Parallelization of computations for finding the rank of a rectangular matrix

Sergey Novikov

Institute of Computer Science, Siedlce University of Natural Sciences and Humanities, 3 Maja Street, 54, 08-110 Siedlce, Poland

Abstract: The paper presents two parallel algorithms for finding the rank of a rectangular matrix and two parallel algorithms for generation of combinations without repetitions represented by Boolean vectors, that are used in an algorithm for finding the rank of a rectangular matrix .

Keywords: parallel algorithm, cluster, control processor, Boolean vector, combinations without repetitions, matrix, fringing minor, determinant, rank.

1 Introduction

As you know, over 75% of all computer problems can be reduced to the problem of solving systems of linear algebraic equations [1]. An important task in solving systems of linear algebraic equations is to find the rank of an $m \times n$ matrix A, where m is the number of rows and n is number of columns of A. Without loss of generality, we can assume that $m \leq n$.

The column rank of a matrix A is the maximum number of linearly independent column vectors of A. The row rank of a matrix A is the maximum number of linearly independent row vectors of A. The column rank and the row rank are always equal. This number (i.e. the number of linearly independent rows or columns) is simply called the rank of A. It is commonly denoted by either $rk(A)$.

The rank of an $m \times n$ matrix cannot be greater than m or n.

One useful application of calculating the rank of a matrix is the computation of the number of solutions for a system of linear equations. According to the Kronecker – Capelli theorem, a system of linear equations is inconsistent if the rank of the augmented matrix is greater than the rank of the coefficient matrix. If, on the other hand, the ranks of these two matrices are equal, the system must have at least one solution. The solution is unique if and only if the rank equals the number of variables. Otherwise the general solution has k free parameters, where k is the difference between the number of variables and the rank. Hence in such case there are an infinite number of solutions.

In the control theory, the rank of a matrix can be used to determine whether a linear system is controllable, or observable.

The easiest way to compute the rank of a matrix A is given by the Gauss elimination method [1,2]. However, Gaussian elimination is not always the fastest algorithm to compute the rank of a matrix, procedure consistent elimination of the matrix elements is difficult for parallelization, when applied to floating point computations on computers, basic Gaussian elimination can be unreliable [3].
Another well-known method of finding the rank an $m \times n$ matrix is based on a study of fringing (bordering) minors [1] with increasing orders (i.e. the amount of ***1, 2,. ..., m***). Having established that a determinant $D(M_i^*) \neq 0$ for a minor M_i^* with the order i (***i***-minor), the conclusion is that $rk(A) \geq i$. Next you need to calculate the determinants of all ***i+1***-minors, fringing the minor M_i^*. If it turns out that each of the ***i+1***-minors is equal to ***0***, then the conclusion is that the $rk(A) = i$.
Using this method you should to calculate a large number of determinants, the order of which can increase to a maximum. The number of minors with the order ***r+1***, each of which spans a ***r***-minor M_r, is equal to ***(m-r)(n-r)*** . It may be necessary to explore $\sum_{i=1}^{m-1}(m-i)(n-i)$ minors for finding the rank of an $m \times n$ matrix. In this regard it is interesting to explore the possibilities of using modern high-performance computational systems for solving the above-mentioned important task.

2 A parallel algorithm for finding the rank of a matrix by the method fringing minors

Multiprocessor computer systems (clusters) allow to accelerate significantly the process of generation of the fringing minors.
Each minor M_r with the order r (***r***-minor) of a matrix A can be formed with the help of the pair of Boolean vectors $(b^r(m), c^r(n))$, where $b^r(m) = b^r_1 b^r_2 \ldots b^r_m$, $c^r(n) = c^r_1 c^r_2 \ldots c^r_n$. Each of these vectors contains a r of single components. Such a pair of vectors uniquely identifies a minor M_r. So, for example, for the matrix

$$A = \begin{Vmatrix} -4 & 3 & 2 & 1 & 0 \\ -2 & 1 & 1 & -4 & 2 \\ 12 & -6 & -6 & 9 & 3 \\ -6 & 4 & 3 & -3 & 1 \end{Vmatrix}$$

with a non-zero $M_2 = \begin{vmatrix} -4 & 3 \\ -2 & 1 \end{vmatrix}$

all ***3***-minors, fringing minor M_2, can be described with the help of the following pairs of Boolean vectors: ***1) $b^3(4) = 1110$, $c^3(5) = 11100$; 2) $b^3(4) = 1110$, $c^3(5) = 11010$; 3) $b^3(4) = 1110$, $c^3(5) = 11001$; 4) $b^3(4) = 1101$, $c^3(5) = 11100$; 5) $b^3(4) = 1101$, $c^3(5) = 11010$; 6) $b^3(4) = 1101$, $c^3(5) = 11001$.***

