Tools for computer investigations of unit groups of modular group algebras

Alexander Konovalov

Centre for Interdisciplinary Research in Computational Algebra
University of St Andrews, Scotland

Arithmetic of Group Rings and Related Objects, RWTH, Aachen, March 22-26, 2010

Outline

- This talk is not about torsion units in integral group rings
- This talk is not about the verification of the modular isomorphism problem for groups of order 512
- Instead of this, the talk is about some recent and less advertised tools for computations in modular group algebras of finite p-groups

Introduction

- $F = \mathbb{F}_p$ – field of p elements
- G – finite p-group
- FG – its integral group ring
- $U(FG)$ – unit group of FG
- $V(FG)$ – subgroup of normalized units

Problem

How to perform efficient computations in $V(FG)$?

- To be able to use fast algorithms, we need to compute its power-commutator presentation:
  - Generators $y_1, \ldots, y_{|G|-1}$
  - Relations $y_i^p = (y_{i+1})^{\alpha_{i,i+1}} \cdots (y_{|G|-1})^{\alpha_{i,|G|-1}}$ for $1 \leq i \leq |G| - 1$
  - Relations $(y_j, y_i) = (y_{j+1})^{\alpha_{j,i,j+1}} \cdots (y_{|G|-1})^{\alpha_{j,i,|G|-1}}$ for $1 \leq i < j \leq |G| - 1$
  - Exponents $\alpha_{i,k}$ and $\alpha_{i,j,k}$ are elements of the set $\{0, \ldots, p-1\}$

Generators for $V(FG)$

- In our case $V(FG)$ has a natural polycyclic generating set $1 + S$, where S is a *weighted basis* of the augmentation ideal of FG (A. Bovdi, 1997):
  - $G = G_1, G_{i+1} = [G_i, G]G_{j^p}$ where j is the smallest non-negative integer such that $j \geq i/p$ is called the *Jennings series* of G
  - $x_{1,1}, \ldots, x_{1,l_1}, \ldots, x_{k,1}, \ldots, x_{k,l_k}$ is a *dimension basis* for G, that is, $\{x_{i,1}G_{i+1}, \ldots, x_{i,l_i}G_{i+1}\}$ is a minimal generating set for the elementary abelian group G_i/G_{i+1}
  - the weighted basis consists of $|G| - 1$ *standard products* $(x_{1,1} - 1)^{\alpha_{1,1}} \cdots (x_{1,l_1} - 1)^{\alpha_{1,l_1}} \cdots (x_{k,1} - 1)^{\alpha_{k,1}} \cdots (x_{k,l_k} - 1)^{\alpha_{k,l_k}}$, $0 \leq \alpha_{i,j} \leq p - 1$
- Thus, we may quickly compute $1 + S$ as explicitly written elements of a group algebra
- But then we need to compute the canonical form of every power and commutator relation with respect to this polycyclic generating set, what involves extensive computations in a group algebra

GAP package LAGUNA

- The algorithm to compute $V(FG)$ for a finite p-group G and a field of p elements F has been implemented in the LAGUNA package for the computational algebra system GAP (since its 2003 release)
- LAGUNA = Lie AlGebras and UNit groups of group Algebras
- The package was started by R. Rossmanith (Jena) under the name LAG for GAP 3.4.4 to compute Lie algebras of group algebras, and later it was taken over and extended (including unit groups functionality) by by V. Bovdi, AK and Cs. Schneider
- LAGUNA can compute $V(KG)$ in two representations: *natural* (slow, operates with explicitly written group algebra elements), and *abstract* (fast, operates with elements of a pc-group)
- Using the bijection between these two representations, one can compute the centre of the pc-group and then find its generators as group algebra elements

Verifying Coleman's counterexample

Coleman (1992) discovered a counterexample to the conjecture that the nilpotency class of $V(KG)$ is equal to the nilpotency class of the wreath product $C_p \wr G'$. We can verify it using the LAGUNA package:

```
gap> G := SmallGroup( 32, 6 );
<pc group of size 32 with 5 generators>
gap> D := DerivedSubgroup(G);
Group([ f3, f5 ])
gap> W := WreathProduct( CyclicGroup(2), D );
<group of size 64 with 3 generators>
gap> NilpotencyClassOfGroup(W);
3
gap> FG := GroupRing( GF(2) , G );
<algebra-with-one over GF(2), with 5 generators>
gap> V := PcNormalizedUnitGroup( FG );
<pc group of size 2147483648 with 31 generators>
gap> NilpotencyClassOfGroup( V );
4
```

