp] ; Y1im^[i2]=Y1im^[i1]-Y1im^[i2]
        jmp     @caseend

       @case1:
        cmp   i, 1
        jne   @case2
        fld   ds:tbyte ptr [bp]  ; Uim[i2]
        fld   ds:tbyte ptr [bx]  ; Ure[i2]
        fld   ds:tbyte ptr [di]  ; Uim[i1]
        fld   ds:tbyte ptr [si]  ; Ure[i1]
        fld   ds:tbyte ptr [di]  ; Uim[i1]
        fld   ds:tbyte ptr [si]  ; Ure[i1]

        fadd  st, st(5)
        fstp  ds:tbyte ptr [si] ; Y1re^[i1]=Y1re^[i1]+Y1im^[i2]
        fsub  st, st(3)
        fstp  ds:tbyte ptr [di] ; Y1im^[i1]=Y1im^[i1]-Y1re^[i2]
        fsub  st, st(3)
        fstp  ds:tbyte ptr [bx] ; Y1re^[i2]=Y1re^[i1]-Y1im^[i2]
```

There are screenshots of the same computations in Pascal and APL:

```
Xc←FFTTransformF2 arg;Cv;Bv;N;S;j;d;n;p;k;s1;s2;r;i1;i2;X;W
Xc Cv Bv N←arg
S←(sqrt 2)÷2 ⋄ j←1 ⋄ d←N÷2 ⋄ n←log2 N

:For p :In 0,⍳n-1
    :For k :In 0,⍳j-1
        s1←k×(2×d) ⋄ s2←s1+d
        :For r :In 0,⍳d-1
            i1 i2←(s1 s2)+r
            X←Xc complGet i2
            :Select k
            :Case 0
                X←X                    ⍝ X×1
            :Case 1
                X←X complMul(0,-1)     ⍝ X×(-i)
            :Case 2
                X←X complMul(S,-S)     ⍝ X×(1-i)×sqrt 2  //-pi÷4
            :Case 3
                X←X complMul((-S),-S)  ⍝ X×-(1+i)×sqrt 2 //-3×pi÷4
            :Else
                W←Cv GetW k            ⍝ rad←-(Bv[k+1])×PI÷N
                X←X complMul W         ⍝ cosv←cos rad ⋄ sinv←sin rad
            :EndSelect                 ⍝ W←(cosv,sinv) ⋄ X←X×W

            Xc←Xc i2 i1 complSetGetSub X
            Xc←Xc i1 i1 complSetGetAdd X
        :EndFor
    :EndFor
    j←j×2 ⋄ d←d÷2
:EndFor
```

