*Bull. Nov. Comp. Center*, Comp. Science, 1 (1993)
© 1993 NCC Publisher

# Implementation of high-accuracy computations in vertical processing systems

A.P. Vazhenin

**An efficient way to achieve the high accuracy of the results of computations is to increase the operands capacity. The best results in terms of the problem solution rate and effectiveness of memory using can be reached in the case when a computer system provides the possibility of dynamic capacity control of processed numbers. The suggested programming system presents an effective tool for Superprecision Parallel ARiTHmetic computations (SPARTH-computations). It is developed for STARAN-like associative array processors (AAP), which is a typical example of vertical processing systems. The system is oriented to solve a large set of vector and matrix operations. Besides, the operands length may be changed dynamically during processing. From the user's viewpoint the system represents a parallel vector processor with programmable word length called SPARTH-processor. This provides the accuracy control of computations directly during solution of the problem. The SPARTH-processor architecture is implemented within the basic AAP-architecture.**

**The paper describes features of the SPARTH-processor architecture and new parallel algorithms of accurate computing of dot products and polynomials, in which the automatic selection of capacity needed for exact calculations is used. The results of an estimation of the SPARTH-processor accuracy are also presented.**

## 1. Introduction

One of the important factors effecting the accuracy of data processing results is rounding in arithmetic operations. The necessity of rounding is closed with the fixed and relatively small length of operands in general purpose computers. The decrease of errors may be achieved by means of using multi-precision arithmetic. Super-long operands may be processed either by using of the specialized coprocessors [8],[14] or by the program implementation of multiprecision arithmetic algorithms in terms of the basic computer architecture [1], [7]. However, the program implementation or micro-program interpretation of high accuracy computations leads (in conditions of usual computer systems) to the fast decrease of problem solution rate and non-effective use of memory.

The development of current integral technology and computer science allows the design of parallel systems functioning with varying operand length. This led to the design of computer systems, which solve problems with an accuracy given before calculations or an accuracy provided by system resources and known to the user after the computation terminates. In [5] a program package of multiple precision integer arithmetic and theoretical numeric computations for CRAY-2 is described. These programs implement multiple precision arithmetic operations employing the pipeline principle. In [9] there are proposed the essentials of the language for high-accuracy computations and the way of implementation of this language in a multi-transputer system.

To perpective computing systems from the viewpoint of effective performance and programming of high accuracy computations one may refer SIMD parallel systems with vertical processing. Their distinctive features combining the possibilities of location and parallel processing of arbitrary length of operands (up to several thousand bits), programmability of data formats, the data masking, etc., allow to consider an effective means to implement Super-precision Parallel ARiTHmetic computations (SPARTH-computations).

The vertical processing is based on the word parallelism. Bit slices of processed array are extracted from memory in a regular way, then they are input to the registers of operating unit, where they are processed by logic schemes. Such processing is called "bit-serial" in [4]. Similar devices are also called systems with "fine-grained" structure. The main feature of any of these systems is the presence of large number of one-bit processing elements (PE) operating in parallel, each having one-bit local memory and performing bit-serial processing on its contents. Wide-spread systems of this type are STARAN [2], DAP [10], MPP [3], CM [15], LUCAS [6], PPS SIMD [12]. Their performance ranges up to several billions of bit operations per second.

The STARAN system [2] is generally called the associative array processor (AAP) since it resulted the evolution of associative processing. The present paper dealts with the programming system of SPARTH-computations for STARAN-like architectures. It is oriented to solve the problems containing many vector and matrix operations and allows the control of computation accuracy during solving the problems.

