Neural, Parallel, and Scientific Computations, **26**, No. 3 (2018), 345-353 ISSN: 1061-5369

SPEEDING UP THE GENERATION OF NEAR-RINGS ON FINITE CYCLIC GROUPS USING PARALLEL PROCESSING IN C#

MARIA MALINOVA[1], ANGEL GOLEV[2], AND ASEN RAHNEV[3]

[1,2,3]Faculty of Mathematics and Informatics
Paisii Hilendarski University of Plovdiv
Plovdiv, BULGARIA

ABSTRACT: The main purpose of this article is to reduce the generation time of near-rings on finite cyclic groups using the parallel processing provided by `C#`. We divide the near-rings into independent subsets and generate them simultaneously. The calculations are further accelerated after refactoring and defining appropriate compiler optimizations.

AMS Subject Classification: 16Y30
Key Words: near-rings, generating near-rings, parallel programming

Received: August 21, 2018; **Accepted:** November 13, 2018;
Published: November 27, 2018. **doi:** 10.12732/npsc.v26i3.9
Dynamic Publishers, Inc., Acad. Publishers, Ltd. https://acadsol.eu/npsc

1. INTRODUCTION

An algebraic system $(G, +, *)$ is a *(left) near-ring* on $(G, +)$, if $(G, +)$ is a group, $(G, *)$ is a semigroup and $a * (b + c) = a * b + a * c$ for $a, b, c \in G$. The left distributive law yields $x * 0 = 0$ for $x \in G$. A near-ring $(G, +, *)$ is called *zero-symmetric*, if $0 * x = 0$ holds for $x \in G$.

J. R. Clay initiated the study of near-rings, whose additive groups are finite cyclic in 1964 [1]. Some sufficient conditions for the construction of near-rings on any finite cyclic groups were obtained. In [2] Jacobson determined the entire structure and number of the left near-rings on a group of prime order. In 1968 all near-rings on cyclic groups of order up to 7 were computed [3]. Later all near-rings on cyclic groups of order 8 [4, 5], of order up to 12 [6], of order up to 13 [7], of order up to 15 [9] as well as of order up to 24 [10, 11] and up to 29 [14], 32 [15], 35 [16] were computed.

We will assume G coincides with the set $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$, $2 \leq n < \infty$ since every cyclic group of order n is isomorphic to the group of the remainders of modulo n. We will denote the functions mapping $\mathbb{Z}_n$ into itself by π, and the addition and the multiplication modulo n we will denote by $+$ and $\cdot$ respectively. The equality $c = a \cdot b$ will be equivalent to the congruence $ab \equiv c \pmod{n}$.

It is known that there exists a bijective correspondence between the left distributive binary operations $*$ defined on $\mathbb{Z}_n$ and the n^n functions π mapping $\mathbb{Z}_n$ into itself. If $r * 1 = b$ defines the function $\pi(r) = b$, then according to [1, Theorem II], the binary operation $*$ is left distributive exactly when, for any $x, y \in \mathbb{Z}_n$, the equality

$$\pi(x) \cdot \pi(y) = \pi(x \cdot \pi(y)) \tag{1}$$

holds.

According to the above result, obtaining the near-rings on $\mathbb{Z}_n$ is equivalent to obtaining the functions π such that equation (1) holds. We represent a near-ring with a list of the values of function $\pi(r)$, $r = 0, \ldots, n-1$.

2. DIVIDING THE NEAR-RINGS INTO SEVERAL INDEPENDENT GROUPS

The goal of this article is to reduce the generation time of near-rings on finite cyclic groups by using parallel processing in `C#` and especially `Parallel.For`.

The program module is developed as a part of the system for generating and researching near-rings [12, 16, 17].

We want to speed up the process of generating near-rings [13] with minimal changes to the core functionality of our software.

We divide the set of near-rings into subsets, which can be generated independently of one another. We use Parallel.For to start multiple instances of our generation algorithm and calculate the subsets simultaneously.

