Bull. Nov. Comp. Center, Comp. Science, 2 (1993), 63–77
© 1993 NCC Publisher

Parallel algorithm for solving systems of linear equations for processors using dynamically changed length of operands

A.P. Vazhenin and V.V. Kozhevnikov

Peculiarity of the most direct algorithms for solving system of linear equations is the use of divisions for the elimination of unknowns. Divisions require a great time for its execution, when the multiprecision arithmetic is used because in this case they are realized by special programs. The paper describes the parallel algorithm implementing the elimination procedure without divisions. Some results of time, speedup and accuracy measurements implemented on a system supporting parallel high-accuracy computations by dynamically changed length of operands are presented.

1. Introduction

One of important factors effecting the accuracy of data processing results is rounding in arithmetic operations. Rounding is necessitated by fixed relatively small length of operands in most computers. Note also that parallel algorithms used in modern supercomputers can provide very high performances. However, they are actually noneffective in minimizing the rounding errors for operations of single or double precision format [1].

Rounding errors may be often avoided by means of multi-precision and interval arithmetics. Super-long operands may be processed either by specialized coprocessors [2], [3], [4], [5], or by program implementation of multiprecision arithmetic algorithms in terms of a basic computer architecture [6], [7], [8]. However, the program implementation or micro-program interpretation of high accuracy computations leads to the fast decrease of problem solution rate and non-effective use of memory. The development of computer science and current VLSI-technology allows for designing parallel systems functioning with varying operand length [9], [10].

An example of such approach is the SPARTH-processor [11], [12], [13] which is implemented within the basic STARAN-like architecture and intended for solving problems containing many vector and matrix operations. A number of new parallel algorithms was developed for solving problems of linear algebra (adaptive accurate dot products, matrix multiplication with multidigital elements, iterative algebraic linear equations solving with dynamic control of result accuracy, etc.), accurate polynomial evaluation, calculating of transcendent functions, etc. SPARTH-processor ensures the similar accuracy of results as the known dedicated programming systems for high-precision computations on sequential computers. Moreover, this accuracy is achieved in this case simultaneously for numerous data sets in corresponding processing elements of massively parallel system. This approach may be used for other massively parallel systems (CM, DAP, MPP, etc.) as well.

The solution of many scientific and applied problems results in the computational solution of systems of linear equations. Peculiarity of the most direct algorithms for solving system of linear equations is the use of divisions for the elimination of unknowns. Divisions require a great time for its execution, when the multiprecision arithmetic is used because in this case they are realized by special programs. Note also that some processors execute this operation by programming way.

The paper shows the parallel algorithm implementing the elimination procedure without division. Some results of time, speedup and accuracy measurements on a SPARTH-processor are presented.

2. Theoretical background

A well-known direct method for solving systems of linear equations of $\boldsymbol{Ax} = \boldsymbol{b}$ is the Gauss elimination [14]. A modification of this approach is the Jordan algorithm [15] which transforms the matrix $\boldsymbol{A}$ to the diagonal mode. Peculiarity of the most algorithms for solving system of linear equations is the use of divisions in the diagonalization of $\boldsymbol{A}$. Usually, great time and/or hardware resources are required to execute these arithmetic operations. It is very important, when the multiprecision arithmetic is used because in this case the arithmetic operations are realized by special programs [6], [7], [9], [10], [13].

The possibility of $\boldsymbol{A}$ transformation using multiplications and subtractions is shown in [16]. We have used this approach to develop the algorithm transforming $\boldsymbol{A}$ without division. In this section its principle peculiarities are explained.

There is given the matrix equation

$$\boldsymbol{AX} = \boldsymbol{B}, \tag{1}$$

where $\boldsymbol{A} \equiv [a_{ij}]_{N\times N}$ is a given $N \times N$ matrix, $\boldsymbol{X} \equiv [x_{ij}]_{N\times M}$ is a matrix of unknowns and $\boldsymbol{B} \equiv [b_{ij}]_{N\times M}$ is a given $N \times M$ matrix of right-hand parts. In other words, (1) is a set of M systems of linear equations with the different right-hand parts.

In general terms, the diagonalization of (1) may be represented as a sequence of N matrix multiplications

$$\boldsymbol{C}_{N-1}\boldsymbol{C}_{N-2}\cdots\boldsymbol{C}_k\cdots\boldsymbol{C}_1\boldsymbol{C}_0\boldsymbol{A}\boldsymbol{X} = \boldsymbol{C}_{N-1}\boldsymbol{C}_{N-2}\cdots\boldsymbol{C}_k\cdots\boldsymbol{C}_1\boldsymbol{C}_0\boldsymbol{B}, \quad (2)$$

where $\boldsymbol{C}_k$ are matrices formed in a special way.

