Sergey NOVIKOV

Institute of Mathematics and Computer Science,
The John Paul II Catholic University of Lublin,
Konstantynów Street 1H, 20-708 Lublin, Poland

Parallelization of computations for generating combinations

Abstract. An effective sequential algorithm and two parallel algorithms for generating combinations without repetitions of m out n of objects, represented by Boolean vectors, are proposed. One of them allows one to calculate starting and ending combinations for the subset, generated by each computing processor. The second algorithm firstly generates short (m-component) vectors on several computing processors. After that, by using special [n/m]-component vectors, it connects the short vectors into n-component Boolean vectors, each of which containing of exactly m units.

Keywords: sequential algorithm, parallel algorithm, Boolean vector, combinations without repetitions, combinations with repetitions, parallelization of computations

1. Introduction

The combinatorial analysis (combinatorics) plays an important role in computer science due to its many applications when designing discrete computing devices and control systems [1]. One of the classic tasks of combinatorics is to generate all the combinations without repetitions of **n** elements taken **m** at a time ((**n**,**m**)- combinations). Matching algorithms have been developed, programmed and used in automated systems of designing digital devices and other applications [1,2]. Combinations in known algorithms are usually represented as a sequence of numbers in the lexicographical order, which is not suitable for some applications. In addition, the conventional representation of combinations makes it difficult to parallelize the computations.

For generating (**n**,**m**)-combinations with large **n** and **m**, it is more useful to represent them as **n**-component Boolean vectors, each of which containing of exactly **m** units (and **n-m** zeros, of course)..

At present, to enhance the performance of computations when solving problems of large dimensions (in engineering, mathematics and other disciplines) high-performance multiprocessor computing systems (supercomputers, clusters) are used. While paralleling solving algorithms of difficult design problems often one needs to parallelize the process generating combinations without repetitions.

Several parallel algorithms for generating combinations without repetitions have been proposed [2, 3, 4, 5, 6].

In particular, the algorithms proposed in [2,3] are focused on multiprocessor associative computing systems, which use SIMD (Single Instruction Multiple Data) architecture. In such systems, information processing comes from associative storage devices, where information is selected not at a certain address, but by its content. In modern supercomputers (see TOP 500) instead of SIMD architecture a MIMD (Multiple Instruction Multiple Data) architecture is used.

In addition, an adaptive algorithm [2] requires arbitrary-precision arithmetic, moreover it is necessary to schedule the combinations, that is, to decide when each combination will be computed, before the moment where each processor can independently generate its combinations subset [4].

Unlike the adaptive algorithm [2], that uses an arbitrary number of independent computing processors $\mathbf{NP \le n!/m!*(n-m)!}$, the parallel algorithm [5] requires a constant number of computing processors. Adaptive algorithms allow to use multiprocessor computing systems more effectively.

In the paper [4] an adaptive parallel algorithm was presented , where each combination is associated with a uniquely determined integer. The numeric representations of (**n**,**m**)-combinations makes difficult to split a task into subtasks for solving them independently on several computing processors ($\mathbf{NP \le n-m+1}$). That is why in the paper [4] real tests are described, where the number of processing processors does not exceed 11.

In the author's paper [6] an adaptive parallel algorithm for generating (**n**,**m**)-combinations presented by Boolean vectors was presented. Such a submission is more convenient for parallelization of computations, if compared with the representation of combinations by sequences of numbers in the lexicographical order. The algorithm [6] allows us to use $\mathbf{NP \le 2^k}$ working processors, where $\mathbf{k \le [m/2]}$ and $\mathbf{k < \log_2 r}$, **r** beind the number of planned to generate processors in our cluster. The drawback of the algorithm is that in the

corresponding multi-processor computing system in some nodes the calculations are duplicated by calculations, that are performed in other nodes.

2. Generating Boolean vectors, corresponding to (n,m)- combinations

We will represent an (**n**,**m**)-combination (**m**-combination of an **n**-set) as an **n**-component Boolean vector $\mathbf{A}= \mathbf{a_n \ldots a_2 a_1}$, where exatly **m** components are equal to 1. For example, we can imagine the **(8**,**4)**-combination **(1,3,4,8)**, selected from the set **{1,2,3,4,5,6,7,8}** by the **8**-component Boolean vector **10001101**, where components $\mathbf{a_1=1}$, $\mathbf{a_3=1}$, $\mathbf{a_4=1}$ and $\mathbf{a_8=1}$. It is obvious, that **n**-component Boolean vector can be regarded as word of length **n** in the alphabet **{0,1}**.

To generate (**n**,**m**)- combinations it is convenient to assume that the source Boolean vector $\mathbf{A = a_n \ldots a_2 a_1}$ contains $\mathbf{m \le [n/2]}$ unit components (1s), because a Boolean vector with a large number of 1s we can always replace by the inverted vector.

In our algorithm the initial combination $\mathbf{A_0}$ is written as an **n**-component Boolean vector $\mathbf{b_m(n)}$ with **m** units, located in positions **1**,**2**,...,**m**. The final combination is written as an **n**-component Boolean vector $\mathbf{A_f = c_m(n)}$ with **m** units, where the last (leftmost) **m** components are equal to 1.

The vector $\mathbf{A_0}$ (also as $\mathbf{A_f}$) contains one block with **m** units. The vector $\mathbf{A_i}$, were $\mathbf{i \neq 0}$ and $\mathbf{i \neq f}$, may contain few blocks with **k** units, were $\mathbf{1 \le k < m}$. Then one of these blocks will call the rightmost block **B1**. Except the rightmost block **B1**, this vector $\mathbf{A_i}$ contains the leftmost block units **B2**.

