IMPROVED COMPUTING PERFORMANCE FOR LISTING COMBINATORIAL ALGORITHMS USING MULTI-PROCESSING MPI AND THREAD LIBRARY

Nguyen Dinh Lau

University of Education and Science, University of Danang, Vietnam

ABSTRACT

*This study builds up two parallel algorithms to improve computing performance for two listing binary and listing permutation algorithms. The problems are extremely interesting and practically applicable in many fields in our daily life. To parallel execution, we divide the data set input and allocate them to the processors. The article focuses on (i) the analysis of the research situation of the related works to compare and evaluate the existing problems of previous works, (ii) the analysis of the input data structure to divide data for the sub processors, (iii) the construction of parallel algorithms - proof of correctness and analysis of computing complexity, and (iv) experiments in multi-processing **MPI** and **Thread** library. Then the comparison of the results of the parallel algorithm with the sequential algorithm and the comparison of the execution time on different sub processors is discussed.*

KEYWORD

Parallel algorithms, listing binary, listing permutation, bounded sequences, substituend, inversion

1. INTRODUCTION

Listing binary and permutation are amazing and appealing problems in discrete mathematics with numerous wide applicability. However, when the input data is large, the listing time is highly long. For example, with input n = 20, the number of binary array is 2^{20}. Therefore, It is crucial to build up parallel algorithms to improve the computing performance for this problem.

In Vietnam, Hoang Chi Thanh has done some Research on combinatorial [3], [4], [5], [6] , [7].

In the world, there are many researchers publishing works related to the field of combinatorial [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18]].

In the article [1] by Nguyen Dinh Lau, a parallel algorithm for listing permutation has been developed, but not yet applied to multi-processing MPI and Thread library. Thus, this paper is inspired by some parts of [1] to rebuild the listing permutation algorithm.

However, in [8], [9], [10], [11], the listing binary sequences algorithm is not improved to cut down on the computing performance. Particularly [3] study by Hoang Chi Thanh focuses on building algorithm based on inversion vector and bounded sequence. However, Hoang Chi Thanh has neither analyzed and proved the complexity of the parallel algorithm, nor experimented in multi-processing MPI and Thread library to compare the processing time between different processors and different data sets.

Therefore, this article has the following new cutting-edge points:

1. Building up a new parallel listing n-binary algorithm to improve computing performance. It deals with the analysis, and proof of the complexity and experiments in the MPI to the examine and compare computing time.

2. Basing on [1] to build up parallel listing permutation algorithm. It involves the analysis, proof the complexity and experiments in in the Thread to analyze and compare computing time.

2. LISTING BINARY SEQUENCE ALGORITHM

2.1. Sequential algorithm

Let n□N. List all binary sequences with *n* length, i.e., sequence $[b_1,..., b_n]$, where b_ib_i□ {0, 1} □ i=1, ..., n.

The number of binary sequences is 2^n and the first sequence s = [0, 0, ..., 0]. For example, given n = 3, we have the 8 following binary sequences: 000, 001, 010, 011, 100, 101, 110, 111

Algorithm 1. Creating a sequential binary sequence with *n* length

```
        Begin
1.      Input n,  s[i]:= 0i = 1,2,..., n
2.      Repeat
3.          Print sequence s[1…n].
4.          i:=n;
5.          While s[i]<>0 then
6.              Begin
7.                  S[i]:=0;
8.                  i:=i-1;
9.              End
10.         If i>=1 then s[i]:=1
11.     Until i=0
12.     End.
```

Assume s[i] in line 2 has the complexity O(n). Lines from 3 to 12 represents 2^n binary sequences. So the complexity of the algorithm is $O(2^n)$.

2.2. PARALLEL ALGORITHMS

Sequential algorithms might take a long time to process if *n* length is large. Therefore, it is necessary to build parallel algorithms to improve computing performance for the algorithms.

This newly-built parallel algorithms use *k* processors $(R_0, R_1,…,R_{k-1})$ with $k = 2^{n'}+ 1$, where n' = 0,1, ..., n-1. The processor R_i receives the output value which is the input value of R_{i+1} (i = 1,2, ..., k-2). Note that the R_0 main processor neither participates in the computation process nor lists binary sequences. R_0 only sends and receives information.

The input on the processors is illustrated as follows:

Let n be the input value, list all binary sequences, let n', then we have the number of *k* processors. Then run the sequential algorithm (Algorithm 1) to list the binary sequence of n' length. After adding the bits 0 on the right, the binary sequence of n' length has a sufficiently long binary sequence of *n* length will be divided by the processor R_0 for the additional processors (R_1, R_2 ... R_{k-1}) as the input value.

For example, given n = 4, n'= 2. Then, the number of processors k = 5 (R_0, R_1, ..., R_4)

The binary sequence n = 4 is: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111

The binary sequence n '= 2 is: 00, 01, 10, 11. Then, the sequences 0000, 0100, 1000, 1100 are the input of four su processors (R_1, R_2, R_3, R_4).

The finishing condition of the four processors (R_1, R_2, R_3, R_4) is 0100, 1000, 1100, 1111.

