TECHNICAL TRANSACTIONS | CZASOPISMO TECHNICZNE
AUTOMATIC CONTROL | AUTOMATYKA
3-AC/2013

ZBIGNIEW KOKOSIŃSKI*

# A PARALLEL DYNAMIC PROGRAMMING ALGORITHM FOR UNRANKING SET PARTITIONS

# RÓWNOLEGŁY ALGORYTM PROGRAMOWANIA DYNAMICZNEGO DO KONWERSJI LICZB PORZĄDKOWYCH W PODZIAŁY ZBIORU

Abstract

In this paper, an $O(n)$ parallel algorithm is presented for unranking set partitions in Hutchinson's representation. A simple sequential algorithm is derived on the basis of a dynamic programming paradigm. In the parallel algorithm, processing is performed in a dedicated parallel architecture combining certain systolic and associative features. The algorithm consists of two phases. In the first phase, a coefficient table is created by systolic computations. Then, $n$ subsequent elements of a partition codeword are computed, in $O(1)$ time each, through associative search operations.

*Keywords*: *set partition, unranking algorithm, associative algorithm, parallel dynamic programming*

Streszczenie

W artykule przedstawiono równoległy algorytm o złożoności $O(n)$ dla wyznaczania podziału zbioru $\{1, ..., n\}$ w reprezentacji Hutchinsona na podstawie jego liczby porządkowej. Prosty algorytm sekwencyjny opiera się na paradygmacie programowania dynamicznego. Algorytm równoległy łączy w sobie cechy programowania systolicznego i asocjacyjnego. Algorytm składa się z dwóch kroków. W pierwszej kolejności, za pomocą obliczeń systolicznych, wyznaczana jest tablica współczynników, zwanych liczbami Williamsona. Następnie, przez asocjacyjne wyznaczanie maksimum zbioru liczb, obliczanych jest $n$ kolejnych elementów reprezentujących podział, każdy w czasie $O(1)$.

*Słowa kluczowe*: *podział zbioru, algorytm konwersji liczby porządkowej w obiekt, algorytm asocjacyjny, równoległe programowanie dynamiczne*

* Ph.D. Zbigniew Kokosiński, email: zk@pk.edu.pl, Department of Automatic Control and Information Technology, Faculty of Electrical and Computer Engineering, Cracow University of Technology.

## 1. Introduction

Listing, ranking and unranking of combinatorial objects is an important problem in computer science and engineering in many application areas [12, 13].

The most common application of unranking algorithms is 'translation' of randomly generated of integer ranks for a class of combinatorial objects into random objects, that are used in generation of instances used in software testing, Monte Carlo algorithms etc. They can also be applied for the generation of ordered sequences of combinatorial objects, in adaptive parallel generation algorithms, and in task partitioning in optimization algorithms constructed on the basis of exhaustive search. Another application of ranking and unranking algorithms is enumerative coding in which selected classes of equiprobable messages are translated into classes of combinatorial objects, providing unique object's indexing within the class. The assigned index and the code of the enumerative class represent the compressed data for the message [27]. Unranking algorithms can also be used for enumeration [3], crossover operations on chromosomes in genetic algorithms [28], metaheuristics etc.

Partitions are widely used to solve optimization problems in bioinformatics, forensic science, and scheduling [7]. In computational molecular biology partitions play an important role in understanding the role of genes in determining global characteristics of species. In a multi-state distribution system, the overall quality of service can be maintained by quick enumeration partitions of the variables used in decision diagrams that model the system [3].

The first known algorithm for generating $(n, m)$-partitions, $1 \leq m \leq n$, i.e partitions of $n$-element set into at most $m$ nonempty blocks is due to Hutchinson [8]. Over the following years, a number of sequential algorithms was developed [4, 6, 11, 22]. An increasing interest in parallel computing also resulted in the development of many parallel algorithms for the partition generation problem in various models of computations [5, 16, 19, 21, 25]. The structure of the set of partitions was investigated and new ranking/unranking techniques were developed satisfying some particular requirements [15, 29]. A fast hardware rank to partition converter has been proposed in [3].

Parallelization of the generation algorithm on the set of objects level is possible by means of an adaptive scheme. The linearly ordered sequence of objects is divided into $k$ subsequences of equal size, where $k$ is the number of available processors. An unranking algorithm is used for determining the first object of each subsequence.

