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 the Thread to analyze and compare computing time.

2. Listing Binary Sequence Algorithm

2.1. Sequential algorithm

Let $n \in \mathbb{N}$. List all binary sequences with $n$ length, i.e., sequence $[b_1, ..., b_n]$, where $b_i \in \{0, 1\}$ for $i = 1, ..., n$.

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]:= 0 i = 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 sub 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.

\[
\begin{align*}
R_1 &\text{ listing binary: } 0000, 0001, 0010, 0011 \\
R_2 &\text{ listing binary: } 0100, 0101, 0110, 0111 \\
R_3 &\text{ listing binary: } 1000, 1001, 1010, 1011 \\
R_4 &\text{ listing binary: } 1100, 1101, 1110, 1111 \\
\end{align*}
\]

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 \( \text{Rank} = 0 \) then \( // \text{main processor } R_0 \)
5. Begin
6. Call Algorithm 1 \( (n') \) \( // \text{listing binary sequence } (t_1, t_2, ..., t_{n'}) \forall i \in 1, ..., 2^{n'} \) of length \( n' \)
7. Create \( 2^n (t_1, t_2, ..., t_{n'}, t_{n'+1}, ..., t_n) := (t_1, t_2, ..., t_{n'}) \cup (0, ..., 0) \forall i \in 1, ..., 2^{n'} \) of length \( n \)
8. Send \( (t_1, t_2, ..., t_{n'}, t_{n'+1}, ..., t_n) \forall i \in 1, ..., 2^{n'} \) 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 the 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 $2^{n-n'} = 2^{4-2} = 4$ 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 ($t_{n'+1}, t_{n'+2}, \ldots, t_n$) $\forall i \in 1, \ldots, 2^{n-n'}$ has length is $n-n'$
7. **Broadcast** ($t_{n'+1}, t_{n'+2}, \ldots, t_n$) $\forall i \in 1, \ldots, 2^{n-n'}$ to $2^{n'}$ sub processors ($P_1, \ldots, P_{k-1}$)
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 ($t_{n'+1}, t_{n'+2}, \ldots, t_n$) $\forall j \in 1, \ldots, 2^{n-n'}$
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].
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.

<table>
<thead>
<tr>
<th>$n$=12</th>
<th>Seq</th>
<th>Par_3p</th>
<th>Par_5p</th>
<th>Par_9p</th>
</tr>
</thead>
<tbody>
<tr>
<td>time (ms)</td>
<td>46761</td>
<td>26710</td>
<td>15617</td>
<td>9162</td>
</tr>
</tbody>
</table>
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 $X_n = \{1, 2, 3, ..., n\}$, ($n \geq 1$). A bijection $\sigma: X_n \rightarrow X_n$ is called a substituend on 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:

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

(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,

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

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

Then the number of substituend on 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 $i, j \in X_n$, $i \neq j$, the pair $(\sigma(i), \sigma(j))$ is an inversion of $\sigma$ if $i < j$ but $\sigma(i) > \sigma(j)$.

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

Set $X_n$ has $n!$ permutations and $n!$ substritueud. 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 $n=4$