Sequential algorithm SAGC(n,m;S)

1. Write the primary combination $\mathbf{A_0 = b_m(n)}$ in the set **S**.

2. For given $\mathbf{i \ge 0}$ and $\mathbf{1 \le k \le m}$, find in $\mathbf{A_i}$ the rightmost block **B1** with **k** units. Units from **B1** located in positions **j**, **j+1**,…,**j+k-1**, but $\mathbf{a_{j+k}=0}$, where $\mathbf{k \ge 1}$, $\mathbf{j=1}$ or $\mathbf{j>1}$ but $\mathbf{a_1 = 0}$, $\mathbf{a_2 =0}$,…, $\mathbf{a_{j-1} =0}$.

3. Find in **B1** the leftmost 1 and move it to the left by one digit, and all other **k -1** units from **B1** move to the right maximum. Modernize the vector $\mathbf{A_i}$ as follows: $\mathbf{a_{j+k-1}=0}$, $\mathbf{a_{j+k} = 1}$, $\mathbf{a_1 =1}$, $\mathbf{a_2 =1}$,…, $\mathbf{a_{k-1} =1}$. After the modernization we obtain $\mathbf{A_{i+1}}$, which we write in **S.**

4. If $\mathbf{A_{i+1} = A_f = c_m(n)}$, move to p. 5. Otherwise, put $\mathbf{A_{i+1} := A_i}$ and move to p.2.

5. Print the set **S.** Stop.

For example, the (**6**,**3**)-combinations generated by **SAGC** are in **S={000111, 001011, 001101, 001110, 010011, 010101, 010110, 011001, 011010, 011100, 100011, 100101, 100110, 101001, 101010, 101100, 110001, 110010, 110100, 111000}**.

It is easy to see that our heuristic algorithm **SAGC** really generates all (**n**,**m**)-combinations.

To this end, we note that for each vector from **S** there is a corresponding number. The number corresponding to the vector $\mathbf{A_{i+1}}$ is higher, compared with the number corresponding to $\mathbf{A_i}$. Consider the vector $\mathbf{A_i}$, where $\mathbf{a_j=1}$, $\mathbf{a_{j+1}=1},\ldots,\mathbf{a_{j+m-1}=1}$ and $\mathbf{a_1=0}$, $\mathbf{a_2=0},\ldots,\mathbf{a_{j-1}=0}$, $\mathbf{a_{j+m}=0}$, $\mathbf{a_{j+m+1}=0},\ldots,\mathbf{a_n=0}$.

For this $\mathbf{A_i}$ the corresponding number is equal to

$\mathbf{N(A_i)} = 2^{j+m-1} + (2^j + 2^{j+1} + \ldots + 2^{j+m-2})$.

The numbers in parenthesis form a geometric progression with the ratio 2. The sum in parenthesis equals $2^{j+m-1} - 2^{j+1}$. Thus, the number that corresponds to the leftmost 1 of $\mathbf{A_i}$, is greater than the sum of numbers corresponding to the remaining **m-1** unit components. Naturally, this number is greater than the sum of the numbers corresponding to **m-1** units, which are located to the right. When the leftmost 1 moves by one digit to the left, the corresponding to this group number exceeds the sum of the numbers, corresponding to the rest of the **m-1** units, irrespective of their location. Thus, the number corresponding to each regular combination $\mathbf{A_{i+1}}$ is greater than the number corresponding to any of the previous combinations. In this way repeating combinations are excluded.

Next, we show that any possible (**n**,**m**) -combinations can be generated by using the algorithm **SAGC**. For this purpose it is enough to specify how to build the previous combination for each considered combination (excluding the primary combination).

We have the following options:

1) the vector $\mathbf{A_i}$ contains the block **B0** with **m** units located the rightmost (primary combination), the leftmost (final combination) or in the middle of $\mathbf{A_i}$;

2) the vector $\mathbf{A_i}$ contains the rightmost block **B1** with $\mathbf{k < m}$ units located the rightmost or in the middle of $\mathbf{A_i}$.

In the case 2), we have three subcases:

2a) **B1** with $\mathbf{k < m}$ units is located on the rightmost and between its the leftmost 1 and the rightmost 1 of next (on the left) block units **B2** is located only one zero, i.e. $\mathbf{a_1=1},\ldots,\mathbf{a_k=1}$, $\mathbf{a_{k+1}=0}$, $\mathbf{a_{k+2}=1}$;

2b) **B1** with $\mathbf{k < m}$ units is located on the rightmost and between its the leftmost 1 and the rightmost 1 of next block (on the left) **B2** with $\mathbf{t \geq 1}$ units are located $\mathbf{l > 1}$ zeros, i.e. $\mathbf{a_1 = 1}, \ldots, \mathbf{a_k = 1}, \mathbf{a_{k+1} = 0}, \ldots, \mathbf{a_{k+l} = 0}, \mathbf{a_{k+l+1} = 1}, \ldots, \mathbf{a_{k+l+t} = 1}$;

2c) **B1** with $\mathbf{k < m}$ units is located in the middle of $\mathbf{A_i}$.

In the case 1), $\mathbf{A_{i-1}}$ differs from $\mathbf{A_i}$ as follows: the rightmost 1 from **B0** is moved with one digit to the right.

In the case 2a), $\mathbf{A_{i-1}}$ differs from $\mathbf{A_i}$ as follows: the rightmost 1 from the block **B2** is moved by one digit to the right, i.e. in $\mathbf{A_i}$ we have $\mathbf{a_{k+1} = 0}$, $\mathbf{a_{k+2} = 1}$ but in $\mathbf{A_{i-1}}$ we have $\mathbf{a_{k+1} = 1}$, $\mathbf{a_{k+2} = 0}$.