Isomorphism problem of unit groups

- **MIP**: $FG \cong FH \Rightarrow G \cong H$?
- **UMIP**: $V(FG) \cong V(FH) \Rightarrow G \cong H$?
- A. Bovdi and Zs. Balogh solved UMIP for 2-groups of maximal class and p-groups with cyclic $\Phi(G)$ for $p > 2$
- Using LAGUNA package, it was verified (AK & A. Krivokhata) that unit groups of modular group algebras of groups of order 32 over the field of two elements are not isomorphic pairwise
- The hardest pair was (32,13) and (32,14). For both groups, $Aut(V(KG))$ has order 2^{149}, but its minimal number of generators is 15 and 16 respectively (determined using the AutPGrp package by B. Eick and E. O'Brien)

Sample runtime to compute $V(FG)$ in pc-presentation

IdGroup	Structure description	Runtime (compute)
$8, 3$	D_8	20 ms
$16, 7$	D_{16}	200 ms
$32, 18$	D_{32}	2.5 s
$64, 52$	D_{64}	1 min
$128, 161$	D_{128}	27 min
$256, 539$	D_{256}	16 hrs 20 min
$27, 3$	$(C_3 \mathrm{x} C_3) : C_3$	1.4 s
$81, 8$	$(C_9 \mathrm{x} C_3) : C_3$	2 min 40 s
$243, 26$	$(C_9 \mathrm{x} C_9) : C_3$	6 hrs 20 min

Measured in GAP 4.4.12 on 8-core Dell Poweredge 2950: 2 quad-core Intel Xeon 2.66 GHz / RAM 16 GB / CentOS Linux 4.5

GAP package UnitLib

- It is worth to wait longer to compute the pc-group isomorphic to $V(FG)$ since it can be operated faster afterwards
- Why not to save it to avoid its repeated computations?
- This idea was implemented in the UnitLib package (2006, AK & E. Yakimenko), which provides the library of normalised unit groups for groups from the GAP Small Groups Library
- Current version covers all p-groups of order not greater than 243
- 21.5 MB of local storage plus 32 MB online data for order 243
- Development addressed many technical issues arising while saving and retrieving the data (storing CodePcGroup in hexa-decimal format, storing DimensionBasis(G) to avoid further interfering with random methods, gzipping large files, providing online access, proper embedding of G into the recovered pc-group, etc.)

UnitLib example

In this example, we show a group which has different sets of bicyclic units of 1st and 2nd kind that generate different subgroups of $V(FG)$:

```
gap> V := PcNormalizedUnitGroupSmallGroup(64,9);
<pc group of size 92233720368547758O8 with 63 generators>
gap> V1 := BicyclicUnitGroupType(V,1); #auxiliary function
<pc group with 154 generators>
gap> V2 := BicyclicUnitGroupType(V,2); #its code not shown
<pc group with 154 generators>
gap> Size(V1);Size(V2);
268435456
268435456
gap> Size(Intersection(V1,V2));
134217728
gap> Size(ClosureGroup(V1,V2));
536870912
gap> StructureDescription(SmallGroup(64,9));
"(C2 x Q8) : C4"
```

Sample runtime to retrieve $V(FG)$ in pc-presentation

IdGroup	Structure description	Runtime (compute)	Runtime (retrieve)	Speedup	Data file (bytes)
$8,3$	D_8	20 ms	3 ms	6.7	78
$16,7$	D_{16}	200 ms	6 ms	33.3	225
$32,18$	D_{32}	2.5 s	26 ms	96.2	1518
$64,52$	D_{64}	1 m	273 ms	219.8	12958
$128,161$	D_{128}	27 m	7 s	192.9	110737
$256,539$	D_{256}	16 h 18 m	9 m 3 s	102.5	925019
$27,3$	$(C_3 \mathrm{x} C_3) : C_3$	1.4 s	13 ms	107.7	793
$81,8$	$(C_9 \mathrm{x} C_3) : C_3$	2 m 40 s	673 ms	237.7	27865
$243,26$	$(C_9 \mathrm{x} C_9) : C_3$	6 h 20 m	4 m 15 s	89.4	895391

Measured in GAP 4.4.12 on 8-core Dell Poweredge 2950: 2 quad-core Intel Xeon 2.66 GHz / RAM 16 GB / CentOS Linux 4.5