We divide the near-rings into at least 6 different subsets, for which the π function has the following preset values:

subset 1: $\pi(0) = 0, \pi(1) = 0, \pi(2) = 0$
subset 2: $\pi(0) = 0, \pi(1) = 0, \pi(2) \geq 1$
subset 3: $\pi(0) = 0, \pi(1) = 1, \pi(2) = 0$
subset 4: $\pi(0) = 0, \pi(1) = 1, \pi(2) \geq 1$
subset 5: $\pi(0) = 0, \pi(1) \geq 2$
subset 6: $\pi(0) \geq 1$

For near-rings over a $\mathbb{Z}_n$ with only the trivial idempotents 0 and 1, subset 6 contains only one near-ring.

	Zero-symmetric	Non-zero-symmetric	Total number
$\mathbb{Z}_3$	6	1	7
$\mathbb{Z}_4$	16	1	17
$\mathbb{Z}_5$	28	1	29
$\mathbb{Z}_6$	65	33	98
$\mathbb{Z}_7$	111	1	112
$\mathbb{Z}_8$	349	1	350
$\mathbb{Z}_9$	1 169	1	1 170
$\mathbb{Z}_{10}$	807	393	1 200
$\mathbb{Z}_{11}$	1 311	1	1 312
$\mathbb{Z}_{12}$	4 467	1 055	5 522
$\mathbb{Z}_{13}$	5 263	1	5 264
$\mathbb{Z}_{14}$	10 505	5 256	15 761
$\mathbb{Z}_{15}$	21 783	6 215	27 998
$\mathbb{Z}_{16}$	16 834 653	1	16 834 654
$\mathbb{Z}_{17}$	72 816	1	72 817
$\mathbb{Z}_{18}$	15 032 215	610 684	15 642 899
$\mathbb{Z}_{19}$	286 380	1	286 381
$\mathbb{Z}_{20}$	876 919	109 847	986 766
$\mathbb{Z}_{21}$	1 164 023	304 834	1 468 857
$\mathbb{Z}_{22}$	2 225 545	1 111 088	3 336 633
$\mathbb{Z}_{23}$	4 371 615	1	4 371 616
$\mathbb{Z}_{24}$	15 821 973	2 619 758	18 441 731
$\mathbb{Z}_{25}$	95 367 449 527 555	1	95 367 449 527 556
$\mathbb{Z}_{26}$	34 749 177	17 400 576	52 149 753
$\mathbb{Z}_{27}$	286 174 087 734	1	286 174 087 735
$\mathbb{Z}_{28}$	207 919 830	19 570 310	227 490 140
$\mathbb{Z}_{29}$	273 300 895	1	273 300 896
$\mathbb{Z}_{30}$	552 602 256	461 986 240	1 014 588 496
$\mathbb{Z}_{31}$	1 089 204 381	1	1 089 204 382
$\mathbb{Z}_{32}$	72 651 402 778 958 352	1	72 651 402 778 958 353
$\mathbb{Z}_{33}$	4 364 742 735	1 092 510 166	5 457 252 901
$\mathbb{Z}_{34}$	8 677 365 263	4 338 542 561	13 015 907 824
$\mathbb{Z}_{35}$	17 373 338 997	1 362 452 660	18 735 791 657

Table 1: Number of near-rings on $\mathbb{Z}_n$, $3 \leq n \leq 35$

For a $\mathbb{Z}_n$ with non-trivial idempotents, there exist non-zero-symmetric near-rings, for which the value of $\pi(0)$ is equal to the non-trivial idempotent. In that case we divide subset 6 into additional subsets - one for each non-trivial idempotent in $\mathbb{Z}_n$ [8].

Shown below is the number of near-rings in the independent subsets that get constructed over $\mathbb{Z}_n, 16 \leq n \leq 24$. The execution time provided for some n does not include saving the generated near-rings into a file or database.

Near-rings on $\mathbb{Z}_{16}$

$\pi(0,0,0,...)$:	1060763
$\pi(0,0,\hat{1},...)$:	3158642
$\pi(0,1,0,...)$:	9710
$\pi(0,1,\hat{1},...)$:	10263
$\pi(0,\hat{2},...)$:	12595275
$\pi(\hat{1},...)$:	1
All n-rs:	16834654

Near-rings on $\mathbb{Z}_{17}$

$\pi(0,0,0,...)$:	17144
$\pi(0,0,\hat{1},...)$:	17942
$\pi(0,1,0,...)$:	17138
$\pi(0,1,\hat{1},...)$:	17952
$\pi(0,\hat{2},...)$:	2640
$\pi(\hat{1},...)$:	1
All n-rs:	72817