In the present paper, we propose a new parallel algorithm for unranking set partitions. The dynamic programming technique is used which was successfully applied in many application areas. So far a dynamic programming paradigm was used in a number of sequential algorithms for unranking combinations [17], partitions [15, 29], t-ary trees [20, 22] and some other combinatorial objects [9, 10]. Although, in general, unranking problems are inherently sequential, a portion of computations can be parallelized. However, many sequential algorithms are not suitable for parallelization. Until now, parallel algorithms were proposed for unranking combinations and t-ary trees [18, 20, 22].

The rest of the paper is organized as follows. The next section introduces a representation of set partitions. Then, a sequential algorithm for unranking of set partitions in Hutchinson's representation is presented in section 3. Section 4 describes an associative algorithm developed on the basis of a parallel dynamic programming method.

## 2. Representation of set partitions

Let us introduce the basic notions used throughout this paper.

Let $\langle A_i \rangle_{i \in I}$ denote an *indexed family of sets* $A_i = A$, where: $A = \{1, ..., m\}$, $I = \{1, ..., n\}$, $1 \le m, n$. Any mapping $f$, which 'chooses' one element from each set $A_1, ..., A_n$ is called a *choice function* of the family $\langle A_i \rangle_{i \in I}$ [23]. With additional restrictions, we can model by choice functions various classes of combinatorial objects [9].

Below, we define choice functions ρ corresponding to partition codewords known from literature [8, 29] (see Table 1).

If additional conditions: 1. $a_1 = 1$ and 2. $a_i \in \{1, ..., \max[a_1, ..., a_{i-1}] + 1\}$, for $2 \le i \le n$, and $i \in I$, are satisfied, then any choice function $\rho = \langle a_i \rangle_{i \in I}$, that belongs to the indexed family $\langle A_i \rangle_{i \in I}$, is called the *partitioning choice function* of this family ($r$-sequence). Set of all partitioning choice functions contains representations of all $m$-block partitions of the set $A$. In Hutchinson's representation of partitions we deal in fact with indexed sets $R_i = \{1, ..., i\} \subseteq A_i$.

Table 1

**Sequences of all ($m$, $n$)-partitions ($1 \le m \le n = 4$, B($n$) = 15)**

| Rank | $r$-sequence | Rank | $r$-sequence | Rank | $r$-sequence |
|---|---|---|---|---|---|
| 1 | 1 1 1 1 | 6 | 1 2 1 1 | 11 | 1 2 2 3 |
| 2 | 1 1 1 2 | 7 | 1 2 1 2 | 12 | 1 2 3 1 |
| 3 | 1 1 2 1 | 8 | 1 2 1 3 | 13 | 1 2 3 2 |
| 4 | 1 1 2 2 | 9 | 1 2 2 1 | 14 | 1 2 3 3 |
| 5 | 1 1 2 3 | 10 | 1 2 2 2 | 15 | 1 2 3 4 |

The number of different $m$-block partitions of $n$-element set for $m \ge 2$ is called the Stirling number of the second kind:

$$S(n, m) = S(n-1, m-1) + mS(n-1, m), \quad \text{for} \quad 0 < m < n \tag{1}$$

where $S(n, n) = 1$, for $n \ge 0$ and $S(n, 0) = 0$, for $n > 0$. The above formula describes construction of the Stirling triangle.

Let us recall the concept of Williamson numbers [15, 29]. The number of different at most $m$-block partitions of the $n$-element set $\rho = \langle r_1, ..., r_{n-\nu}, r_{n-\nu+1}, ..., r_n \rangle \in \langle R_i \rangle_{i \in I}$ with constant $\langle r_1, ..., r_{n-\nu} \rangle$ and $\max\{r_1, ..., r_{n-\nu}\} = \mu$ is called Williamson number:

$$W_n^m(\nu, \mu) = \mu W_n^m(\nu - 1, \mu) + W_n^m(\nu - 1, \mu + 1) \tag{2}$$

where $\mu, \nu \ge 1$, $W_n^m(1, \mu) = 1$, for $1 \le \mu \le m$, and $W_n^m(1, \mu) = 0$ for $\mu > m$.