In the case 2b), $\mathbf{A_{i-1}}$ differs from $\mathbf{A_i}$ as follows: the rightmost 1 from the block **B2** is moved by one digit to the right ($\mathbf{a_l = 1}$) and all 1s from **B1** are moved by **k** digits to the left, i.e. in $\mathbf{A_{i-1}}$ we have $\mathbf{a_{l-1} = 1}, \ldots, \mathbf{a_{l-k} = 1}, \mathbf{a_{l-k-1} = 0}, \ldots, \mathbf{a_1 = 0}$.

In the case 2c), $\mathbf{A_{i-1}}$ differs from $\mathbf{A_i}$ as follows: the rightmost 1 from the block **B1** is moved by one digit to the right.

The availability of ways to build for each combination of the previous combination one proves the fact that using the heuristic algorithm **SAGC** we can generate all possible (**n**,**m**)-combinations. Thus, $\mathbf{/S/ = C_n^m}$.

3. Parallelization the process of generating combinations

3.1. Generating Boolean vectors of length of a machine word

The rules of building the previous combination for each regular one help us to parallelize the process of generating the (**n**,**m**)-combinations on $\mathbf{q > 1}$ processes.

To this end, the control processor defines the starting $\mathbf{A_{j,0}}$ and ending $\mathbf{A_{j,k}}$ combinations for the subset $\mathbf{S_j}$, generated by each computing processor $\mathbf{p_j}$, where $\mathbf{1 \leq j \leq q}$.

For example, it is easy to show the method to parallelize the generation of (**n**,**m**)-combinations on $\mathbf{q = m}$ processes.

To parallel the computations with the help of our parallel algorithm **PAGC1 (n,m;S),** it is necessary to perform the following three steps:

1) The control processor $\mathbf{p_0}$, using the algorithm

A1(n,m ; $\mathbf{A_{1,0}, A_{2,0}, \ldots, A_{m,0}, A_{1,k}, A_{2,k}, \ldots, A_{m,k}}$ **)**, assigns the previous combinations $\mathbf{A_{j,0}}$:

$\mathbf{A_{1,0}}$ with **m** units, located on the rightmost, for the computing processor $\mathbf{p_1}$; $\mathbf{A_{2,0}}$ with **m** units, located on the positions **i**, **i-1**, …, **i-m+1**, where **i=[(n-m)/2]+m+1**, for the computing processor $\mathbf{p_2}$; …; $\mathbf{A_{j,0}}$ with **j-2** units, located in the positions **n**, **n-1**,…,**n-(j-3)**, and all other **m-(j-2)** units, located on the rightmost for the computing processor $\mathbf{p_j}$ with $\mathbf{3 \le j \le m}$.

Then $\mathbf{p_0}$ assigns ending combinations $\mathbf{A_{j,k}}$ as Boolean vectors:

$\mathbf{A_{1,k}}$ with **m-1** units, located in the positions **i**, **i-1**,…, **i-m+2**, with **i=[(n-m)/2]+m+1**, with one 0, located in the position **i-m+1**, and with one 1, located in the position **i-m**, for the computing processor $\mathbf{p_1}$; $\mathbf{A_{2,k}}$ with **m** units, located in the positions **n-1**, **n-2**,…, **n-m**, for the computing processor $\mathbf{p_2}$; …; $\mathbf{A_{j,k}}$ with **j-2** units, located in the positions **n, n-1,…,n-(j-3)**, where $\mathbf{3 \le j \le m-1}$, with one 0, located in the position **n-j+2**, and with **m-(j-2)** units, located in the positions **n-j+1**, **n-j**, …, **n-m-1**, for the computing processor $\mathbf{p_j}$, where $\mathbf{3 \le j \le m-1}$;

$\mathbf{A_{m,k}}$ with **m** units, located in leftmost positions, for the computing processor $\mathbf{p_m}$.

For example, for **n=10, m=4** we have $\mathbf{A_{1,0}}$ **= 0000001111**, $\mathbf{A_{1,k}}$ **=0011101000**, $\mathbf{A_{2,0}}$ **= 0011110000**, $\mathbf{A_{2,k}}$ **=0111100000**, $\mathbf{A_{3,0}}$ **= 1000000111**, $\mathbf{A_{3,k}}$ **=1011100000**, $\mathbf{A_{4,0}}$ **= 1100000011**, $\mathbf{A_{4,k}}$ **=1111000000**.

2) Then each computing processor $\mathbf{p_j}$, by using the algorithm $\mathbf{A2(A_{j,0}, A_{j,k}; S_j)}$, i.e. **SAGC**, generates (**n**,**m**)-combinations, writes it in the set $\mathbf{S_j}$, where $\mathbf{j \in \{1,\ldots,m\}}$, and sends the results to the control processor $\mathbf{p_0}$.

3) The control processor $\mathbf{p_0}$, by using the algorithm $\mathbf{A3(S_1, S_2,\ldots, S_m ; S)}$, sums the sets ($\mathbf{S_1 \cup S_2 \cup \ldots \cup S_m = S}$) and ends the generation of all the (**n**,**m**)-combinations.

It is easy to suggest a method to parallelize the process of generating (**n**,**m**)-combinations on $\mathbf{q > m}$ processes.

For maximum parallelization we propose the following method to build initial combinations of subsets for generating by computing processors.

Let us build the Boolean vectors $\mathbf{A_j}$ containing two blocks of units: the rightmost block **B1** with **j=m-k** units; the leftmost block **B2** with $\mathbf{k \le m-2}$ units, located in the digits **n**,**n-1**,…,**n-(k-1)**, where $\mathbf{k \in \{0,1,...,m-2\}}$.