<table>
<thead>
<tr>
<th>No</th>
<th>Permutation</th>
<th>inversion</th>
<th>Inversion vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 2 3 4</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>2</td>
<td>2 1 3 4</td>
<td>1 0 0 0</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>3</td>
<td>2 3 1 4</td>
<td>2 0 0 0</td>
<td>0 0 0 2</td>
</tr>
<tr>
<td>4</td>
<td>2 3 4 1</td>
<td>3 0 0 0</td>
<td>0 0 0 3</td>
</tr>
<tr>
<td>5</td>
<td>1 3 2 4</td>
<td>0 1 0 0</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>6</td>
<td>3 1 2 4</td>
<td>1 1 0 0</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>7</td>
<td>3 2 1 4</td>
<td>2 1 0 0</td>
<td>0 0 1 2</td>
</tr>
<tr>
<td>8</td>
<td>3 2 4 1</td>
<td>3 1 0 0</td>
<td>0 0 1 3</td>
</tr>
<tr>
<td>9</td>
<td>1 3 4 2</td>
<td>0 2 0 0</td>
<td>0 0 2 0</td>
</tr>
<tr>
<td>10</td>
<td>3 1 4 2</td>
<td>1 2 0 0</td>
<td>0 0 2 1</td>
</tr>
<tr>
<td>11</td>
<td>3 4 1 2</td>
<td>2 2 0 0</td>
<td>0 0 2 2</td>
</tr>
<tr>
<td>12</td>
<td>3 4 2 1</td>
<td>3 2 0 0</td>
<td>0 0 2 3</td>
</tr>
<tr>
<td>13</td>
<td>1 2 4 3</td>
<td>0 0 1 0</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>14</td>
<td>2 1 4 3</td>
<td>1 0 1 0</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>15</td>
<td>2 4 1 3</td>
<td>2 0 1 0</td>
<td>0 1 0 2</td>
</tr>
<tr>
<td>16</td>
<td>2 4 3 1</td>
<td>3 0 1 0</td>
<td>0 1 0 3</td>
</tr>
<tr>
<td>17</td>
<td>1 4 2 3</td>
<td>0 1 1 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>18</td>
<td>4 1 2 3</td>
<td>1 1 1 0</td>
<td>0 1 1 1</td>
</tr>
<tr>
<td>19</td>
<td>4 2 1 3</td>
<td>2 1 1 0</td>
<td>0 1 1 2</td>
</tr>
<tr>
<td>20</td>
<td>4 2 3 1</td>
<td>3 1 1 0</td>
<td>0 1 1 3</td>
</tr>
<tr>
<td>21</td>
<td>1 4 3 2</td>
<td>0 2 1 0</td>
<td>0 1 2 0</td>
</tr>
<tr>
<td>22</td>
<td>4 1 3 2</td>
<td>1 2 1 0</td>
<td>0 1 2 1</td>
</tr>
<tr>
<td>23</td>
<td>4 3 1 2</td>
<td>2 2 1 0</td>
<td>0 1 2 2</td>
</tr>
<tr>
<td>24</td>
<td>4 3 2 1</td>
<td>3 2 1 0</td>
<td>0 1 2 3</td>
</tr>
</tbody>
</table>

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 \( \mathbb{Z} \). Let n be a positive integer, assume that p and q are two integer sequences of length n and denoted as follows:

\[
p=(p_1p_2...p_n), \quad q=(q_1q_2...q_n) | \quad p_i, q_i \in \mathbb{Z}, \forall i \in 1, ..., n
\]

We have the following definition:

1) \( p \leq q \) If and only if \( p_i \leq q_i \forall i \in 1, ..., n \)
2) \( p < q \) If and only if \( \exists j \in (1 ... n): p_j < q_j \) and \( p_i \leq q_i : \forall i \in 1, ..., n \) 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 \( s \leq t \leq g \).

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

\[
t\in \mathbb{Z} \land s_i \leq t_i \leq g_i \forall i \in (1 ... n)(3)
\]

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

<table>
<thead>
<tr>
<th>( N_o )</th>
<th>Bounded sequence t</th>
<th>( N_o )</th>
<th>Bounded sequence t</th>
<th>( N_o )</th>
<th>Bounded sequence t</th>
<th>( N_o )</th>
<th>Bounded sequence t</th>
<th>( N_o )</th>
<th>Bounded sequence t</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 0 0 0</td>
<td>5</td>
<td>0 1 0</td>
<td>9</td>
<td>0 0 2 0</td>
<td>13</td>
<td>0 1 0</td>
<td>17</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>2</td>
<td>0 0 0 1</td>
<td>6</td>
<td>0 1 1</td>
<td>10</td>
<td>0 0 2 1</td>
<td>14</td>
<td>0 1 0</td>
<td>18</td>
<td>0 1 1 1</td>
</tr>
<tr>
<td>3</td>
<td>0 0 0 2</td>
<td>7</td>
<td>0 1 2</td>
<td>11</td>
<td>0 0 2 2</td>
<td>15</td>
<td>0 1 0</td>
<td>19</td>
<td>0 1 1 2</td>
</tr>
<tr>
<td>4</td>
<td>0 0 0 3</td>
<td>8</td>
<td>0 1 3</td>
<td>16</td>
<td>0 1 3</td>
<td>20</td>
<td>0 1 1 3</td>
<td>24</td>
<td>0 1 2 3</td>
</tr>
</tbody>
</table>