R_1 listing binary: 0000, 0001, 0010, 0011
R2 listing binary: 0100, 0101, 0110, 0111
R3: listing binary 1000, 1001, 1010, 1011
R4: listing binary 1100, 1101, 1110, 1111
The following is the parallel algorithm

Algorithm 2. Creating a parallel binary sequence

```
1.  Begin
2.  Input n, n’
3.  k :=$2^{n'}$+1
4.  If Rank=0 then // main processor $R_0$
5.     Begin
6.        Call Algorithm 1 (n’) // listing binary sequence $(t_1, t_2, \ldots t_{n'})_i \forall i \in 1,\ldots,2^{n'}$ of length n’
7.        Create $2^{n'}$ $(t_1, t_2, \ldots t_{n'}, t_{n'+1},\ldots,t_n)_i := (t_1, t_2, \ldots t_{n'})_i \cup (0,\ldots,0)_i \forall i \in \underbrace{1,\ldots,2^{n'}}$ of length n
             n-n’element
8.        Send $(t_1, t_2, \ldots t_{n'}, t_{n'+1},\ldots,t_n)_i \forall i \in 1,\ldots,2^{n'}$ to $2^{n'}$ ($P_1$, …., $P_{k-1}$) sub processors
9.     End
10. For i:=1 to k-1 do
11.       Begin
12.          Listing binary sequences in the corresponding processor segment
13.          Send the result to $R_0$
14. End;
15. $R_0$ print results
16. End.
```

Let $2^n = 2^{n'} + 2^{n-n'}$ where $2^{n'}$= k-1 (R_1, R_2, ..., R_{k-1}), then if R_1 initiates a binary sequence with a value 0 and n' length is 0... 000, R_2 initiates binary sequence with a value 1 and n' length is 0 ... 001, R_3 initiates binary sequence with n' length is 0 ... 010, R_4 with n' length' is 0 ... 011, Continue to R_{k-1}. Each processor R_1 to R_{k-1} connects the $2^{n-n'}$ binary sequence has *n-n'* length to left of the sequence. Send the results to processor R_0. R_0 prints results and ends.

Example 2: Given n= 4, n'= 2, then k = 5, then R_1 holds 00, R_2 holds 01, R_3 holds 10, R_4 holds 11.

R_0 lists $2^{n-n'}$= 24-2 = 4 binary sequences with the length n-n' = 4-2 = 2 : 00, 01, 10, 11. Then broadcast 00, 01, 10, 11 to sub processors. The processors R_1, R_2, R_3, R_4 receive data from the Broadcast command, Then connect the sequence 00 to the left of the sequences in R_1, then R_1 shows: 0000, 0001, 0010, 0011. R_2 represents: 0100, 0101, 0110, 0111. R_3 shows: 1000, 1001, 1010, 1011. R_4 represents: 1100, 1101, 1110, 1111

Algorithm 2 is rewritten as algorithm 3 as follows:

Algorithm 3. Creating a parallel binary sequence by data Broadcast

```
1.  Begin
2.  Input n, n'
3.  k :=$2^{n'}$+1
4.  If Rank=0 then //Main processor $R_0$
5.  Begin
6.      Call Algorithm 1 (n-n') // listing binary sequence $(t_{n'+1}, t_{n'+2}, \ldots t_n)_i \forall i \in 1, \ldots, 2^{n-n'}$ has length is n-n'
7.  Broadcast $(t_{n'+1}, t_{n'+2}, \ldots t_n)_i \forall i \in 1, \ldots, 2^{n-n'}$ to $2^{n'}$ sub processors ($P_1$, …., $P_{k-1}$)
8.  End
9.  For i:=1 to k-1 do
10.     Begin
11.         $R_i$ create binary sequence has of length n' with value i-1 is $(t_1, t_2, \ldots t_{n'})_i$
12.         $R_i$ connect $(t_1, t_2, \ldots t_{n'})_i$ into the left sequence $(t_{n'+1}, t_{n'+2}, \ldots t_n)_j \forall j \in 1, \ldots, 2^{n-n'}$
13.         Send the results to $R_0$
14.       End;
15. $R_0$print results
16. End.
```

2.3. Experimental Results

The world of parallel multiple instruction, multiple data, or MIMD, computers is, for the most part, divided into distributed-memory and shared-memory systems. From a programmer's point of view, a distributed-memory system consists of a collection of core-memory pairs connected by a network, and the memory associated with a core is directly accessible only to that core. See Figure 1 [21], [22], [23], [24].

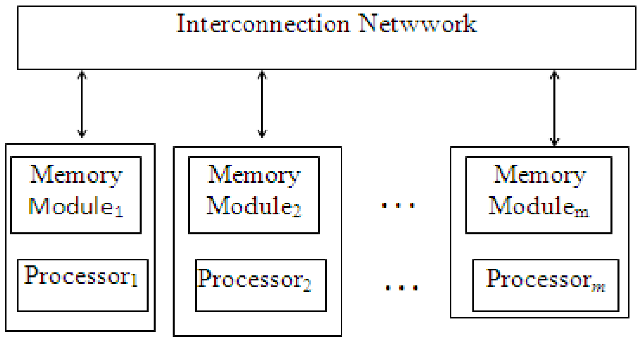
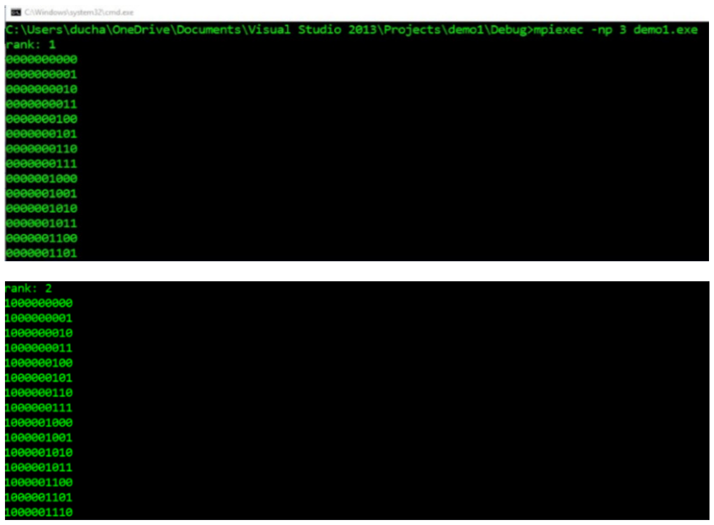


Figure 1. Model of adistributed-memory system

In message-passing programs, a program running on one core-memory pair is usually called a process, and two processes can communicate by calling functions: one process calls a send function and the other calls a receive function. The implementation of message-passing that we'll be using is called MPI, which is an abbreviation of Message-Passing Interface. MPI is not a new programming language. It defines a library of functions that can be called from C, C, and Fortran programs. We'll learn about some of MPI's different send and receive functions.