Parallelising $V(FG)$

- To save the group $V(FG)$ in the library first we need to compute it, and larger orders require a substantial runtime to do this
- The GAP package SCSCP (2008, AK & S. Linton) package implements an SCSCP protocol (Symbolic Computation Software Composability Protocol) – a remote procedure call framework that allows to connect multiple instances of the same of different computer algebra systems and exchange data between them using the OpenMath format
- One of applications built on top of SCSCP is the Master-Worker skeleton that allows to run parallel computations with one master distributing individual tasks to multiple (local or remote) workers
- Parallelising an own code, the user should make a decision about remote procedures and data exchanged, keeping in mind optimal workload distribution and efficient data marshalling

Parallelising $V(FG)$

- We compute in parallel all power and commutator relations
- Relying on the fact that identical group algebras can be restored from the GAP Small Groups Library on every worker
- The result is returned in a form of GAP strings containing words in polycyclic generators
- Still there is sequential phase at the end to combine all collected data and form the resulting pc-group
- The rest of technicalities about the design of remote procedures is out of the scope of this talk

Realtime to compute $V(FG)$ in pc-presentation

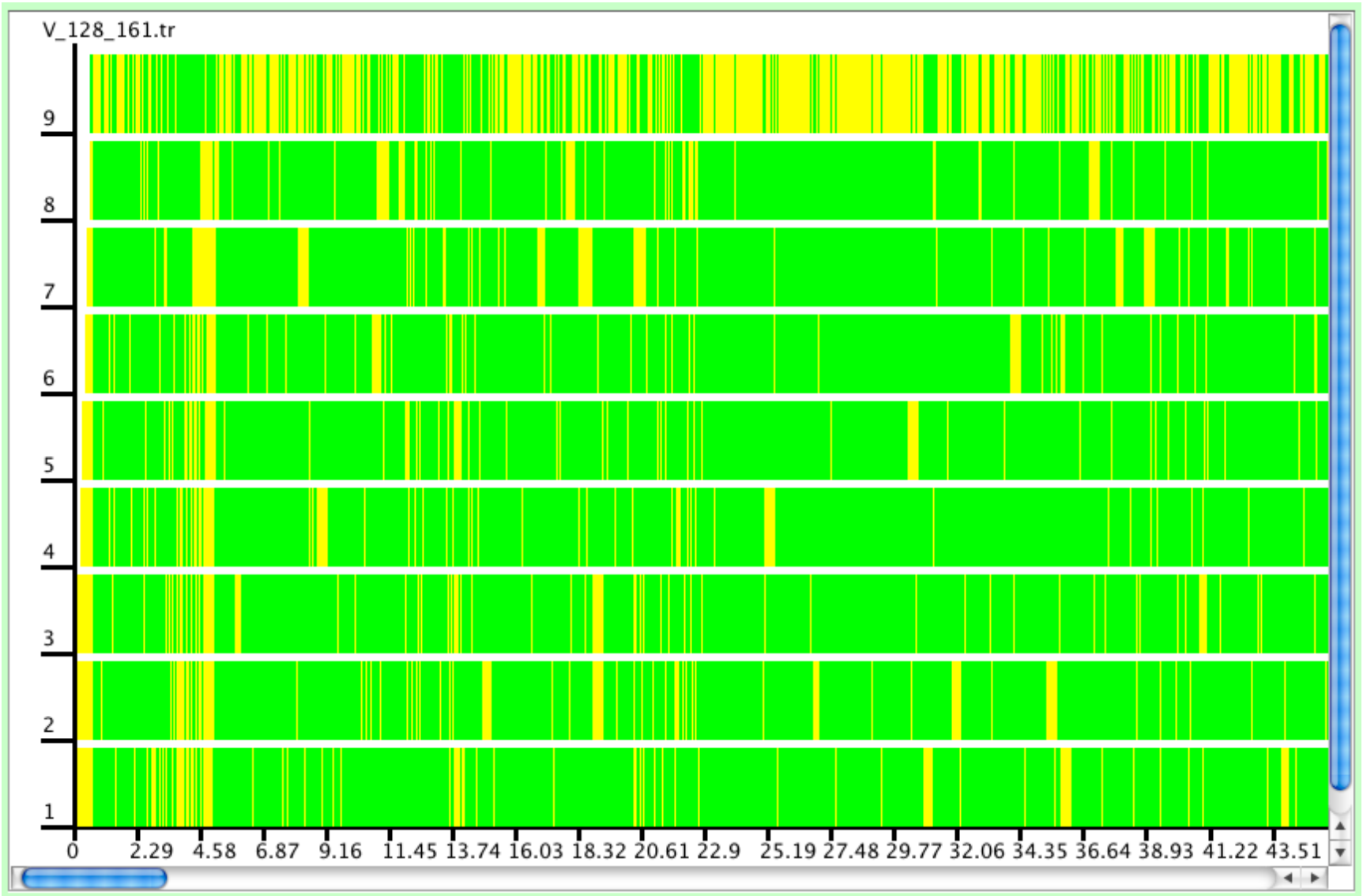
IdGroup	Structure description	Runtime (sequential)	Runtime (parallel)	Ratio seq / par
$8, 3$	D_8	20 ms	0.4 s	0.05
$16, 7$	D_{16}	200 ms	1.5 s	0.13
$32, 18$	D_{32}	2.5 s	8.8 s	0.28
$64, 52$	D_{64}	1 min	42 s	1.43
$128, 161$	D_{128}	27 min	6 min 25 s	4.21
$256, 539$	D_{256}	16 hrs 20 min	2 hr 54 min	5.63
$27, 3$	$(C_3 \mathrm{x} C_3) : C_3$	1.4 s	7.1 s	0.20
$81, 8$	$(C_9 \mathrm{x} C_3) : C_3$	2 min 40 s	1 min 11 s	2.25
$243, 26$	$(C_9 \mathrm{x} C_9) : C_3$	6 hrs 20 min	1 hr 13 min	5.21