Matrix multiplications are implemented for both parts of equation (1). Therefore, (2) is equivalent to (1). Another way of considering the procedure of elimination of unknowns is in deriving the matrix

$$\boldsymbol{D} = \boldsymbol{C}_{N-1}\boldsymbol{C}_{N-2}\cdots\boldsymbol{C}_k\cdots\boldsymbol{C}_1\boldsymbol{C}_0\bar{\boldsymbol{A}}, \quad (3)$$

where $\bar{\boldsymbol{A}} \equiv [\bar{a}_{ij}]_{N\times L}$ is the expanded matrix, $L = N + M$ and

$$\bar{a}_{ij} = \begin{cases} a_{ij} & \text{if } 0 \le j \le N-1, \\ b_{i,j-N} & \text{if } N \le j \le L-1. \end{cases}$$

The conversion is started from the zeroth row of $\bar{\boldsymbol{A}}$. The matrix $\boldsymbol{C}_0 \equiv [c_{ij}]_{N\times N}$ is composed from $\bar{\boldsymbol{A}}$ by

$$c_{ij} = \begin{cases} 1 & \text{if } i = j = 0, \\ \bar{a}_{00} & \text{if } i = j, \\ -\bar{a}_{i0} & \text{if } j = 0, \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to verify that the matrix multiplication $\boldsymbol{A}_0 = \boldsymbol{C}_0\bar{\boldsymbol{A}} \equiv [a_{ij}^{(0)}]_{N\times L}$ is reduced to

$$a_{ij}^{(0)} = \begin{cases} \bar{a}_{ij} & \text{if } i = 0, \\ \bar{a}_{00}\bar{a}_{ij} - \bar{a}_{i0}\bar{a}_{0j} & \text{otherwise.} \end{cases}$$

After implementation of this operation the elements of the zeroth column of $\boldsymbol{A}_0$ are all zero except $a_{00}^{(0)}$. Execute similar transformations for the following rows. In the k-th iteration the matrix $\boldsymbol{C}_k \equiv [c_{ij}^{(k)}]_{N\times N}$ is composed from $\boldsymbol{A}_{k-1}$ by

$$c_{ij}^{(k)} = \begin{cases} 1 & \text{if } i = j = k, \\ a_{kk}^{(k-1)} & \text{if } i = j, \\ -a_{ik}^{(k-1)} & \text{if } j = k, \\ 0 & \text{otherwise,} \end{cases} \quad (4)$$

and the elimination procedure is implemented by

$$a_{ij}^{(k)} = \begin{cases} a_{ij}^{(k-1)} & \text{if } i = k, \\ a_{kk}^{(k-1)}a_{ij}^{(k-1)} - a_{ik}^{(k-1)}a_{kj}^{(k-1)} & \text{otherwise.} \end{cases} \tag{5}$$

After implementation of this operation the elements of the k-th column of $\boldsymbol{A}_k$ are all zero except $a_{kk}^{(k)}$. Therefore, if N similar transformations for $0 \le k \le N-1$ are executed then the matrix $\boldsymbol{D} \equiv [d_{ij}]_{N\times L}$ is formed. It contains the elements:

$$d_{ij} = \begin{cases} a_{ij}^{(N-1)} & \text{if } i = j, \\ 0 & \text{if } 0 \le j \le N, \\ a_{ij}^{(N-1)} & \text{otherwise.} \end{cases} \tag{6}$$

It is easy to obtain the result matrix $\boldsymbol{X} \equiv [x_{ij}]_{N\times M}$ by

$$x_{ij} = \frac{d_{ik}}{d_{ii}}, \tag{7}$$

where $k = j + N$.

Theorem 1. *A sequential algorithm* (5) *for solving the matrix equation* (1) *has an arithmetic operation count of* $O_A(N^2(N+M))$.

Proof. As shown above, the solution of matrix equation (1) is implemented in two stages: the diagonalization of $\boldsymbol{A}$ by (5) and the calculation of matrix $\boldsymbol{X}$ by (7). Two multiplications and one subtraction are necessary to compute each $a_{ij}^{(k)}$. Therefore, the matrix $\boldsymbol{A}_k = \boldsymbol{C}_k\boldsymbol{A}_{k-1} \equiv [a_{ij}^{(k)}]_{N\times L}$ can be obtained in $O_A(NL)$-time, and the computation of $\boldsymbol{D}$ is implemented in $O_A(N^2L)$-time. To obtain the matrix $\boldsymbol{X}$, it is necessary to execute $N \times M$ divisions. Thus, an arithmetic operation count for solving the matrix equation (1) is $O_A(N^2(N+M))$. □

Corollary 1. *Let* $\boldsymbol{A}$ *be a given* $N \times N$ *matrix. The computation of the inverse matrix* $\boldsymbol{A}^{-1}$ *by algorithm* (5) *can be implemented in* $O_A(N^3)$*-time.*

Proof. Let $\boldsymbol{B} = \boldsymbol{E}$, where $\boldsymbol{E}$ is the $N \times N$ unit matrix. The solution of matrix equation (1) is $\boldsymbol{X} = \boldsymbol{A}^{-1}$. Therefore, an arithmetic operation count of matrix inversion is $O_A(N^3)$, because $N = M$. □