I used MPI to parallelize the computation and got exact results. Moreover, the execution time by parallel algorithms is much shorter than one by sequential algorithm. If n = 10, n'= 1, then the number of processors k = $2^{n'}$+ 1 = 3. Only Rank 1 and rank 2 do calculations and send results to Rank 0.



```
C:\Users\ducha\OneDrive\Documents\Visual Studio 2013\Projects\demo1\Debug>mpiexec -np 3 demo1.exe
rank: 1
0000000000
0000000001
0000000010
0000000011
0000000100
0000000101
0000000110
0000000111
0000001000
0000001001
0000001010
0000001011
0000001100
0000001101
```

```
rank: 2
1000000000
1000000001
1000000010
1000000011
1000000100
1000000101
1000000110
1000000111
1000001000
1000001001
1000001010
1000001011
1000001100
1000001101
1000001110
```

Figure 2. Demo result

Table 1. The execution time (ms) with n = 12 on the sequential (Seq) and parallel (Par)

n=12	Seq	Par_3p	Par_5p	Par_9p
ime (ms)	46761	26710	15617	9162

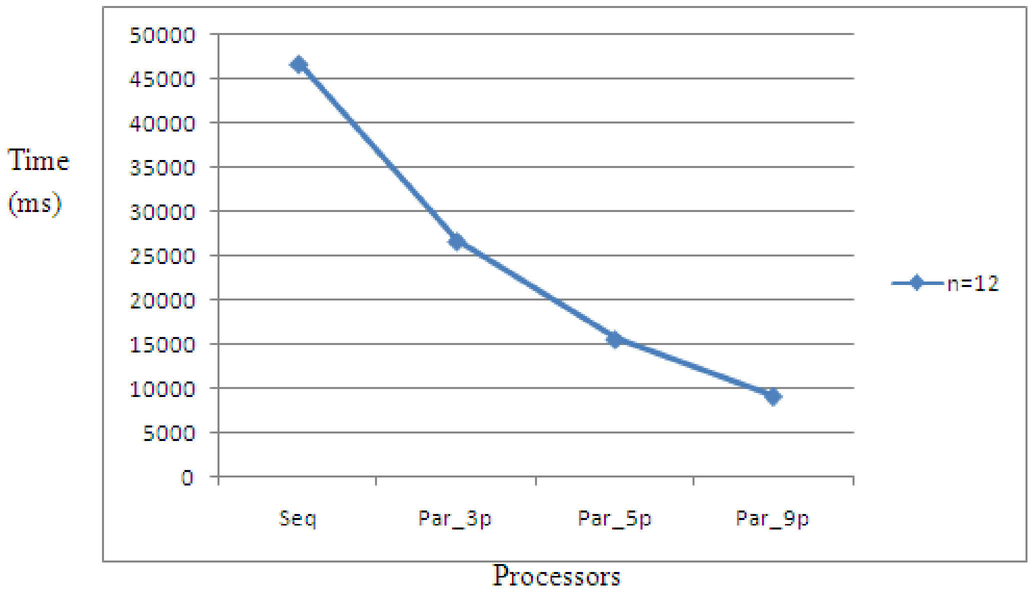


Figure 3. The graph illustrates execution time of the binary sequence with n = 12 on the Processors

It is noted that when n is big, the parallel algorithm will reduce the execution time as compared to the sequential algorithm. When we increase the number of processors, the execution time will decrease dramatically. However, when we increase the number of processors at a certain point, execution time does not reduce but increases.

3. The Algorithm Listing Permutations Of N Elements

3.1. Substitution, Inversion

Based on linear algebra theory and the study [1], the concepts of substituend, inversion are presented as follows:

Let set $X_n = \{1, 2, 3, ..., n\}$, $(n\geq1)$. A bijection $\sigma: X_n \rightarrow X_n$ is called a substituendon the set X_n

The set of all substituends on the set X_n is labeled S_n

Substituend $\sigma: X_n \rightarrow X_n$ is demonstrated as follows:

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & \ldots\ldots\ldots\ldots .n \\ \sigma(1) & \sigma(2) & \sigma(3) & \ldots\ldots\ldots .\sigma(n) \end{pmatrix} \quad (1)$$

where $\sigma(i)$ is the image of the element $i \in X_n$ written on the bottom line, in the same column as i.

For example.

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 4 & 1 \end{pmatrix} \quad (2)$$

is the substituendon the set $X_4 = \{1, 2, 3, 4\}$ determined by: σ (1) = 3, σ (2) = 2, σ (3) = 4, σ (4) = 1.

Then the number of substituendson the set X_n is equal to the number of permutations on that set and is n !. Thus, S_n has n! elements.

Suppose there exists a substituend on the set X_n. with i, j ∈ X_n, i ≠ j, the pair (σ (i), σ (j)) is aninversion of σ if i <j but σ (i)> σ(j).

For example. Let X_3, the substituend$\sigma_2 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}$has two inversions: (2, 1), (3, 1). the substituend$\tau_2 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}$has three inversions: (3, 2), (3, 1), (2, 1).

Set X_n has n! permutations and n! substituend. the inversion sequence on every substituend can be defined as follows: the value of inversion of element 1 in the substituend is assigned to that inversion sequence, the value of inversion of element 2 in the substituend is assigned to the inversion sequence. Let's continue with this for n elements. The following is the inversion sequence with n = 4.