```
D:\...\M_NEW\M_NEW2L9.PAS *         DOS     Line        337/505    Col
{-----------------------------------------------------------------------
{ procedure FFT3 computatio of DFT complex sequence Wre^, -Wim^
{-----------------------------------------------------------------------
 il:=1;
 nl:=n2;
 for lmf:=0 to mf-1 do
 begin
  for i:=0 to il-1 do
  begin ipn:=2*i*nl; ipk:=ipn+nl;
   for j:=0 to nl-1 do
  begin
    i1:=(ipn+j)*2; i2:=(ipk+j)*2;
    ytre:=Wre^[i2];ytim:=Wre^[i2+1];
        case i of
    0:begin fs:=ytre; fs1:= ytim; end;
    1:begin fs:=ytim; fs1:=-ytre; end;
    2:begin fs := (ytre+ytim)*S1;
            fs1:= (ytim-ytre)*S1; end;
    3:begin fs := (ytim-ytre)*S1;
            fs1:=-(ytim+ytre)*S1; end;
    else begin
            if i mod 2 = 0 then begin cosv:=cre[i];sinv:=-cre[i+1]
                                else begin sinv:=cre[i-1]; cosv:=cre[i]
            fs :=ytre*cosv+ytim*sinv;
            fs1:=ytim*cosv-ytre*sinv; end;
    end;
    Wre^[i2]:=Wre^[i1]-fs; Wre^[i2+1]:=Wre^[i1+1]-fs1;
    Wre^[i1]:=Wre^[i1]+fs; Wre^[i1+1]:=Wre^[i1+1]+fs1;
   end;
  end;
  nl:=nl div 2; il:=il*2;
 end;
```

APL takes much less space (lines) to develop the same computation.
APL code looks:

- compact;
- closer to mathematical model;
- easier to understand, analyze and, as a result, improve.

Note. Using complex arithmetic from Dyalog APL/W v.12 the APL code would look much simpler.

MULTI-DIGIT MULTIPLICATION (EXAMPLE 1 OF OPTIMISATION). STANDARD AND DIAGONAL METHODS

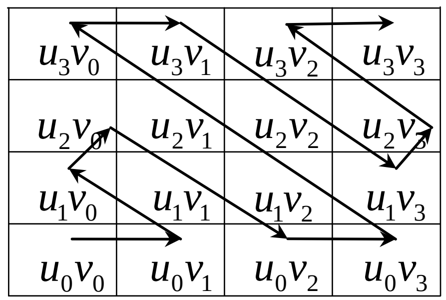
Consider computation:

$$R_{2N} = \left(\sum_{i=0}^{N-1} u_i 2^{\omega i}\right) \cdot \left(\sum_{i=0}^{N-1} v_i 2^{\omega i}\right) = \sum_{i=0}^{2N-1} r_i 2^{\omega i},$$

where U_N, V_N, R_{2N} – N- and $2N$-digit positive integers: $U_N = (u_{N-1}u_{N-2}...u_0) = \sum_{i=0}^{N-1} u_i 2^{\omega i}$, $V_N = (v_{N-1}v_{N-2}...v_0) = \sum_{i=0}^{N-1} v_i 2^{\omega i}$, $R_{2N} = (r_{2N-1}r_{2N-2}...r_0) = \sum_{i=0}^{2N-1} r_i 2^{\omega i}$, ω – number of bits in one word (ω=16, 24, 32 or 64 bits), $0 \le u_i, v_i, r_i < 2^{\omega}$. The complexity is N^2 one-word multiplications.

				u_3	u_2	u_1	u_1
				v_3	v_2	v_1	v_1
				u_3v_0	u_2v_0	u_1v_0	u_0v_0
			u_3v_1	u_2v_1	u_1v_1	u_0v_1	
		u_3v_2	u_2v_2	u_1v_2	u_0v_2		
	u_3v_3	u_2v_3	u_1v_3	u_0v_3			
[illegible]	r_6	r_5	r_4	r_3	r_2	r_1	r_0

Standard method of multiplication of two 4-digit numbers ($4N^2 + 3N$ memory reads needed)



Diagonal scheme of multiplication of two 4-digit numbers ($2N^2 + 2N$ memory reads needed)

Each diagonal is calculated on registers that reduces the number of memory reads. It gives possibility to reduce performance twice and this kind of optimization is not considered.

MULTI-DIGIT MULTIPLICATION (EXAMPLE 2 OF OPTIMISATION) KARATSUBA-OFMAN METHOD

It gives possibility to reduce complexity.

Let`s consider U_{2N} and V_{2N} – positive integers, each of them is allocated in $2N$ ω-bit words. The numbers U_{2N} and V_{2N} could be shown as:

$$U_{2N} = H(U_{2N}) \cdot 2^{\omega N} + L(U_{2N}),\ V_{2N} = H(V_{2N}) \cdot 2^{\omega N} + L(V_{2N}),$$

where operators $H(U_{2N})$, $H(V_{2N})$ and $L(U_{2N})$, $L(V_{2N})$ gives high and low parts of U_{2N}, V_{2N}, respectively.

If abbreviations $X = 2^{\omega N}$, $HU = H(U_{2N})$, $HV = H(V_{2N})$, $LU = L(U_{2N})$, $LV = L(V_{2N})$ are used, than multiplication $U_{2N} \cdot V_{2N}$ could be shown as:

$$U_{2N} \cdot V_{2N} = (H(U_{2N}) \cdot 2^{\omega N} + L(U_{2N})) \cdot (H(V_{2N}) \cdot 2^{\omega N} + L(V_{2N})) = (HU \cdot X + LU) \cdot (HV \cdot X + LV) =$$
$$= HU \cdot HV \cdot X^2 + (HU \cdot HV + LU \cdot LV - (HU - LU) \cdot (HV - LV)) \cdot X + LU \cdot LV$$

or

$$U_{2N} \cdot V_{2N} = (H(U_{2N}) \cdot 2^{\omega N} + L(U_{2N})) \cdot (H(V_{2N}) \cdot 2^{\omega N} + L(V_{2N})) =$$
$$= HU \cdot HV \cdot X^2 + (HU + LU) \cdot (HV + LV) - HU \cdot HV - LU \cdot LV) \cdot X + LU \cdot LV.$$

There are three N-word operations of multiplications ($HU \cdot HV$ and $LU \cdot LV$ are repeated twice) instead of four N-word operations using standard method. 25% of multiplications are reduced using one level of splitting $2N$-word number into two N-word numbers. Using splitting on more levels reduces complexity more.

If $N = 2^n$ than it is needed 3^n one-word simple multiplications but that`s required more $4N\sum_{i=0}^{n-1}\left(\frac{3}{2}\right)^n$ additions (and subtractions). This method has a limit of using due to additional operations needed for recursive calls.

EXAMPLE OF MULTIPLICATION 4-DIGIT NUMBERS BASED ON KARATSUBA-OFMAN METHOD

$$U_{2N} \cdot V_{2N} = (H(U_{2N}) \cdot 2^{\omega N} + L(U_{2N})) \cdot (H(V_{2N}) \cdot 2^{\omega N} + L(V_{2N})) = (HU \cdot X + LU) \cdot (HV \cdot X + LV) =$$

$$= HU \cdot HV \cdot X^2 + (HU \cdot HV + LU \cdot LV - (HU - LU) \cdot (HV - LV)) \cdot X + LU \cdot LV$$

$X = 2^{\omega 2}$, $U = V = (1,1,1,1)$

$HU = HV = LU = LV = (1,1)$

$$\begin{array}{cccc}
 & & 1 & 1 \\
 & & 1 & 1 \\
\cline{3-4}
 & & 1 & 1 \\
 & 1 & 1 & \\
\hline
0 & 1 & 2 & 1
\end{array}$$

Standard method of multiplication of numbers $HU \cdot HV = LU \cdot LV$

$$\begin{array}{cccccccc}
 & & & & 1 & 1 & 1 & 1 \\
 & & & & 1 & 1 & 1 & 1 \\
\cline{5-8}
 & & & & 1 & 1 & 1 & 1 \\
 & & & 1 & 1 & 1 & 1 & \\
 & & 1 & 1 & 1 & 1 & & \\
 & 1 & 1 & 1 & 1 & & & \\
\hline
\mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{3} & \mathbf{2} & \mathbf{1}
\end{array}$$

Standard method of multiplication of two 4-digit numbers

$$\begin{array}{cccccccc}
 & & & & 0 & 1 & 2 & 1 \\
 & & 0 & 1 & 2 & 1 & & \\
 & & 0 & 1 & 2 & 1 & & \\
0 & 1 & 2 & 1 & & & & \\
\hline
\mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{3} & \mathbf{2} & \mathbf{1}
\end{array}$$

Total sum of subtotals

The results are the same using different methods: standard and Karatsuba-Ofman.

Using Karatsuba method it is enough to compute only once 2-digit numbers instead of 4-digit numbers due to structure of numbers (1,1,1,1). Karatsuba method works in the rest cases.

MULTI-DIGIT CYCLIC CONVOLUTION

Operation of cyclic convolution length N of two sequences X_N и Y_N:

$$R_N = X_N \otimes Y_N\,,\ R_N = (r_0,\dots,r_{N-1})\,,\ r_k = \sum_{p=0}^{N-1} x_p y_{\langle p+k\rangle_N}\,,\ k=\overline{0,N-1}.$$

Cyclic convolution length $N=4$ could be shown as:

$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ x_3 & x_0 & x_1 & x_2 \\ x_2 & x_3 & x_0 & x_1 \\ x_1 & x_2 & x_3 & x_0 \end{bmatrix} \cdot \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix},\quad \begin{array}{l} r_0 = x_0\cdot y_0 + x_1\cdot y_1 + x_2\cdot y_2 + x_3\cdot y_3 \\ r_1 = x_0\cdot y_1 + x_1\cdot y_2 + x_2\cdot y_3 + x_3\cdot y_0 \\ r_2 = x_0\cdot y_2 + x_1\cdot y_3 + x_2\cdot y_0 + x_3\cdot y_1 \\ r_3 = x_0\cdot y_3 + x_1\cdot y_0 + x_2\cdot y_1 + x_3\cdot y_2 \end{array},\quad \begin{array}{c|cccc} x_0 & y_0 & y_1 & y_2 & y_3 \\ x_1 & y_1 & y_2 & y_3 & y_0 \\ x_2 & y_2 & y_3 & y_0 & y_1 \\ x_3 & y_3 & y_0 & y_1 & y_2 \\ \hline & r_0 & r_1 & r_2 & r_3 \end{array},\quad \begin{array}{c|cccc} y_0 & x_0 & x_3 & x_2 & x_1 \\ y_1 & x_1 & x_0 & x_3 & x_2 \\ y_2 & x_2 & x_1 & x_0 & x_3 \\ y_3 & x_3 & x_2 & x_1 & x_0 \\ \hline & r_0 & r_1 & r_2 & r_3 \end{array}.$$

where 16 one-word multiplications are needed using standard approach.
There are two approaches to get R_N. The approach on the right is more convenient to describe and make an experiment to find simpler view:

$$\begin{aligned} a_0 &= (x_0 + x_1 + x_2 + x_3)\cdot(y_0 + y_1 + y_2 + y_3), \\ a_1 &= (x_0 + x_1 - x_2 - x_3)\cdot(y_0 + y_1 - y_2 - y_3), \\ a_2 &= (x_0 - x_1 + x_2 - x_3)\cdot(y_0 - y_1 + y_2 - y_3), \\ a_3 &= (x_0 - x_1 - x_2 + x_3)\cdot(y_0 - y_1 - y_2 + y_3), \\ r_0 &= 1/4(a_0 + a_1 + a_2 + a_3), \\ r_1 &= 1/4(a_0 + a_1 - a_2 - a_3) - t\,, \\ r_2 &= 1/4(a_0 - a_1 + a_2 - a_3), \\ r_3 &= 1/4(a_0 - a_1 - a_2 + a_3) + t\,. \end{aligned}$$

$$R_4 \leftarrow \frac{1}{4} W_4 \cdot A_4 + T_4\,,\ A_4 \leftarrow (W_4\cdot X_4)\cdot(W_4\cdot Y_4)\,,$$

$$t \leftarrow (x_1 - x_3)\cdot(y_0 - y_2) = x_1y_0 + x_3y_2 - x_1y_2 - x_3y_0\,,$$

$$X_4 \leftarrow \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix},\ Y_4 \leftarrow \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix},\ W_4 \leftarrow \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix},\ T_4 \leftarrow \begin{bmatrix} 0 \\ -t \\ 0 \\ +t \end{bmatrix}.$$

It looks like APL style due of using vectors and arrays.

It will be shown that APL gives possibility to analyze, develop and improve more complicated algorithms.

MULTIPLICATION ALGORITHM BASED ON MULTI-DIGIT CYCLIC CONVOLUTION

Multiplication of two multi-digit values $U_4 = (u_0, u_1, u_2, u_3)$, $V_4 = (v_0, v_1, v_2, v_3)$ length of 4 based on cyclic convolution $R_8 = (u_0, u_1, u_2, u_3, 0,0,0,0) \otimes (0,0,0,0, v_3, v_2, v_1, v_0)$ could be shown like:

$$
\begin{array}{c|cccccccc}
u_0 & 0 & 0 & 0 & 0 & v_3 & v_2 & v_1 & v_0 \\
u_1 & 0 & 0 & 0 & v_3 & v_2 & v_1 & v_0 & 0 \\
u_2 & 0 & 0 & v_3 & v_2 & v_1 & v_0 & 0 & 0 \\
u_3 & 0 & v_3 & v_2 & v_1 & v_0 & 0 & 0 & 0 \\
0 & v_3 & v_2 & v_1 & v_0 & 0 & 0 & 0 & 0 \\
0 & v_2 & v_1 & v_0 & 0 & 0 & 0 & 0 & v_3 \\
0 & v_1 & v_0 & 0 & 0 & 0 & 0 & v_3 & v_2 \\
0 & v_0 & 0 & 0 & 0 & 0 & v_3 & v_2 & v_1 \\
\hline
 & r_0 & r_1 & r_2 & r_3 & r_4 & r_5 & r_6 & r_7
\end{array}
$$

Multiplication of two 4-digit values based on convolution

$$
\begin{array}{c|cccccccc}
u_0 & 0 & 0 & 0 & 0 & v_3 & v_2 & v_1 & v_0 \\
u_1 & 0 & 0 & 0 & v_3 & v_2 & v_1 & v_0 & 0 \\
u_2 & 0 & 0 & v_3 & v_2 & v_1 & v_0 & 0 & 0 \\
u_3 & 0 & v_3 & v_2 & v_1 & v_0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
 & r_0 & r_1 & r_2 & r_3 & r_4 & r_5 & r_6 & r_7
\end{array}
$$

Multiplication of two 4-digit values Based on convolution with zero lines

$$
\begin{array}{c|cccccccc}
u_0 & & & & & v_3 & v_2 & v_1 & v_0 \\
u_1 & & & & v_3 & v_2 & v_1 & v_0 & \\
u_2 & & & v_3 & v_2 & v_1 & v_0 & & \\
u_3 & & v_3 & v_2 & v_1 & v_0 & & & \\
\hline
 & r_0 & r_1 & r_2 & r_3 & r_4 & r_5 & r_6 & r_7
\end{array}
$$

Multiplication of two 4-digit values based on convolution without zero lines

$$
\begin{array}{cccccccc}
 & & & & u_3 & u_2 & u_1 & u_0 \\
 & & & & v_3 & v_2 & v_1 & u_0 \\
\hline
 & & & & u_0v_3 & u_0v_2 & u_0v_1 & u_0v_0 \\
 & & & u_1v_3 & u_1v_2 & u_1v_1 & u_1v_0 & \\
 & & u_2v_3 & u_2v_2 & u_2v_1 & u_2v_0 & & \\
 & u_3v_3 & u_3v_2 & u_3v_1 & u_3v_0 & & & \\
\hline
r_7 & r_6 & r_5 & r_4 & r_3 & r_2 & r_1 & r_0
\end{array}
$$

Multiplication of based on standard Method

COMPUTATION OF CONVOLUTION LENGTH OF N=2^n BASED ON METHOD PITASSI – DEVISA

Input and output sequences of convolution $R_N = X_N \otimes Y_N$, $N = 2^n$, are linked:

$$E(R_N) = E(X_N) \otimes E(Y_N) + O(X_N) \otimes O(Y_N),\ O(R_N) = E(X_N) \otimes O(Y_N) + O(X_N) \otimes U(E(Y_N)).$$

$$E(X_8) = \begin{bmatrix} x_0 \\ x_2 \\ x_4 \\ x_6 \end{bmatrix},\ O(X_8) = \begin{bmatrix} x_1 \\ x_3 \\ x_5 \\ x_7 \end{bmatrix},\ U(Y_4) = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_0 \end{bmatrix},\ D(Y_4) = \begin{bmatrix} y_3 \\ y_0 \\ y_1 \\ y_2 \end{bmatrix}.$$

$$E(R_N) = 1/2\,(A_{N/2} + S_{N/2}),$$

$$O(R_N) = 1/2\,(A_{N/2} - S_{N/2}) + C_{N/2}.$$

$$A_{N/2} = (E(X_N) + O(X_N)) \otimes (E(Y_N) + O(Y_N)),$$

$$S_{N/2} = (E(X_N) - O(X_N)) \otimes (E(Y_N) - O(Y_N)),$$

$$C_{N/2} = O(X_N) \otimes (U(E(Y_N)) - E(Y_N)).$$

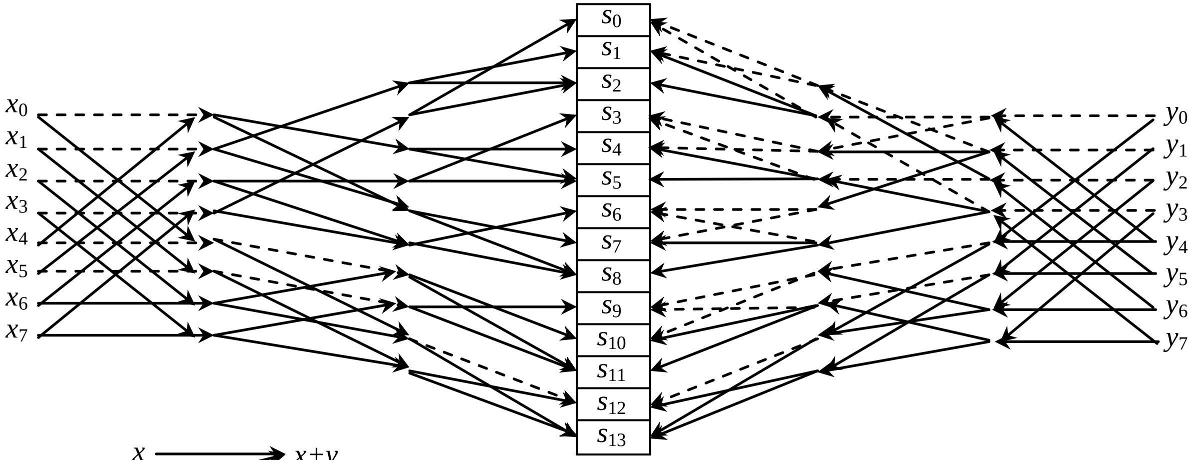
y_0	x_0	x_7	x_6	x_5	x_4	x_3	x_2	x_1
y_1	x_1	x_0	x_7	x_6	x_5	x_4	x_3	x_2
y_2	x_2	x_1	x_0	x_7	x_6	x_5	x_4	x_3
y_3	x_3	x_2	x_1	x_0	x_7	x_6	x_5	x_4
y_4	x_4	x_3	x_2	x_1	x_0	x_7	x_6	x_5
y_5	x_5	x_4	x_3	x_2	x_1	x_0	x_7	x_6
y_6	x_6	x_5	x_4	x_3	x_2	x_1	x_0	x_7
y_7	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7

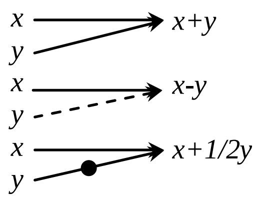
Standard method of computation of convolution length of N=8

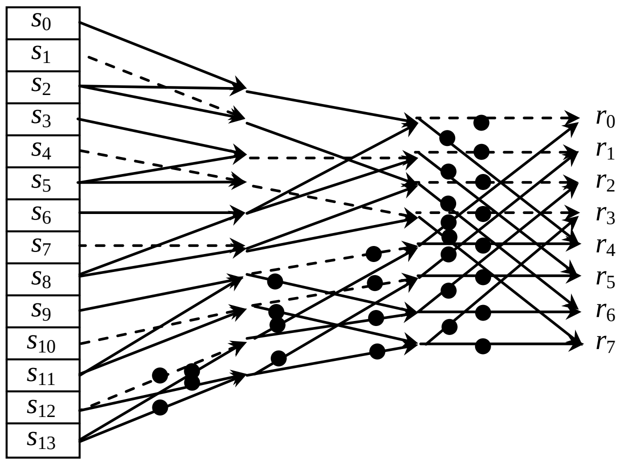
$E(Y_N)$	y_0	x_0	x_6	x_4	x_2	x_7	x_5	x_3	x_1
	y_2	x_2	x_0	x_6	x_4	x_1	x_7	x_5	x_3
	y_4	x_4	x_2	x_0	x_6	x_3	x_1	x_7	x_5
	y_6	x_6	x_4	x_2	x_0	x_5	x_3	x_1	x_7
$O(Y_N)$	y_1	x_1	x_7	x_5	x_3	x_0	x_6	x_4	x_2
	y_3	x_3	x_1	x_7	x_5	x_2	x_0	x_6	x_4
	y_5	x_5	x_3	x_1	x_7	x_4	x_2	x_0	x_6
	y_7	x_7	x_5	x_3	x_1	x_6	x_4	x_2	x_0
		r_0	r_2	r_4	r_6	r_1	r_3	r_5	r_7
		$E(R_N)$				$O(R_N)$			

Parisection method of computation of convolution length of N=8

ANALYSIS OF ALGORITHM BASED ON FAST HAAR TRANSFORM (XX3FHAARXY)







The user should adopt
for the language.
APL could be easily adopted
for the user`s needs.