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 \( X_n=\{1, 2, 3, ..., n\} \), \( n \geq 1 \). 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. Let\( s=(s_1s_2...s_n) \) and \( g=(g_1g_2...g_n) \) be two bounds. The sequences \( t=(t_1t_2...t_n) \) are bounded sequences. Let \( C \) be the number of bounded sequences t. Then we have:

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

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. Until $i=0$
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

$$g_0 = \left(\begin{array}{c} 0 \ldots 0 \ p \ n-1 \end{array}\right) \ (5)$$

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: $s_0 = 0\ldots0\ (n\ number\ 0)$ and the largest sequence:
The smallest sequence of the 2 segment is:

\[\begin{align*}
    p-1 \ N_0 & 0 \\
    n-p \ N_0 & 0 \\
    s_1 &= (0 \ldots 0 \ 1 \ 0 \ldots 0) \\
\end{align*}\]

(6)

Thus, the sequence \(g_i\) (\(i = 0, \ldots k-1\)) is obtained by finding the bounded sequence of \(s'[i] = 0 \ldots 0\) \((p-1 \ N_0 \ 0)\) and \(g'[i] = i, i = 1, \ldots, p-1\). After finding the bounded sequence, 0 is inserted to the left side of the bounded sequence and \(p, \ldots, n-1\) are inserted to the right side of the bounded sequence. We have \(g_i\)

Based on \(g_{i-1}\), \(s_i\) is found as follows:

\[m = \text{Max}(j, g_{i-1}[j] < g[j])\]

The value of \(s_i[1]\) to \(s_i[m-1]\) is unchanged, ie \(g_{i-1}[a], a = 1, \ldots, m-1\)

\[s_1[i] = 0, i = m + 1, \ldots, n\]

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

<table>
<thead>
<tr>
<th>N_0</th>
<th>Bounded sequence</th>
<th>N_0</th>
<th>Bounded sequence</th>
<th>N_0</th>
<th>Bounded sequence</th>
<th>N_0</th>
<th>Bounded sequence</th>
<th>N_0</th>
<th>Bounded sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 0 0 0</td>
<td>5</td>
<td>0 0 1 0</td>
<td>9</td>
<td>0 0 2 0</td>
<td>13</td>
<td>0 1 0 0</td>
<td>17</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>2</td>
<td>0 0 0 1</td>
<td>6</td>
<td>0 0 1 1</td>
<td>10</td>
<td>0 0 2 1</td>
<td>14</td>
<td>0 1 0 1</td>
<td>18</td>
<td>0 1 1 1</td>
</tr>
<tr>
<td>3</td>
<td>0 0 0 2</td>
<td>7</td>
<td>0 0 1 2</td>
<td>11</td>
<td>0 0 2 2</td>
<td>15</td>
<td>0 1 0 2</td>
<td>19</td>
<td>0 1 1 2</td>
</tr>
<tr>
<td>4</td>
<td>0 0 0 3</td>
<td>8</td>
<td>0 0 1 3</td>
<td>12</td>
<td>0 0 2 3</td>
<td>16</td>
<td>0 1 0 3</td>
<td>20</td>
<td>0 1 1 3</td>
</tr>
<tr>
<td></td>
<td>(s_1,g_1)</td>
<td></td>
<td>(s_2,g_2)</td>
<td></td>
<td>(s_3,g_3)</td>
<td></td>
<td>(s_4,g_4)</td>
<td></td>
<td>(s_5,g_5)</td>
</tr>
<tr>
<td></td>
<td>= (0000,0003)</td>
<td></td>
<td>= (0010,0013)</td>
<td></td>
<td>= (0020,0023)</td>
<td></td>
<td>= (0010,0013)</td>
<td></td>
<td>= (0110,0113)</td>
</tr>
<tr>
<td></td>
<td>Segment 1:</td>
<td></td>
<td>Segment 2:</td>
<td></td>
<td>Segment 3:</td>
<td></td>
<td>Segment 4:</td>
<td></td>
<td>Segment 5:</td>
</tr>
<tr>
<td></td>
<td>(s_1,g_1)</td>
<td></td>
<td>(s_2,g_2)</td>
<td></td>
<td>(s_3,g_3)</td>
<td></td>
<td>(s_4,g_4)</td>
<td></td>
<td>(s_5,g_5)</td>
</tr>
<tr>
<td></td>
<td>= (0000,0003)</td>
<td></td>
<td>= (0010,0013)</td>
<td></td>
<td>= (0020,0023)</td>
<td></td>
<td>= (0010,0013)</td>
<td></td>
<td>= (0110,0113)</td>
</tr>
<tr>
<td></td>
<td>Segment 6:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(s_6,g_6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>= (0120,0123)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.3.2. PARALLEL ALGORITHM