We will also use the k-component Boolean vector $e(k)_i = e_1\, e_2 \ldots e_k$ with the only non-zero component of $e_i = 1$, where $1 \le i \le k$, in our algorithms of generating vectors $b^r(m)$ and $c^r(n)$.

The algorithm ***A1*** to generate vectors $b^{r+1}(m)$ for the set S_1, elements of which identify the rows of $(r+1)$-minors , fringing r-minor M_r, has the form

$b^r(m) = b^r{}_1\, b^r{}_2 \ldots b^r{}_m$, $S_1 = \varnothing$, $i:=1$;
while $i \le m$ **do**
if $b^r{}_i = 0$ **then**
begin $b^{r+1}(m) := b^r(m) \vee e(m)_i$; $S_1 := S_1 \cup \{ b^{r+1}(m) \}$;
end;
$i:=i+1$;
print S_1
end.

The algorithm ***A1*** allows us to generate also the vectors $c^{r+1}(n)$ for the set S_2 , elements of which identify the columns of $(r+1)$-minors, fringing r-minor M_r. After slight modernization the algorithm (***A1***') takes the following form

$c^r(n) = c^r{}_1\, c^r{}_2 \ldots c^r{}_n$, $S_2 = \varnothing$, $i:=1$;
while $i \le n$ **do**
if $c^r{}_i = 0$ **then**
begin $c^{r+1}(n) := c^r(n) \vee e(n)_i$; $S_2 := S_2 \cup \{ c^{r+1}(n) \}$;
end;
$i:=i+1$;
print S_2
end.

Our parallel algorithm for finding the rank of a matrix by the method of fringing minors can be described as follows:

1. *Selection of non-zero a minor*

The control processor p_0 finds a non-zero element of an $m \times n$ matrix A, where $m \le n$, with the help of the algorithm ***A0(A; $b^r(m)$, $c^r(n)$)***. This element $a_{ij} \ne 0$ is a minor with the first order (***1***-minor), We can identify the ***1***-minor by a pair of Boolean vectors $b^r(m)$, $c^r(n)$, where $b^r(m) := e(m)_i$, $c^r(n) := e(n)_j$. It is obvious that in case of absence of non-zero elements in the matrix we have $rk(A) = 0$. The transition to the paragraph 6.

Then the control processor sends vectors $b^r(m)$ and $c^r(n)$ as the input data for computing processors p_1 and p_2 of our cluster.

2. *The generation of Boolean vectors, identifying the fringing minors*

A computing processor p_1 generates the set S_1, elements of which identify the rows $(r+1)$-minors, fringing the r-minor M_r^* , with the help of our algorithm ***A1($b^r(m)$; S_1)***.

At the same time a computing processor p_2 generates the set S_2, elements of which identify the columns of $(r+1)$-minors, fringing the r-minor M_r^*, with the help of our algorithm $A1'(c^r(n); S_2)$.

Then each computing processor sends to the control processor p_0 the sets S_1 and S_2 .

3. The formation of pairs of vectors to build fringing minors

The control processor p_0 composes the pairs of this vectors to build fringing $(r+1)$-minors for the r-minor M_r^* with the help of the algorithm $A2(S_1, S_2;(b^{r+1}(m), c^{r+1}(n))$. After that p_0 sends one of the pairs together with the matrix A as a source of data for each of the computing processor.