## 2. Architechture of SPARTH-processor

Figure 1 shows the SPARTH-processor architecture implemented within the basic AAP-architecture. The main elements of SPARTH-processor are: vector registers $VR_0 - VR_{v-1}$ intended for the location of super-digital vector operands; high-precision parallel summator (HPS) containing fields $VS_0 - VS_3$ and having the capacity necessary for the performance of arithmetic operations without rounding; scalar registers or scalars $S$ where scalar operands are located; operational AAP-registers $X, Y, M$; index registers or indices $I$ which are unsigned integers and intended for storage of constants defining the number of loop iterations, modes of access to vector registers, etc.; registers for temporal storage of masks $RM_0 - RM_{l-1}$ located in the special field of the multidimensional access (MDA) memory and intended for storing the masks and bit slices resulting from performance of parallel vector operations (overflow, search operations, etc.).

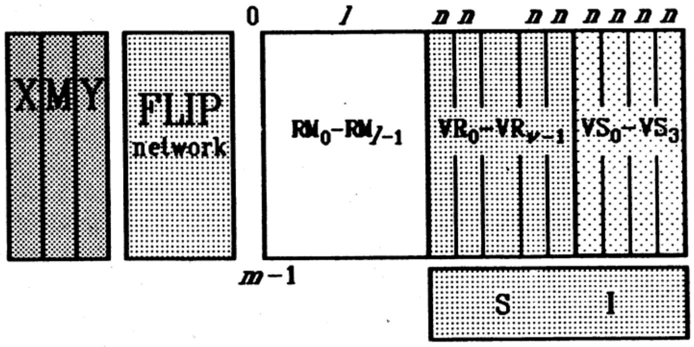

**Figure 1.** The SPARTH-processor architecture

All parallel operations are performed for unmasked elements of vector registers. The masks for these instructions should be loaded to the $M$-register. Functions of the $X$- and $Y$-registers are similar to their assignment in AAP. They provide processing of bit slices. A bit slice resulting of the parallel operation performance is located in the $Y$-register. It may be written either in the $RM$-register or in the $M$-register, or operated processed by procedures of response processing.

Computations in the SPARTH-processor may be performed in two modes: with the fixed or dynamic accuracy. The first mode is characterized by the constant capacity of operands in computations. In this case, the number $v$ of vector registers available is determined by required capacity

$$v = \lfloor \frac{s - 4n - l}{n} \rfloor, \tag{1}$$

where $n$ is an ordered capacity (bits), $l$- number of $RM$-registers and $s$ -

size of local AAP memory. The user is able to choose between the problem solution rate, the amount of processed data and required accuracy.

In the dynamic accuracy mode, the operand capacity may alter in the interval defined by the user. The number of available vector registers is formed according to the maximum ordered capacity value (see (1)). Computations may be started with relatively small capacity of operands. If needed, the SPARTH-processor may be switched to a next capacity limit by means of precision control procedures.

In the SPARTH-processor three types of data are used: *integer* data type; *fixed-point* type format (without integer part); *real* type format (numbers with integer and fractional parts).

Processor instructions provide effective interaction of subsystems and execution of high-precision parallel computations. Arithmetic operations are executed in two stages. At first, the exact result (without rounding) is formed in HPS, then it is stored into destination registers using the rounding operations for multiplication and division. In the dynamic accuracy mode, the data located in HPS may be stored in vector registers without rounding.

In Figure 1 $m$ denotes the number of PE's in AAP. All $m$ components of vectors may be performed in parallel. Therefore, the parallel *addition* and *subtraction* have an arithmetic operation count of $O_A(1)$ and a bit operation count of $O_B(n)$ (for $n$-digital numbers). Operation counts of both parallel *multiplication* and *division* are $O_A(1)$ and $O_B(n^2)$.

The accurate *sum of vector elements* is executed by *recursive doubling* in groups. Each of these groups may contain $2, 4, 8, \cdots, 2^i, \cdots, m$ components. The sum is computed for each group in parallel and have operation counts of $O_A(\log_2 N)$ and $O_B(n \log_2 N)$, where $N = 2^i$.

The change-over of capacity limits from $n$ to $2n$ bits has a bit operation count of $O_B(vn)$.