```
      xx3fhaarxy
+ 8 x0y7+ 8 x1y0+ 8 x2y1+ 8 x3y2+ 8 x4y3+ 8 x5y4+ 8 x6y5+ 8 x7y6
+ 8 x0y6+ 8 x1y7+ 8 x2y0+ 8 x3y1+ 8 x4y2+ 8 x5y3+ 8 x6y4+ 8 x7y5
+ 8 x0y5+ 8 x1y6+ 8 x2y7+ 8 x3y0+ 8 x4y1+ 8 x5y2+ 8 x6y3+ 8 x7y4
+ 8 x0y4+ 8 x1y5+ 8 x2y6+ 8 x3y7+ 8 x4y0+ 8 x5y1+ 8 x6y2+ 8 x7y3
+ 8 x0y3+ 8 x1y4+ 8 x2y5+ 8 x3y6+ 8 x4y7+ 8 x5y0+ 8 x6y1+ 8 x7y2
+ 8 x0y2+ 8 x1y3+ 8 x2y4+ 8 x3y5+ 8 x4y6+ 8 x5y7+ 8 x6y0+ 8 x7y1
+ 8 x0y1+ 8 x1y2+ 8 x2y3+ 8 x3y4+ 8 x4y5+ 8 x5y6+ 8 x6y7+ 8 x7y0
+ 8 x0y0+ 8 x1y1+ 8 x2y2+ 8 x3y3+ 8 x4y4+ 8 x5y5+ 8 x6y6+ 8 x7y7
```