Table 2. Substituend, inversion sequence and inversion vector sequence with n=4

N_o	Permutation	inversion	Inversion vector
1	1 2 3 4	0 0 0 0	0 0 0 0
2	2 1 3 4	1 0 0 0	0 0 0 1
3	2 3 1 4	2 0 0 0	0 0 0 2
4	2 3 4 1	3 0 0 0	0 0 0 3
5	1 3 2 4	0 1 0 0	0 0 1 0
6	3 1 2 4	1 1 0 0	0 0 1 1
7	3 2 1 4	2 1 0 0	0 0 1 2
8	3 2 4 1	3 1 0 0	0 0 1 3
9	1 3 4 2	0 2 0 0	0 0 2 0
10	3 1 4 2	1 2 0 0	0 0 2 1
11	3 4 1 2	2 2 0 0	0 0 2 2
12	3 4 2 1	3 2 0 0	0 0 2 3
13	1 2 4 3	0 0 1 0	0 1 0 0
14	2 1 4 3	1 0 1 0	0 1 0 1
15	2 4 1 3	2 0 1 0	0 1 0 2
16	2 4 3 1	3 0 1 0	0 1 0 3
17	1 4 2 3	0 1 1 0	0 1 1 0
18	4 1 2 3	1 1 1 0	0 1 1 1
19	4 2 1 3	2 1 1 0	0 1 1 2
20	4 2 3 1	3 1 1 0	0 1 1 3
21	1 4 3 2	0 2 1 0	0 1 2 0
22	4 1 3 2	1 2 1 0	0 1 2 1
23	4 3 1 2	2 2 1 0	0 1 2 2
24	4 3 2 1	3 2 1 0	0 1 2 3

Table 1 shows that a permutation always has an Inversion vector and an Inversion vector always has a permutation. Thus, instead of looking for the permutation of n elements in the order of the dictionary methods. The study comes up with a new idea is that to work on the permutations by finding the Inversion vector sequence. Inversion vector sequence (bounded sequence) is created with the initial sequence 0 0 0 0 and with final sequence 0 1 2 3 with n = 4.

3.2. BOUNDED SEQUENCES

The *set of integers* is represented by the letter *Z*. Let n be a positive integer, assume that p and q are two integer sequences of length n and denoted as follows:

$p=(p_1p_2\ldots p_n)$, $q=(q_1q_2\ldots q_n)| \; p_i, q_i \in Z, \forall i \in 1,\ldots,n$

We have the following definition:

1) $p \leq q$ If and only if $p_i \leq q_i \forall i \in 1,\ldots,n$

2) $p < q$ If and only if $\exists j \in (1 \ldots n)$: $p_j < q_j$ and $p_i \leq q_i : \forall i \in 1,\ldots,n)$ and $i \neq j$

Bounded sequence problems are demonstrated as follows:

Given two integer sequences s and g of length n, such that s <g, find all sequences t of length n such that s≤t≤g

Let $s=(s_1s_2\ldots s_n)$ and $g=(g_1g_2\ldots g_n)$, be two bound. The sequence $t=(t_1t_2\ldots t_n)$ must satisfy:

$$t_i \in Z \wedge s_i \leq t_i \leq g_i \forall i \in (1 \ldots n) \textbf{(3)}$$

Example: Let s = (0 0 0 0), g = (0 1 2 3) be two bounds, integer sequences t satisfy s≤t≤g. Thus, t is arranged in ascending dictionary order as in the following table:

Table 3.Bounded sequence t with s=(0 0 0 0), g=(0 1 2 3)

N$_o$	**Bounded sequence t**	**N$_o$**	**Bounded sequence t**	**N$_o$**	**Bounded sequence t**	**N$_o$**	**Bounded sequence t**	**N$_o$**	**Bounded sequence t**	**N$_0$**	**Bounded sequence t**
1	0 0 0 0	5	0 0 1 0	9	0 0 2 0	13	0 1 0 0	17	0 1 1 0	21	0 1 2 0
2	0 0 0 1	6	0 0 1 1	10	0 0 2 1	14	0 1 0 1	18	0 1 1 1	22	0 1 2 1
3	0 0 0 2	7	0 0 1 2	11	0 0 2 2	15	0 1 0 2	19	0 1 1 2	23	0 1 2 2
4	0 0 0 3	8	0 0 1 3	12	0 0 2 3	16	0 1 0 3	20	0 1 1 3	24	0 1 2 3

Theorem 1. Given two bounds s = (0 ... 0) (with n elements 0) and g = (0 1 2 ... n-1). The bounded sequence t satisfy $s \leq t \leq g$ which is the inversion vector of the set X_n= {1, 2, 3,..., n}, (n ≥ 1). The sequence t is equals to n! and the inversion Vector s = (0 ... 0) corresponds to the permutation (1 2 ... n) and the inversion Vector g = (0 1 2 ... n-1) corresponds to the permutation (n n-1 ... 1) .

Proof: See [1]

Theorem 2. Lets=$(s_1s_2\ldots s_n)$ and $g=(g_1g_2\ldots g_n)$ be two bounds. The sequences $t=(t_1t_2\ldots t_n)$ are bounded sequences. Let C be the number of bounded sequences t. Then we have:

$$C=\prod_{i=1}^{n}(g_i - s_i + 1) \textbf{(4)}$$

Proof: See [1]

Algorithm 4. Creating bounded sequence (s(n), g(n))

```
1.  BEGIN
2.  Input  n, s[i], g[i], i=1,…,n //s, g: two bounds
3.  t[i]:=s[i], i=1,…,n
4.  Repeat
5.  Print t[i], i=1,…,n
6.  i:=n;
7.  While t[i] =g[i] do
8.  Begin
        a.  t[i]:=s[i];
        b.  i:=i-1;
9.  End;
10. If i>=1 then t[i]:=t[i]+1;
11. Untill i=0
12. END.
```

3.3. PARALLEL ALGORITHM LISTING PERMUTATIONS OF N ELEMENTS

Algorithm finding the permutation of n elements by the dictionary method is sometimes challenging to determine the input and the end conditions of the processors. Thus, it is crucial to propose a parallel algorithm to find the permutations of n elements based on the bounded sequence to divide the bounded sequences for the processors.

3.3.1. THE IDEAS OF THE ALGORITHMS

If n increases, then the permutation is very large (n!). Therefore, a parallel algorithm must be built to improve computing performance.