Similarly to the Gauss elimination, the strategy using the pivot element may be used to improve the stability of method (5). Let the k-th iteration be executed. Select the element a_{kl}^{k-1} from the k-th row of $\boldsymbol{A}_{k-1}$, which

$|a_{kl}^{(k-1)}| \geq |a_{kj}^{(k-1)}|$ for $0 \leq j, l \leq N-1$. We name it the pivot element. The column in which a_{kl}^{k-1} is placed is named the pivot column. Implement the permutation of k-th and l-th rows of $\boldsymbol{A}_{k-1}$. After this operation the pivot element is placed on the diagonal of $\boldsymbol{A}_{k-1}$. The elimination procedure is implemented for the l-th row by

$$a_{ij}^{(k)} = \begin{cases} a_{ij}^{(k-1)} & \text{if } i = l, \\ a_{ll}^{(k-1)} a_{ij}^{(k-1)} - a_{il}^{(k-1)} a_{lj}^{(k-1)} & \text{otherwise.} \end{cases} \tag{8}$$

Now we give a numerical example of this algorithm. Let $\boldsymbol{AX} = \boldsymbol{B}$ be the system defined by

$$\begin{cases} 2x_0 + 5x_1 - 8x_2 = 8, \\ 4x_0 + 3x_1 - 9x_2 = 9, \\ 2x_0 + 3x_1 - 5x_2 = 7. \end{cases}$$

Compose the expanded matrix

$$\bar{\mathbf{A}} = \left[\begin{array}{ccc|c} 2 & 5 & -8 & 8 \\ 4 & 3 & -9 & 9 \\ 2 & 3 & -5 & 7 \end{array}\right].$$

Now we can calculate $\boldsymbol{D}$.

- **Iteration 0:**

$$\boldsymbol{C}_0\bar{\boldsymbol{A}} = \begin{bmatrix} 1 & 0 & 0 \\ 9 & \mathbf{-8} & 0 \\ 5 & 0 & \mathbf{-8} \end{bmatrix} \left[\begin{array}{ccc|c} 2 & 5 & \mathbf{-8} & 8 \\ 4 & 3 & -9 & 9 \\ 2 & 3 & -5 & 7 \end{array}\right] = \left[\begin{array}{ccc|c} 2 & 5 & -8 & 8 \\ -14 & \mathbf{21} & 0 & 0 \\ -6 & 1 & 0 & -16 \end{array}\right].$$

- **Iteration 1:**

$$\boldsymbol{C}_1\boldsymbol{C}_0\bar{\boldsymbol{A}} = \begin{bmatrix} \mathbf{21} & -5 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & \mathbf{21} \end{bmatrix} \cdot \boldsymbol{C}_0\bar{\boldsymbol{A}} = \left[\begin{array}{ccc|c} 112 & 0 & -168 & 168 \\ -14 & 21 & 0 & 0 \\ \mathbf{-112} & 0 & 0 & -336 \end{array}\right].$$

- **Iteration 2:**

$$\boldsymbol{D} = \boldsymbol{C}_2\boldsymbol{C}_1\boldsymbol{C}_0\bar{\boldsymbol{A}} = \left[\begin{array}{ccc|c} 0 & 0 & 18816 & 18816 \\ 0 & -2352 & 0 & -4704 \\ -112 & 0 & 0 & 336 \end{array}\right].$$

Therefore, the equivalent system is

$$\begin{cases} 18816x_2 = 18816, \\ -2352x_1 = -4704, \\ -112x_0 = -336, \end{cases}$$

and the solution of the initial system is $x_0 = 3$, $x_1 = 2$, $x_2 = 1$.

The proposed algorithm is more computation intensive than the Gauss elimination. However, as shown below it is well suitable for parallel implementation.

3. Architecture of SPARTH-processor

The programming system for parallel computations with dynamically changed length of operands is described in more detail in papers [11], [13]. Here we explain only its main features needed to analyze the parallel realization of algorithm (5). From the user's viewpoint the system represents a parallel vector processor with programmable word length called SPARTH-processor. It is one for Superprecision Parallel ARiTHmetic computations (SPARTH-computations).

Figure 1 shows the SPARTH-processor architecture implemented within the STARAN-architecture. The main elements of SPARTH-processor are:

- super-digital vector registers $VR_0 - VR_{v-1}$ (up to 512 bits);
- high-precision parallel summator (HPS) $VS_0 - VS_3$;
- scalar registers S;
- operational STARAN-registers X, Y, M;
- index registers I for storage of constants defining the number of loop iterations, modes of access to vector registers, etc.;
- registers for temporal storage of masks $RM_0 - RM_{l-1}$ intended for storing the masks and bit slices resulting from performance of parallel vector operations (overflow, search operations, etc.);
- the FLIP-interconnection network.