The above recursive formula describes the construction of the Williamson triangle for a given value of $m$. In particular, when $n = \nu$ and $\mu = 1$, the following equation holds:

$$W_n^m(n,1) = \sum_{l=0}^{m} S(n,l) \tag{3}$$

Table WT containing a part of the Williamson triangle, for $n = m = 6$, is shown in Table 2. Williamson numbers $W_6^6(\nu, \mu)$ are stored in elements WT[ν, μ] of the table WT. If $n = m$, then the number of all partitions of $n$-element set with at most $n$ blocks is called Bell number $B_n$:

$$B_n = W_n^n(n,1) = \sum_{l=0}^{m} S(n,l) \tag{4}$$

The above formulas, (1)–(4) come from the literature. The terms 'Williamson number' and 'Williamson triangle' were first proposed by the author in [15]. Two-dimentional tables containing parts of Williamson triangle are used in dynamic programming unranking algorithms for set partitions.

Table 2

**Construction of the table WT for ($m$, $n$)-partitions**
**($1 \le m \le n = 6$, $B(n) = 203$)**

| $j$ = | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $i$ = | | | | | | |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 3 | 4 | 5 | 6 | |
| 3 | 5 | 10 | 17 | 26 | | |
| 4 | 15 | 37 | 77 | | | |
| 5 | 52 | 151 | | | | |
| 6 | 203 | | | | | |

Let us now introduce the lexicographic order of the set of all choice functions of the family $\langle A_i \rangle_{i \in I}$.

For the given choice functions $\rho = \langle d_1, ..., d_k \rangle$ and $\gamma = \langle g_1, ..., g_k \rangle$, we say that δ is less then γ according to the *increasing lexicographic order*, if and only if there exists $i \in \{1, ..., k\}$, satisfying $d_i < g_i$, and $d_j = g_j$, for every $j < i$.

For given choice functions $\delta = \langle d_1, ..., d_k \rangle$ and $\gamma = \langle g_1, ..., g_k \rangle$, we say that δ is less than γ according to the *decreasing lexicographic order*, if and only if there exists $i \in \{1, ..., k\}$ satisfying $d_i > g_i$ and $d_j = g_j$, for every $j < i$.

## 3. Sequential dynamic programming algorithm

We will start from the construction of the sequential algorithm given in [15], which is a modification of the Williamson's algorithm [29]. In the algorithm UnrankPart, the table WT is used, which includes a part of a modified Williamson Triangle (see Table 2).

In the unranking algorithm given below, the table WT is already given. The following restrictions on pairs $(i, j)$, $k \le n$ should be taken into account during construction of this table (2 and 3 optionally):

1) $n \le n_{max}$, where: $n_{max}$ is any natural number (size of the triangle),
2) $1 \le k \le k_{max} \le n_{max}$ (columns from 1 to $k_{max}$),
3) $0 \le n - k = r \le n_{max} - k_{min}$ (rows from 1 to $r + 2$).

### Algorithm UnrankPart

*Input:* $n$ – number power of the set, Index – rank of the choice function ρ representing set partition ($1 \le \text{Index} \le B(n)$, Table WT with elements WT[$i, j$] containing Williamson numbers $W_n^n(i, j)$, for $1 \le j \le n$.
*Output:* Table R with choice function ρ.
*Method:* Computations proceed with partitions ranks in increasing lexicographic order. In each step 6.1, the element mWT[$i, j$] with maximum $m$ satisfying the given inequality is selected and next value R[$n - i + 1$] is obtained in step 6.1.2. After $n$ iterations we obtain the partition ρ with given index.

```
1-5 initialization phase
1. Index=Index–1;
2. for i=1 to n do R[i]=1;
3. i=n–1;
4. j=1;
5. m=j;
6. while Index > 0 do
   6.1. if mWT[i, j] ≤ Index
            then
            6.1.1. Index=Index–mWT[i, j];
            6.1.2. R[n–i+1]=R[n–i+1]+m;
            6.1.3. if j < R[n–i+1] then j=R[n–i+1];
            6.1.4. m=j;
            6.1.5. i=i–1;
            else
      6.1.6. m=m–1;
7. return R.
```

*Example 1*

For the input data given below compute table R using the algorithm UnrankPart.

*Input:*

n=4, Index(R)=10.