The idea of parallel algorithms is to utilize k processors, which have a main processor called processor 0, and sub processors called k-1. The main processor receives the sequence s [i] and g [i] that are the two bounds as in algorithm 4. The main processor will find k bound sequences and send these k sequences for the sub processors to find the bounded sequences and convert

$$g_0 = (\overbrace{0 \ \dots \ 0}^{p} \ p \ n\text{-}1) \ \textbf{(5)}$$

them into permutation sequences. *k* Processor depends on p with k: = p !, p = (2, 3, ..., n-1). Given that p is chosen, the first bound has the smallest sequence: s_0= 0…0 (n number 0) and the largest sequence:

he smallest sequence of the 2 segment is:

$$s_1 = (\overbrace{0 \ldots 0}^{p-1\ N_o\ 0}\ 1\ \overbrace{0 \ldots 0}^{n-p\ N_o\ 0}) \qquad (6)$$

Thus, the sequence g_i (i = 0, ... k-1) is obtained by finding the bounded sequence of s'[i] = 0 ... 0 (p-1 N_o 0) and g'[i] = i, i = 1, ..., p-1. After finding the bounded sequence, 0 is inserted to the left side of the bounded sequence and p, ..., n-1 are inserted to the right side of the bounded sequence. We has g_i

Based on $g_{i-1.}$ s_i. is found as follows:
m=Max(j, g_{i-1}[j]<g[j], The value of s_i[1] to s_i[m-1] is unchanged, ie g_{i-1}[a], a = 1, ... , m-1
s_i[m]: = g_{i-1}[m] +1
s_1[i] = 0, i = m + 1, ..., n

For example: Let n = 4, choose p = 3, k = 6, then we have 3! = 6 segments. These six segments are allocated to 6 sub processors shown in Table 3
:

Table 4. Six segments are allocated to 6 sub processors

N_o	**Bounded sequence t**	**N_o**	**Bounded sequence t**	**N_o**	**Bounded sequence t**	**N_o**	**Bounded sequence t**	**N_o**	**Bounded sequence t**	**N_o**	**Bounded sequence t**
1	0 0 0 0	5	0 0 1 0	9	0 0 2 0	13	0 1 0 0	17	0 1 1 0	21	0 1 2 0
2	0 0 0 1	6	0 0 1 1	10	0 0 2 1	14	0 1 0 1	18	0 1 1 1	22	0 1 2 1
3	0 0 0 2	7	0 0 1 2	11	0 0 2 2	15	0 1 0 2	19	0 1 1 2	23	0 1 2 2
4	0 0 0 3	8	0 0 1 3	12	0 0 2 3	16	0 1 0 3	20	0 1 1 3	24	0 1 2 3
Segment 1: (s_1,g_1) =(0000,0003)		Segment 2: (s_2,g_2) =(0010,0013)		Segment 3: (s_3,g_3) =(0020,0023)		Segment 4: (s_4,g_4) =(0100,0103)		Segment5:(s_5,g_5) =(0110,0113)		Segment 6: (s_6,g_6) =(0120,0123)	

3.3.2. PARALLEL ALGORITHM

Processors number k= p!; p=(2, 3,…,n-1)

Algorithm 5: Parallel algorithm finding permutation of n elements

```
{
1.      Input n, p (p$\in \{2,3,\ldots,n-1\}$)
2.      s[i]:=0 $\forall i = 1,\ldots,n$
3.      g[i+1]:=i $\forall i = 0,1,\ldots,n-1$
4.      k:=p!; p=(2, 3,…,n-1) // k is processors
5.      //The main processor  finds  k subsegments, then divides to the subprocessors
If k=1 (Rank =1) then
{

    // Find the bounded by algorithm 4 and send data to subprocessors
        5.1.    s'[i]=0, i=1,…,p-1
        5.2.   g'[i]=i, i=1,…,p-1
        5.3.   $c_j$ :=Algorithm 4 (s'(i), g'(i)), j=1,…,k.
        5.4.   Send(s[i]=0, $\forall i$ =1,…,n to $p_1$)
        5.5.   Send ($c_j$ to $p_j$ (j=1,…,k)
        5.6.   Send ($c_j$) to $p_{j+1}$ (j=1 to k-1)
    6.7. Send g[i] in step 4 to subprocessors
}

6.      // Subprocessors perform concurrently
{
        6.1.   Receive(data)
        6.2.   Insert element 0 to the left of $c_j$ (j = 1, ..., k) // j is the index of the k
               processors
        6.3.   Insert the elements p, p + 1, ... n-1 to the right of $c_j$ (j = 1, ..., k)
        6.4.   $g_j$:=$c_j$ (j=1,2,…,k) //$g_j$ is the largest bound sequence.
        6.5.   The subprocessor $p_1$ initiates $s_1$: $s_1$[i]: = 0 ∀i = 1, ..., n // $s_1$ is the smallest
               bound sequence on processor $p_1$.
    // the Subprocessor $p_2$, $p_3$,…,$p_k$ find the smallest bound sequence as follows::

        6.6.   i:=n;
          6.7.   While $c_{j-1}$[i] =g[i] do
        6.8.        Begin
          6.9.          $c_{j-1}$[i]:=0;
          6.10.         i:=i-1;
          6.11.     End;
          6.12.  If i>=1 then $c_{j-1}$[i]:=$c_{j-1}$[i]+1;
          6.13.  $s_j$[i]:=$c_{j-1}$[j], i=1,…,n, j=2,…,k
    7.      $t_j$[i] :=Algorithm 4 ($s_j$(i), $g_j$(i)), j=1,…,k, i=1,…,n.
    8.      Convert all bounded sequences $t_j$[i] to permutation sequences
    9.      Send permutations sequences to main processor.
    10.     The main processor print results and ends.
```

Theorem 3: The Parallel algorithm is TRUE.