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 length of operands. In this case, the number v of available vector registers is determined by required capacity

$$v = \left\lfloor \frac{s - 4n - l}{n} \right\rfloor, \tag{9}$$

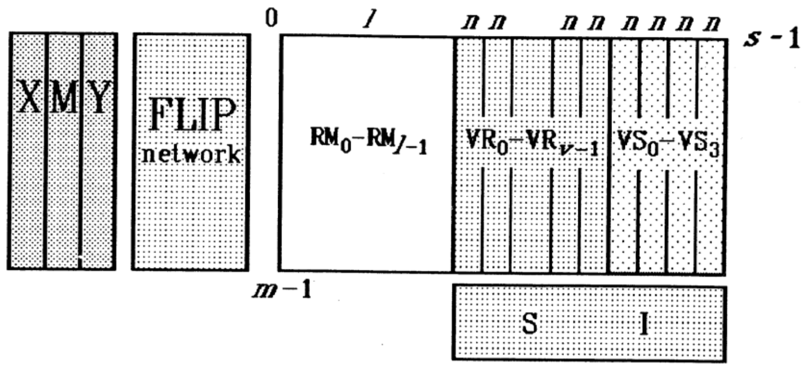

Figure 1. The SPARTH-processor architecture

where n is the ordered capacity (bits), l is the number of RM-registers and s is the size of parallel memory of STARAN. 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 capacity of operands may be altered in the interval defined by the user. To implement this, 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* type format;
- *fixed-point* type format (without integer part);
- *real* type format (with integer and fractional parts).

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 without rounding.

In Figure 1 m denotes the number of processing elements (PE's). All m components of vectors are processed in parallel. Therefore, if we assume the duration of operations with words to be a time unit, then parallel *addition* and *subtraction* of vectors have an arithmetic operation count of $O_A(1)$ and a bit operation count of $O_B(n)$ (for n-bit words). Operation counts of both parallel *multiplication* and *division* are $O_A(1)$ and $O_B(n^2)$. 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 [17].

The peculiarity of the SPARTH-library implementation is that it is reduced to few basic operations, which execute structural transformations of vectors, and provide special types of computations. This allows effective mapping of algorithms on SPARTH-processor architecture. Some important operations of this kind are the expansion and compression of vectors. These operations were developed to provide the processing of vectors of arbitrary N. The **vector-expansion** forms from an initial vector

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

contained k groups of N components each for $m \geq N \cdot k$ a vector

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

where $N_1 = 2^{\lceil \log_2 N \rceil}$ and, c is a constant. The **vector-compression** transforms $\boldsymbol{X}$ having k groups of $N_1 = 2^i$ each to

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

where $2^{i-1} < N < N_1$ and $x, \cdots$ is a "tail" of length $k(N_1 - N)$. The operation counts of both vector-expansion and vector-compression procedures in the SPARTH-processor are $O_A(\lceil \log_2 N \rceil)$, and $O_B(n \lceil \log_2 N \rceil)$.

Another type of structural transformations of vectors is implemented by the procedures executing the data duplicating. One such procedure called the **element-duplicating** makes from an initial vector $\boldsymbol{X}$ having k groups of $N = 2^i$ components each N copies of the l-th element inside each group by

$$\acute{\boldsymbol{X}} = \{\underbrace{x_l^0, \cdots, x_l^0}_{N}, \cdots, \underbrace{x_l^{k-1}, \cdots, x_l^{k-1}}_{N}\}$$

in $O_A(\log_2 N)$- and $O_B(n \log_2 N)$-time. The other procedure called .the **group-duplicating** forms N copies of the selected l-th group by

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

in $O_A(\lceil \log_2 k \rceil)$- and $O_B(n \lceil \log_2 k \rceil)$-time.

4. Features of parallel algorithm

The parallel realization of algorithm (5) can be implemented by different ways depending on the relation between the problem size of N and M, and the number of PE's of m. We distinguish three types of this relation: $m \geq N(N+M)$, $N^2 \leq m \leq N(N+M)$ and $N \leq m \leq N^2$.

Theorem 2. *The solution of matrix equation* (1) *containing n-digital elements by algorithm* (5) *can be obtained in* $O_A(N\lceil\log_2(N+M)\rceil)$*- and* $O_B(Nn(n+\lceil\log_2(N+M)\rceil))$*-time for SPARTH-processor having* $m \geq N(N+M)$ *PE's.*

Proof. As shown in Figure 2 (for $N = 4$, $M = 2$, and $k = 2$), the expanded matrix $\bar{\boldsymbol{A}}$ composed from $\boldsymbol{A}$ and $\boldsymbol{B}$ is placed in the vector register by columns. The elimination procedure (5) is implemented by parallel subtraction of vectors placed in **VR0** and **VR1** taking into account the mask **RM2** in $O_A(1)$- and $O_B(n)$-time. The **minuend vector VR0** is formed by multiplication of the matrix $\bar{\boldsymbol{A}}$ and the pivot element in $O_A(1)$- and $O_B(n^2)$-time. The **subtrahend vector VR1** is the result of parallel vector multiplication. One such vector is obtained by **group-duplicating** the pivot column for $N+M$ groups in $O_A(\lceil\log_2(N+M)\rceil)$- and $O_B(n\lceil\log_2(N+M)\rceil)$-time. The other vector is resulted by the **parallel element-duplicating** of the k-th element executed for groups of N components each in $O_A(\log_2 N)$- and $O_B(n\log_2 N)$-time. Therefore, operation counts of each iteration (5) are $O_A(\lceil\log_2(N+M)\rceil)$ and $O_B(n^2+n\lceil\log_2(N+M)\rceil)$, and the diagonalization of (1) can be implemented in $O_A(N\lceil\log_2(N+M)\rceil)$- and $O_B(Nn(n+\lceil\log_2(N+M)\rceil))$-time.