4. The building of fringing minors and calculation of the determinants

Every computing processor p_i, where $i \in \{1,2,...,(m-r)*(n-r)\}$, builds $(r+1)$-minors with the help of the algorithm $A3(A, (b^{r+1}(m), c^{r+1}(n)); M_{r+1})$. After that p_0 computes the determinant $D(M_{r+1})$ for built submatrices M_{r+1} with the help of the algorithm $A4(M_{r+1} ; D(M_{r+1}))$. To compute the determinant you can use the method of Gauss elimination [1,2].
The algorithm $A4$ for the calculation of the determinant of a square $n \times n$ matrix A has the following form

$k:=1$
$While\ k \le n-1\ do$
$\quad i=k+1$
$\quad While\ i \le n\ do$
$\quad\quad t_{ik} := a_{ik} / a_{kk}$
$\quad\quad i:=i+1$
$\quad\quad\quad j=k+1$
$\quad\quad\quad While\ j \le n\ do$
$\quad\quad\quad a_{ij} := a_{ij} - t_{ik}*a_{kj}$
$\quad\quad\quad j: = j+1$
$k: = k+1$
$Det\ A = a_{11}*a_{22}*\ldots*a_{nn}$

Note that the task of finding out, if a determinant is equal to 0 , is solved much easier (using modular arithmetic) objectives of the actual computation of this determinant [4].
After that each computing processor p_i sends the obtained results (a pair of vectors $(b^{r+1}(m), c^{r+1}(n))$ and the value of the $(r+1)$ - minor) to the control processor p_0 .

5. Adoption of a decision on the continuation of computing

The control processor p_0 analyzes the results with the help of the algorithm $A5((b^{r+1}(m), c^{r+1}(n)), D(M_{r+1}) ; rk(A), b^r(m), c^r(n))$. If each of the $(r+1)$-minors,

fringing ***r***-minor M_r^*, is equal to zero, then the rank of the matrix ***A*** is equal to ***rk(A) = r.*** The end of the calculations. The transition to the paragraph 6. If there is a minor M_{r+1}^* with $D(M_{r+1}^*) \neq 0$, then the $rk(A) \geq r$. In this case we must continue to calculate.
Then $b^r(m) := b^{r+1}(m)$, $c^r(n) := c^{r+1}(n)$.

The transition to paragraph 2.

6. *Stop*.

Our parallel algorithm ***APRM1*** for finding the rank of a matrix by means of research fringing ***r***-minors implements the following computer schedule
$S_{R+1}(APRM1)=((A0, p_0), (A1, p_1, p_2), (A2, p_0), (A3, p_{1,\ldots}, p_R), (A4, p_{1,\ldots}, p_R), (A5, p_0)),$
where a record (Aj,p_i) indicates that the processor p_i implements the algorithm Aj.
We are finishing the process of finding the rank of the matrix discussed above as an example.
Our ***3***-minor M_3^*, identified by vectors ***1110*** and ***11010*** , is not zero. Then $rk(A) \geq 3$. Only two ***4***-minors fringe the minor M_3^*. The first of them, identified by vectors ***1111*** and ***11110***, is zero, and the second, identified by vectors ***1111*** and ***11011***, is equal to ***-270***. Thus it is established, that ***rk(A)=4.***
We can use this algorithm for matrices with a large number of zero elements, as well as for matrices with equal (or almost equal) rows, or columns. For such matrices there is a high probability that the rank of ***A*** is significantly smaller than ***m*** .

3 Finding the rank of a matrix by means of research *r*-minors with descending orders

We must solve the task of finding the rank of a matrix ***A*** for matrices with different properties. The considered task can be solved faster through research minors, starting with a minor of maximum the order, if it is strong possibility, that the rank of the matrix is not much smaller than ***m***.
So, for example, to find the rank of the matrix ***A***, discussed above, there are twelve ***2***-minors, fringing the minor $M_1 = -4$, six ***3***-minors, fringing the minor M_2, determined by the vectors 1100 and 11000, and two ***4***-minors, fringing the minor M_3, defined by vectors 11100 and 11010. It turned out that ***rk(A)=4***. It would be enough to explore the whole five ***4***-minors for finding the rank of this matrix by means of research minors, starting with a minor of maximum order.
The rank of an $m \times n$ matrix can be found as follows.
We generate all the maximal minors with the ***m***-order. For this, we need to generate all possible combinations without repetitions of ***n*** elements taken ***m*** at a time. We form the ***m***-minor M_m for each combination represented by vectors $c^m(n)$, $b^m(m)$ and calculate the corresponding determinant. If it turns out, that the formed minor has $D(M_m) \neq 0$, then the $rk(A)= m$. We must explore $(m-1)$-minors using vectors $b^{m-1}(m)$ and $c^{m-1}(n)$, if the determinant of each of the formed ***m***-minor is equal to ***0***. Calculations are repeated until the rank of our matrix is found.