Proof:

First, we need to prove that the bound sequences s_j and g_j on k processors satisfy the formula (3), ie, s_j and g_j are in the bounded sequence with the smallest bound sequence s[i]: = 0 ∀i = 1, ..., n, and the largest bound sequence g[i + 1]: = i ∀i = 0,1, ..., n-1.

g_j is computed in step 6.3 in the parallel algorithm by inserting 0 to the left of t and inserting p, p + 1, n-1 to the right of c_j, then g_j [i] ≤g[i], i = 1, ..., n. s_{j+1} is based on the g_j given from steps 7.6 to 7.13. there always exists s[i] ≤ s_j[i], i = 1, ..., n. Thus s_j and g_j satisfy the formula (3) with 2 bound sequences s[i] and g[i], i = 1, ..., n.

Next, we prove that the total number of bounded sequences in the *k* processors is n!

When p (p∈ {2,3, ..., n-1}) is chosen, the number of processors involved in finding the bounded sequences is k = p! (Note that the number of processors to find bounded sequences are equal.)

The smallest bound sequence of p_1 is s_1 = (0 ... .0) (n number 0) and the largest bound sequence is g_1 based on formula (5). According to the solution in (4), the number of bounded sequence in segment 1 that the subprocessor p_1 has done is $\prod_{i=p}^{n-1}(i+1)$. Each subprocessor will also find the number of bounded sequences equal to $\prod_{i=p}^{n-1}(i+1)$. In addition, s_j and g_j are two bound sequences on the subprocessor p_j, then $s_j[i] = g_j[i], \forall i = 1, \ldots, p$ and $s_j[i] = 0, \forall i = p + 1, \ldots, n$ and $g_j[i] = i, \forall i = p, \ldots, n$-1.. Applying the formula (4) to the two bound sequences s_j and g_j, the number of bounded sequence to each processor is $\prod_{i=p}^{n-1}(i+1)$=(p+1).(p+2). …. . n. On the other hand, we have the number of processors k = p! So the number of buonded sequences by the *k* processors is:

k.(p + 1). (p + 2). …. . n = p! (p + 1). (p + 2). …. . n = n! Thus, the number of bounded sequences on the *k* processors is n! which is equal to permutation n!.☐

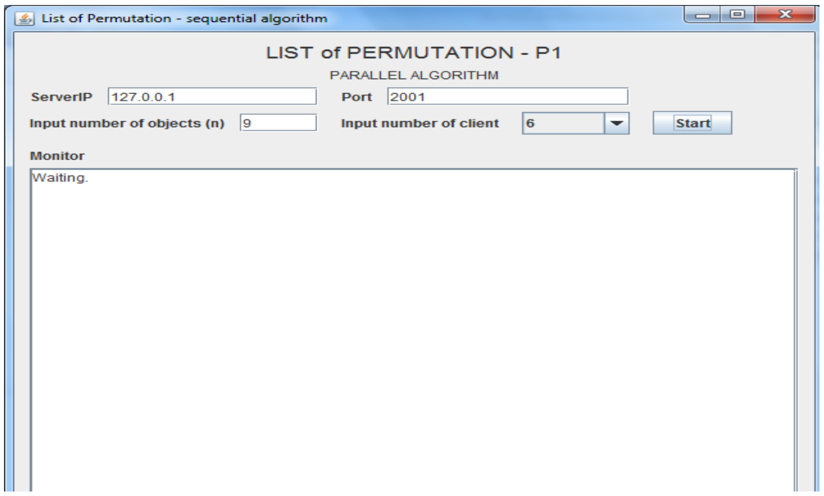
3.3.3. Experimental Results

The algorithm is implemented in the computer with its configuration:

Processor: corei7 2.6GHz and disk: write 28-30 Mb/s

- Interface on the main processor P_1: In this main interface, we need to select n and the number of subprocessors (Figure 4).

- Interface for the sub processors P_i (i=1,2....,k) (Figure 5)
- Resulting interface on the main processor P_1. The permutation result is saved as a file (Figure 6).


List of Permutation - sequential algorithm

LIST of PERMUTATION - P1

PARALLEL ALGORITHM

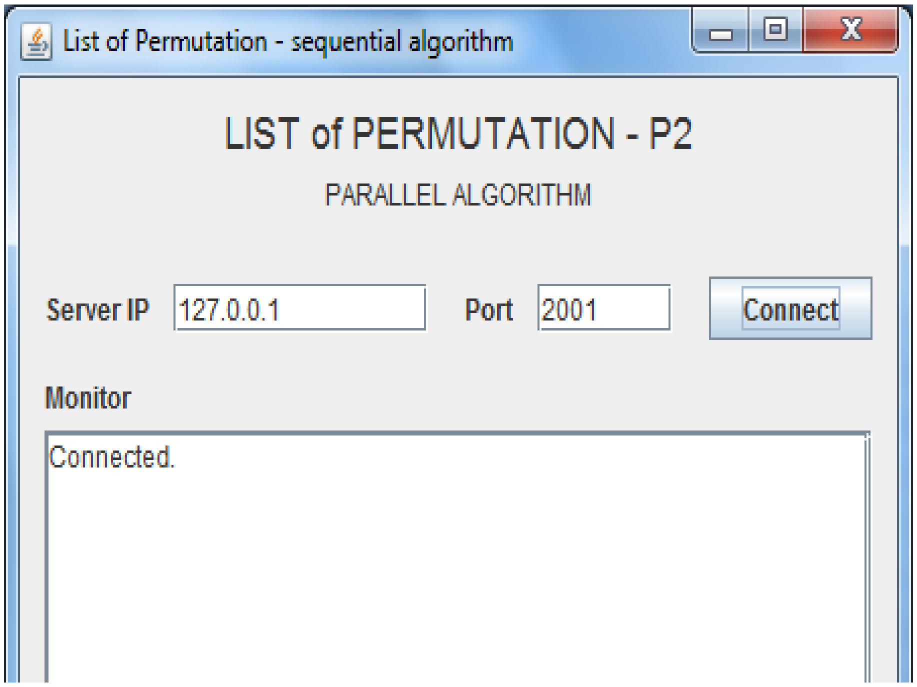
ServerIP 127.0.0.1 Port 2001

Input number of objects (n) 9 Input number of client 6

Monitor

Waiting.