Measured in GAP 4.4.12 on 8-core Dell Poweredge 2950: 2 quad-core Intel Xeon 2.66 GHz / RAM 16 GB / CentOS Linux 4.5

$V(KG)$ for D_{128}=SmallGroup(128,161) – overview

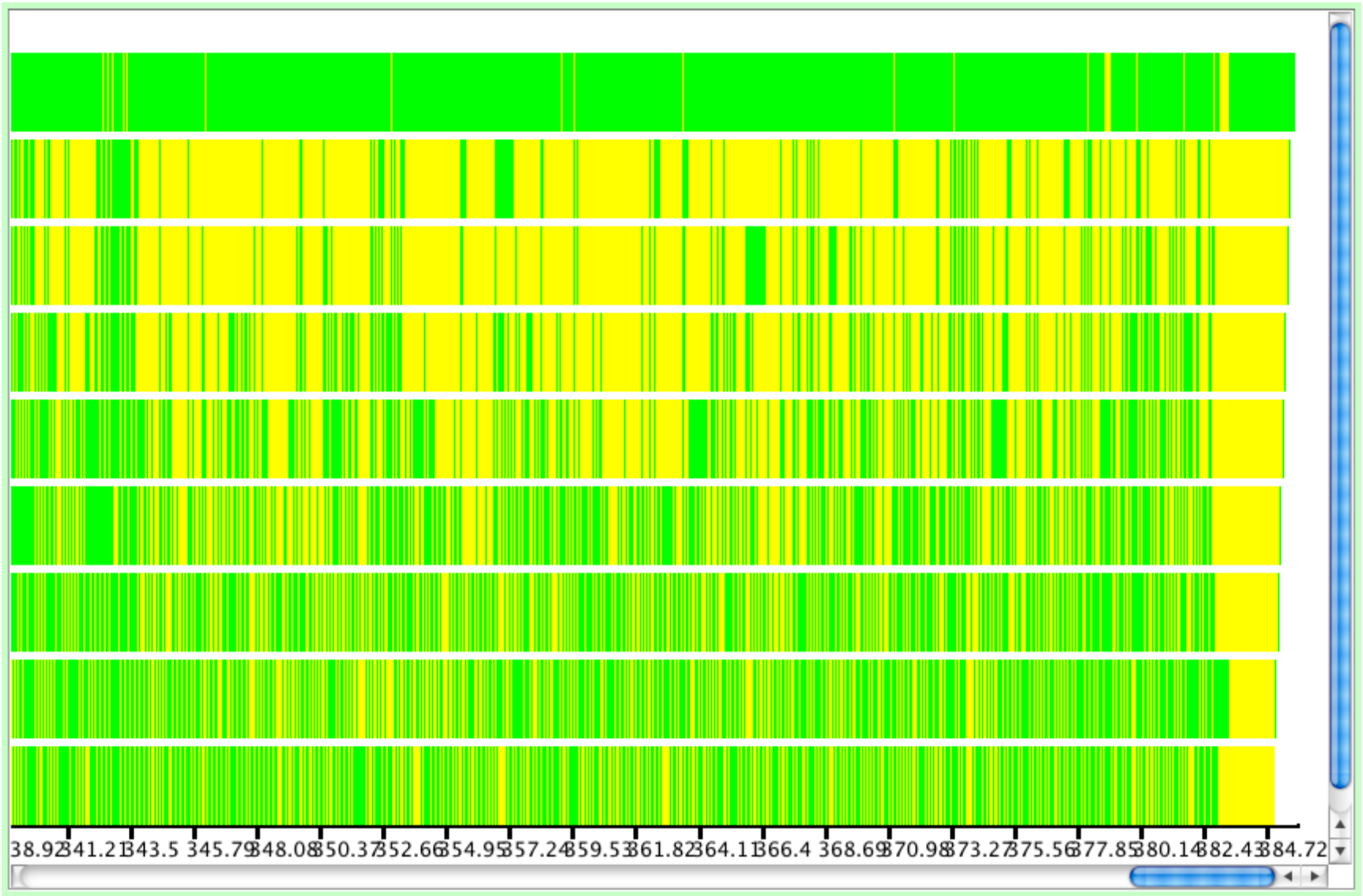
Measured in GAP 4.4.12 on 8-core Dell Poweredge 2950: 2 quad-core Intel Xeon 2.66 GHz / RAM 16 GB / CentOS Linux 4.5