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

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. \(k := p!; p = (2, 3, \ldots, n-1) \// k\) 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. \( s'[i]=0, \; i=1,\ldots,p-1 \)
5.2. \( g'[i]=i, \; i=1,\ldots,p-1 \)
5.3. \( c_j := \text{Algorithm 4} \left( s'(i), \; g'(i) \right), \; j=1,\ldots,k \).
5.4. Send \( s[i]=0, \; \forall i =1,\ldots,n \) to \( p_1 \)
5.5. Send \( c_i \) to \( p_j \) \( (j=1,\ldots,k) \)
5.6. Send \( c_i \) to \( p_{j+1} \) \( (j=1 \text{ to } k-1) \)
5.7. 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 \( c_j \) \( (j = 1, \ldots, k) \) // j is the index of the k processors
6.3. Insert the elements \( p, \; p+1, \ldots, n-1 \) to the right of \( c_j \) \( (j = 1, \ldots, k) \)
6.4. \( g_j = c_j \; \text{(j=1,2,...,k)} \) // \( g_j \) is the largest bound sequence.
6.5. The subprocessor \( p_1 \) initiates \( s_1: \; s_1[i] = 0 \; \forall i = 1, \ldots, n \) // \( s_1 \) is the smallest bound sequence on processor \( p_1 \).
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. \( s[i] := c_{j-1}[j], \; i=1,\ldots,n, \; j=2,\ldots,k \)
7. \( t[i] := \text{Algorithm 4} \left( s(i), \; g(i) \right), \; j=1,\ldots,k, \; i=1,\ldots,n \)
8. Convert all bounded sequences \( t[i] \) 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 \( s[i] := 0 \; \forall i = 1, \ldots, n \), and the largest bound sequence \( g[i+1] := i \; \forall i = 0,1, \ldots, n-1 \).

\( g_j \) 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 \( g_j[i] \leq g[i] \), \( i = 1, \ldots, n \). \( s_{j+1} \) is based on the \( g_j \) given from steps 7.6 to 7.13. there always exists \( s[i] \leq s_j[i], \; i = 1, \ldots, n \). Thus \( s_j \) and \( g_j \) satisfy the formula (3) with 2 bound sequences \( s[i] \) and \( g[i], \; i = 1, \ldots, n \).

Next, we prove that the total number of bounded sequences in the \( k \) processors is \( n! \).
When \( p (p \in \{2, 3, ..., n-1\}) \) 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 \( s_1 = (0 \ldots 0) \) (\( n \) 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 \( s_j[i] = 0, \forall i = p + 1, \ldots, n \), \( s_j \) 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 \( s_j[i] = 0, \forall i = p + 1, \ldots, n \). Applying the formula (4) to the two bound sequences \( s_j \) and \( g_j \), the number of bounded sequence to each processor is \( \prod_{i=p}^{n-1}(i + 1) = (p+1)(p+2) \ldots . n \). On the other hand, we have the number of processors \( k = p! \). So the number of bounded sequences by the \( k \) processors is:

\[
k.(p + 1). (p + 2). \ldots . n = p! (p + 1). (p + 2). \ldots . n = n! \]

Thus, the number of bounded sequences on the \( k \) processors is \( n! \) which is equal to permutation \( n! \). \( \square \)

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

![Figure 4. Interface of main processor P1](image-url)
Figure 5. Interface of sub processors P_i

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)

<table>
<thead>
<tr>
<th>N</th>
<th>Seq</th>
<th>Par_2p</th>
<th>Par_6p</th>
<th>Par_24p</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>323</td>
<td>234</td>
<td>100</td>
<td>97</td>
</tr>
<tr>
<td>10</td>
<td>3200</td>
<td>1879</td>
<td>691</td>
<td>151</td>
</tr>
<tr>
<td>11</td>
<td>45985</td>
<td>24061</td>
<td>8481</td>
<td>2190</td>
</tr>
<tr>
<td>12</td>
<td>1143542</td>
<td>581107</td>
<td>210590</td>
<td>27228</td>
</tr>
</tbody>
</table>

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 subprocessors 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


