(Published in Proc. 3d Int. Conf. Parallel Processing and Applied Mathematic "PPAM'99", Kazimierz Dolny, Poland, Sept. 1999)

A Method for Mapping DSP Algorithms into the Pentium MMX Architecture

A.M.Sergyienko*, D.V.Korchev, J.S.Kanevski**

*Department of Computer Engineering, National Technical University of Ukraine, KPI-2020, pr. Peremogy, 37, Kiev, 252056, Ukraine. E-mail: aser@comsys.ntu-kpi.kiev.ua

**Institute of Math. & Computer Science, Technical University of Coszalin, Coszalin, Poland. E-mail: kanievsk@tu.koszalin.pl

Abstract

In the representation a new method for mapping DSP algorithms into MMX architecture is considered. The method is based on the matrix-graph method for mapping regular algorithms into SIMD processor arrays. Then the Pentium MMX architecture is considered as a four 16-bit processor linear array. According to the method, the reduced dependence graph is mapped into configuration of the structure which corresponds to the SIMD structure and to configuration of events. Finally, time slices of the latter are exchanged by assembly instructions of the MMX instructions set. The example of the algorithm mapping illustrates this method, and proves its effectiveness.

1. Introduction

The Intel Pentium MMX microprocessor has the instruction set which is expanded for multimedia applications, called MMX technology. The MMX technology has the facilities to achieve DSP application performance approximately equal to one of the high end signal microprocessors. Usually customers use DSP MMX library functions or hand made assembly codes with MMX instructions. When a specific and complex DSP algorithm is programmed both approaches are ineffective ones because of low load balancing of the processor and labour consumable programming process. In such case the automatic programming tools are needed. But the demand on such tools is still not satisfied.

Architecture of the MMX kernel of the Pentium microprocessor can be considered as a SIMD architecture with the constrained processor number. A new programming tool can be designed on the base of the appropriate method for mapping DSP algorithms into such architecture.

A set of methods for mapping DSP algorithms into application specific structure are known, for example, described in [1], but they do not consider the SIMD architecture. The methods for mapping regular algorithms into constrained systolic type arrays like described in [2,3] can be used for such purpose. But the set of algorithms which can be directly mapped into MMX architecture using these methods is very limited.

In this work the simplified four processor SIMD model of the MMX kernel is selected and a new method for mapping DSP algorithms into it is proposed. This method is deriving by adapting the method for mapping unimodular loop nests into application specific structures, described in [4].

2. Structure of the processor array.

For most DSP applications the architecture of the MMX core can be approximated by the array of four 16- bit processor units (PUs) which is illustrated by the Fig.1.

Here due to the superscalar nature of the Pentium processor, each of PUs is computing simultaneously one or two instructions, which follow from U and V instruction pipelines. The CPU core implements the control flow of the algorithm and calculates the address stream to the cache RAM, which 64 bit quad word is divided to four 16 bit words. The inner structure of each PU is represented by the Fig 2. All of instructions except multiplication are calculated for a single clock cycle. The multiplication is calculated for three clock cycles. The data interprocessor exchange is implemented by shift instructions [5].

3. Method for mapping data flow graphs into application specific structure.

The mapping method compendiously described below is well suited for mapping DSP algorithms into application specific structures and was published in [4,6,7]. In this paper it is adapted to programming MMX applications. Often DSP algorithms are described by data flow graphs (DFG). In DFG operator nodes represent operations of addition or multiplication, a chain of *k* delay nodes represents delay of a signal variable to *k* iterations, edges represent data flows. DFG can be derived by respective mapping of reduced dependence graph G_{AR} of an unimodular loop nest [8]. In the graph G_{AR} also nodes represent operators but weighted by *k* edge represent dependence of the data which is delayed to *k i*iterations.

Both DFG and reduced dependence graph G_{AR} can be represented in *n*- dimensional space Z^n . For most DSP algorithms for one dimensional signal processing it is enough to operate with n=4 dimensional space. Each of *N* nodes of the graph which denotes the algorithm operator is represented by the vector - node $\boldsymbol{K}_i$, i=1,...,N. The coordinates of the vector $\boldsymbol{K}_i$ signify iteration number, clock number in the iteration, processing unit (PU) in which the respective operator is implemented, and its type. Each of *M* edges of the graph which denotes the data dependence or variable moving is represented by the vector - edge $\boldsymbol{D}_j = \boldsymbol{K}_i - \boldsymbol{K}_{i-1}$, j=1,...,M, besides, vector $\boldsymbol{D}_{N+1} = K_1$.