Near-rings on $\mathbb{Z}_{18}$

$\pi(0,0,0,...)$:	1628601
$\pi(0,0,\hat{1},...)$:	3225457
$\pi(0,1,0,...)$:	35213
$\pi(0,1,\hat{1},...)$:	37264
$\pi(0,\hat{2},...)$:	10105680
$\pi(\hat{1},...)$:	610684
All n-rs:	15642899

Near-rings on $\mathbb{Z}_{19}$

$\pi(0,0,0,...)$:	67983
$\pi(0,0,\hat{1},...)$:	70729
$\pi(0,1,0,...)$:	67987
$\pi(0,1,\hat{1},...)$:	70730
$\pi(0,\hat{2},...)$:	8951
$\pi(\hat{1},...)$:	1
All n-rs:	286381

Near-rings on $\mathbb{Z}_{20}$

$\pi(0,0,0,...)$:	202136
$\pi(0,0,\hat{1},...)$:	209430
$\pi(0,1,0,...)$:	137077
$\pi(0,1,\hat{1},...)$:	145465
$\pi(0,\hat{2},...)$:	182811
$\pi(\hat{1},...)$:	109847
All n-rs:	986766
Calc.time:	1 sec.

Near-rings on $\mathbb{Z}_{21}$

$\pi(0,0,0,...)$:	272334
$\pi(0,0,\hat{1},...)$:	284679
$\pi(0,1,0,...)$:	273049
$\pi(0,1,\hat{1},...)$:	289987
$\pi(0,\hat{2},...)$:	43974
$\pi(\hat{1},...)$:	304834
All n-rs:	1468857
Calc.time:	2 sec.

Near-rings on $\mathbb{Z}_{22}$

$\pi(0,0,0,...)$:	537665
$\pi(0,0,\hat{1},...)$:	551683
$\pi(0,1,0,...)$:	538583
$\pi(0,1,\hat{1},...)$:	552062
$\pi(0,\hat{2},...)$:	45552
$\pi(\hat{1},...)$:	1111088
All n-rs:	3336633
Calc.time:	4 sec.

Near-rings on $\mathbb{Z}_{23}$

$\pi(0,0,0,...)$:	1068269
$\pi(0,0,\hat{1},...)$:	1087942
$\pi(0,1,0,...)$:	1068259
$\pi(0,1,\hat{1},...)$:	1087955
$\pi(0,\hat{2},...)$:	59190
$\pi(\hat{1},...)$:	1
All n-rs:	4371616
Calc.time:	5 sec.

Near-rings on $\mathbb{Z}_{24}$

$\pi(0,0,0,...)$:	3209493
$\pi(0,0,\hat{1},...)$:	3409926
$\pi(0,1,0,...)$:	2181369
$\pi(0,1,\hat{1},...)$:	2266558
$\pi(0,\hat{2},...)$:	4754627
$\pi(\hat{1},...)$:	2619758
All n-rs:	18441731
Calc.time:	16 sec.

The number of near-rings in the subsets is proportional. There are exceptions for $\mathbb{Z}_n$ with non-trivial idempotents or nilpotents of order 2.

For a $\mathbb{Z}_n$ with non-zero nilpotents and $n > 16$, there is a big group of previously described near-rings, which are skipped during generation – for n=25, n=8, n=32 these near-rings are so many, that they cannot be generated in real-time. The structure of these near-rings is described in [11, Theorem 9] and in [15] for n=32.

Due to the non-zero nilpotents of order 2 over $\mathbb{Z}_{16}$, $\mathbb{Z}_{18}$, $\mathbb{Z}_{20}$, $\mathbb{Z}_{24}$, the number of near-rings in their subsets is also not proportional.

If the contents of these big subsets are ignored, the number of near-rings is at least doubled with each subsequent n. This also holds true for the execution time.

3. IMPLEMENTATION OF THE PARALLEL PROCESSING IN C#

Our goal is to be able to compute different subsets simultaneously. To avoid multithreading problems, every near-rings subset has its own copy of the main data structures we use.