*Solution:*

Index=9, R=[1,1,1,1], i=3, j=1, m=1.
Index=9 > 0, 1*WT[3,1]=5 ≤ 9, Index=4, R[2]=2, j=2, m=2, i=2.
Index=4 > 0, 2*WT[2,2]=6 > 4, m=1.
Index=4 > 0, 1*WT[2,2]=3 ≤ 4, Index=1, R[3]=2, j=2, m=2, i=1.
Index=1 > 0, 2*WT[1,2]=2 > 1, m=1.
Index=1 > 0, 1*WT[1,2]=1 ≤ 1, Index=0, R[4]=2, j=2, m=2, i=0.

*Output*: R=[1,2,2,2].

*Theorem 1*

Algorithm UrankPart is correct and its asymptotic computational complexity is $O(n^2)$.

*Proof*

The set of all $B(n)$ partitions can be depicted in the form of a rooted ordered tree of height $n$ ($n$ + 1, when root is included). The structure of this tree reflects the recursive structure of the partition set (see Fig. 1). Each node of the tree is labeled by a Williamson number that is equal to the number of leaves of the subtree rooted in this node (labels 1 for leaves are omitted in Fig. 1). Nodes with isomorphic subtrees are labeled with the same Williamson numbers.

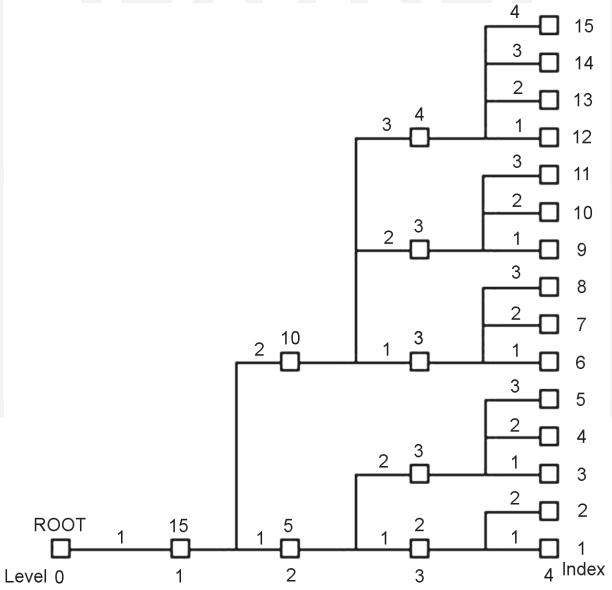


Fig. 1. The rooted ordered tree of all $B(4) = 15$ partitions with edge and node labels

Edges connecting ancestors with their descendants are also labeled. The single edge coming from the root is labeled 1. The number of descendants of a given node exceeds by one the highest edge label on the path leading from the root to this node. Edges connecting a parental node with its children receive successive integers as their labels. A sequence of edge labels on a path leading from the root to a leave, represents a set partition. Traversing the tree in

preorder and listing all paths from the root to subsequent leaves – by sequences of edge labels – is equivalent to generation (enumeration) of all $B(n)$ partitions in increasing lexicographic order. Let us assign to all such paths their ranks in lexicographic order. Unranking the object with the given rank is equivalent to finding in the tree the path corresponding to *Index*, $1 \leq Index \leq B(n)$. We determine the path with rank *Index* by examining sizes of ordered subtrees on the consequtive levels starting from the root. In order to do this, the current relative *Index* of the choice function ρ is compared with $mW_n^n(i,j)$, where $W_n^n(i,j)$ are node labels and $1 \leq m \leq j$. Maximum $m$ satisfying inequality in step 6.1 of the algorithm UnrankPart is determined, Index is updated and the next item of the required object is obtained. In each level $i$, no more than $j$ comparisons are made. Single iteration with complexity $O(n)$ is repeated $O(n)$ times. Hence, the total complexity of the algorithm is $O(n^2)$.

## 4. Parallel dynamic programming algorithm

The original contribution of the paper is parallelization of the sequential algorithm. In the first phase, the table WT is precomputed by systolic computations according to formula (2). Thus, the table WT contains coefficients used in our parallel dynamic algorithm suitable for a hardware acceleration in an associative computational structure.