Sets of vectors $\boldsymbol{K}_i$ and $\boldsymbol{D}_j$ form respective matrices *K* and *D* which together with the graph G_{AR} incidence matrix *A* form an algorithm configuration $C_A = (K,D,A)$. The configuration C_A is equal to the composition of structure configuration $C_S = (K_S,D_S,A)$ and configuration of events $C_T = (K_T,D_T,A)$, namely

$$K = \left(K_S^T, K_T^T\right)^T, \qquad D = \left(D_S^T, D_T^T\right)^T;$$

where vector-node $\boldsymbol{K}_{Si} \in K_S$, represent coordinates of PU where *i*-th operator is implemented, vector-edge $\boldsymbol{D}_{Sj} \in D_S$ represent relative coordinates of communication line for *j*-th variable, vector-node $\boldsymbol{K}_{Ti} \in K_T$ represent clock period of this operator implementation and vector-edge $\boldsymbol{D}_{Tj} \in D_T$ represent delay of this variable moving. Another words, configuration C_S represents the graph of the processor structure, and the configuration C_T represents the operator time schedule.

The following definitions and statements are true for configurations C_A, C_S, C_T . The configuration C_A is correct if $\boldsymbol{K}_i \neq \boldsymbol{K}_j$; i,j = 1,...,N, $i \neq j$, i.e. all of vectors-nodes are placed in the space separately.

There is a linear dependence between matrices: $D = KA$; $K = D_oA_o^{-1}$, where A_o is the incidence matrix for the maximum spanning tree of the graph G_{AR}, and D_o is the matrix of vectors-edges of this tree.

Correct configuration C_A can be transformed into equivalent configuration C_A' by any injection function. For example, the following transformations give equivalent configurations: permutations of vectors $\boldsymbol{K}_i$, multiplications of the matrix K and non-singular matrices P.

The sum of vectors-edges $\boldsymbol{D}_j$,which belong to any loop of the graph G_{AR} must be equal to zero.

The configuration C_T is correct if $\boldsymbol{D}_{tj}\geq 0$, where $\boldsymbol{D}_{tj}$ is unweighted dependence vector of the graph G_{AR}, inequality has lexicographic meaning, j =1,...M. Besides, the given algorithm is implemented in pipelined manner correctly iff

$$\forall \boldsymbol{K}_{Tl}\in K_T(\boldsymbol{K}_{Tl}=(i,q)^T, q\in(0,1,...,L-1)), \qquad (1)$$

where $\boldsymbol{K}_{Ti}$ is not incident to edge $\boldsymbol{D}_{Tj}=(p,0)^T$ weighted by p, L is the period of time between two consecutive the same input operand loadings or is the latency of the algorithm implementation.

Searching for algorithm mapping consists in deriving configurations C_A, C_S, C_T which are optimised according to given criterion. Directed searching for optimised configurations is implemented taking into account mentioned above definitions, dependencies and constraints.

At the first stage of the mapping, the searching for the space component C_S is implemented. The forming of the matrix K_S consists of distributing M_k operators of the k-th type among $]M_k/L[$ processing units of the k-th type. As a result, M_S groups of equal columns are formed in the matrix K_S , where M_S is the number of PUs in the resulting structure. The goal of this process is resource allocation and resource assignment.

At the second stage, the time component C_T of the mapping is searched for. Derived matrices K_T and D_T must satisfy the condition of algorithm configuration correctness, correctness of the configuration of events, condition, that the sum of vectors-edges $\boldsymbol{D}_j$,which belong to any loop of the graph G_{AR} must be equal to zero, and condition (1). Besides, if the operator represented by $\boldsymbol{K}_{Tl}$ is calculated for d clock cycles, then the norm $R(\boldsymbol{D}_{Tj})=iL+q$ of the vector $\boldsymbol{D}_{Tj}=(i,q)^T$ must be no less than d . The clock period in which the operator represented by

$\boldsymbol{K}_{Tl}=(i,q)^T$ is implemented is equal to $t = R(\boldsymbol{K}_{Tl})= iL+q$. As a result, the operator schedule is derived.