The effectiveness of this approach can be improved using modern multiprocessor computational systems.

For realization of this approach required an algorithm for generation of combinations without repetitions.

4 Generation of combinations without repetitions represented by Boolean vectors

We will represent a combination without repetitions of n elements taken m at a time, where $m \le n$, by the n-component Boolean vector $b^m(n) = b_1 b_2 \dots b_n$, which contains a m of single components (units). For example, we can imagine the combination $(1,3,4,7)$, selected from the set $\{1,2,3,4,5,6,7,8\}$, by the 8-component Boolean vector $b^4(8) = 10110010$, where components $b_1 = b_3 = b_4 = b_7 = 1$. It is obvious, each n-component Boolean vector can be regarded as the word of length n in the alphabet $\{0,1\}$.

An algorithm was proposed, where combinations are presented as a sequence of integers and generated in the lexicographical order [5]. We can offer a more efficient and convenient for parallelization algorithm for generation of combinations without repetitions, where combinations are presented as Boolean vectors.

The initial combination C_0 is written as the n-component Boolean vector $b^m(n) = b_1 b_2 \dots b_n$, in which the first (far left) m components are equal to 1, and all other components are equal to 0. Another combination C_{i+1} we obtain from the previous C_i as follows. We find the right-most «unit» in the C_i (let it be placed in the position of j) and move it to the right by one bit, provided that, in position $j+1$ is written 0. If in the position $j+1$ of the combination C_i was recorded 1 or a sign of the end of the word, we get the new combination C_{i+1} by another way.

We find the right-most « unit » in the C_i, which is recorded to the left of the block of s «right-wing» units, where $1 \le s \le m-1$. We move it on one bit to the right (from position j to position $j+1$) , but in the positions $j+2, j+3, \dots, j+s+1$ we move all s «right-wing» units.

After the completion of the building of the C_{i+1} we build a new combination from the previous $C_i := C_{i+1}$.

If it turns out that to the left of the block of s «right-wing» units have no «units» in the C_i, the final combination represents the n - component Boolean vector, in which the most right-wing m components are equal to 1, and all other components are equal to 0.

Our algorithm $A11(m,b^m(n); S)$ for generation of combinations without repetitions, used below in a parallel algorithm for finding the rank of a matrix, has the form:

$j:=m;\ S:=\{b^m(n)\}$
while $j \le n-1$ **do**
begin
  if $b_j = 1$ **and** $b_{j+1} = 0$ **then**
    begin $b^m(n) := b^m(n) \oplus e_j$; $b^m(n) := b^m(n) \vee e_{j+1}$; $S := S \cup \{b^m(n)\}$
    end
  until $b_{j+1} = 1$ **or** $j = n$

```
begin
 j:=n ; d:= 0;
while  j ≥ 1  and b_j =1  do
if   b_j =1  then
begin d:= d+1 ; b^m (n):=b^m (n)  ⊕  e_j  ; j:=j-1
end
until b_j = 0 ;
    while  j ≥ 1  do
     begin
         if   b_j =0  then  j: = j-1
         until b_j = 1 ;
         begin b^m (n):=b^m (n)  ⊕  e_j  ; b^m (n):=b^m(n)  ∨ e_{j+1} ;
        end
         i:= 1; k:=j+1;
        while i ≤ d   do
        begin
        k:=k+1;  b^m (n):=b^m (n)  ∨ e_k ; i:= i+1
        end
     end
       S:= S ∪  {b^m (n) }
  end
j:= j+d+1
end;
write S. Stop.
```

Example 1. Let $n=5$, $m = 3$

The initial combination is $b^3(5)= C_0 = 11100$. Then we have $C_1 = 11010$, $C_2 = 11001$. To obtain C_3 we move the «unit», recorded to the left of the block of $s=1$ «right-wing» units, by one digit (from the second to the third). In addition we move to the fourth position block of $s=1$ the «right wing» of the units. Thus we get $C_3 = 10110$. Then we have $C_4 = 10101$. The following combination is equal to the $C_5 = 10011$. Block «right-wing» units already contains two units.