First, we build **n-m** Boolean vectors for **B1** with **m** units:

$\mathbf{A_1}$, where the leftmost 1 of the rightmost block **B1** is located in the digit **m**;

$\mathbf{A_2}$, where the leftmost 1 of the rightmost block **B1** is located in the digit **m+1**;...;

$\mathbf{A_{n-m}}$, where the leftmost 1 of the rightmost block **B1** is located in the digit **n-1**.

Then, we build **n-m** Boolean vectors for **B1** with **m-1** units:

$\mathbf{A_{n-m+1}}$, where the leftmost 1 of the rightmost block **B1** is located in the digit **m-1**;

$\mathbf{A_{n-m+2}}$, where the leftmost 1 of the rightmost block **B1** is located in the digit **m**;...; $\mathbf{A_{2(n-m)}}$, where the leftmost 1 of the rightmost block **B1** is located in the digit **n-2**.

At last, we build **n-m** Boolean vectors for **B1** with **j= m-k=2** units, where the leftmost 1s of rightmost blocks **B1** are located in digits: **2**, **3**,…,**n-k-1**.

For each rightmost block **B1** with **j** units, where $\mathbf{j \in \{2,\ldots,m\}}$, we can build **n-m** such **n**-component Boolean vectors.

Thus, our method allows one to build **(n-m)*(m-1)** **n**-component Boolean vectors.

However, not every one of these vectors is appropriate as an initial combination of the subset $\mathbf{S_j}$ for generation by some computing processor $\mathbf{p_j}$, where $\mathbf{1 \le j \le (n-m)*(m-1)}$, according to our method.

As initial combinations are not suitable the following vectors:

1) The vector $\mathbf{A_j}$, containing **B1** with **m** units, located in the digits **n-1**, **n-2** ,…, **n-m**;

2) The vector $\mathbf{A_j}$, containing **B1** with **m-k** units, and the leftmost block **B2** with **k** units, located in the digits **n**, **n-1**,…, **n-k+1**, such that between it's the rightmost 1 and the leftmost 1 of **B1** there is only one zero.

Every vector with properties described above is not suitable as the initial combination of a subset, for generation by one of computing processors according to our method, since this subset has the cardinality equal to 1.

Therefore, the vectors with described properties we propose to use as final combinations $\mathbf{A_{j,k}}$ of $\mathbf{S_j}$. For $\mathbf{A_{j,k}}$ we build the initial combination $\mathbf{A_{j,0}}$ as follows. The leftmost block **B2** with **k** units in $\mathbf{A_{j,0}}$ coincides with **B2** from $\mathbf{A_{j,k}}$, one 1 is located in the digit **n-k-1**, but the other **m-k-1** units (from **B1** in $\mathbf{A_{j,0}}$) are located in the digits **a**, **a-1**, …, **a-(m-k-2)**, where $\mathbf{a = (n-k-1) -](n-m)/2[}$, $\mathbf{k \in \{0,1,\ldots,m-3\}}$.

Notice, that the vector $\mathbf{A_{j,0}}$ with two 1s in the rightmost block **B1** and **m-2** units in the leftmost block **B2**, located in the digits **n**, **n-1**,…, **n-k+1**, such that between it's the rightmost 1 and the leftmost 1 of **B1** is located only one zero, may be used as an initial combination of the last subset of combinations.

For each initial combination, built by our method, the control processor can easily assign the final combination.

For example, in the case **n=9**, **m=4**, we have: $\mathbf{A_{1,0}}$ = 000001111, $\mathbf{A_{1,k}}$ = 000011101, $\mathbf{A_{2,0}}$ = 000011110, $\mathbf{A_{2,k}}$ = 000111010, …, $\mathbf{A_{4,0}}$ = 001111000, $\mathbf{A_{4,k}}$ = 010011010, $\mathbf{A_{5,0}}$ = 010011100, $\mathbf{A_{5,k}}$ = 011110000,…, $\mathbf{A_{9,0}}$ =100111000, $\mathbf{A_{9,k}}$ = 101001010, $\mathbf{A_{10,0}}$ = 101001100, $\mathbf{A_{10,k}}$ = 101110000,…, $\mathbf{A_{15,0}}$ = 110110000, $\mathbf{A_{15,k}}$ = 111100000.

Thus, the algorithm **PAGC1** allows to parallelize the generation of (**n**,**m**)-combinations on **(n-m)*(m-1)** processes.

The ease summation of the results is one of the advantages of the described approach. The advantages of the proposed approach is also the possibility to parallelize the calculations on **(n-m)*(m-1)** processes.

The parallel algorithm **PAGC1** is effective for generating of (**n**,**m**)-combinations when **n** does not exceed the size of one machine word.

Modern processors general purpose computers usually use machine words with 32 or 64 bits.

Therefore, we propose a different approach for generation of (**n,m**)-combinations with large **n** and **m**.

3.2. Reduction of the problem of generating (n,m)-combinations to tasks of generating short vectors

The task generating the (**n**,**m**)-combinations represented by Boolean vectors can be reduced to the solution of several tasks generating short Boolean vectors. To this end, we can divide an **n**-component Boolean vector into **s = [n/m]** parts, each of the length **k = m**, except the rightmost. The length of the rightmost part of an **n**-component Boolean vector is equal to $\mathbf{k_s=n-k(s-1)}$. Each of short Boolean vectors contains $\mathbf{i_j \le m}$ units.

The sum of the 1s in all **s** parts must satisfy the condition

$$\mathbf{i_1 + i_2 + \ldots + i_s = m}. \qquad (*)$$

We can generate the short (**k**-component and $\mathbf{k_s}$-component) vectors in parallel on several (not more than **2m**) processors working independently. The number of 1s in each of these vectors varies from **0** to **m**.