During its generation, every subset needs 4 one-dimensional arrays to hold the values of π and some additional data. This means that in our software we have 4 jagged arrays (C# arrays of arrays) - `pi`, `pi_2`, `pi_n`, `pi_ptr`, with each near-rings subset using one row from each jagged array. The initialization is shown below.

```
for (int k = 0; k < subsetsnum; k++)
{
    pi[k] = new int[n];
    pi_2[k] = new int[n];
    pi_n[k] = new int[n];
    pi_ptr[k] = new List<int>[n];
    for (int i = 0; i < n; i++) pi_ptr[k][i] = new List<int>(n+2);
}
```

Why we use simple arrays as the best solution for storing this data is discussed in the next section of the article.

We change the main function for generating near-rings so that it accepts as input parameters the starting and ending values of the π function. These values uniquely define the subsets described in the previous section.

```
private static void MakeNearRings(int[] pi, int[] pi_2, int[] pi_n,
     List<int>[] pi_ptr)
```

After setting the initial values of the π function and some other parameters, we start the parallel calls to the main generator function.

```
Parallel.For(0, subsetsnum, k =>
     {MakeNearRings(pi[k], pi_2[k], pi_n[k], pi_ptr[k])});
```

If enough processing cores are available, the calculation of all subsets can happen simultaneously and then the execution time for the program depends only on the time needed to calculate the largest subset of near-rings.

Shown below is the initialization of some structures and variables used in the program and the calls to the function for generating near-rings. Of particular interest is the specific way in which we utilize `Parallel.For`.

```
mod_n = new int[n * n];
nilp = new int[n];
idemp = new int[n];
nr_result = new StringBuilder[pn];

Parallel.For(0, n * n, index => { mod_n[index] = index % n; });
```

```
CalcNilpotents(n, nilp, out nilp_n);
CalcIdempotents(n, idemp, out idemp_n);

subsetnum = 4 + idemp_n;

InitPi(0, new int[]{0,0,0}, new int[]{0,0,1});
InitPi(1, new int[]{0,0,1}, new int[]{0,1,0});
InitPi(2, new int[]{0,1,0}, new int[]{0,1,1});
InitPi(3, new int[]{0,1,1}, new int[]{0,2});
InitPi(4, new int[]{0,2}, new int[]{1});
InitPi(5, new int[] { 1 }, new int[]{2});

if (subsetnum > 6)
{
   for (int id = 2; id < idemp_n; id++)
   {
      InitPi(4 + id, new int[] { idemp[id] },
         new int[]{id < idemp_n-1 ? idemp[id+1] : n});
   }
}

Parallel.For(0, subsetnum, k =>
{
   MakeNearRings(pi[k], pi_2[k], pi_n[k], pi_ptr[k]);
});
```

4. CONCLUSIONS REACHED DURING THE OPTIMIZATION PROCESS

The algorithm described in this paper utilizes arrays for storing intermediate data – this is the best approach for our use case, because there is nothing more than direct memory access involved in the reading and writing of intermediate results. If there are multiple threads accessing the same jagged array at the same time, they are always doing their work onto different rows – there is no fighting for resources or locking involved.

The C# language has two types of multi-dimensional arrays – rectangular and jagged (an array of arrays). We use the second, because we are able to pass a reference to a single row to the near-rings generator code – rectangular arrays don't have this functionality.

We tested separating the jagged arrays into single rows, organized within an object instance of a helper class. This made the code more readable, but decreased the speed of generation due to the overhead of using objects for millions of read/write operations.

Finally, we experimented with moving the data structures onto the stack. Overall, this did not affect our execution time, mostly because our data structures are small and get cached anyway. We utilized the new Span$\langle T\rangle$ functionality of C# from mid 2018 (Span$\langle T\rangle$ on the stack), but as mentioned this did not produce any noticeable speed gains.

5. ADDITIONAL TECHNIQUES FOR SPEEDING UP THE GENERATION OF NEAR-RINGS

We made additional optimizations to our project, which sped up the generation 4 times.

5.1. PLATFORM AND COMPILER OPTIMIZATIONS

The Microsoft platform .NET Core, which shares most of its API with the .NET Framework, is used for creating applications where the goal is speed and scalability. Moving the code base onto .NET Core halved the execution time for any given n.