Data transmission instructions allow the user to assign different operations of data exchange between subsystems of SPARTH-processor using the properties of the FLIP interconnection network [2]. They support different procedures of the parallel transmission of unmasked elements of vector registers, and selective communication of the data chosen by means of index registers. For parallel transfer operations, elements may be interchanged. There are three types of permutations: the *mirror*, the *cyclic shift* and the *mixed permutation* which use the both previous types (see [13] for details). Data transmission instructions have an arithmetic operation count of $O_A(1)$ and $O_B(n)$.

## 3. Accurate scalar products

A fundamental algorithm in regard to basic numerical methods is the one for the computation of the scalar product (dot product) of vectors $\mathbf{X} \equiv [x_i]$ and $\mathbf{Y} \equiv [y_i]$:

$$P = \mathbf{X} \cdot \mathbf{Y} = \sum_{i=0}^{N-1} x_i \cdot y_i.$$

The computation of this formula can be made in a SPARTH-processor employing the above mentioned instructions of *multiplication of vectors* $\mathbf{X}$ and $\mathbf{Y}$, and *sum of vector elements.*

**Proposition 1.** *The operation counts of scalar product computation in a SPARTH-processor are $O_A(\log_2 N)$ and $O_B(n^2 + n \cdot \log_2 N)$.*

As indicated above, the sum of vector elements is computed for $N = 2^i$. Special procedures was developed to provide the processing of vectors of arbitrary $N$.

**Definition 1.** Let $\mathbf{X}$ contain $k$ groups of $N$ components each:

$$\mathbf{X} = \{x_0^0, x_1^0, \cdots, x_{N-1}^0, \cdots, x_0^{k-1}, x_1^{k-1}, \cdots, x_{N-1}^{k-1}\}.$$

The procedure forming for $m \geq N \cdot k$ a vector

$$\acute{\mathbf{X}} = \{\underbrace{x_0^0, \cdots, x_{N-1}^0, c, \cdots, c}_{N_1}, \cdots, \underbrace{x_0^{k-1}, \cdots, x_{N-1}^{k-1}, c, \cdots, c}_{N_1}\},$$

where $N_1 = 2^{\lceil \log_2 N \rceil}$ and, $c$ is a constant, we call **vector-expansion.**

To calculate a sum, it is necessary to set $c = 0$.

**Definition 2.** Let $\mathbf{X}$ contain $k$ groups of $N_1 = 2^i$ each. The procedure forming for $m \geq N_1 \cdot k$ a vector

$$\acute{\mathbf{X}} = \{\underbrace{x_0^0, x_1^0, \cdots, x_{N-1}^0, \cdots, x_0^{k-1}, x_1^{k-1}, \cdots, x_{N-1}^{k-1}}_{N \cdot k}, x, \cdots\},$$

where $N_1 = 2^{\lceil \log_2 N \rceil}$ and $x, \cdots$ - a "tail" of length $k(N_1 - N)$, we call **vector-compression.**

**Proposition 2** [16]. *The operation counts of both vector-expansion and vector-compression procedures on a SPARTH-processor are $O_A(\lceil \log_2 N \rceil)$ for arithmetic operations, and $O_B(n \lceil \log_2 N \rceil)$ for bit operations.*