If $\mathbf{k_s = k=m}$, it's enough to generate only $\sum_{i=0}^{m} C_k^i$ short vectors. These vectors we can place in sets $\mathbf{S_0(k)}$, $\mathbf{S_1(k)}$,…, $\mathbf{S_m(k)}$ **,** where the set $\mathbf{S_i(k)}$ contains vectors, each of which contains exactly **i** 1s. It is obvious, that $\mathbf{|S_i(k)| = C_k^i}$. The set $\mathbf{S_0(k)}$ consists of one vector, whose all **k** components are equal to 0. Similarly, the set $\mathbf{S_m(k)}$ consists of one vector, whose all **k** components are equal to 1.

Each set $\mathbf{S_i(k)}$ of short vectors (excepting $\mathbf{S_0(k)}$ and $\mathbf{S_m(k)}$) can be generated by using the sequential algorithm **SAGC** for generating (**k,i**)-combinations on one of **m-1** processors $\mathbf{p_1, p_2, \ldots, p_{m-1}}$.

If $\mathbf{k_s \neq k=m}$, still **m** processing processors are required for generating **m** sets of $\mathbf{k_s}$-component vectors $\mathbf{S_i(k_s)}$, where $|\mathbf{S_i(k_s)}| = \mathbf{C_{ks}^i}$ and $\mathbf{i \in \{0,1,\ldots,m\}}$. Each vector from $\mathbf{S_i(k_s)}$ contains exactly **i** units. In the case $\mathbf{k_s \neq k=m}$, it is required to generate $\sum_{i=0}^{m} C_k^i$ **m**-component vectors and $\sum_{i=0}^{m} C_{ks}^i$ short vectors with $\mathbf{k_s}$ components, where $\mathbf{i \in \{0,1,\ldots,m\}}$.

After completing the generation of short Boolean vectors, the latest have to be merged into **n**-component Boolean vectors, each of which containing exactly **m** units.

The easiest way to combine short vectors in an **n**-component we have when $\mathbf{s = 2}$.

In this case, for even **n** we have $\mathbf{k=k_s=m=n/2}$, and for odd **n** we get $\mathbf{k=[n/2]}$, $\mathbf{k_s=n-k}$.

To merged the short vectors into **n**-component Boolean vectors we perform the operation of the Cartesian product of sets $\mathbf{S_i(k)}$ and $\mathbf{S_j(k_s)}$, such that $\mathbf{i+j=m}$.

The number of (**n**,**m**)-combinations, that are generated in this way, is computed by the formula

$$C_n^m = \sum_{i=0}^{m} C_k^{m-i} * C_{ks}^i .$$

Thus you can generate (**n**,**m**)-combinations when $\mathbf{k_s}$ does not exceed the size of one machine word and $\mathbf{m \leq [n/2]}$.

To generate (**n,m**)-combinations with large values **n** and **m** we can divide corresponding **n**-component Boolean vectors into $\mathbf{s \geq 3}$ parts.

In the case $\mathbf{s > 2}$, to combine short vectors in an **n**-component vector, a special connecting **s**-component vector $\mathbf{t=t_s t_{s-1} \ldots t_1}$ is required, where the component $\mathbf{t_j}$ is equal to $\mathbf{i_j}$, $\mathbf{i_j \in \{0,1,2,\ldots,m\}}$, $\mathbf{j \in \{1,2,\ldots,s\}}$, $\mathbf{i_1 + i_2 + \ldots + i_s = m}$.

Easily seen that the problem of generation the connecting **s**-component vectors, satisfying the condition (*), can be reduced to the known problem generating of the options distribution of **m** objects (“balls”) into **s** categories (“urns”), for which we can use the well-known algorithms generating combinations with repetitions [1].

So, in the case $\mathbf{s = 3}$, $\mathbf{m = 4}$, all combinations with repetitions are enumerated in the following set **{1111, 1112, 1122, 1222, 2222, 1113, 1123, 1223, 2223, 1133, 1233, 2233, 1333, 2333, 3333}** and the corresponding connecting **3**-component vectors, satisfying the condition $\mathbf{i_1 + i_2 + i_3 = 4}$, are enumerated in the following set **{400,310,220,130,040, 301,211,121,031, 202,112,022, 103,013,004}**.

To generate the set **S** of all connecting **s**-component vectors, satisfying the condition (*), it is convenient to use the algorithm proposed below.

Sequential algorithm SAGCV(s,m;S)