In a large set of different exemplars of mapping results an optimum mapping is searched. Some heuristics can be applied to derive a quick solution, such as list scheduling, force directed scheduling, loop folding, or left - edge algorithm, etc. [1]. The advantages of this method consist in the following. Both stages of the mapping deriving can be executed in different order or simultaneously providing best optimisation strategy by time constrained scheduling and functional pipelining. The pipelined PUs with the given stage number can be taken into account. After some adaptation this method is well suited for mapping algorithms into MMX architecture.

4. Method for mapping data flow graphs into MMX architecture.

Due to described above MMX structure model the maximum PU loading is achieved by the following conditions. Up to four operators of the same type must be calculated simultaneously. That means that its vector nodes $\boldsymbol{K}_i$ in the algorithm configuration C_A must be different on each other only in the coordinate of the PU number, i.e. they form a line which is perpendicular to the time axis. According to MMX instruction semantic, the data movings must be preferably between registers or memory cells of the same PUs. The data movings between neighbouring PUs are supported by shift instructions. The line of four vector- nodes $\boldsymbol{K}_i$ of equal type like addition, multiplication, etc., and vectors-edges $\boldsymbol{D}_j$, which are incident to them and equal to each other, is mapped to a single MMX instruction.

Also the following must be taken into consideration. Up to two MMX instructions can be calculated simultaneously due to the superscalar nature of the processor. One source operand of the instruction is allocated in the same register as the destination operand is. The irregular data movings must be implemented by usual move type instructions or by the sequence of instructions of packed AND, OR, shift, addition and multiplication using proper masks and constants. According to strict sequential consistency of computing, the latent delay between storing the operand into memory and using it in another calculations can be equal to several clock cycles, and the real delay can be unpredictable due to cache coherency implementation. Therefore, it is preferable to store such operands in MMX registers.

The method for mapping data flow graphs into MMX architecture consists in the following. The latent period L =3,4 ... is selected. Two stages of the method for mapping data flow graphs into application specific structure are implemented. By this the SIMD structure illustrated by the fig.1,2 is selected as the target one.

On the third stage the derived algorithm configuration is optimized to fit both SIMD structure and MMX instruction set. For this purpose up to four multiplication or addition nodes $\boldsymbol{K}_i$ are gathered to form a line which is perpendicular to time axes. Then the nodes in these lines are permutated to satisfy the condition that vectors-edges $\boldsymbol{D}_j$, which are incident to them must be equal to each other. When such condition is not satisfied, then functional equivalent transforms are implemented which consist in addition of operators like AND, OR, shift, addition and multiplication using proper masks and constants. Also the delay- type vectors-nodes are introduced into vectors-edges $\boldsymbol{D}_j$ which are not incident to multiplication nodes, until $R(\boldsymbol{D}_{Tj})$=1. The delay- type vectors-nodes are mapped into quarter parts of MMX registers. These transforms can disagree with the conditions which were satisfied in the first two stages of the synthesis. Then the process is repeated from the first stage, and the latent period L can be exchanged . This process is repeated until all of nodes and edges can be covered by graphs (stencils), which represent MMX instructions.

On the last stage the derived algorithm configuration C_A is taken into consideration. The nodes $\boldsymbol{K}_i$ =(k,l,j, i,q) of k -th type, which are calculated in i- th iteration and q-th clock cycle of this iteration, and in j th PU, form a set of up to four nodes, when k =*const* , q= *const*, $j \in (0,1,2,3)$. Then this set of nodes is represented by a proper MMX instruction, which source and target operands are derived by the coordinate l of nodes which are adjacent to these ones. The derived instructions form the assembly program loop body, in which the instructions stay in the order according to the rising of the clock cycle q of the respective node set. According to the program pipelining technique, operators of address calculating and iteration counting as well as prologue and epilogue operator groups are added to the resulting program.