To compute the matrix of solutions of $\boldsymbol{X}$ (Figure 3), it is necessary to execute $1+\log_2 N$ parallel data transmissions and permutations in groups containing $2N, 4N, \ldots, N^2, 2^{\lceil\log_2(N+M)\rceil}$ elements each according the mask RM1 in $O_A(\log_2 N)$- and $O_B(n\log_2 N)$-time. After that, parallel division is implemented in $O_A(1)$- and $O_B(n^2)$-time. Because the elimination procedure is more computation intensive than the forming of matrix $\boldsymbol{X}$, we have the desired result. □

If $N \neq 2^i$, then the vector-expansion of $\bar{\boldsymbol{A}}$ is implemented, and computations are provided for $N_1 = 2^{\lceil\log_2 N\rceil}$.

Theorem 3. *The solution of matrix equation* (1) *containing n-digital elements by algorithm* (5) *can be obtained in* $O_A(N(\lceil\frac{NM}{m}\rceil+1)\lceil\log_2 N\rceil)$*- and* $O_B(Nn(\lceil\frac{NM}{m}\rceil+1)(n+\lceil\log_2 N\rceil))$*-time for SPARTH-processor having* $N^2 \leq m \leq N(N+M)$ *PE's.*

VR0	**RM0**	**RM1**	**VR0**	**VR1**	**RM2**	**VR0**
a_{00}	0	0	$a_{00}a_{22}$	$a_{02}a_{20}$	1	$a_{00}a_{22} - a_{02}a_{20}$
a_{10}	0	0	$a_{10}a_{22}$	$a_{12}a_{20}$	1	$a_{10}a_{22} - a_{12}a_{20}$
a_{20}	0	1	$a_{20}a_{22}$	$a_{22}a_{20}$	0	a_{20}
a_{30}	0	0	$a_{30}a_{22}$	$a_{32}a_{20}$	1	$a_{30}a_{22} - a_{32}a_{20}$
a_{01}	0	0	$a_{01}a_{22}$	$a_{02}a_{21}$	1	$a_{01}a_{22} - a_{02}a_{21}$
a_{11}	0	0	$a_{11}a_{22}$	$a_{12}a_{21}$	1	$a_{11}a_{22} - a_{12}a_{21}$
a_{21}	0	1	$a_{21}a_{22}$	$a_{22}a_{21}$	0	a_{21}
a_{31}	0	0	$a_{31}a_{22}$	$a_{32}a_{21}$	1	$a_{31}a_{22} - a_{32}a_{21}$
a_{02}	0	0	$a_{02}a_{22}$	$a_{02}a_{22}$	1	0
a_{12}	0	0	$a_{12}a_{22}$	$a_{12}a_{22}$	1	0
a_{22}	1	1	$a_{22}a_{22}$	$a_{22}a_{22}$	0	a_{22}
a_{32}	0	0	$a_{32}a_{22}$	$a_{32}a_{22}$	1	0
a_{03}	0	0	$a_{01}a_{22}$	$a_{02}a_{23}$	1	$a_{03}a_{22} - a_{02}a_{23}$
a_{13}	0	0	$a_{11}a_{22}$	$a_{12}a_{23}$	1	$a_{13}a_{22} - a_{12}a_{23}$
a_{23}	0	1	$a_{21}a_{22}$	$a_{22}a_{23}$	0	a_{23}
a_{33}	0	0	$a_{31}a_{22}$	$a_{32}a_{23}$	1	$a_{33}a_{22} - a_{32}a_{23}$
b_{00}	0	0	$b_{00}a_{22}$	$a_{02}b_{20}$	1	$b_{00}a_{22} - a_{02}b_{20}$
b_{10}	0	0	$b_{10}a_{22}$	$a_{12}b_{20}$	1	$b_{10}a_{22} - a_{12}b_{20}$
b_{20}	0	1	$b_{20}a_{22}$	$a_{22}b_{20}$	0	b_{20}
b_{30}	0	0	$b_{30}a_{22}$	$a_{32}b_{20}$	1	$b_{30}a_{22} - a_{32}b_{20}$
b_{01}	0	0	$b_{01}a_{22}$	$a_{02}b_{21}$	1	$b_{01}a_{22} - a_{02}b_{21}$
b_{11}	0	0	$b_{11}a_{22}$	$a_{12}b_{21}$	1	$b_{11}a_{22} - a_{12}b_{21}$
b_{21}	0	1	$b_{21}a_{22}$	$a_{22}b_{21}$	0	b_{21}
b_{31}	0	0	$b_{31}a_{22}$	$a_{32}b_{21}$	1	$b_{31}a_{22} - a_{32}b_{21}$