```
yt←xt←calct←(0 1)⍴0
⍝X     0 1 2 3 4 5 6 7  (my)
xt⍪←⊂'       -       +' ⍝ 0
xt⍪←⊂'   -       +    ' ⍝ 1
xt⍪←⊂'   -   -   +   +' ⍝ 2
xt⍪←⊂'     -       +  ' ⍝ 3
xt⍪←⊂' -       +      ' ⍝ 4
xt⍪←⊂' -   -   +   +  ' ⍝ 5
xt⍪←⊂'     - -     + +' ⍝ 6
xt⍪←⊂' - -     + +    ' ⍝ 7
xt⍪←⊂' - - - - + + + +' ⍝ 8
xt⍪←⊂'   -   +   -   +' ⍝ 9
xt⍪←⊂' -   +   -   +  ' ⍝ 10
xt⍪←⊂' - - + + - - + +' ⍝ 11
xt⍪←⊂' - + - + - + - +' ⍝ 12
xt⍪←⊂' + + + + + + + +' ⍝ 13
⍝Y     0 1 2 3 4 5 6 7
yt⍪←⊂' - - + - + + - +' ⍝ 0
yt⍪←⊂' + - + + - + - -' ⍝ 1
yt⍪←⊂' +     + -     -' ⍝ 2
yt⍪←⊂' - + - + + - + -' ⍝ 3
yt⍪←⊂' - + + - + - - +' ⍝ 4
yt⍪←⊂'     + -     - +' ⍝ 5
yt⍪←⊂'   +   +   -   -' ⍝ 6
yt⍪←⊂'   +   -   -   +' ⍝ 7
yt⍪←⊂'       -       +' ⍝ 8
yt⍪←⊂' + + - - + + - -' ⍝ 9
yt⍪←⊂' + - - + + - - +' ⍝ 10
yt⍪←⊂'   -   +   -   +' ⍝ 11
yt⍪←⊂' - + - + - + - +' ⍝ 12
yt⍪←⊂' + + + + + + + +' ⍝ 13
⍝XY       0 1 2 3 4 5 6 7 8 9 A B C D
calct⍪←⊂'-4  -4      -4  -4-2  -2-1 1'
calct⍪←⊂'       4   4-4  -4   2-2 1 1'
calct⍪←⊂'   4-4         4-4 2   2-1 1'
calct⍪←⊂'        -4 4   4-4  -2 2 1 1'
calct⍪←⊂' 4   4       4   4-2  -2-1 1'
calct⍪←⊂'      -4  -4 4   4   2-2 1 1'
calct⍪←⊂'  -4 4          -4 4 2   2-1 1'
calct⍪←⊂'        4-4  -4 4  -2 2 1 1'
```

FINDING APPROACH FOR WALSH TRANSFORM CALCULATION (XX3FWALSH5/6G)

Developing algorithm based on Walsh transform was possible due to using APL as approach to analyze, check results and improve code and mathematical model.

```
AX0     0 1 2 3 4 5 6 7  (my)
xt1⍪←⊂' + + + + + + + +' ⍝ 0
xt1⍪←⊂' + + + + - - - -' ⍝ 1
xt1⍪←⊂' + + - - + + - -' ⍝ 2
xt1⍪←⊂' + + - - - - + +' ⍝ 3
xt1⍪←⊂'     2 2    -2-2' ⍝ 3
xt1⍪←⊂' + - + - + - + -' ⍝ 0
xt1⍪←⊂' + - + - - + - +' ⍝ 1
xt1⍪←⊂' + - - + + - - +' ⍝ 2
xt1⍪←⊂' + - - + - + + -' ⍝ 3
xt1⍪←⊂'     2-2    -2 2' ⍝ 3

xt1⍪←⊂'   2   2   2   2' ⍝ 0
xt1⍪←⊂'   2   2  -2  -2' ⍝ 1
xt1⍪←⊂'   2  -2   2  -2' ⍝ 2
xt1⍪←⊂'   2  -2  -2   2' ⍝ 3
xt1⍪←⊂'       4      -4' ⍝ 0
```

```
AY      0 1 2 3 4 5 6 7  (my)
yt1⍪←⊂' + + + + + + + +' ⍝ 0
yt1⍪←⊂' + + + + - - - -' ⍝ 1
yt1⍪←⊂' + + - - + + - -' ⍝ 2
yt1⍪←⊂' + + - - - - + +' ⍝ 3
yt1⍪←⊂' 2 2    -2-2    ' ⍝ 0
yt1⍪←⊂' + - + - + - + -' ⍝ 0
yt1⍪←⊂' + - + - - + - +' ⍝ 1
yt1⍪←⊂' + - - + + - - +' ⍝ 2
yt1⍪←⊂' + - - + - + + -' ⍝ 3
yt1⍪←⊂' 2-2    -2 2    ' ⍝ 0

yt1⍪←⊂'                ' ⍝ 0
yt1⍪←⊂'-2       2      ' ⍝ 1
yt1⍪←⊂'-2   2  -2   2  ' ⍝ 2
yt1⍪←⊂'     2      -2  ' ⍝ 3
yt1⍪←⊂'-2   2   2  -2  ' ⍝ 0
```

```
AX0        0 1 2 3 4 5 6 7 8 9 A B C D E
calct1⍪←⊂' 1 1 1 1   1 1 1 1            '
calct1⍪←⊂' 1 1-1-1-1 1 1-1-1-1          '
calct1⍪←⊂' 1-1 1-1   1-1 1-1            '
calct1⍪←⊂' 1-1-1 1 1 1-1-1 1 1          '
calct1⍪←⊂' 1 1 1 1  -1-1-1-1   1 1 1 1  '
calct1⍪←⊂' 1 1-1-1-1-1-1 1 1 1 1 1-1-1-1'
calct1⍪←⊂' 1-1 1-1  -1 1-1 1   1-1 1-1  '
calct1⍪←⊂' 1-1-1 1 1-1 1 1-1-1 1-1-1 1 1'
```

Sorting is needed to get data looked like transform

That`s better to use FWT (instead of Haar transform) as there are a lot of common parts

```
xt1⍪←⊂' + + + + + + + +' ⍝ a0 w0
xt1⍪←⊂' + - + - + - + -' ⍝ a1 w4
xt1⍪←⊂' + + - - + + - -' ⍝ a2 w2
xt1⍪←⊂' + - - + + - - +' ⍝ a3 w6
xt1⍪←⊂' + + + + - - - -' ⍝ a4 w1
xt1⍪←⊂' + - + - - + - +' ⍝ a5 w5
xt1⍪←⊂' + + - - - - + +' ⍝ a6 w3
xt1⍪←⊂' + - - + - + + -' ⍝ a7 w7
xt1⍪←⊂'   2   2   2   2' ⍝ a0-a1  w0-w4
xt1⍪←⊂'   2  -2   2  -2' ⍝ a2-a3  w2-w6
xt1⍪←⊂'   2   2  -2  -2' ⍝ a4-a5  w1-w5
xt1⍪←⊂'   2  -2  -2   2' ⍝ a6-a7  w3-w7
xt1⍪←⊂'     2 2    -2-2' ⍝ a4-a6  aa0
xt1⍪←⊂'     2-2    -2 2' ⍝ a5-a7  aa1
xt1⍪←⊂'       4      -4' ⍝ aa0-aa1
```