The flexible switching of capacity limits in SPARTH-processor allows an automatic adaptation to the range of data values. As indicated in SPARTH-program given below, the algorithm of exact scalar product is implemented in the following stages: parallel conversion of input floating-point numbers with automatic selection of capacity needed for exact representation of these numbers; *vector-expansion* of **X** and **Y**; calculation of dot products in parallel; *vector-compression* of results and transformation of results to floating-point format.

```
            ************************************************************
        *********** COMPUTING OF EXACT SCALAR PRODUCTS **********
            ******** X → VR0; Y → VR1; DESTINATION → VR2 ******
            ************************************************************

         BEGSPARF  X'2000'           * Start of SPARTH-program
         DPRECISN  REAL,512,64       * Set dynamic capacity mode
                                     * Format REAL
                                     * Max. limit - 512bit
                                     * Initial limit - 64bit
         INDEXC    N,K,INPUT         * N - Size of vectors
                                     * K - Number of vectors
         INDEX     I,J,N1,NK,N1K     * Define work indices
         LOG2IUP   N,J               * J := ⌈log2 N⌉
         POWER2    J,N1              * N1 := 2^J
         MULI      N,K,NK            * NK := N·K
         MULI      N1,K,N1K          * N1K := N1·K
         SETMI     RM0,0,NK,CLR      * Set mask in RM0
         SETMI     RM1,0,N1K,CLR     * Set mask in RM1
         MOVM      RM0,M             * RM0 → M
FIRST    DNORM     VRO,VR2,2         * Convert X from floating-point
         BRZ       CORRES            * to SPARTH-format
         MPRECISN  VR0,VR1           * If signed bits were lost then
                                     * modify capacity exept VR0,VR1
         BN        FIRST             * and repeate conversion
CORRES   SPRECISN  VRO               * Fit VR0 to current capacity
         MOVV      VR2,VR0           * VR2 → VR0
SECOND   DNORM     VR1,VR2,2         * Convert Y from floating-point
         BRZ       PRODUCT           * to SPARTH-format
         MPRECISN  VR1               * If signed bits were lost then
                                     * modify capacity exept VR1
         BN        SECOND            * and repeate conversion
PRODUCT  SPRECISN  VR1               * Fit VR1 to current capacity
         MOVV      VR2,VR1           * VR2 → VR1
         MULV      VR0,VR1,VR2       * VR2 → VR0·VR1
         STRETCH   VR2,VR2,N         * Expansion of VR2
         MOVM      RM1,M             * M → RM1
         SUMV      VR2,VR2,K,RM1     * Sum of vector elements in VR2
         COMPRESS  VR2,VR2,N1,1      * Compression of sum in VR2
         NORM      VR2,VR2,2         * Transformation to host-format
         STOP
         ENDSPARF                    * End of SPARTH-program
```

To estimate the accuracy of scalar product evaluating in SPARTH-processor we have used the "hard" input data from [11]. Examples of such data are shown in Table 1. In [11] test results of parallel systems SIEMENS/Fujitsu VP400-EX and CRAY-2, and high-accuracy arithmetic subroutine library ACRITH on IBM-4381 [7] are also described.

**Table 1**

| $\mathbf{X}^{(1)}$ | $\mathbf{Y}^{(1)}$ | $\mathbf{X}^{(2)}$ | $\mathbf{X}^{(2)}$ |
|---|---|---|---|
| 27182818280 | 14862497000000 | 5772156649 | 47737146470000000 |
| -31415926540 | 8783669879000000 | 3010299957 | 1850490 |
| 14142135620 | -223749200000 | 0 | 0 |
| 5772156649 | 47737146470000000 | 27182818280 | 14862497000000 |
| 0 | 0 | -31415926540 | 8783669879000000 |
| 3010299957 | 1850490 | 14142135620 | -223749200000 |

The results from Table 2 show that the accuracy of dot products computations in SPARTH-processor is similar to ACRITH. However, SPARTH provides very high performance because all scalar products are computed simultaneously.