For obtaining C_6 we move the «unit», recorded to the left of the block of $s=2$ the «right» units, by one digit (from the first to the second). In addition to this move in the third and fourth position the block of $s=2$ the «right wing» of the units. We get $C_6 = 01110$. Next, we obtain $C_7= 01101$, $C_8= 01011$, $C_9= 00111$. Finally we get

$S=\{11100, 11010, 11001, 10110, 10101, 10011, 01110, 01101, 01011, 00111\}$
$=\{123, 124, 125,134, 135,145, 234,235,245,345\}.$

We can parallelize the process of generation of combinations without repetitions in the process of finding the rank of an $m \times n$ matrix.

5 A parallel algorithm for generation of combinations without repetitions

We can parallelize the task of generation of all possible combinations of n elements taken m at a time, where $m \le n$, on 2^k processes, where $k < m$, $2^k \le R$, R - number of planned for generation of computing processors.
Our parallel algorithm ***APGC1*** for generation of combinations without repetitions can be described as follows:

1. *Preparation of data for parallelization*

The control processor p_0 prepares for each computing processor p_i input data with the help of the algorithm $A10P1(m, n, R; k, \alpha_i(k))$. The input data are the numbers n, m, k, as well as Boolean vector $\alpha_i(k)$ of length k. Boolean vector $\alpha_i(k)$ is recorded in the binary notation of the number i. The number i, where $0 \le i \le 2^k-1$, indicates the number of the i-th group of combinations, planned for generation by a computing processor p_i . We'll plan a computing processor, which denote by the number 2^k, for generation of the 0-th group of combinations and plan computing processors $p_1, p_2, \ldots, p_{2^k-1}$ for generation 1-th, 2-th, …, 2^k-1-th groups of combinations.

2. *Generation of a i-th group of combinations without repetitions*

Each computing processor p_i first generates the $(n-k)$ - component Boolean vector $b^s(n-k)=b_1^{\,s}\, b_2^{\,s} \ldots b_{n-k}^{\,s}$ with the help of the algorithm $A11P1(m, n, k, \alpha_i(k) ; S_i)$. The first (far left) s components of this vector are equal to 1, i.e. $b_1^{\,s}=b_2^{\,s}=\ldots=b_s^{\,s}=1$. The value of s is determined by the formula $s=m-\alpha_i(1)$, where $\alpha_i(1)$ is the number of units in the word α_i. In the rest of the $n-k-s$ positions of this Boolean vector $b^s(n-k)$ should be written down 0. The presence of the vector $b^s(n-k)$ allows us to take advantage of the described above serial algorithm ***A11*** for generation of combinations without repetitions of $n-k$ elements taken s at a time. The initial combination is a vector: $b:=b^s(n-k)$, $m:=s, n:=n-k$. Each computing processor p_i generates C_{n-k}^{s} of Boolean vectors of length $n-k$ with the help of the serial algorithm ***A11***. After this each processor p_i appends to the left up to each of these vectors the prefix $\alpha_i(k)$ and forms the set S_i of vectors with length n. Thus the process of generation of the i-th group combinations will be completed. Next each processor p_i sends the set S_i to the control processor.

3. *The completion of the generation of all possible combinations of n elements taken m at a time*

Our control processor sums the sets ($S_0 \cup S_1 \cup \ldots \cup S_{2^k-1}=S$) received from the computing processors with the help of the algorithm $A12P1(S_0, S_1, \ldots, S_{2^k-1}; S)$ and

completes the process of generation of all possible combinations of ***n*** elements taken ***m*** at a time.
Our parallel algorithm APGC1 for generation of all possible combinations of n elements taken m at a time uses R=2k computing processors and implements the following computer schedule

$$S_{2^{k}+1}^{k}(APGC1)=(\,(A10P1,\,p_0),\,(A11P1,\,p_1,\,\ldots,\,p_R),\,(A12P1,\,p_0)\,),$$

where a record (Aj,p_i) indicates that the processor p_i implements the algorithm Aj.
We prove that our parallel algorithm really generates all C_n^m combinations without repetitions of n elements taken m at a time by the method of mathematical induction on the parameter k, where k is the prefix length.
For $k=1$ we have two sets of vectors (S_0 and S_1). The prefix of each of the combinations from the set S_0 is $\alpha_0 = 0$, and the prefix of each of the combinations from the set S_1 is $\alpha_1 = 1$. Hence it follows

$$|\,S_0\,| = C_{n-1}^{m-\alpha_0(1)} = C_{n-1}^{m},$$

$$|\,S_1\,| = C_{n-1}^{m-\alpha_1(1)} = C_{n-1}^{m-1}.$$

It is obvious that $S_0 \cap S_1 = \varnothing$.