Figure 2. Executing the elimination procedure in SPARTH-processor

Proof. In this case, the matrix $\boldsymbol{A}$ is placed in vector register. $\lceil\frac{NM}{m}\rceil$ additional vector registers are needed to store $\boldsymbol{B}$. Therefore, the **group-duplicating** is implemented in $O_A(\lceil\log_2 N\rceil)$- and $O_B(n\lceil\log_2 N\rceil)$-time, and the **element-duplicating** is implemented in $O_A((\lceil\frac{NM}{m}\rceil+1)\lceil\log_2 N\rceil)$- and $O_B(n(\lceil\frac{NM}{m}\rceil+1)\lceil\log_2 N\rceil)$-time. Operation counts for (5) are $O_A(\lceil\frac{NM}{m}\rceil+1)$ and $O_B(n^2(\lceil\frac{NM}{m}\rceil+1))$. This allows to obtain the desired asymptotic estimations of execution time. $\square$

Theorem 4. *The solution of matrix equation* (1) *containing n-digital elements by algorithm* (5) *can be obtained in* $O_A(N\lceil\frac{N+M}{\lfloor\frac{m}{N}\rfloor}\rceil\lceil\log_2 N\rceil)$- *and* $O_B(Nn\lceil\frac{N+M}{\lfloor\frac{m}{N}\rfloor}\rceil(n+\lceil\log_2 N\rceil))$*-time for SPARTH-processor having* $N \leq m \leq N^2$ *PE's.*

VR0	**RM0**	**RM1**	**VR0**	**RM1**	**VR0**	**VR1**	**VR1**
a_{00}	1	0	a_{00}	0	a_{00}	b_{00}	$x_{00} = b_{00}/a_{00}$
0	0	1	a_{11}	0	a_{11}	b_{10}	$x_{10} = b_{10}/a_{11}$
0	0	1	0	1	a_{22}	b_{20}	$x_{20} = b_{20}/a_{22}$
0	0	1	0	1	a_{33}	b_{30}	$x_{30} = b_{30}/a_{33}$
0	0	1	a_{00}	0	a_{00}	b_{10}	$x_{01} = b_{01}/a_{00}$
a_{11}	1	0	a_{11}	0	a_{11}	b_{00}	$x_{11} = b_{11}/a_{11}$
0	0	1	0	1	a_{22}	b_{20}	$x_{21} = b_{21}/a_{22}$
0	0	1	0	1	a_{33}	b_{30}	$x_{31} = b_{31}/a_{33}$
0	0	1	0	1	a_{00}	*	*
0	0	1	0	1	a_{11}	*	*
a_{22}	1	0	a_{22}	0	a_{22}	*	*
0	0	1	a_{33}	0	a_{33}	*	*
0	0	1	0	1	a_{00}	*	*
0	0	1	0	1	a_{11}	*	*
0	0	1	a_{22}	0	a_{22}	*	*
a_{33}	1	0	a_{33}	0	a_{33}	*	*
b_{00}	0	0	b_{00}	0	b_{00}	*	*
b_{10}	0	0	b_{10}	0	b_{10}	*	*
b_{20}	0	0	b_{20}	0	b_{20}	*	*
b_{30}	0	0	b_{30}	0	b_{30}	*	*
b_{01}	0	0	b_{01}	0	b_{01}	*	*
b_{11}	0	0	b_{11}	0	b_{11}	*	*
b_{21}	0	0	b_{21}	0	b_{21}	*	*
b_{31}	0	0	b_{31}	0	b_{31}	*	*

Figure 3. Computing the matrix of solutions

Proof. $\lfloor \frac{m}{N} \rfloor$ columns can be stored in one vector register, and $\lceil \frac{N+M}{\lfloor \frac{m}{N} \rfloor} \rceil$ vector registers are needed to place $N+M$ columns of $\bar{\boldsymbol{A}}$. Therefore, $\lceil \frac{N+M}{\lfloor \frac{m}{N} \rfloor} \rceil$ arithmetic and duplicating operations are needed to implement each iteration (5), and the diagonalization of $\bar{\boldsymbol{A}}$ can be executed in $O_A(N\lceil \frac{N+M}{\lfloor \frac{m}{N} \rfloor} \rceil \lceil \log_2 N \rceil)$- and $O_B(Nn\lceil \frac{N+M}{\lfloor \frac{m}{N} \rfloor} \rceil (n + \lceil \log_2 N \rceil))$-time.

To calculate $\boldsymbol{X}$, the diagonal elements of $\boldsymbol{A}$ are moved into one vector register in $O_A(\lceil \frac{N}{\lfloor \frac{m}{N} \rfloor} \rceil \lceil \log_2 N \rceil)$- and $O_B(n\lceil \frac{N}{\lfloor \frac{m}{N} \rfloor} \rceil \lceil \log_2 N \rceil)$-time. Then $\lceil \frac{M}{\lfloor \frac{m}{N} \rfloor} \rceil$ parallel divisions are executed.