**Table 2**

| Computing System | $P_1$ | $P_2$ |
|---|---|---|
| VP-400-EX (Scalar Mode) | 0.462915718600000E+10 | -0.115474320000000E+10 |
| VP-400-EX (Vector Mode) | -0.334189890000000E+09 | 0.000000000000000E+00 |
| CRAY-2 (Scalar Mode) | -0.179496213989999E+13 | -0.170625964441600E+13 |
| CRAY-2 (Vector Mode) | -0.110380659550720E+13 | -0.110380659550720E+13 |
| IBM PC/AT (Double prec.) | 0.186091654600000E+10 | 0.436495923200000E+10 |
| IBM4381 (ACRITH) | -0.100657107000000E+10 | -0.100657107000000E+10 |
| SPARTH | -0.100657107000000E+10 | -0.100657107000000E+10 |

## 4. Polinomial evaluation

Usually the evaluation of a polinomial

$$P(x) = a_{p-1}x^{p-1} + \cdots + a_1x^1 + a_0 = \sum_{i=0}^{p-1} a_i x^i$$

is done via Horner's scheme. This leads to a linear first order recurrence for which a vectorization is possible only with the help of an expansion method like *recursive doubling* or *cyclic reduction.*

We use another simple and fast method to evaluate a polinomial. It is to compute $P(x) = \mathbf{A} \cdot \mathbf{X}$ as the *dot product* of $\mathbf{A} = \{a_{p-1}, \cdots, a_1, a_0\}$ and $\mathbf{X} = \{x^{p-1}, \cdots, x, 1\}$.

| $VR0$ | $RM0$ | $VS2$ | $VS3$ | $RM0$ | $VS2$ | $VS3$ | $RM0$ | $VR2$ |
|---|---|---|---|---|---|---|---|---|
| $c_0$ | 0 | * | $c_0$ | 0 | * | $c_0$ | 0 | $c_0$ |
| $c_1$ | 1 | $c_0$ | $c_0c_1$ | 0 | * | $c_0c_1$ | 0 | $c_0c_1$ |
| $c_2$ | 1 | $c_1$ | $c_1c_2$ | 1 | $c_0$ | $c_0c_1c_2$ | 0 | $c_0c_1c_2$ |
| $c_3$ | 1 | $c_2$ | $c_2c_3$ | 1 | $c_0c_1$ | $c_0c_1c_2c_3$ | 0 | $c_0c_1c_2c_3$ |
| $c_4$ | 1 | $c_3$ | $c_3c_4$ | 1 | $c_1c_2$ | $c_1c_2c_3c_4$ | 1 | $c_0c_1c_2c_3c_4$ |
| $c_5$ | 1 | $c_4$ | $c_4c_5$ | 1 | $c_2c_3$ | $c_2c_3c_4c_5$ | 1 | $c_0c_1c_2c_3c_4c_5$ |
| $c_6$ | 1 | $c_5$ | $c_5c_6$ | 1 | $c_3c_4$ | $c_3c_4c_5c_6$ | 1 | $c_0c_1c_2c_3c_4c_5c_6$ |
| $c_7$ | 1 | $c_6$ | $c_6c_7$ | 1 | $c_4c_5$ | $c_4c_5c_6c_7$ | 1 | $c_0c_1c_2c_3c_4c_5c_6c_7$ |

**Figure 2.** Calculating of prefix product in a SPARTH-processor

**Definition 3.** Let **C** contain $p$ components $\mathbf{C} = \{c_{p-1}, \cdots, c_1, c_0\}$. The procedure forming a vector $\tilde{\mathbf{C}} = \{\tilde{c_{p-1}}, \cdots, \tilde{c_1}, \tilde{c_0}\}$, where $\tilde{c_j} = \prod_{i=0}^{j} c_i$, we call **prefix product.**

**Proposition 3.** *The operation counts of prefix product computation in a SPARTH-processor are* $O_A(\log_2 p)$ *and* $O_B(n^2 \log_2 p)$.

PROOF. As shown in Figure 2 (for $p = 8$), the computation of *prefix product* in SPARTH-processor may be executed using parallel data transmission with *cyclic shifts* of vector and masks, and parallel *multiplication.* The number of $\tilde{c_j}$ is doubled in each step. Therefore, the total number of steps is $\log_2 p$. We have now the desired result, because the operation counts are $O_A(1)$ and $O_B(n)$ (for transmission), and $O_A(1)$ and $O_B(n^2)$ (for multiplication). □

The vector **X** may be computed using the *prefix product* of vector $\acute{\mathbf{X}} = \{\underbrace{x, x, \cdots, x}_{p-1}, 1\}$.