Therefore

$$|\,S_0 \cup S_1\,| = |\,S_0\,| + |\,S_1\,| = C_{n-1}^{m} + C_{n-1}^{m-1} = C_n^m.$$

Let us assume (induction assumption) that for the prefix length equal to k , is equal to

$$|\,S_0 \cup S_1 \cup S_2 \cup \ldots S_{2^k-1}\,| = C_n^m. \qquad (*)$$

Let prefix length be equal to $k+1$.
Each prefix length $k+1$ can be written in the form of a prefix length k to which it is written right side 1 or 0. For each prefix α_i with length k there are two prefixes with length $k+1$ different the last letter. For example, for α_i with length k there is a prefix $\alpha'_i = \alpha_i\,0$ and the prefix $\alpha'_l = \alpha_i\,1$ with length $k+1$. These words are prefixes with length $k+1$ of Boolean vectors (words), which represent combinations of sets S'_j and S'_l .
We have

$$|\,S'_j\,| = C_{n-(k+1)}^{m-\alpha'_j(1)} = C_{n-(k+1)}^{m-\alpha_i(1)} = C_{n-(k+1)}^{s}$$

$$| S'_l |= C^{m-\alpha'_l(1)}_{n-(k+1)} = C^{m-\alpha_i(1)-1}_{n-(k+1)} = C^{s-1}_{n-(k+1)}$$

From here we get

$$| S'_j \cup S'_l |=| S'_j | + | S'_l |= C^{s}_{n-(k+1)} + C^{s-1}_{n-(k+1)} = C^{s}_{n-k},$$

where $s = m-\alpha_i(1)$.
Thus, for each prefix α_i with length k, where $0 \le i \le 2^k-1$, there are two prefix $\alpha'_j = \alpha_i 0$ and $\alpha'_l = \alpha_i 1$ with length $k+1$. Each element of the set S'_j has a prefix with length equal to $\alpha'_i = \alpha_i 0$ and each element of the set S'_l has a prefix length equal to $\alpha'_l = \alpha_i 1$. The sum of powers of these sets S'_j and S'_l is equal to the power of the sets S_i.

We have

$$| S'_0 \cup S'_0 \cup S'_0 \cup ... S'_{2^{k+1}-1} |=| S_0 \cup S_1 \cup S_2 \cup ... S_{2^k-1} |.$$

On the basis of (*) we conclude

$$| S'_0 \cup S'_1 \cup S'_2 \cup ... S'_{2^{k+1}-1} \cup |= C^m_n.$$

Thus proven, if we parallelize computations on the 2^k processes by the above method it would generate all possible combinations without repetitions of n elements taken m at a time. This provides high performance our computing system.
However, the above method requires too much computing processors. This leads to an increase of the cost of our computing system.
In addition, some of these processors may duplicate calculations that other computing processors perform.
For example, if $k = 3$, each of the processors p_1 and p_2 will generate the same set of $(n-3)$ - component vectors with $m-1$ single components $S(n-3,m-1)$, where

$$| S(n-3, m-1) |= C^{m-1}_{n-3}.$$

Then for every vector from this set $S(n-3,m-1)$ the processor p_1 appends to the left up the prefix 001, and for every vector from this set $S(n-3,m-1)$ the processor p_2 appends to the left up the prefix 010. Thus will be constructed the sets S_1 and S_2 of vectors with length n.
In order to reduce the cost of computations and eliminate duplications of calculations we offer the following algorithm. We can parallelize the task of generation of all possible combinations of n elements taken m at a time, where $m \le n$, on $k=R-1$ processes, where R - number of planned for generation of computing processors.

Our parallel algorithm ***APGC2*** for generation of combinations without repetitions can be described as follows:

1. *Preparation of data for parallelization*

The control processor p_0 prepares for each computing processor $p_1, p_2, \ldots, p_{k+1}$ input data with the help of the algorithm ***A10P2(m, n, k; i, b^i(k), m-i, b^{m-i}(n-k))***. Here ***i, k, m, n*** are the numbers, where $0 \le i \le k < m < n$, $k = R-1$, and $b^i(k)$, $b^{m-i}(n-k)$ are Boolean vectors with length k and $n-k$, in which the first (far left) k ($n-k$ for $b^{m-i}(n-k)$) components are equal to 1, and all other components are equal to 0. The number i, where $0 \le i \le k$, indicates the number of the i-th group of combinations, planned for generation by a computing processor p_i. We'll plan a computing processor, which denote by the number R, for generation of the 0-th group of combinations and plan computing processors $p_1, p_2, \ldots, p_{R-1}$ for generation 1-th, 2-th, …, k-th groups of combinations.