1) The primary vector $\mathbf{V^0}$, where $\mathbf{i_1 = m}$, $\mathbf{i_2 = 0}$, …, $\mathbf{i_s = 0}$, write in the set **S**.
2) To get the next vector $\mathbf{V'=i'_s i'_{s-1}\ldots i'_1}$, find in the previous vector $\mathbf{V=i_s i_{s-1}\ldots i_1}$ the leftmost component $\mathbf{t_j}$ with $\mathbf{i_j \neq 0}$, where $\mathbf{j < s}$, and modify **V**. We have $\mathbf{i'_t = i_t}$ for $\mathbf{t \in \{1,\ldots,s-1\}}$, $\mathbf{t \neq j}$, $\mathbf{t \neq j+1}$ and $\mathbf{i'_j = i_j - 1}$, $\mathbf{i'_{j+1} = 1}$. Then write **V'** in the set **S**. If $\mathbf{j+1 = s}$, transition to p. 3. Otherwise, move to the p. 2.
3) To get the next vector **V'**, find in the previous vector **V** the leftmost $\mathbf{t_j}$ with $\mathbf{i_j \neq 0}$, where $\mathbf{j < s}$. If $\mathbf{j=s-1}$, transition to p. 4. Otherwise, modify **V**: $\mathbf{i'_t = i_t}$ for $\mathbf{t \in \{1,\ldots,s-1\}}$, $\mathbf{t \neq j}$, $\mathbf{t \neq j+1}$ and $\mathbf{i'_j = i_j - 1}$, $\mathbf{i'_{j+1} = i_s + 1}$, $\mathbf{i'_s = 0}$. Write **V'** in the set **S** and move to p. 2.
4) To get the next vector **V'**, modify **V**: $\mathbf{i'_t = i_t}$ for $\mathbf{t \in \{1,\ldots,s-2\}}$, $\mathbf{t \neq s}$, $\mathbf{t \neq s-1}$ and $\mathbf{i'_{s-1} := i_{s-1} - 1}$, $\mathbf{i'_s := i_s + 1}$. Then write **V'** in the set **S.** If $\mathbf{i'_s = m}$, move to p. 5. If $\mathbf{i'_{s-1} = 0}$, transition to p. 3. Otherwise, move to the p. 4.
5) Print the set **S.** Stop.

For example, all connecting **4**-component vectors, satisfying the condition $\mathbf{i_1 + i_2 + i_3 + i_4 = 4}$, generated by **SAGCV** are in **S= {0004, 0013, 0103, 1003, 0022, 0112, 1012, 0202, 1102, 2002, 0031, 0121, 1021, 0211, 1111, 2011, 0301, 1201, 2101, 3001, 0040, 0130, 1030, 0220, 1120, 2020, 0310, 1210, 2110, 3010, 0400, 1300, 2200, 3100, 4000}**.

The algorithm **SAGCV** really generates all connecting **s**-component vectors, satisfying the condition (*).

It is clear that repeating combinations are excluded.

Next, we show that any of possible connecting **s**-component vectors (satisfying the condition (*)) can be generated by using the algorithm **SAGCV**. For this purpose, it is enough to specify how to build the previous vector **V** for each vector **V'** (excluding the primary vector).

There are the following options:

1) the value of the leftmost component $\mathbf{t_j}$ with $\mathbf{i'_j \neq 0}$ in **V'** is equal to **1**;

2) the value of the leftmost $\mathbf{t_j}$ component with $\mathbf{i'_j \neq 0}$ in **V'** is greater then **1**.

In the case 1), one can build **V** (for **V'**) as follows: $\mathbf{i_t = i'_t}$ for $\mathbf{t \in \{1,\ldots,s\}}$, $\mathbf{t \neq j}$, $\mathbf{t \neq j-1}$ and $\mathbf{i_{j-1} = i'_{j-1}+1}$, $\mathbf{i_j = 0}$.

In the case 2), for $\mathbf{j=s}$ one can build the vector **V** (for **V'**) as follows: put $\mathbf{i_t = i'_t}$ for $\mathbf{t \in \{1,\ldots,s-2\}}$ and $\mathbf{i_{s-1} = i'_{s-1}+1}$, $\mathbf{i_s = i'_s - 1}$.

For $\mathbf{j < s}$ one can build the vector **V** (for **V'**) as follows: put $\mathbf{i_t = i'_t}$ for $\mathbf{t \in \{1,\ldots,s-1\}}$, $\mathbf{t \neq j}$, $\mathbf{t \neq j-1}$, $\mathbf{t \neq s}$ and $\mathbf{i_{j-1} = i'_{j-1}+1}$, $\mathbf{i_j = 0}$, $\mathbf{i_s = i'_j - 1}$.

Thus, the algorithm **SAGCV** allows to generate any connecting **s**-component vector, satisfying the condition (*).

Denote by H_s^m the number of all connecting **s**-component vectors, satisfying the condition (*).

It is obvious, that

$$H_s^m = \sum_{j=0}^{m} H_{s-1}^{m-j} .$$

We can parallelize the process generation the connecting **s**-component vectors, satisfying the condition (*), on **m+1** processes. To this end, each working processor generates a subset of all **(s-1)**-component vectors $(\mathbf{t_s\, t_{s-1} \ldots\, t_2})$, satisfying the condition $\mathbf{i_2 + \ldots + i_s = m-j}$, where $\mathbf{j \in \{0,1,\ldots,m\}}$, and appends for each vector the additional component $\mathbf{t_1}$ equal to **j**. Then the control processor $\mathbf{p_0}$ sums received subsets and gets the set of all connecting **s**-component vectors.

Let $\mathbf{T(n,m,s,k,k_s)}$ denote the set of all connecting **s**-component vectors with parameters **n**, **m**, **s**, **k**, $\mathbf{k_s}$.

It is obvious, that

$$\mathbf{|\, T(n,m,s,k,\, k_s\,)| = H_s^m = C_{s+m-1}^m}.$$

To put together the short vectors from the sets $\mathbf{S_0(k)}$, $\mathbf{S_1(k),\ldots,\, S_m(k)}$, $\mathbf{S_0(k_s)}$, $\mathbf{S_1(k_s)}$,…, $\mathbf{S_m(k_s)}$ in a single **n**-component Boolean vector, one has to perform the operation of the Cartesian product of sets, the numbers of which are indicated in the corresponding connecting **s**-component vector $\mathbf{t = t_s t_{s-1} \ldots t_1}$.