```
yt1⍪←⊂' + + + + + + + +' ⍝ a0 w0
yt1⍪←⊂' + - + - + - + -' ⍝ a1 w4
yt1⍪←⊂' + + - - + + - -' ⍝ a2 w2
xy1⍪←⊂' + - - + + - - +' ⍝ a3 w6
xy1⍪←⊂' + + + + - - - -' ⍝ a4 w1
yt1⍪←⊂' + - + - - + - +' ⍝ a5 w5
yt1⍪←⊂' + + - - - - + +' ⍝ a6 w3
yt1⍪←⊂' + - - + - + + -' ⍝ a7 w7
yt1⍪←⊂'                ' ⍝ 0      y6-y0
yt1⍪←⊂'-2   2  -2   2  ' ⍝ 2      y2-y4
yt1⍪←⊂'-2       2      ' ⍝ 1      y0-y2
yt1⍪←⊂'     2      -2  ' ⍝ 3      y4-y6
yt1⍪←⊂' 2 2    -2-2    ' ⍝ a4+a6
yt1⍪←⊂' 2-2    -2 2    ' ⍝ a5+a7
yt1⍪←⊂'-2   2   2  -2  ' ⍝ 0
```

```
AX0        0 1 2 3 4 5 6 7 8 9 A B C D E
calct2⍪←⊂' 1 1 1 1 1 1 1 1              '
calct2⍪←⊂' 1 1-1-1 1 1-1-1        -1-1  '
calct2⍪←⊂' 1-1 1-1 1-1 1-1              '
calct2⍪←⊂' 1-1-1 1 1-1-1 1         1 1  '
calct2⍪←⊂' 1 1 1 1-1-1-1-1 1 1 1 1      '
calct2⍪←⊂' 1 1-1-1-1-1 1 1 1 1-1-1-1 1-1'
calct2⍪←⊂' 1-1 1-1-1 1-1 1 1-1 1-1      '
calct2⍪←⊂' 1-1-1 1-1 1 1-1 1-1-1 1 1-1 1'
```

It was iterative process where on each iteration the mathematical model was improved using APL and APL code was improved using better mathematical approach.

Reducing the number of inverse FWTs (xx3fwalsh6kmOp8).

On initial phase there were used two different fast transforms (Walsh and Hadamar) and the link between both transforms (calcHadamar2Walsh).

```
xx3fwalsh6;flag;len;perv;orgv;i;xy;
⍝ ... Calculation with using Walsh.

k←32
mf←5

x1←k k⍴0
:For i :In ⍳k
    x1[i;i]←1
:EndFor
y1←x1

x1←calcHadamar x1 mf
x1←calcHadamar2Walsh x1 mf
nl←k
iv←il←1
:For lmf :In ⍳mf-2
    row←0
    list←⍳nl÷2
    list2←⍳nl
    listeven←(2|list2)/list2

    :For i :In ⍳iv
        x1⍪←x1[row+list;]-x1[row+list2~list;]
        y1⍪←calcWalshShiftLeft(y1[row+listeven;])
        row+←nl
    :EndFor

    y1←calcWalshStepAll y1 k nl lmf

    nl÷←2
    il×←2
    iv×←3
:EndFor
```

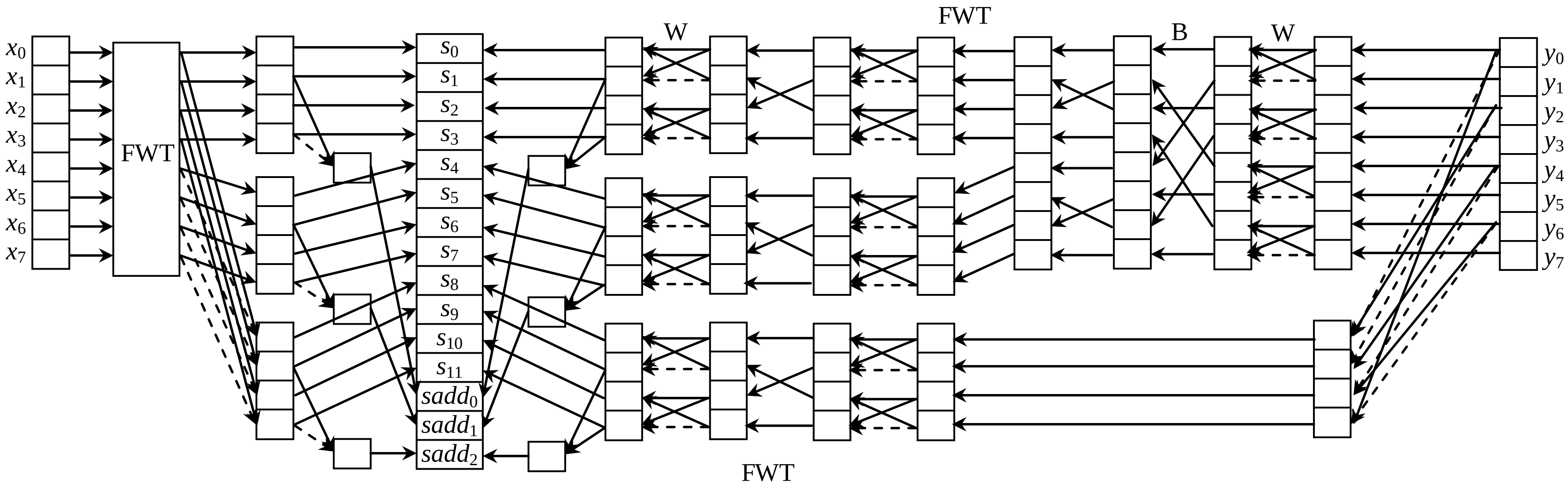
```
⍝X0           0 1 2 3 4 5 6 7 8 9 A B C D E
calct2⍪←⊂' 1 1 1 1 1 1 1 1              '
calct2⍪←⊂' 1 1-1-1 1 1-1-1        -1-1  '
calct2⍪←⊂' 1-1 1-1 1-1 1-1              '
calct2⍪←⊂' 1-1-1 1 1-1-1 1         1 1  '
calct2⍪←⊂' 1 1 1 1-1-1-1-1 1 1 1 1      '
calct2⍪←⊂' 1 1-1-1-1-1 1 1 1 1-1-1-1 1-1'
calct2⍪←⊂' 1-1 1-1-1 1-1 1 1-1 1-1      '
calct2⍪←⊂' 1-1-1 1-1 1 1-1 1-1-1 1 1-1 1'
```

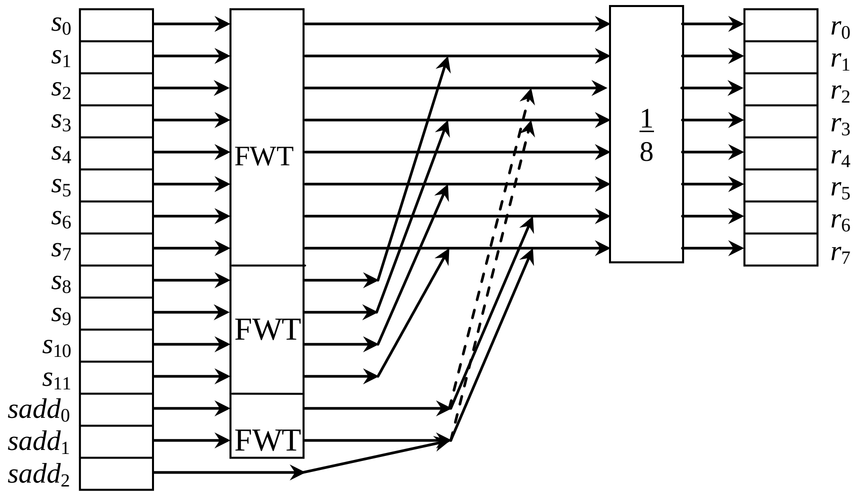
$$CT = \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ \hline 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{array}\right]$$

$$H = \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ \hline 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{array}\right]$$

From developer point of view that`s better to have common subfunctions instead of separate independent functions. The developer need to deliver functions for both fast transforms (Walsh and Hadamar).

COMPUTATIONAL SCHEME OF CONVOLUTION LENGTH OF 8





There are a lot of FWTs different lengths across scheme.

APL IS A TOOL TO FIND APROACH

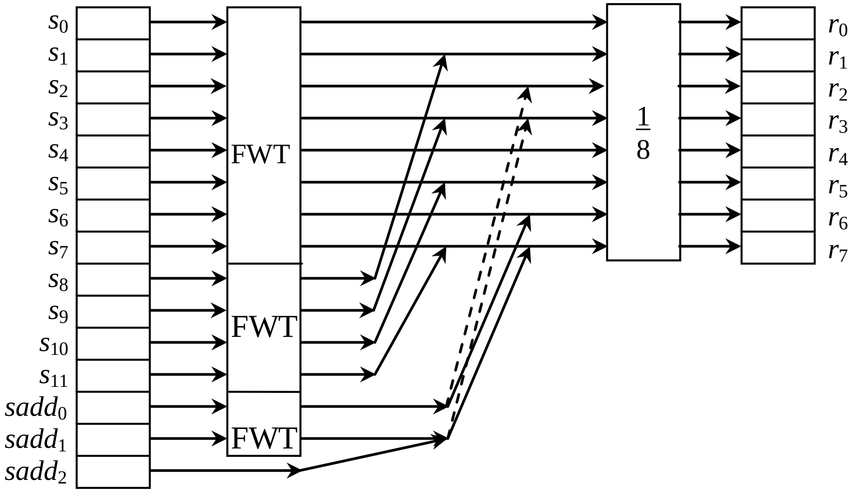
The approach is proved afterwards using mathematical formulas (xx3fwalsh6).