5. Example of the algorithm programming.

Consider an example of the calculation of the function $y_i=arctg(x_i)$ for the array of arguments x_i. This function is calculated by the following polynomial approximation $arctg(x)=0.999x-0.289x^3+0.079\ x^5$, and is often used in DSP applications. This example is selected because of its relative complexity to show the advantages of proposed method comparing to the hand made programs.

The polynome is factorised as the following: $y_i = c_1\, x_i + c_2(x_i^2\, x_i) + c_3(x_i^2(x_i^2\, x_i))$. Consider the resulting algorithm configuration has the latent period L =5 clock cycles. Then six multiplications of the algorithm can be implemented on the couple of PUs of the SIMD structure. This means, that four PUs of the SIMD structure can calculate two algorithms in parallel.

Then first and second stages of the synthesis are implemented. The resulting algorithm configuration C_A is illustrated by the fig.3. Here coordinates i, q, j represent iteration number, clock cycle in the iteration, and PU number, respectively. Circles represent registers, circles with plus sign and with cross sign represent addition and multiplication operators, respectively. Two algorithms are implemented in parallel on the PUs 0, 1 and 2, 3 , respectively. Then the input dates are loaded in the packed format as the following : (0, $x_i^{'}$,0, x_i), the coefficients are stored in register mm6 : (c_1, c_2, c_1, c_2), and in register mm7 : (0, c_3, 0, c_3), the results are stored as the following : (0, $y_i^{'}$,0, y_i).

At the third stage the derived algorithm configuration is optimised to fit both SIMD structure and MMX instruction set. The resulting optimised algorithm configuration C_A is illustrated by the fig.4. Here circles with vi sign denotes the OR operations.

At the fourth stage of the program synthesis sets of up to two nodes of the equal type are searched which are calculated in the q -th clock cycle. Then these sets of nodes are represented by a proper MMX instruction, which are collected into the following table. The derived instructions form the assembly program loop body, which consists of about twenty instructions.

Table. MMX instructions derived by the algorithm mapping.

Clock cycle *q*	MMX instruction			
0	`pmulhw MM1,MM0`	`movq MM4,MM1`	`movq MM3,MM7`	`paddw MM5,MM4`
1	`movq MM0,xi`	`movq MM3,MM0`	`pslld MM4,16`	`pmulhw MM3,MM1`
2	`pmulhw MM0,xi`	`movq MM2,MM4`	`movq MM4,MM2`	`movq yi,MM5`
3	`por MM2,MM1`	`pmulhw MM1,MM3`	`psrld MM4,16`	`movq MM5,MM4`
4	`movq MM1,MM0`	`pmulhw MM2,MM6`	`paddw MM5,MM3`	

The performance of derived program was proven by the VTune programming tool. Due to the superscalar nature of the processor and the fact that U-pipe and V-pipe of it is fully loaded, the latent period of derived program implementation is equal to 10 instruction cycles, and the one result calculating lasts 36 cycles, when all of dates are in the cache RAM. Only three of 19 MMX instructions make access to the RAM which proves the high grade of data reuse. Also taking into account two algorithms implemented in parallel, each result y_i is calculated approximately only for 5 clock cycles.

5.Conclusion .

Implementation DSP algorithms in MMX architecture has a set of advantages, like the possibility to achieve performance approximately equal to one of the high end signal microprocessors, and combining DSP and other applications. But the demand on automatic programming tools is still not satisfied. In this work the simplified four processor SIMD model of the MMX kernel is selected and a new method for mapping DSP algorithms into it is proposed. This method is derived by adapting the method for mapping unimodular loop nests into application specific structures, described in [4].

The method consists of four stages. At the first stage, the searching for the space component of the algorithm mapping into application specific structure is implemented. At the second stage, the time component of the mapping is searched for and algorithm configuration is derived. At the third stage is optimised to fit both SIMD structure and MMX instruction set. And at the fourth stage sets of nodes of the algorithm configuration are represented by MMX instructions, which form the assembly program loop body.

The method helps to derive programs which fully implement the parallelism of the MMX kernel of the Pentium microprocessor and can be used for the development new complex DSP applications and library functions. It also can be adapted to another microprocessor families which implement the expanded instruction set for multimedia applications. An automatic programming tool which implements this method is now under development.