**Proposition 4.** *The operation counts of polinomial evaluation in a SPARTH-processor are* $O_A(\log_2 p)$ *and* $O_B(n^2 \log_2 p + n^2 + n \log_2 p)$.

PROOF. The polinomial evaluation may be implemented in three stages: the forming of $\acute{\mathbf{X}}$ in $O_A(1)$ and $O_B(n)$ time steps loading a scalar $x$ into the vector register, calculation of **X** by *prefix product* and *dot product* of **A** and **X**. Therefore, total operation counts of polinomial evaluation are $O_A(\log_2 p)$ and $O_B(n^2 \log_2 p + n^2 + n \log_2 p)$. □

**Proposition 5.** *The relative speedup of polinomial evaluation in SPARTH-processor is* $O(\frac{p}{[\log_2 p]+1})$.

PROOF. The polinomial evaluation in sequential computers is done via Horner's scheme in $p$ multiplications and $p$ sums. Therefore, the sequential bit operation count is $O_B(pn(n+1))$. Thus, the ratio of sequential and parallel bit operation counts is $O\left(\frac{p}{\lfloor\log_2 p\rfloor+1}\right)$, because $\log_2 p = \lfloor\log_2 p\rfloor$ for $p = 2^i$. If $p \neq 2^i$, the computation is implemented in $p_1 = \lfloor\log_2 p\rfloor + 1$ using the *vector-expansion.* □

**Corollary 1.** *The maximal relative speedup of polinomial evaluation in SPARTH-processor is* $O\left(\frac{mp}{2^{\lceil\log_2 p\rceil}(\lfloor\log_2 p\rfloor+1)}\right)$.

PROOF. A SPARTH-processor can process $N_1 = \frac{m}{2^{\lceil\log_2 p\rceil}}$ simultaneously in groups of $2^{\lceil\log_2 p\rceil}$ components each using the properties of the FLIP interconnection network. If all $m$ processing channels are used, then the maximal relative speedup is $O\left(\frac{mp}{2^{\lceil\log_2 p\rceil}(\lfloor\log_2 p\rfloor+1)}\right)$. □

To estimate the accuracy of polinomial evaluation we have used the "hard" input data from [11]. The coefficients were defined as

$$a_i = \begin{cases} -16^7, & \text{for } i = 8 \\ 16^4, & \text{for } i = 11 \\ 16^i, & \text{in other cases} \end{cases} \qquad (i = 0, \cdots, 15).$$

The input data are $x_j = 16 + j \cdot 16^7$, where $(j = -6, -5, \cdots, -1, 0, 1, \cdots, 4)$.

Polinomials were computed simultaneously for all values $x_i$ using a procedure similar to the calculation of dot products. It is an automatic adaptation to the range of data values. As shown in Table 3, the polinomial evaluation in SPARTH-proccessor allows calculation of absolutly exact results.

## 5. Conclusion

The comparison with known dedicated programming systems for high-accuracy computations on sequential computers shows that SPARTH-processor ensures similar accuracy of results. Moreover, it provides very high performance due to effective use of massively parallelism.

The presented programming system may be considered as an intermediate language for the translation from high-level parallel scientific languages (FORTRAN-XCS, PASCAL-XCS, C-XCS, etc.) It may simplify the structure of such compilers, because the SPARTH-instructions are relatively large vector operations.