A simple parallel unranking algorithm implementing associative memory search operations (*no greater then* and *maximum*) may be sketched as follows:

### Algorithm UnrankPart-Par

*Input:*
$n$ – number power of the set, Index – rank of the choice function ρ representing set partition ($1 \leq$ Index $\leq B(n)$, Table WT with elements WT[$i$, $j$] containing Williamson numbers $W_n^n(i,j)$, for $1 \leq j \leq n$.
*Output*: Table $R$ with choice function ρ.
*Method*: Computations proceed with partitions ranks in increasing lexicographic order. In order to determine elements of $R$, an associative search is used. In each step 5.1 all elements WT[$i$, $m$], $1 \leq m \leq j$, satisfying the given inequality are selected. Then, element with maximum $m$ coordinate is selected and next value $R[n–i+1]$ is obtained in step 5.4. After $n$ iterations we obtain the partition ρ with given index.

*1-4 initialization phase*
1. Index = Index–1;
2. R[1]=1;
3. i=n–1;
4. j=1;
5. **while** Index > 0 **do**
   5.1. **search in parallel for** all m=j **downto** 0: mWT[i, j] ≤ Index;
   5.2. **select maximum** m;
   5.3. Index = Index–mWT[i, j];
   5.4. R[n–i+1]=R[n–i+1]+m+1;

5.5. **if** j < R[n–i+1] **then** j=R[n–i+1];
5.6. i=i–1;
6. **return** R.

*Example 2*

For the input data given below compute table R using the algorithm UnrankPart-Par.

*Input:*

n=4, Index(R)=10.

*Solution:*

Index=9, R[1]=1, i=3, j=1.
Index=9 > 0, 1*WT[3,1]=5 ≤ 9, m=1, Index=4, R[2]=2, j=2, i=2.
Index=4 > 0, 2*WT[2,2]=6 > 4, 1*WT[2,2]=3 ≤4, m=1, Index=1, R[3]=2, j=2, i=1.
Index=1 > 0, 2*WT[1,2]=2 > 1, 1*WT[1,2]=1 ≤1, m=1, Index=0, R[4]=2, j=2, i=0.

*Output:* R=[1,2,2,2].

*Theorem 2*

Algorithm UnrankPart-Par is correct and its asymptotic computational complexity is $O(n)$.

*Proof*

The unranking algorithm is a parallel version of the algorithm UnrankPart. Correctness of the method results from the proof of Theorem 1. Parallel search in step 5.1 is organized in an associative manner. Value Index, which is a pattern register is simultaneously compared with all values $mW_n^n(i,j)$, $0 \le m \le j$. This reduces the search time to $O(1)$. The maximum value m in step 5.2 is computed associatively in $O(1)$ time. Then, the next value R[n–i+1] is determined. Each iteration in step 5 has the time complexity $O(1)$. Hence, the total complexity of the algorithm is $O(n)$.

## 5. Concluding remarks

The presented algorithm UnrankPart-Par is the next in the line of unranking algorithms for classes combinatorial objects, developed by the author on the basis of parallel dynamic scheme and oriented on acceleration of the required computations by fast associative operations. It can be implemented in all cases when the time of unranking of $(n, m)$-partitions is of critical importance. Both systolic and associative components of the parallel computations are scalable. The $O(n)$ parallel implementation improves the computational complexity of the sequential unranking algorithm for set partitions UnrankPart by factor n. The original contribution of the present paper are also Theorems 1 and 2 with proofs of algorithms' finiteness and exactness.

## References


[1] Akl S.G., *Parallel computation: models and methods*, Prentice Hall, 1997, 475-509.

[2] Akl S.G., Stojmenovič I., *Generating combinatorial objects on a linear array of processors*, [in:] Zomaya A.Y. (editor): *Parallel Computing: Paradigms and Applications*, Int. Thompson Comp. Press, 1996, 639-670.

[3] Butler J.T., Sasao T., *Hardware index to set partition converter*, Proc. ARC'2013, Lecture Notes in Computer Science, Vol. 7806, 2013, 72-83.

[4] Djokič B. et al., *A fast iterative algorithm for generating set partitions*, The Computer Journal, Vol. 32, 1989, 281-282.