References.

[1]. The synthesis approach to digital system design. Ed.: P.Michel, U.Lauther, P. Duzy, Kluwer Academic Pub. 1992.

[2]. Kung S.Y. VLSI processor arrays. Prentice Hall, Englewood Cliffs, 1988.

[3]. Wyrzykowsky R., Kanevski J.S., Maslenikov O, Sergyienko A. Mapping recursive algorithms into processor arrays.\ Proc. Int. Workshop "Parallel Numerics' 94", M.Vajtersic, P.Zinterhof, eds., Smolenice (Slovakia), 1994, pp 169-191.

[4]. A. Sergyienko, A. Guzinski, Ju. Kanevski, A method for mapping unimodular loops into application specific parallel architectures, In Proc. 2-nd Int. Conf. on Parallel Procesing and Applied mathematics. PPAM’97. Zacopane, Poland, Sept. 2-5, 1997, p. 362-371.

[5]. Intel Architecture MMX Instruction Set. http:// developer. intel.com/ drg/mmx/ manuals/ prm/ .

[6]. J. S. Kanevski, A. M. Sergyenko, H. Piech, A method for the structural synthesis of pipelined array processors, In Proc. 1-st Int. Conf. on Parallel Processing and Applied Math. - PPAM’94. Czestochowa (Poland), 1994, pp.100-109.

[7]. Yu. S. Kanevskiy, L. M. Loginova, A. M. Sergienko, Structured Design of Recursive Digital Filters, Enginering Simulation, 1996, V.13, pp. 381-390.

[8]. VLSI and Modern Signal Processing, Ed. by S.Y.Kung, H.Whitehouse, T.Kailath, Prentice Hall, 1985.