Because the elimination procedure is more computation intensive than the forming of matrix $\boldsymbol{X}$, we have the desired result. $\square$

Given theorems confirm the possibility of effective SIMD-parallelization of method (5). Note also that the selection of the pivot element is a usual search operation for systems of this kind. It can be implemented in $O_A(1)$– and $O_B(n)$–time.

Rounding errors can be avoided by the accurate calculation of (5). To do this, multiplications and subtractions are implemented in HPS without rounding. The dynamically changed length of operands allows exact calculations for a sequence of iterations (5). The necessity of switching to next capacity limit can be selected, for example, by analyzing overflows for integer type of data. The fast growing of the operand length can be avoided by scaling down the values of elements placed in rows of $\bar{\boldsymbol{A}}$. It is possible to implement this operation as a shift of numbers to the least-significant bits for all elements of each row.

5. Results of numerical experiments

In this section the results of some numerical experiments are presented. Computations were implemented in STARAN-like system with following characteristics: $m = 256$, execution times of addition and multiplication are $t_a \cong 6n\tau$ and $t_m \cong 6n^2\tau$ correspondingly, where $\tau \cong 200ns$ is the average execution time of each parallel bit operation.

Figure 4 shows the dependence of the execution time from the problem size of N and the length of operands of n (in bits) for $M = N$. Computations were implemented according to the relation between N and m. One of three parallel algorithms described in Section 4 was selected to support the optimal solution rate.

Figure 5 contains results of estimating a relative speedup defined as a ratio between times needed for sequential (in one processing channel) and parallel solutions.

To estimate the accuracy of proposed algorithm we have used the "hard" input data. One example of such data is the following system:

$$\begin{bmatrix} 1 & & & \\ 1.07 & 1 & & \\ 1.02 & 1.10 & 1 & \\ \alpha_1 & \alpha_2 & \beta & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2.07 \\ 3.12 \\ \gamma \end{bmatrix},$$

where $\alpha_1 = 0.993 \cdot 10^{14}$, $\alpha_2 = -(\alpha_1 + 4)$, $\beta = -0.34 \cdot 10^{-3}$ and $\gamma = -4.00017$.

Computations were implemented in the following stages:

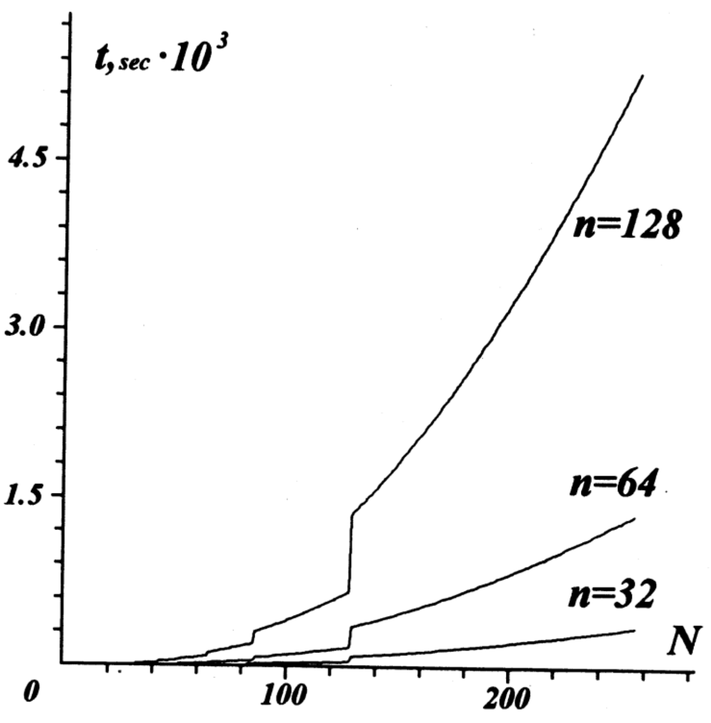

Figure 4. Execution time

- parallel conversion of input floating point numbers to SPARTH-format with an automatic selection of capacity needed for their exact representation;
- parallel solution of initial system executing the iteration (5) without rounding;
- parallel transformation of results to the floating-point format.

It is easy to verify that the capacity needed for the exact representation of this matrix is 128 bits. Computations implemented for this capacity limit allow to obtain the exact solution. It is $x_0 = x_1 = x_2 = 1$ and $x_3 = 0.17 \cdot 10^{-3}$.