```
statusm←1 5⍴1(k÷4)k 1 k
ixstatus←1

:While 1≤ixstatus
    flag←0
    :If </statusm[ixstatus;1 2]
        flag←1
    :ElseIf =/statusm[ixstatus;1 2]
        :If ixstatus=1
            flag←1
        :Else
            :If =/statusm[ixstatus-1;1 2]
            :AndIf ≠/statusm[ixstatus;1 3]
                flag←1
            :EndIf
        :EndIf
    :EndIf

    :If flag
        statusv←statusm[⊃⍴statusm;]
        ktmp←statusv[3]
        statusm⍪←1(statusv[1])(ktmp÷2)(+/statusm[ixstatu
        ixstatus+←1
      :If ixstatus=2
      :AndIf =/statusm[1;1 2]
         statusm[ixstatus;2 3]÷←2
      :EndIf
      statusm[ixstatus;5]←statusm[ixstatus;3]
   :EndIf

   :If ≥/statusm[ixstatus;1 2]
      rowdist←statusm[ixstatus-1;4]
      rowsrc←statusm[ixstatus;4]
      steps ktmp←statusm[ixstatus;1 3]
          :For i :In ⍳ktmp÷steps
              rowdist+←steps
              :For count :In ⍳steps
                  xy1[rowdist;]+←xy1[rowsrc;]
                  rowdist+←1
                  rowsrc+←1
              :EndFor
          :EndFor

          :If ixstatus=2
          :AndIf 4=÷/statusm[1 2;3]
              rowsrc-←ktmp
              :For i :In ⍳ktmp÷steps
                  rowdist+←steps
                  :For count :In ⍳steps
                      xy1[rowdist;]-←xy1[rowsrc;]
                      rowdist+←1
                      rowsrc+←1
                  :EndFor
              :EndFor
          :EndIf

          statusm[ixstatus-1;1]×←2
          statusm[ixstatus-1;5]+←statusm[ixstatus;5]
          statusm←statusm[⍳ixstatus-1;]
          ixstatus-←1
          :If ixstatus=1
          :AndIf ≥/statusm[ixstatus;1 2]
              ixstatus-←1
          :EndIf
      :EndIf
:EndWhile

prnreal xy1(⍳k)0
```

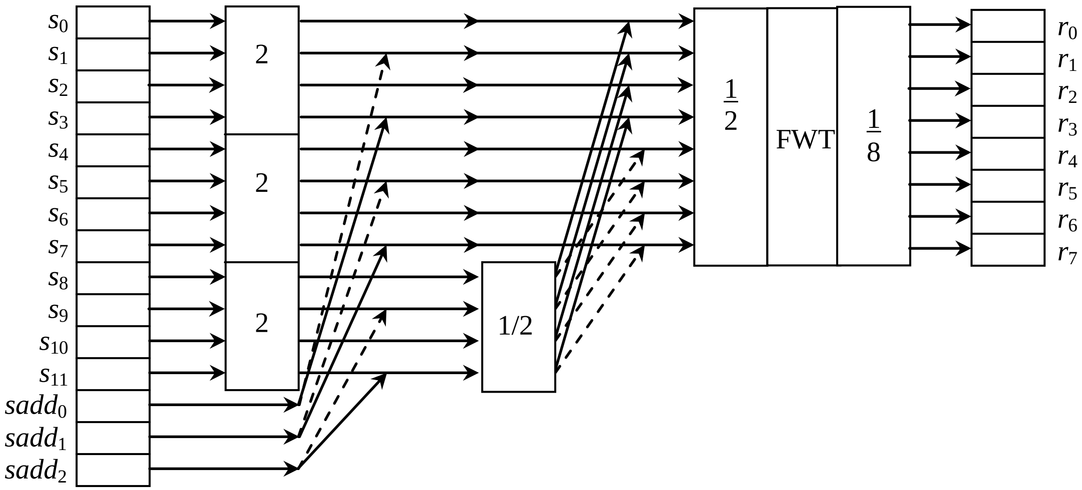
The number of FWTs different length defines complexity of the scheme.

IMPROVEMENT IN COMPUTATIONAL SCHEME

The old one:



The new one:

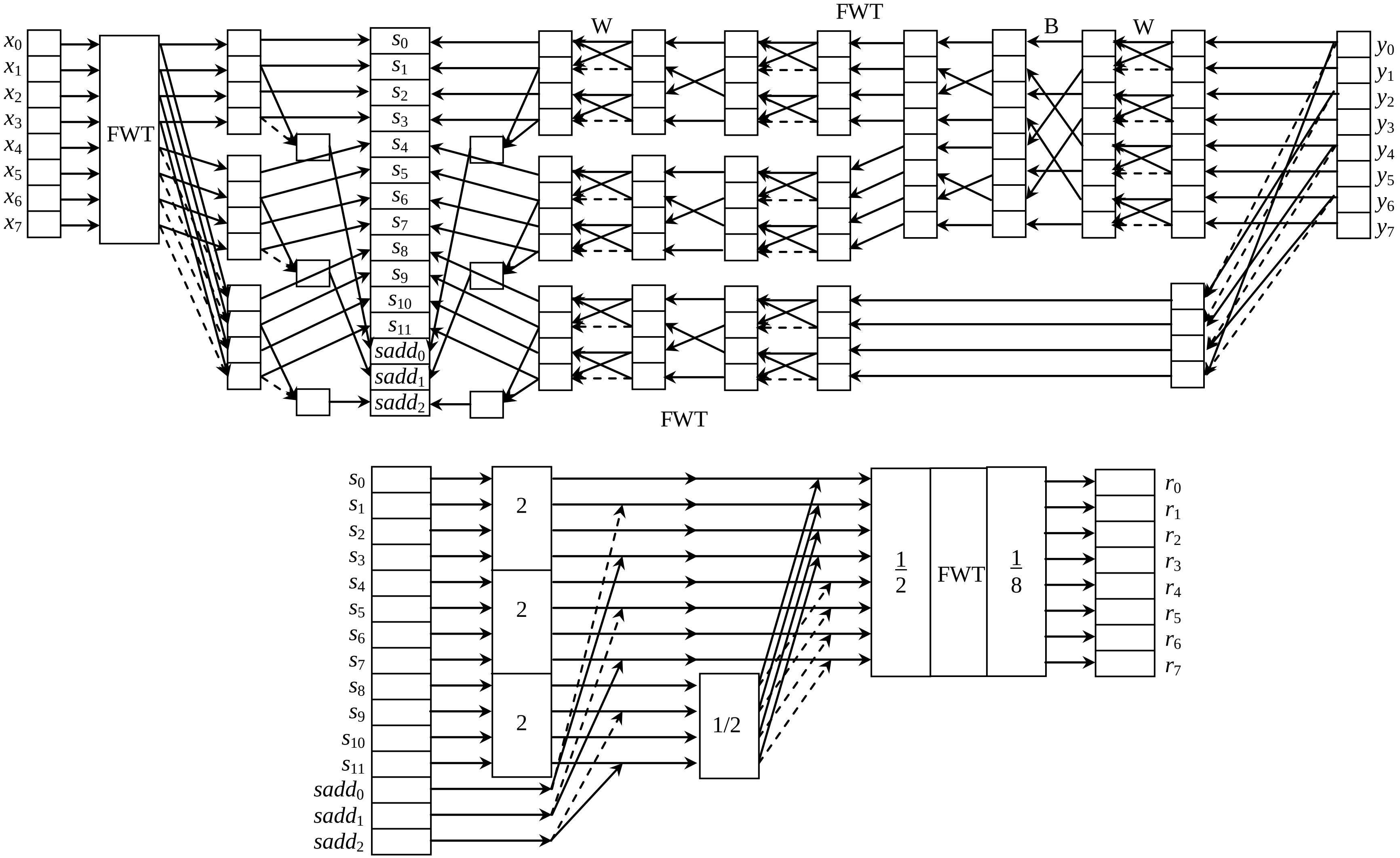