Most functions were also annotated with AggresiveInlining flags which, in combination with compiler flags for code optimization, additionally reduced the execution time.

5.2. CODE ANALYSIS USING VISUAL STUDIO TOOLS

The Visual Studio tools for analyzing running code gave us insight into bottlenecks in the code base. With some refactoring of heavy code blocks, we were able to achieve a 50% increase in speed over the above mentioned results from switching to .NET Core.

6. CONCLUSION

We have created a module for generating near-rings on finite cyclic groups, using the parallel processing provided by `C#`. The near-rings on $\mathbb{Z}_n$ are divided into at least 6 independent subsets and then multiple instances of the generation algorithm are started. If enough processor cores are available the generation may be speed up around 5 times. Further optimizations were applied during refactoring, code analysis and porting the code to a newer framework. The module is part of a system for generating and researching near-rings.

ACKNOWLEDGEMENTS

This work was partially supported by the project FP17-FMI-008 of the Scientific Fund of the University of Plovdiv Paisii Hilendarski, Bulgaria.

REFERENCES

[1] Clay J. R., The near-rings on a finite cycle group, *Amer. Math. Monthly*, **71** (1964), 47–50.

[2] Jacobson R. A., The structure of near-rings on a group of prime order, *Amer. Math. Monthly*, **73** (1966), 59–61.

[3] Clay J. R., The near-rings on groups of low order, *Math. Zeitschr.*, **104** (1968), 364–371.

[4] Pilz G., *Near-rings*, North-Holland, Amst., **23** (1977).

[5] Pilz G., *Near-rings*, North-Holland, Amst., Revised edition, **23** (1983).

[6] Yerby R., H. Heatherly, Near-Ring Newsletter, **7** (1984), 14–22.

[7] Rakhnev A. K., G. A. Daskalov, Construction of near-rings on finite cyclic groups, *Math. and Math. Education*, Sunny Beach, Bulgaria, (1985), 280–288.

[8] Rakhnev A. K., On near-rings, whose additive groups are finite cyclics, *Compt. rend. Acad. bulg. Sci.*, **39** No. 5 (1986), 13–14.

[9] Aichinger E., F. Binder, J. Ecker, R. Eggetsberger, P. Mayr and C. Nöbauer. SONATA: Systems Of Nearrings And Their Applications, Package for the group theory system GAP4. Johanes Kepler University Linz, Austria, (2008). http://www.algebra.uni-linz.ac.at/sonata/

[10] Rahnev A., A. Golev, Some New Lower Bounds for the Number of Near-rings on Finite Cyclic Groups, *Int. Journal of Pure and Applied Mathematics*, **59**, No.1 (2010), 59–75.

[11] Rahnev A. K., A. A. Golev, Computing Near-rings on Finite Cyclic Groups, *Compt. rend. Acad. bulg. Sci.*, **63**, book 5 (2010), 645–650.

[12] Golev A., A. Rahnev, Computing Classes of Isomorphic Near-rings on Cyclic Groups of Order up to 23, *Scientific Works, Plovdiv University*, **37**, book 3, Mathematics (2010), 53–66.

[13] Golev A., Algorithms for Generating Near-rings on Finite Cyclic Groups, *Proceedings of the Anniversary International Conference REMIA 2010, Plovdiv*, 255–262, 10-12 December 2010.

[14] Golev A. A., A. K. Rahnev, Computing Near-rings on Finite Cyclic Groups of Order up to 29, *Compt. rend. Acad. bulg. Sci.*, **64**, No. 4, (2011), 461–468.

[15] Golev A., A. Rahnev, New results for near-rings on finite cyclic groups, *Proceedings of Annual Workshop "Coding Theory and Applications"*, 51–54, Gabrovo, December 2011.

[16] Pavlov N., A. Golev, A. Rahnev, Distributed Software system for Testing Near-RIngs Hypotheses and New Constructions for Near-Rings on Finite Cyclic Groups, *Int. Journal of Pure and Applied Mathematics*, **90**, No.3 (2014), 345–356.

[17] Malinova M., A. Golev, A. Rahnev, Generating SQL Queries for Filtering Near-Rings on Finite Cyclic Groups, *Int. Journal of Pure and Applied Mathematics*, **119**, No.1 (2018), 225–234.