6. Conclusion

Our experience of the implementation of SPARTH-processor confirms that it can be a serious basis for parallel high-accuracy computations. A great attention is directed to experiments with ill-conditioned matrices like the Hilbert one. In this case we not only employ very-long operand words

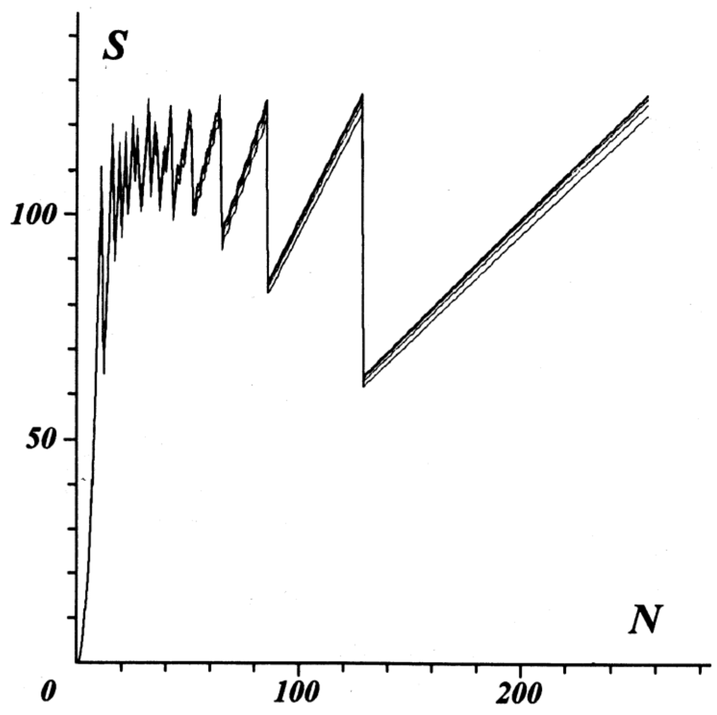

Figure 5. Relative speedup

but excluding the intermediate division operations. The proposed algorithm illustrates this approach.

Note also that the given results were obtained for a system having relatively low performance. The modern computers are significantly different from a system used in our investigations (for example, $m = 65536$, $\tau \cong 50ns$, $t_a \cong 2n\tau$ and $t_m \cong 2n^2\tau$ for CM-computers). This can allow very effective implementation of proposed method.

References

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

[2] A.L. Lucke, Programmed word length computer, Proceedings of A.C.M. National Meeting, 1967, 57–65.

[3] M.S. Cohen, T.E. Hull, V.C. Hamacker, CADAC: a controlled-precision decimal digital arithmetic Unit, IEEE Trans. on Comput., 32, 1983, 370–377.

[4] B. Serpette, J. Vullemin, J.C. Herve, A portable efficient package for arbitrary-precision arithmetic, PRL reports, DEC, Paris, Research Lab., 85, France.

[5] J.J. Thomas, S.R. Parker, Implementing exact calculations in hardware, IEEE Trans. Comput. 36, 1987, 764–768.

[6] Fr. Krückeberg, Arbitrary accuracy with variable precision arithmetic, Interval Mathematics 1989, Proc. Int. Symp., Freiburg, FRG, 1985, eds. K. Nickel, Lecture Notes in Comput. Sci., 212, 95–101.

[7] E. Lange, Implementation and test of the ACRITH facility in a System/370, IEEE Trans. on Comput. 36, 1987, 1088–1096.

[8] J.H. Bleher, S.M. Rump, U. Kulish, M. Metzger, Ch. Ullrich, W. Walter, FORTRAN-SC: a study of a FORTRAN extension for engineering/scientific computation with access to ACRITH, Computing 39, Springer, Berlin, 1987, 93–110.

[9] N.N. Mirenkov, Parallel Computers for Overcoming Rounding Errors, Prepr. of Institute of Mathematics, OBC-08, Novosibirsk, 1979 (in Russian).

[10] D. Buell, R. Ward, A multiprecise integer arithmetic package, The Journal of Supercomputing, 3, 1989, 89–107.

[11] A. Vazhenin, Programming system of High-accuracy computations for associative array processor, Proc. on CONPAR-90/VAPP-IV, Volume of special technical contributions, Zürich, 1990, C69–C72.

[12] A. Vazhenin, Hardware and algorithmic support of high-accuracy computations in vertical processing systems, Proc. of Int. Conf. "Parallel Computing Technologies", 1993, Obninsk, Russia, Vol. 1 ReSCo J.-S.Co., Moscow, 1993, 149–162.

[13] A. Vazhenin, Implementation of high-accuracy computations in vertical processing systems, Bulletin of the Novosibirsk Computing Center, Series: Computer Science, Issue: 1, NCC Publisher, Novosibirsk, 1993, 51–62.

[14] J.R. Rice, Matrix Computations and Mathematical Software, McGraw-Hill Book Company, New York, 1981.

[15] A.G. Kurosh, Course of Higher Algerbra, FISMATGIS, Moscow, 1963 (in Russian).

[16] D.C. Fadeev, V.N. Fadeeva, Computational Methods of Linear Algebra, FISMATGIS, Moscow, 1963 (in Russian).

[17] K.J. Thurber, Large-scale Computer Architecture: Parallel and Associative Processors, Rochelle Park, Hayden Book Comp., New York, 1976.