**Table 3**

| $j$ | $P_j$(skalar) VP400-EX | $P_j$(vector) VP400-EX | $P_j$(Horner) VP400-EX | $P_j$(ACRITH) | $P_j$(SPARTH) |
|---|---|---|---|---|---|
| −6 | -4831838152 | -4831838140 | -4831838162.0 | -4831838139.25 | -4831838133.2500 |
| −5 | -4026531800 | -4026531788 | -4026531810.5 | -4026531789.81 | -4026531783.8125 |
| −4 | -3221225448 | -3221225436 | -3221225456.0 | -3221225437.00 | -3221225431.0000 |
| −3 | -2415919096 | -2415919084 | -2415919098.5 | -2415919080.81 | -2415919074.8125 |
| −2 | -1610612728 | -1610612716 | -1610612738.0 | -1610612721.25 | -1610612715.2500 |
| −1 | -805306360 | -805306348 | -805306374.5 | -805306358.31 | -805306352.3125 |
| 0 | 8 | 4 | 8.0 | 8.00 | 14.0000 |
| 1 | 805306376 | 805306372 | 805306377.5 | 805306377.68 | 805306383.6875 |
| 2 | 1610612744 | 1610612740 | 1610612750.0 | 1610612750.75 | 1610612756.7500 |
| 3 | 2415919112 | 2415919108 | 2415919125.5 | 2415919127.18 | 2415919133.1875 |
| 4 | 3221225480 | 3221225476 | 3221225504.0 | 3221225507.00 | 3221225513.0000 |

This approach may be used as well for other massively parallel systems (DAP,MPP,CM,etc.). It can also lead to the developing of new computer architectures with overcoming rounding errors.

## References

[1] O. Aberth, Precise scientific computation with a microprocessor, IEEE Trans. Comput., Vol.C-33, No.8, 1984, 685-690.

[2] K.E. Batcher, STARAN series E, Proc. 1977 Int. Conf. Parallel Processing, New York, IEEE, 1977, 140-143.

[3] K.E. Batcher, Design of massively parallel processor, IEEE Trans. on Comput., Vol.C-29, No.9, 1980, 325-336.

[4] K.E. Batcher, Bit-serial parallel processing systems, IEEE Trans. on Comput., Vol.C-31, No.5, 1982, 377-384.

[5] D. Buell, R. Ward, A multiprecise integer arithmetic package, The Journal of Supercomputing, Vol.3, No.2, 1989, 89-107.

[6] Ch. Fernstrom, I. Krusela, B. Svensson, LUCAS associative array processor: design, programming and application studies, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 216, 1986.

[7] E. Lange, Imlementation and test of the ACRITH facility in a System/370, IEEE Trans. on Comput., Vol.C-36, No.9, 1987, 1088-1096.

[8] A.L. Lucke, Programmed word length computer proceedings, A.C.M. National Meeting, 1967, 57-65.

[9] N.N. Mirenkov, Parallel Programming for Multimodule Computing Systems, Radjo i svyaz, Moscow, 1989 (In Russian).

[10] D. Parkinson, The distributed array processor (DAP), Computer Physics Communications, North-Holland, Vol.28, 1983, 325-336.

[11] D. Ratz, The effects of the arithmetic of vector computers on basic numerical methods, Contributions to Computer Arithmetic and Self-Validating numerical methods, IMACS, Scientific Publishing Co., 1990, 499-514.

[12] K. Richter, A parallel computer system SIMD, Artificial intelligence and information, Inter. Conf.: Control Systems of Robots, 1984, 309-313.

[13] K.J. Terber, Large-scale Computer Architecture: Parallel and Associative Processors, Rochelle Park, Hayden Book Comp., 1976.

[14] J.J. Thomas, S.R. Parker, Implementing exact calculations in hardware, IEEE Trans. Comput., Vol.C-36, No.6, 1987, 764-768.

[15] L.W. Tucker, G.G. Robertson, Architecture and application of the connection machine, Computer, Vol.21, No.8, 1988, 26-38.

[16] A.P. Vazhenin, A.E. Vartazaryan, Parallel matrix multiplication with multidigital elements, Preprint Vychisl. Tsentr SO RAN, Novosibirsk, 973, 1992 (in Russian)