```
⍝ Prepare convolution computition of length 8,...
ktmp←4
:For i :In ⌽⍳n-2
    colsrc←colv[i]
    colsrcnumber←colsrc÷2
    colsrcv←colsrc+⍳colsrcnumber
    ktmpcount←colsrcnumber÷ktmp
    distv←⊃,/(ktmpcount⍴⊂⍳ktmp)+(ktmp×2)×¨(¯1+⍳ktmpcc
    rv[distv]+←rv[colsrcv]÷2
    rv[distv+ktmp]-←rv[colsrcv]÷2
    ktmp×←2
:EndFor
⍝ Computition convolution length of K
rv[⍳k]÷←2                    ⍝ Because of Point (A)
rv[⍳k]←(calcFWT rv[⍳k])÷k
```

There are more coefficients like (2, 1/2, 1/8, etc) but there is only one FWT.

Coefficients (2, 1/2, 1/8, etc) don`t add more multiplications as they could be replaced with bit-shift operations.

FINAL COMPUTATIONAL SCHEME OF CONVOLUTION LENGTH OF 8



Resulted scheme is applicable for parallel computational model as there are a lot of the same blocks. APL is useful tool to get parallel computational model as a result.

ANALYSIS OF FFT ALGORITHM

Algorithm (using Pascal) looks overcomplicated taking into account there are complex numbers. Some part of the code commented as there was attempt to improve algorithm making changes in the code. It takes time.

```
{------------------------------------------------------------
{Find vectors Y1 & Z1 to compute DFT lower length
{------------------------------------------------------------
{  writeln('Move to bigger length');{}
  ytre := Wre^[0]-Wre^[1];
  ztre := Zre^[0]-Zre^[1]; i:=n*2;
  Wre^[i  ] := ytre*ztre;
  Wre^[i+1] := 0;

  ytre := Wre^[0]+Wre^[1];
  ztre := Zre^[0]+Zre^[1];
  Wre^[0] := ytre*ztre;
  Wre^[1] := 0;

  i:=n2*2;
  ytre := Wre^[i]; ytim := Wre^[i+1];
  ztre := Zre^[i]; ztim := Zre^[i+1];
  Wre^[i  ]:= ytre*ztre-ytim*ztim;
  Wre^[i+1]:= ytim*ztre+ytre*ztim;

  prnw(Wre^);{}

  for i:=1 to n2-1 do begin
    t:=lst[i];
    if t mod 2 = 0 then begin cosv:=cre[t  ];sinv:=-cre[t+1];
                   else begin sinv:=cre[t-1];cosv:= cre[t  ];
    i1:=i*2; i2:=(n-i)*2;

{   writeln(cosv:16:5, sinv:16:5);
    cosv:=cos(2*PI*i/k);sinv:=sin(2*PI*i/k);
    writeln(cosv:16:5, sinv:16:5);
    writeln(i:2,(2*PI*i/k):16:5,lst[i]:3);}

    y1i1re :=Wre^[i1]/2; y1i1im := Wre^[i1+1]/2;
    y1i2re := Wre^[i2]/2; y1i2im := Wre^[i2+1]/2;
    z1i1re := Zre^[i1]/2; z1i1im := Zre^[i1+1]/2;
    z1i2re := Zre^[i2]/2; z1i2im := Zre^[i2+1]/2;
```

```
    fs := y1i1re-y1i2re;      ft := z1i1re-z1i2re;
    fs1:= y1i1im+y1i2im;      ft1:= z1i1im+z1i2im;

    y1i1re := y1i1re+y1i2re; z1i1re := z1i1re+z1i2re;
    y1i1im := y1i1im-y1i2im; z1i1im := z1i1im-z1i2im;

    y2i1re := fs*cosv+fs1*sinv;       z2i1re := ft*cosv+ft1*s
    y2i1im := fs*sinv-fs1*cosv;       z2i1im := ft*sinv-ft1*c
{Wre^[i]}
    ytre:= y1i1re-y2i1im;             ztre:= z1i1re-z2i1im;
    ytim:= y1i1im-y2i1re;             ztim:= z1i1im-z2i1re;

    Wre^[i1  ]:= ytre*ztre-ytim*ztim;
    Wre^[i1+1]:=-ytim*ztre-ytre*ztim;
{Wre^[n-i]}
    ytre:= y1i1re+y2i1im;             ztre:= z1i1re+z2i1im;
    ytim:=-y1i1im-y2i1re;             ztim:=-z1i1im-z2i1re;

    Wre^[i2  ]:= ytre*ztre-ytim*ztim;
    Wre^[i2+1]:=-ytim*ztre-ytre*ztim;
  end;

  prnw(Wre^);{}

{------------------------------------------------------------
{Find Y1 & Z1
{------------------------------------------------------------
{ writeln('Move to lower length');{}
  i1:=n*2; i2:=n2*2;
  fs := Wre^[0]/2; fs1 := Wre^[i1]/2;
  Wre^[0 ] := fs+fs1;   Wre^[1 ]:=fs-fs1;
  {Wre^[i2] := Wre^[i2]; Wim^[n2]:=Wim^[n2];}

  for i:=1 to n2-1 do begin
    t:=lst[i];
    if t mod 2 = 0 then begin cosv:=cre[t  ];sinv:=-cre[t+1];
```

It was worth to spent time on rewriting Pascal code and improving algorithm analyzing APL code. The complicated algorithm looks simpler using APL code.

There are two functions similar functions FFTUnpackF2 and FFTPackF2

```
Rv←FFTMainF2;N;Uv;Uv;Cv;Bv;Xc;Yc;Zc
N←16
Uv←Uv←Nρ1

Cv Bv←FFTPreCalcF2 N
Xc Yc←FFTInitF2 Uv Uv N
Xc←FFTTransformF2 Xc Cv Bv N
Yc←FFTTransformF2 Yc Cv Bv N
Xc←FFTBinaryInverseF2 Xc Bv N
Yc←FFTBinaryInverseF2 Yc Bv N
Xc←FFTUnpackF2 Xc Cv Bv N
Yc←FFTUnpackF2 Yc Cv Bv N
Zc←FFTMultiF2 Xc Yc N
Zc←FFTPackF2 Zc Cv Bv N
Zc←FFTComplexF2 Zc N
Zc←FFTTransformF2 Zc Cv Bv N
Zc←FFTBinaryInverseF2 Zc Bv N
Zc←FFTComplexF2 Zc N
Rv←FFTSaveRes2 Zc N
Xc←FFTUnpackF2 arg;Cv;Bv;N;f;X;r;k;W;Xr;XNr;XNrC;A;S
⍝ Transfrom from N to 2N
Xc Cv Bv N←arg
f←1 ⍝ first element index

X←Xc complGet 0
Xc←Xc complSet 0(((Re X)+(Im X)),0)
Xc←Xc complSet N(((Re X)-(Im X)),0)

:For r :In ⍳(N÷2)-1
    k←Bv[f+r]      ⍝ ... K←2×N ⋄ rad←-(2×PI×r)÷K
    W←Cv GetW k    ⍝ ... cosv←cos rad ⋄ sinv←sin rad
                   ⍝ ... W←cosv,sinv
    Xr←Xc complGet r
    XNr←Xc complGet(N-r)
    XNrC←complComplex XNr

    A←(Xr+XNrC)÷2
    S←((Xr-XNrC)÷2)complMul(0,-1)          ⍝ 1÷i=-i
    S←W complMul S