2. *Generation of a **i**-th group of combinations without repetitions*

Each computing processor p_i with the help of the algorithm ***A11P2(i, b^i(k), m-i, b^{m-i}(n-k); S_i)*** first generates all k - component Boolean vectors, each of which contains i single components. Processor p_i then generates all $(n-k)$ - component Boolean vectors, each of which contains $m-i$ single components. We can use the described above serial algorithm ***A11*** for these goals.
Processor p_i then writes all the generated vectors with length k to the set S_{i1} and writes all the generated vectors with length $n-k$ to the set S_{i2}, where

$$| S_{i1} | = C_k^i, | S_{i2} | = C_{n-k}^{m-i}.$$

After this processor p_i appends to the left up to each of vector from the set S_{i2} the vector from the set S_{i1} and forms the set S_i of vectors with length n, where

$$| S_i | = C_k^i * C_{n-k}^{m-i}.$$

Thus the process of generation of the i-th group combinations without repetitions n elements taken m at a time will be completed.
After that processor p_i sends the set S_i to the control processor.

3. *The completion of the generation of all possible combinations of **n** elements taken **m** at a time*

Our control processor sums the sets ($S_0 \cup S_1 \cup \ldots \cup S_k = S$) received from the computing processors with the help of the algorithm ***A12P2(S_0, S_1,…, S_k; S)*** and completes the process of generation of all possible combinations of n elements taken m at a time.

Our parallel algorithm APGC2 for generation of all possible combinations of n elements taken m at a time uses R=k+1 computing processors and implements the following computer schedule

S_{k+2} **(APGC2)=((A10P2,** p_0**), (A11P2,** p_1**, ...,** p_{k+1} **), (A12P2,** p_0**)),**

where a record **(Aj,**p_i**)** indicates that the processor p_i implements the algorithm **Aj.**

The corresponding computational system requires far less computing processors (than computational system for the algorithm ***APGC1***), none of which not duplicates computations from other computing processors.

Due to the way of parallelization of calculations according to the algorithm ***APGC2*** we can solve our task of generating combinations for the large values of parameter ***k*** (up to 32), and, consequently, the parameters ***n, m***.

Example 2. Let ***n=10, m=6, R=5, k=4.***

1. Data for parallelization

$b^0(4)=0000$, $b^6(6)= 111111$ for p_5
$b^1(4)=1000$, $b^5(6)= 111110$ for p_1
$b^2(4)=1100$, $b^4(6)= 111100$ for p_2
$b^3(4)=1110$, $b^3(6)= 111000$ for p_3
$b^4(4)=1111$, $b^2(6)= 110000$ for p_4

*2. Generation of a **i**-th group of combinations*

$|S_{01}|=1$, $|S_{02}|=1$, $|S_0|=1$, $S_0=\{0000111111\}$;
$|S_{11}|=4$ $|S_{12}|=6$ $|S_1|=24$; , $S_1=\{1000111110, 0100111110,\ldots, 0010011111, 0001011111\}$;
$|S_{21}|=6$, $|S_{22}|=15$, $|S_2|=90$; $S_2=\{1100111100, 1010111100,\ldots, 0101001111, 0011001111\}$;
$|S_{31}|=4$, $|S_{32}|=20$, $|S_3|=80$; , $S_3=\{1110111000, 1101111000,\ldots, 1011000111, 0111000111\}$;
$|S_{41}|=1$, $|S_{42}|=15$, $|S_4|=15$, $S_4=\{1111110000, 1111101000, \ldots, 1111000101, 1111000011\}$.

3. $S_0 \cup S_1 \cup S_2 \cup S_3 \cup S_4 = S$, where $|S|= C_{10}^6=210$.