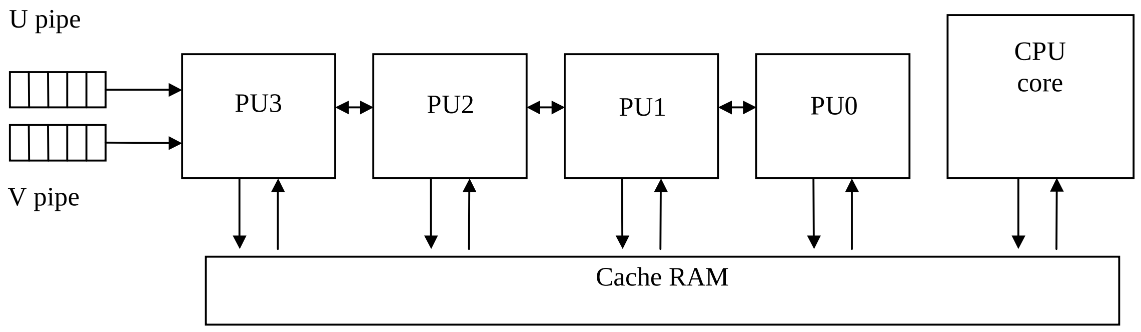

Fig.1. SIMD structure model of the MMX processor core

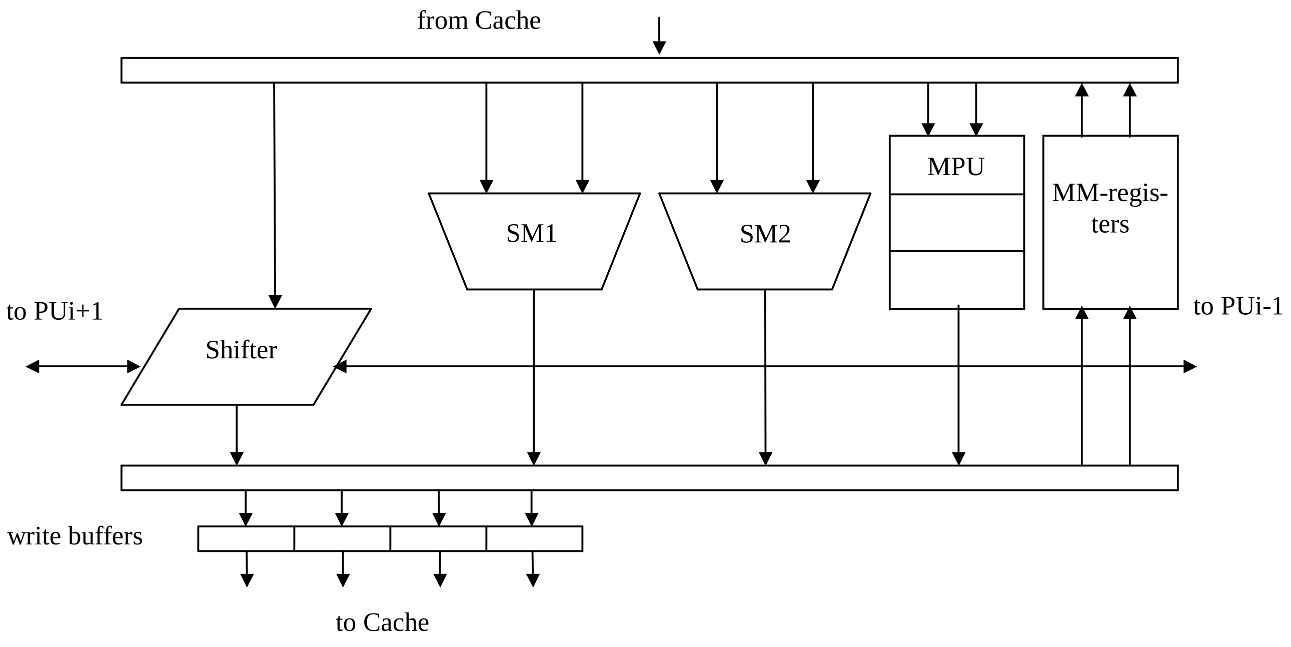

Fig.2. Structure of the PU.