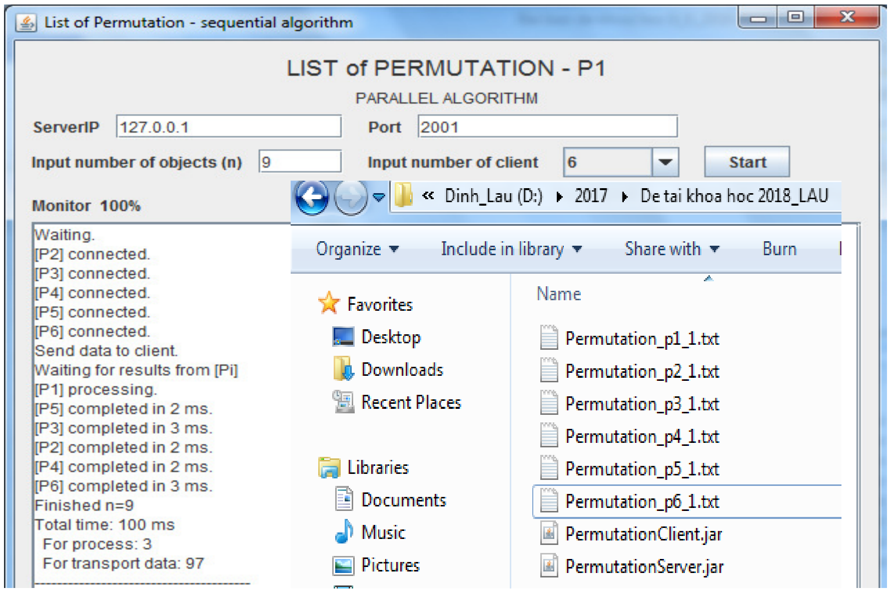
Figure 4. Interface of main processor P_1



Server IP 127.0.0.1 Port 2001

```
Connected.
```

Figure 5. Interface of sub processors P_i



ServerIP 127.0.0.1 Port 2001

Input number of objects (n) 9 Input number of client 6

Monitor 100%

```
Waiting.
[P2] connected.
[P3] connected.
[P4] connected.
[P5] connected.
[P6] connected.
Send data to client.
Waiting for results from [Pi]
[P1] processing.
[P5] completed in 2 ms.
[P3] completed in 3 ms.
[P2] completed in 2 ms.
[P4] completed in 2 ms.
[P6] completed in 3 ms.
Finished n=9
Total time: 100 ms
  For process: 3
  For transport data: 97
```

Figure 6. Interface results of the main processor P_1

Table 4. The execution time (ms) on the sequential (Seq) and parallel (Par) (n=9 to 12)

N	Seq	Par_2p	Par_6p	Par_24p
9	323	234	100	97
10	3200	1879	691	151
11	45985	24061	8481	2190
12	1143542	581107	210590	27228

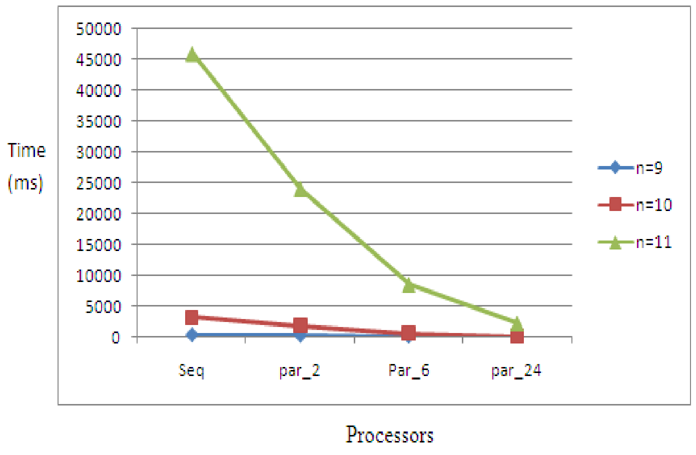


Figure 7. The graph illustrates time listing permutation of n elements by the subprocessors

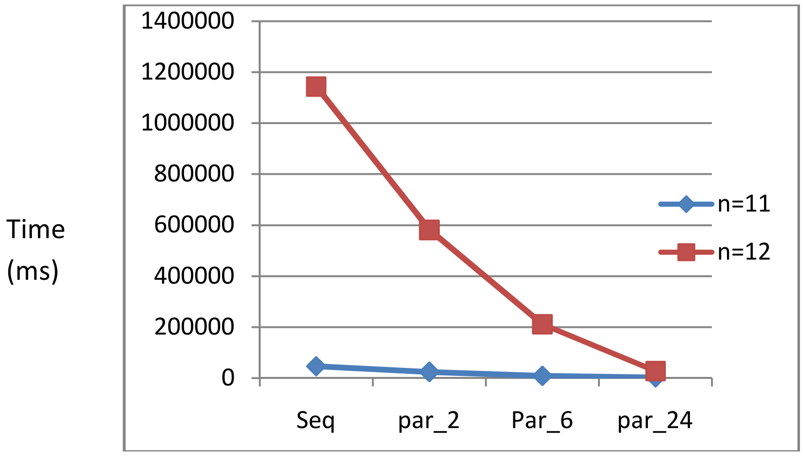


Figure 8. The graph illustrates time listing permutation of n=11 and n=12 by the subprocessors

Remarks: a close look at Table 4, Figure 7 and Figure 8 shows that if n is large enough, the parallel computation time is much lower than the sequential computation time. When the sub processors increase in number, the computation time will decrease. When n increases to 1 unit, the number of permutations increases dramatically, so the computation time goes up sharply (Figure 8). However, if you abuse and increase too many processors, the computation time will also go up.