6 A parallel algorithm for finding the rank of a matrix by means of research *r*-minors with descending orders

Our parallel algorithm can be described as follows:

1. Construction of initial Boolean vectors $l^k(m)$ *and* $l^k(n)$

The control processor p_0 generates Boolean vectors $l^k(m)$ and $l^k(n)$ with the help of the algorithm ***A10(m,n;*** $l^k(m)$, $l^k(n)$**)** . These vectors are different lengths, but each

of them contains exactly k "units" in the extreme left positions. In the beginning of the calculation is taken $k:= m$. After that the control processor p_0 sends these vectors $l^k(m)$ and $l^k(n)$ as the input data to computing processors p_1, p_2.

2. Generation of combinations without repetitions

A computing processor p_1 generates the set S_1^k, the elements of which are combinations (Boolean vectors with length m) to identify the lines of k-minors, with the help of the algorithm $A11(k, l^k(m) ; S_1^k)$. Similarly, a computing processor p_2 generates the set S_2^k, the elements of which are combinations (Boolean vectors with length n) to identify the columns of k-minors, with the help of the algorithm $A11'(k,l(n)^k ; S_2^k)$.
We can parallelize the process of generation of combinations without repetitions using the parallel algorithms ***APGC1*** or ***APGC2*** described above.

3. The formation of pairs of vectors to build k*-minors*

The control processor forms a pair of vectors to build a k-minor and sends one of the pairs together with the matrix A as a source of data to computing processors with the help of the algorithm $A13(S_1^k, S_2^k ;(b^k, c^k))$. The maximum number of the pairs of this vectors does not exceed the value $M = C_m^k * C_n^k$. We have high performance of our problem solution if the number of computing processors equals to value $R=M$. In this case the control processor sends only one of the pairs of this vectors for each computing processor. If we have $R < M$, our control processor sends $]M/R[$ of the pairs of this vectors for each computing processor.

4. Construction of k *- minors and calculation of the determinants*

Every computing processor p_i builds k-minor with the help of the algorithm $A14((b^k, c^k); M_k)$. Then it computes the determinant $D(M_k)$ for the submatrix M_k with the help of the algorithm $A15(M_k ; D(M_k))$.

5. The adoption of a decision on the continuation of computing

The processor p_0 analyzes the results with the help of the algorithm $A16(D(M_k) ; rk(A), b^k, c^k)$. If at least one of k-minors is not equal to zero, the rank of the matrix A is equal to $rk(A) = k$. The end of the calculation. The move to step 6. If every k-minor is equal to zero, the $rk(A) \leq k$. In this case, we must decide about the continuation of the calculations. Then $k:=k-1$ and the control processor forms new vectors $l^k(m)$ and $l^k(n)$.
The transition to paragraph 2.

6. Stop.

Our parallel algorithm ***APRM2*** for finding the rank of a matrix by means of research ***r***-minors with the descending orders implements the following computer schedule
S_{R+1} ***(APRM2)=((A10, p_0), (A11, p_1 ,p_2), (A13, p_0), (A14, $p_{1,\ldots,}$ p_R),(A15, $p_{1,\ldots,}$ p_R), (A16, p_0)),***where a record ***(Aj,p_i)*** indicates that the processor p_i implements the algorithm ***Aj.***

7 Conclusion

Our studies suggest the following conclusions.
You can use modern high-performance multiprocessor computer systems (clusters) to solve the important problem of finding of the rank of a rectangular matrix.
With this purpose we propose two parallel algorithm for finding the rank of an $m \times n$ matrix. One of them allows you to find the rank by paralleling process of generation of fringing minors. Another algorithm allows you to parallelize calculations in the process of finding the rank by research minors with descending orders. For the study of minors also proposed two parallel algorithms for generation of combinations without repetitions.
The use of multiprocessor computing systems and the proposed parallel algorithms will allow to increase considerably the speed of solving the task of finding the rank of a rectangular matrix, which needs to be addressed when checking of the compatibility of a system of linear algebraic equations.

References

1. Demmel J.W. Applied Numerical Linear Algebra, SIAM, 1997. - 431 pages.
2. Gauss C.F. Beiträge zur Theorie der algebraischen Gleichungen. – Gött., 1849.
3. V. Strassen, Gaussian elimination is not optimal, Num. Math., 13(4): 1969. 354–356.
4. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 31.3: Modular arithmetic, pp. 862–868.
5. Witold Lipski. Kombinatoryka dla programistów, WNT, 2004. -274 s.