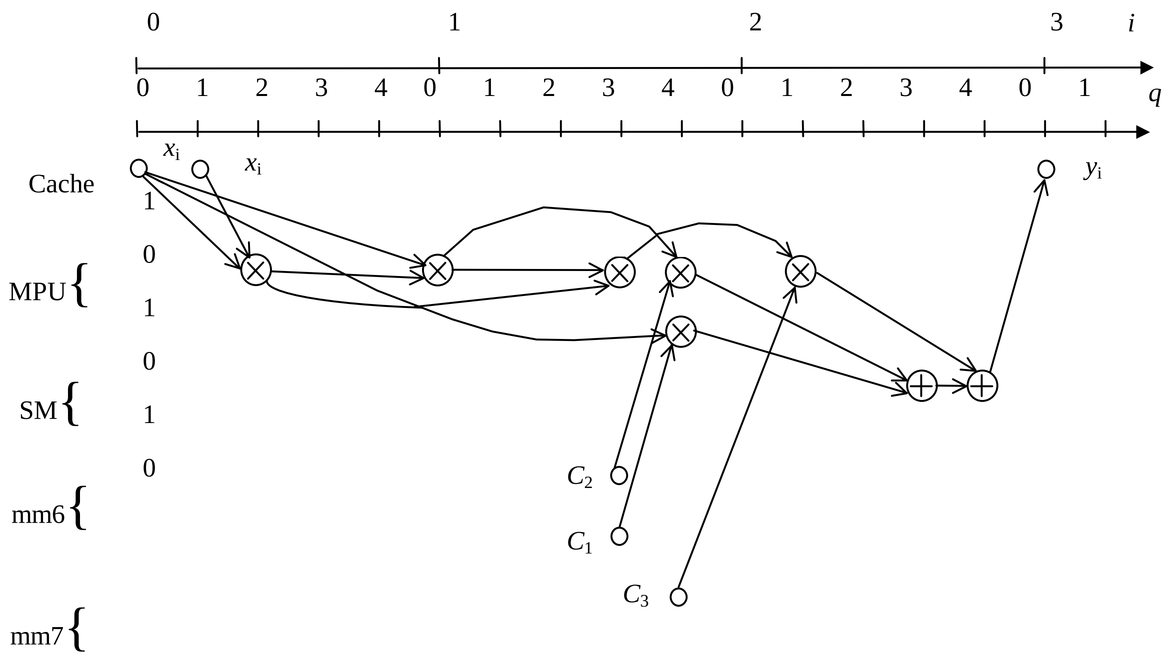

Fig.3. Initial algorithm configuration

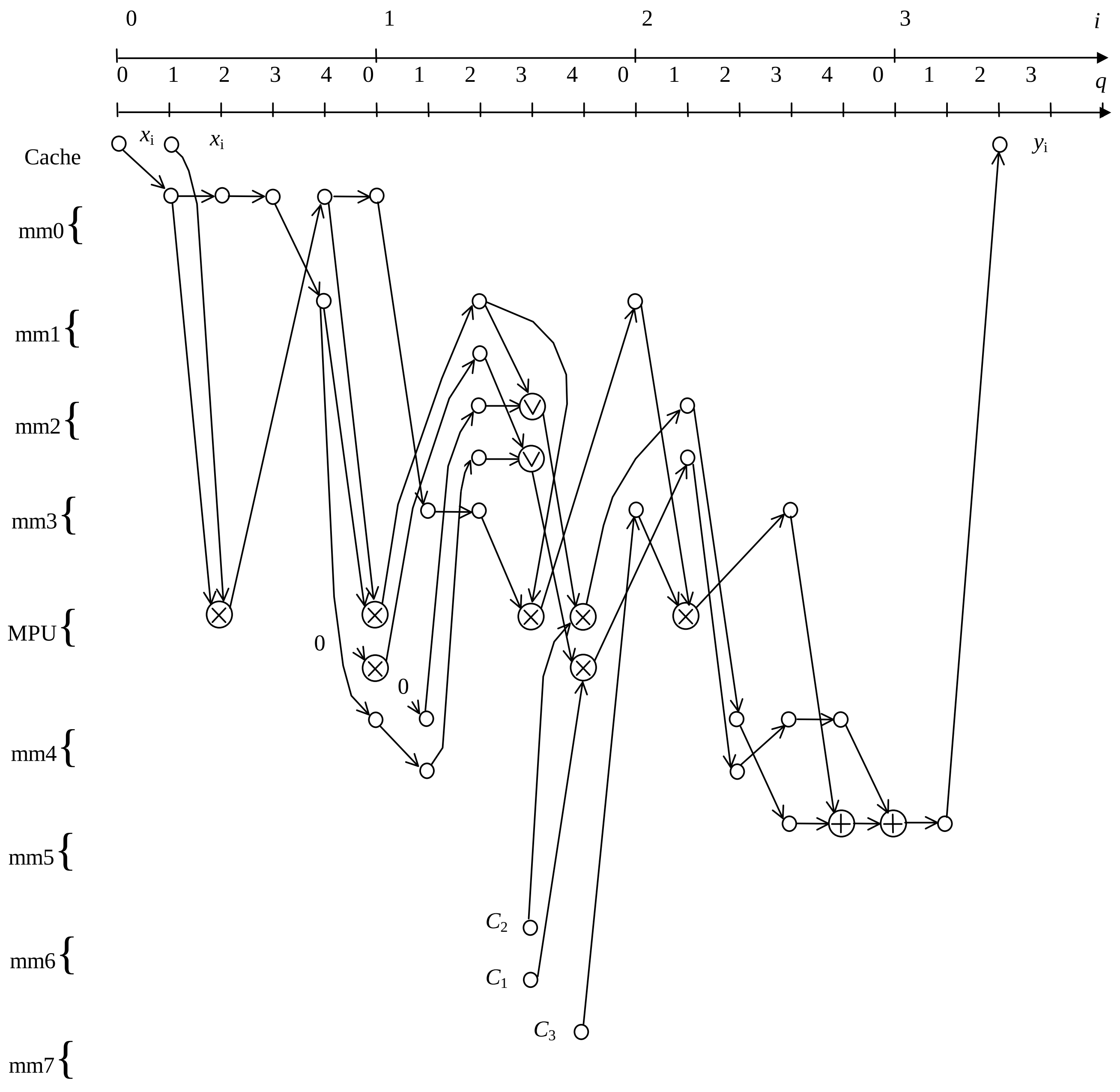

Fig.4. Resulting algorithm configuration