4. CONCLUSION

The paper solves the problem of improved computing performance for two listing binary sequences and listing permutations with sufficiently large n. It is an interesting and innovative idea in case n is large. This newly-built parallel algorithm was experimental with large n and with numerous different sub processors. This paper is devoted to building up a general algorithm for multiple processors. Last but not least, it demonstrates the correctness and experiments in multi-processing MPI and Thread library.

REFERENCES

1. Nguyen Dinh Lau, Parallel algorithm list permutations,@ 2017,ISBN: 978-604-67-1009-7, 23-24/11/2017, Quy Nhon, Binh Dinh, Vietnam, pp 348-353.

2. Nguyen Dinh Lau, Parallel algorithm for the graph, Doctoral dissertation, University of Technology, The University of Da Nang, 2015.

3. Hoang Chi Thanh, Parallel Generation of Permutations by Inversion Vectors,Proceedings of IEEE-RIVF International Conference on Computing and Communication Technologies, IEEE, ISBN: 978-1-4673-0308-8, 2012, pp.129-132.

4. Hoang Chi Thanh, Parallelizing a new algorithm for the set partition problem, Annals UMCS Information AIX, 2(2010) pp. 21-28, DOI:10.2478/v10065-010-0049-1, 2010, (http://dlibra.umcs.lublin.pl/dlibra/plain-content?id=12053)

5. Hoang Chi Thanh, Nguyen Thi Thuy Loan. Nguyen Duy Ham, *From Permutations to Iterative Permutations*, International Journal of Computer Science Engineering and Technology**,** Vol 2, Issue 7, 2012, pp. 1310-1315.

6. Hoang Chi Thanh*, Parallel combinatorial algorithms for multi-sets and their applications,* International Journal of Software Engineering and Knowledge Engineering, Vol. 23, No. 01, 2013, pp. 81-99

7. Hoàng Chi Thanh, *Inheritance principle and some bounded sequence problems,* The *Journal of Computer Science and Cybernetics,* T.29 S.1, 2013, pp. 79-91.

8. Ivan Stojmenovic, *Listing combinatorial objects in parallel*, The international journal of parallel emergent and distributed systems, vol. 21, no. 2, April 2006, pp. 127–146.

9. Akl, S.G., Gries, D. and Stojmenovic, I., *An optimal parallel algorithm for generating combinations*, Information Processing Letters, 33, 1989, pp. 135–139.

10. Akl, S.G., Meijer, H. and Stojmenovi, I., *An optimal systolic algorithm for generating permutations in lexicographic order*, Journal of Parallel and Distributed Computing, 20(1), 1994, pp. 84–91.

11. Akl, S.G. and Stojmenovic I., *Parallel algorithms for generating integer partitions and compositions*, The Journal of Combinatorial Mathematics and Combinatorial Computing, 13, 1983, pp. 107–120.

12. Chen, G.H. and Chern, M.S., *Parallel generation of permutations and combinations*, BIT, 26, 1986, pp. 277–283.

13. Cosnard, M. and Ferreira, A.G., *Generating permutations on a VLSI suitable linear network*, The Computer Journal, 32(6),1989, pp. 571–573.

14. Djokic, B., Miyakawa, M., Sekiguchi, S., Semba, I. and Stojmenovic, I., *Parallel algorithms for generating subsets and set partitions*. In: T. Asano, T. Ibaraki, H. Imai and T. Nishizeki (Eds.) Proceedings of SIGAL International Symposium on Algorithms, Tokyo, Japan, Lecture Notes in Computer Science, Vol. 450, 1990, pp. 76–85.

15. Even, S., Algorithmic Combinatorics (New York: Macmillan). Er, M.C., 1988, *A parallel algorithm for cost-optimal generation of permutations of r out of n items,* Journal of Information & Optimization Sciences, 9, 1973, pp. 53–56.

16. Elhage, H. and Stojmenovic, I., *Systolic generation of combinations from arbitrary elements*, Parallel Processing Letters, 2(2/3), 1992, pp. 241–248.

17. Gupta, P. and Bhattacharjee, G.P., *Parallel generation of permutations*, The Computer Journal, 26(2), 1983, pp. 97–105.

18. Kapralski, A., *New methods for the generation of permutations, combinations, and other combinatorial objects in parallel*, Journal of Parallel and Distributed Computing, 17, 1993, pp. 315–326.

19. Seyed H. Roosta, *Parallel Processing and Parallel Algorithms, Theory and Computation,*USA,Springer 1999.

20. Steve Fortune and James Wyllie, *Parallelism in random access machines*, STOC '78 Proceedings of the tenth annual ACM symposium on Theory ofcomputing, 1978, pp 114-118.

21. Nguyen Dinh Lau, Tran Quoc Chien, Phan Phu Cuong, Le Hong Dung, *On the implementation of Goldberg's maximum Flow Algorithm in extended mixed network*, International Journal of computer Science & Information Technology, Vol 9, No 6 pp. 93-102, 2017.

22. Nguyen Dinh Lau, Tran Quoc Chien,*Algorithm to Find Maximum Concurent Multicommodity Linear Flow with Limited Cost on Extended Traffic Network with Single Regulating Coeffitient on Two-Side Lines*, The International Journal of Computer Networks & Communications, V 9 N2, pp**:** 57-67, 2017.

23. Nguyen Dinh Lau, Tran Quoc Chien,*Traveling Salesman Problem in Distributed Envirenment*, Computer Sciencs & Information Technology (CSIT), Fourth International Conference on Advanced Information Technologies and Applications (ICAITA 2015), pp. 19-28, 2015.

24. Peter S. Pacheco, *An Introduction to Parallel Programming*, Morgan Kaufmann Publishers is an imprint of Elsevier, ISBN 978-0-12-374260-5 (hardback), 2011