[5] Djokič B. et al., *Parallel algorithms for generating subset and set partitions*, Proc SIGAL Int. Symposium on Algorithms, Tokyo 1990.

[6] Er M.C., *A fast algorithm for generating set partitions*, The Computer Journal, Vol. 31, 1988, 283-284.

[7] Hankin R.K.S., West L.J., *Set partitions*, J. of Statistical Software, Vol. 23, Code Snippet 2 (December 2007), available at: http://www.jstatsoft.org/

[8] Hutchinson G., *Partitioning algorithms for finite sets*, Comm. ACM, Vol. 6, 1963, 613-614.

[9] Kapralski A., *New methods for the generation of permutations, combinations and other combinatorial objects in parallel*, Journal of Parallel and Distributed Computing, Vol. 17, 1993, 315-326.

[10] Kapralski A., *Modelling arbitrary sets of combinatorial objects and their sequential and parallel generation*, monograph, Studia Informatica, Vol. 21, No. 2 (40), 2000.

[11] Kaye R., *A Gray code for set partitions*, Information Processing Letters, Vol. 5, 1976, 171-173.

[12] Knuth D.E., *The art of computer programming. Fundamental algorithms*, Addison-Wesley, 3rd edition, 1997.

[13] Knuth D.E., *The art of computer programming*, Pre-fascicle 3B. *Generating all partitions*, Addison-Wesley, 2004.

[14] Kokosiński Z., *On generation of permutations through decomposition of symmetric groups into cosets*, BIT, Vol. 30, 1990, 583-591.

[15] Kokosiński Z., *Circuits generating combinatorial objects for sequential and parallel computer systems*, Monografia 160, Politechnika Krakowska, Kraków 1993.

[16] Kokosiński Z., *Mask and pattern generation for associative supercom-puting*, Proc. 12th Int. Conf. Applied Informatics AI'94, Annecy 1994, 324-326.

[17] Kokosiński Z., *Algorithms for unranking combinations and their applica-tion*, Proc. 7th Int. Conf. Parallel and Distributed Computing and Systems PDCS'95, Washington D.C., USA, 1995, 216-224.

[18] Kokosiński Z., *Unranking combinations in parallel*, Proc. Int. Conf. Parallel and Distributed Processing Techniques and Applications PDPTA'96, Sunnyvale, CA, USA, Vol. I, 1996, 79-82.

[19] Kokosiński Z., *On parallel generation of set partitions in associative processor architectures*, Proc. Int. Conf. Parallel and Distributed Processing Techniques and Applications PDPTA'99, Las Vegas, USA, Vol. III, 1999, 1257-1262.

[20] Kokosiński Z., *A parallel dynamic programming algorithm for unranking t-ary trees*, Proc. PPAM'2003, Lecture Notes in Computer Science, Vol. 3019, 2004, 255-260.

[21] Kokosiński Z., *A new algorithm for generation of exactly m-block set partitions in associative model*, Proc. PPAM'2005, Lecture Notes in Computer Science, Vol. 3911, 2006, 67-74.
[22] Kokosiński Z., *Parallel enumeration of t-ary trees in ASC SIMD model*, Int. J. Computer Science and Network Security, Vol. 11, No. 12, 2011, 38-49.
[23] Mirsky L., *Transversal theory*, Academic Press, 1971.
[24] Semba I., *An efficient algorithm for generating all partitions of the set {1, 2, ..., n}*, Journal of Information Processing, Vol. 7, 1984, 41-42.
[25] Stojmenovič I., *An optimal algorithm for generating equivalence relations on a linear array of processors*, BIT, Vol. 30, 1990, 424-436.
[26] Stojmenovič I., *On random and adaptive parallel generation of combinato-rial objects*, Int. Journal of Computer Mathematics, Vol. 42, 1992, 125-135.
[27] Tomic R.V., *Quantized indexing: background information*, Technical Report TR05-0625, 1st Works Corporation, June 25, 2005, 39.
[28] Üçoluk G., *A method for chromosome handling of r-permutations of n-element set in genetic algorithms*, Proc. 4th Int. Conf. on Evolutionary Compu-ting ICEC'97, Indianapolis, USA, 55-58, 1997.
[29] Williamson S.G., *Ranking algorithms for lists of partitions*, SIAM J. of Computing, Vol. 5, 1976, 602-617.