If, for example, $\mathbf{S_1(4)}$ **={0001, 0010, 0100,1000}**, $\mathbf{S_2(4)}$**={0011,0101,0110, 1001, 1010, 1100}**, $\mathbf{S_1(5)}$ **={00001, 00010, 00100, 01000, 10000}** and **T(13,4,3,4,5)={400, 310, 220, 130,**

040, 301, 211, 121 ,031, 202, 112, 022, 103, 013, 004}, then the connecting **3**-component vector **211∈ T(13,4,3,4,5)** allows one to connect three short vectors in 120 **13**-component vectors:

0011 0001 00001,**0011 0010 00001**,**0011 0100 00001**,**01010001 00001**, … , **1100 1000 00001**, **1100 1000 00010**, **1100 1000 00100**, **1100 1000 01000**, **1100 1000 10000**.

With the help of all 15 connecting **3**-component vectors from **T(13,4,3,4,5)**, we can generate all 715 (**13**,**4**)-combinations, presented by **13**-component vectors, each of which contains exactly 4 units.

Thus, with the help of the all corresponding connecting **s**-component vectors from **T(n,m,s,k,k$_s$)**, one can generate all the **n**-component Boolean vectors, corresponding to all combinations without repetitions of **m** out **n** of objects.

3.3. Parallel algorithm PAGC2(n,m;S)

To parallel the computations with the help of our parallel algorithm **PAGC2(n,m;S)**, it is necessary to perform the following nine steps.

1. Preparing input data for generating of short Boolean vectors

The control processor **p$_0$**, using the algorithm **A1(m, n; s, k, k$_s$)**, prepares the input data for generating short Boolean vectors and sends its to the processing processors **p$_1$**,…,**p$_{2m-1}$**.

When assigning the value of **k**, we have to take into account the limitation: the sequential algorithm **SAGC** must be able to generate all **k$_s$**-component Boolean vectors, each of which containing **i$_j$ ≤ m** units. It is more conveniently, to determine the value of **k** to be equal to **m**, but the algorithm can be applied also in the case of **m < k**.

2. Generation short Boolean vectors

For **j ∈ {1,…,2m-1}** each processing processor **p$_j$**, by using the algorithm

SAGC(j,k; S$_j$(k)), generates (**k**, **j**)-combinations or (**k$_s$**, **j**)-combinations and sends to the control processor **p$_0$** the set **S$_j$(k)** or **S$_j$(k$_s$)** with short Boolean vectors. Obviously, may not generate short vectors from sets **S$_0$(m)**, **S$_m$(m)**, **S$_0$(k$_s$)**. However, in the case **k$_s$ > k=m** we must generate short vectors from **S$_m$(k$_s$)**.

3. Summarizing sets of short Boolean vectors

The control processor **p$_0$**, by using the algorithm **A3(m, n, s, k, k$_s$; S, j)**, sums received sets of short vectors (**S$_0$(k) ∪ S$_1$(k) ∪…∪ S$_m$(k) ∪ S$_0$(k$_s$) ∪ S$_1$(k$_s$) ∪ …∪ S$_m$(k$_s$) = S**), prepares

the input data and sends to processing processors the parameters, required to generate connecting **s**-component vectors: the set **S** of all short vectors and value of the component $\mathbf{t_1}$, i.e. $\mathbf{i_1 = j}$, where $\mathbf{j \in \{0,1,2,\ldots, m\}}$.

4. Generation connecting vectors

Each $\mathbf{p_i}$, where $\mathbf{i \in \{1,\ldots,m+1\}}$, using the algorithm $\mathbf{A4(m, n, s, k, k_s, j; T_j (n, m, s, k, k_s))}$, for the given parameter $\mathbf{i_1 = j}$ generates (with the help of the algorithm **SAGCV**) connecting **(s-1)**-component vectors, satisfying the condition $\mathbf{i_2 + \ldots + i_s = m-j}$, and appends for each vector the additional component equal to **j** . Then it sends the set of connecting **s**-component vectors $\mathbf{T_j(n,m,s,k,k_s)}$ to $\mathbf{p_0}$.

5. Summarizing sets of connecting vectors

The control processor $\mathbf{p_0}$ with the help of the algorithm $\mathbf{A5(T_1(n,m,s,k,k_s),..., T_{m+1}(n,m,s, k, k_s); T (n,m,s, k, k_s))}$ sums received subsets and gets the set $\mathbf{T (n,m,s,k,k_s)}$ with all connecting **s-**component vectors.

6. Preparing input data for connection of short Boolean vectors

The control processor $\mathbf{p_0}$, by using the algorithm

$\mathbf{A6(T(n,m,s,k,k_s),S; (t_s t_{s-1} \ldots t_2 t_1)_i, S_{i1},\ldots, S_{is})}$, prepares input data, required to connect short vectors, i.e. a connecting **s**-component vector together with corresponding sets of short vectors.

The maximum number of processing processors, required for parallel connection of short vectors, equals to $\mathbf{r = |T (n,m,s,k,k_s)|}$. Thus, the algorithm **PAGC2** allows one to use

$\mathbf{NP \le (s+m-1)*(s+m-2)*\ldots*(m+1) / (s-1)!}$ computing processors, where $\mathbf{s=[n/m]}$.

However, to reduce the cost of the solution for large **n** and **m**, the control processor $\mathbf{p_0}$ may not prepare the data for all **r** computing processors, but only for one group ($\mathbf{g_l}$) from $\mathbf{q \le r}$ processors, where $\mathbf{l \in \{1,\ldots,L\}}$, $\mathbf{L=]r/q[}$.

So the first step on p.6 is to check the condition $\mathbf{l \le L}$. If the condition is met, then $\mathbf{p_0}$ prepares the data for the next group, and moves to p.7. Otherwise, $\mathbf{p_0}$ moves to p.9.