$V(KG)$ for D_{128}=SmallGroup(128,161) – start



Measured in GAP 4.4.12 on 8-core Dell Poweredge 2950: 2 quad-core Intel Xeon 2.66 GHz / RAM 16 GB / CentOS Linux 4.5

$V(KG)$ for D_{128}=SmallGroup(128,161) – end



Measured in GAP 4.4.12 on 8-core Dell Poweredge 2950: 2 quad-core Intel Xeon 2.66 GHz / RAM 16 GB / CentOS Linux 4.5

Comments to the reader

- The 1st diagram shows the whole computation
  - row 9 (top) is master, rows 1-8 are workers
  - green = busy, yellow = idle
  - ideally, master $\longrightarrow$ yellow, worker $\longrightarrow$ green
  - however, this is not always the case because of irregularity
- The 2nd diagram shows more detailed view of the initial segment
  - clearly seen the border between computing powers and commutators at 4.58 seconds
  - in the beginning, workers are kept busy sufficiently well
- The 3rd diagram shows more detailed view of the final segment
  - as you go down the powers of augmentation ideal, individual tasks become shorter and the load on master increases, while some workers are being idle more often
  - clearly seen the sequential phase at the end

Some facts for order 729

- G = SmallGroup(729,13) = $(C_3 \mathrm{x} (C_{27} : C_3)) : C_3$
  - It took 3 days to compute $V(FG)$ in GAP 4.4.12 on 8-core Dell Poweredge 2950: 2 quad-core Intel Xeon 2.66 GHz / RAM 16 GB / CentOS Linux 4.5) using 1 master and 8 workers (including 5 hours for the final sequential stage to construct the resulting PcGroup)
    - It may take even longer for other groups
    - but we can add more workers on different computers
    - and do, for example, the complete cycle (computation and saving the result) for the same G = SmallGroup(729,13) less than in 20 hours (three 8-core Intel servers, with dual quad-core Intel Xeon 5570 2.93GHz / RAM 48 GB RAM / CentOS Linux 5.3)

Further work

- Complete the library for all 504 groups of order 729
- Coming soon GAP 4.5 release with GMP support:
  $\longrightarrow$ faster integer arithmetic
  $\longrightarrow$ faster work with CodePcGroup
- For larger groups, it might be faster/shorter to store the presentation itself rather then CodePcGroup
- With future computers, this can make it possible to compute the library for the order 256 (56092 groups)
- What about orders 5^5 (77-easy), 5^6 (684-?), 2^9 (10494213-???)
- But even now one can compute such groups on demand and save their description for further usage
- Group ring community is highly encouraged to use these tools, and your feedback is most welcome

THANK YOU !!!