    Xc←Xc complSet r(A+S)
    Xc←Xc complSet(N-r)(complComplex A-S)
:EndFor
```

```
Rv←FFTMainF2Opt;Uv;Uv;Xc;Yc;N;Cv;Bv;Zc
N←16
Uv←Uv←Nρ1

Cv Bv←FFTPreCalcF2 N
Xc Yc←FFTInitF2 Uv Uv N
Xc←FFTTransformF2 Xc Cv Bv N
Yc←FFTTransformF2 Yc Cv Bv N
Xc←FFTBinaryInverseF2 Xc Bv N
Yc←FFTBinaryInverseF2 Yc Bv N
Xc←FFTUnpackPackF2 Xc Cv Bv N 0
Yc←FFTUnpackPackF2 Yc Cv Bv N 0
Zc←FFTMultiF2 Xc Yc N
Zc←FFTComplexF2 Zc N
Zc←FFTUnpackPackF2 Zc Cv Bv N 1
Zc←FFTTransformF2 Zc Cv Bv N
Zc←FFTBinaryInverseF2 Zc Bv N
Zc←FFTComplexF2 Zc N
Rv←FFTSaveRes2 Zc N
Zc←FFTPackF2 arg;Zc;Cv;Bv;N;f;Z0re;ZNre;re;im;r;k;W;Zr;ZNr;ZNrC;A;S
Zc Cv Bv N←arg
f←1 ⍝ first element index

Z0re←Zc complGet 0
ZNre←Zc complGet N
re←((Re Z0re)+(Re ZNre))÷2
im←((Re Z0re)-(Re ZNre))÷2
Zc←Zc complSet 0(re,im)

:For r :In ⍳(N÷2)-1
    k←Bv[f+r]          ⍝ ... K←2×N ⋄ rad←-(2×PI×r)÷K
    W←Cv GetW k        ⍝ ... cosv←cos rad ⋄ sinv←sin rad
                       ⍝ ... W←cosv,sinv
    Zr←Zc complGet r
    ZNr←Zc complGet(N-r)
    ZNrC←complComplex ZNr

    A←(Zr+ZNrC)÷2
    S←((Zr-ZNrC)÷2)complMul(0,-1)          ⍝ 1÷i=-i
    S←(complComplex W)complMul S

    Zc←Zc complSet r(A-S)
    Zc←Zc complSet(N-r)(complComplex A+S)
:EndFor
```

Changing order of FFTPackF2 and FFTComplexF2 gives possibility to use common function for both FFTUnpackF2 and FFTPackF2

```
Xc←FFTUnpackPackF2 arg;Xc;Cv;Bv;N;X;mode;f;X0re;XNre;re;im;r;k;W;Xr;XNr;XNrC;A;S
⍝ Transfrom from N <-> 2N
Xc Cv Bv N mode←arg
f←1 ⍝ first element index

:If mode=0
    X←Xc complGet 0
    Xc←Xc complSet 0(((Re X)+(Im X)),0)
    Xc←Xc complSet N(((Re X)-(Im X)),0)
:Else
    X0re←Xc complGet 0
    XNre←Xc complGet N
    re←((Re X0re)+(Re XNre))÷2
    im←((Re X0re)-(Re XNre))÷2
    Xc←Xc complSet 0(complComlex(re,im))
:EndIf

⍝Xc←Xc complSet(N÷2)(complComlex(Xc complGet(N÷2)))

:For r :In ⍳(N÷2)-1
    k←Bv[f+r]           ⍝ ... K←2×N ⋄ rad←-(2×PI×r)÷K
    W←Cv GetW k         ⍝ ... cosv←cos rad ⋄ sinv←sin rad
                        ⍝ ... W←cosv,sinv
    Xr←Xc complGet r
    XNr←Xc complGet(N-r)
    XNrC←complComplex XNr

    A←(Xr+XNrC)÷2
    S←((Xr-XNrC)÷2)complMul(0,-1)          ⍝ 1÷i=-i
    S←W complMul S

    Xc←Xc complSet r(A+S)
    Xc←Xc complSet(N-r)(complComplex A-S)
:EndFor
```

APL DESCRIBES PARALLEL MODELS

ALGORITHM. Computation of multi-digit convolution length of $N = 2^n$ based on FWT and parallel model.

Input: X_N, Y_N – sequences length of $N = 2^n$, $n \geq 3$.

Output: $R_N \leftarrow X_N \otimes Y_N$ – convolution of X_N, Y_N.

Step 0. Initialization.

$$X \leftarrow ((X_{N=2^n})_0) \leftarrow X_N\,;\; Y \leftarrow ((Y_{N=2^n})_0) \leftarrow Y_N\,.$$

Step 1. FWT of initial section

$$((X_N)_0)\,.\;((X_N)_0) \leftarrow W_N \cdot ((X_N)_0)\,.$$

Step 2. DWT of the rest sections of X as linear combinations of sections of X.

$$X \leftarrow (X, V_{M/2}),\; V_{M/2} \leftarrow L((X_M)_j) - H((X_M)_j),$$

$$j = \overline{0, T-1},\; M = 2^{n-i},\; T = 3^i,\; i = \overline{0, n-3}.$$

Step 3. DWT of the vector Y (except two last iterations).

Step 3a. For i from 0 to $n-3$

Step 3b. $Y \leftarrow (Y, V_{M/2})$, $V_{M/2} \leftarrow U(E(Z_M)) - E(Z_M)$,

Step 3c. $((Y_M)_j) \leftarrow \begin{bmatrix} B(L((Y_M)_j)) \\ \hline B(H((Y_M)_j)) \end{bmatrix}$, $((Y_M)_j) \leftarrow B(W(Z_M))$,

Step 3d. $Z_M \leftarrow ((Y_M)_j)$, $j = \overline{0, T-1}$, $M = 2^{n-i}$, $T = 3^i$.

Step 3e. End For i.

Step 4. Last two iteration of FWT.

$$((Y_4)_j) \leftarrow W(B(W((Y_4)_j))),\; j = \overline{0, 3^{n-2} - 1} \ldots\ldots$$

```
xx3fwalsh6u5g;k;n;xv;yv;ktmp;j;colv;sink;col;leny;listv;xaddv
⍝ ... Multiplication via Walsh (there are only vectors).
⍝ ... Operators E,O,U,L,H,B,W are used.

k n←16 4    ⍝ k=16, n=4

⍝ Input vectors (sequence)
xv←((k÷2)⍴1),((k÷2)⍴0)  ⍝ xv=1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
yv←((k÷2)⍴0),((k÷2)⍴1)  ⍝ yv=0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

xv←calcFWT xv           ⍝ xv=8 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0

⍝ Main loop
ktmp j colv←k 1 ⍬
:For sink :In 0,⍳n-3
    col←0
    leny←⊃⍴yv               ⍝ Get lingth of yv
    colv,←leny

    :For sink :In ⍳j     ⍝ ⍳j     (or ⍳3)  means (1..3)
        listv←col+⍳ktmp ⍝ ⍳ktmp (or ⍳16) means (1..16)
        xv,←(opLv(xv[listv]))-(opHv(xv[listv]))
        yv⍪←(opUv(opEv(yv[listv])))-(opEv(yv[listv]))


        ⍝ Execute one step in Walsh transform
        yv[listv]←opBv opWv yv[listv]
        yv[opLv listv]←opBv opLv yv[listv]
        yv[opHv listv]←opBv opHv yv[listv]
        col+←ktmp           ⍝ col←col+ktmp
    :EndFor

    ktmp÷←2                 ⍝ ktmp←ktmp÷2
    j×←3                    ⍝ j←j×3
:EndFor

⍝ Finish Walsh transformation for yv
yv←opWv⊃,/opBv¨(4 splitv opWv yv)
```

It is very important to have a balance between number of steps (iterations) and number of basic operations on each step to build fast algorithm using parallel model.

REFERENCES

- Rivest R.A., Shamir A., Adleman L. A method for obtaining digital signatures and public-key cryptosysteme / ASS. Comput. Math., Vol. 21, № 2, 1978. – P. 120–126.
- Karatsuba A.A., Ofman U.P. Multi-digit multiplication using automats / DAS USSR, 145 (1962), P. 293–294 [in Russian].
- Pitassi D.A. Fast convolution using the Walsh transform / Applicat. Walsh Functions. – 1971. – April. – P. 130–133.
- Davis W.F. A class of efficient convolution algorithms / Applicat. Walsh Functions. – 1972. – March. – P. 318–329.
- Sadihov R. H., Sharenkov A.V. Fast convolution algorithms / Automatica, № 3, 1986. – P. 71–75.