7. Connection of short Boolean vectors

By using the algorithm **A7**$((t_s t_{s-1} \ldots t_2 t_1)_i, S_{i1}, \ldots, S_{is}; C_j)$, each processing processor p_j, where $j \in \{1,\ldots,q\}$, $q \le r = \left| T(n,m,s,k,k_s) \right|$, performs the operation of the Cartesian product of sets with the indices from the corresponding connecting **s**-component vector $(t_s t_{s-1} \ldots t_1)_i$ and sends the subset C_j of **n**-component Boolean vectors to the control processor.

8. Summarizing sets of (n,m)-combinations

The control processor p_0, using the algorithm **A8**$(C_1,\ldots, C_q; C^l)$, where $l \in \{1,\ldots,L\}$, sums the solutions $(C_1 \cup \ldots \cup C_q = C^l)$ and moves to p. 6.

9. Ending generation of (n,m)-combinations sets

The control processor p_0, using the algorithm **A9**$(C^1,\ldots,C^L;C)$, sums the solutions $(C^1 \cup \ldots \cup C^L = C)$ and ends the generation (**n**,**m**)-combinations without repetitions.

Interaction of sequential algorithms in the composition of the proposed parallel algorithm **PAGC2** for generating (**n**,**m**)-combinations implements the computer system, operating according to the following schedule

H(PAGC2) $= ($ **(A1,** p_0 **),** $(SAGC, p_1, \ldots, p_{2m-1})$, **(A3,** p_0 **),** $(A4, p_1, \ldots, p_{m+1})$, **(A5,** p_0 **),** $\downarrow^1$ **(A6,** p_0 **),** $(A7, p_1, \ldots, p_q)$, **(A8,** p_0 **)** $\uparrow^1$ **, (A9,** p_0 **)),**

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

The parallel algorithm **PAGC2** contains the following 9 serial algorithms: **A1** –for preparation of the data for generating short Boolean vectors; **SAGC -** for enumerating short Boolean vectors; **A3** –for summation of short Boolean vectors sets and for preparation of the data for generating of connecting **s**-component vectors; **A4** –for generating of connecting **s**-component vectors; **A5**–for summation subsets of connecting **s**-component vectors; **A6** –for preparation the data for connection of short vectors using a special connecting **s**-component vector; **A7** –to perform the product of short vectors sets using a special connecting **s**-component vector; **A8** –for summation of subsets of **n**–component vectors; **A9** –for ending the generation of (**n**,**m**)-combinations without repetitions.

4. Conclusion

This article examines how to parallelize the process of generating the combinations without repetitions, represented by Boolean vectors. Such a submission is more convenient for parallelization of computations, compared with representation of combinations by sequences of numbers ordered lexicographically.

For generation the Boolean vectors corresponding to (**n**,**m**)-combinations the effective sequential algorithm **SAGC** is proposed. It is proved that the heuristic algorithm **SAGC** allows one to generate all possible (**n**,**m**)-combinations.

Based on **SAGC**, two parallel algorithms are proposed.

The suggested adaptive parallel algorithms **PAGC1** and **PAGC2** allows one to solve tasks of high dimensionality and to use for this purpose a multiprocessor computing system with an arbitrary number of independent working processors.

The advantage of our adaptive parallel algorithm **PAGC1** is the ease decomposition of large task into subtasks for solving them independently on several computing processors, as well as the very simple summation of generated combinations subset. **PAGC1** uses an arbitrary number of independent computing processors $\mathbf{NP \le (n-m)*(m-1),}$ that is significantly higher than the similar characteristic of the algorithm [4].

While the parallel algorithm **PAGC1** is effective for generating (**n,m**)-combinations when **n** does not exceed the size of one machine word, our adaptive parallel algorithm **PAGC2** allows one to generate (**n**,**m**)-combinations when **n** is equal to the size of several machine words.

The algorithm **PAGC2** firstly allows one to generate short (**m-**component) vectors on several computing processors. For connecting the short vectors into **n**-component Boolean vectors in **PAGC2** is need the special algorithm **SAGCV** to generate connecting **[n/m]** -component vectors.

However, this algorithm allows one to parallelize the generation of (**n,m**)-combinations on a larger number of processes. **PAGC2** uses an arbitrary number of independent computing processors $\mathbf{NP \le (s+m-1)*(s+m-2)*\ldots*(m+1) / (s-1)!}$, where $\mathbf{s = [n/m]}$.

Thus, we can conclude that the algorithms **PAGC1** and **PAGC2** are competitive.

References

1. Lipski W.: Kombinatoryka dla programistów, Warszawa, Wyd. Naukowo-Techniczne, WNT, 274 pages, 2004.

2. Akl S.G.: Adaptive and optimal parallel algorithms for enumerating permutations and combinations, The Computer Jouma1, 30. pp. 433 – 436, 1987.

3. Kokosiński Z.: On parallel generation of combinations in associative processor architectures, Tech. Rep. 96-1-007, University of Aizu, Aizu-Wakamatsu, Japan, 14 pages, 1996.

4. Torres M., Goldman A., Barrera J.: A parallel algorithm for enumerating combinations. Proceeding of the 2003 International Conference on parallel processing. https://pdfs.semanticscholar.org/d153/cf9b621cb3cbfdf943f5bde1d84c6b01a930.pdf

5. Zhou B.B., Brent R., Qu X., Liang W.F.: A novel parallel algorithm for enumerating combinations. In International Conference on Parallel Processing, Volume 2, pp. 70 – 73, 1996.

6. Novikov S.: Parallelization of computations for finding the rank of a rectangular matrix. Studia Informatica, Systems and information technology, Volume 1-2(17), Wyd. UPH, Siedlce, pp. 49 - 62, 2013.