Parallel Construction of Finite Solvable Groups*

Anton Betten

Department of Mathematics
University of Bayreuth
Germany

Abstract. An algorithm for the construction of finite solvable groups of small order is given. A parallelized version under PVM is presented. Different models for parallelization are discussed.

1 Group Extensions

A finite group G is called *solvable*, if there exists a chain of normal subgroups $1 = G_0 < G_1 < \ldots < G_s = G$ such that each group G_i is normal in its succesor G_{i+1} and that the index $[G_{i+1} : G_i]$ is prime for $i = 0, \ldots, s-1$.

Consider the simplest situation of a *group extension* G with a normal subgroup N of prime index $p = [G : N]$ (compare Huppert I, 14.8 [3]). Take an arbitrary $g \in G \backslash N$. Then $G = \langle N, g \rangle$ and the factor group G/N consists of the cosets $N, Ng, Ng^2, \ldots, Ng^{p-1}$. Each coset Ng^i with $i \not\equiv 0 \bmod p$ generates the factor group. Because of $(Ng)^p = Ng^p = N$ one gets $g^p = h \in N$. As N is normal in G, $g^{-1}ng = n^g$ is an element of N for any $n \in N$. Conjugation with the fixed element $g \in G$ defines an automorphism of N since $(n_1 n_2)^g = g^{-1} n_2 g g^{-1} n_2 g = n_2^g n_2^g$. This *inner automorphism* of G considered as an automorphism of N is not generally an inner automorphism of N. Define $\alpha_g : N \to N, n \mapsto n^g$, the associated automorphism. What is known about α_g?

i) $g^p = h \in N \cap \langle g \rangle$, a cyclic and therefore abelian subgroup, so $h^g = h^{\alpha_g} = h$.
ii) $g^p = h$ implies for any $n \in N$: $n^{g^p} = n^{\alpha_g^p} = n^h$, so that $\alpha_g^p = \mathrm{inn}_h$, where inn_h is the inner automorphism of N induced by conjugation with h.

On the other hand, one easily verifies that for any group N and any pair of elements $h \in N$ and $\alpha \in \mathrm{Aut}(N)$ there exists a group G of order $p \cdot |N|$, if

i) $h^\alpha = h$ and
ii) $\alpha^p = \mathrm{inn}_h$.

One obtains the group extension by introducing an element g with $g^p := h$ and $g^{-1}ng := n^\alpha$. $\langle N, g \rangle$ defines a group G of order $p \cdot |N|$ and, by definition, N is normal in G.

Elements $h \in N$ and $\alpha \in \mathrm{Aut}(N)$ where N is a fixed group are called *admissible*, if the conditions i) and ii) of above are fullfilled. Let now h and α

* This research was supported by Procope

be an arbitrary admissible pair for prime p and group N. Define the group extension $\mathrm{Ext}(N, p, h, \alpha)$ to be the group generated by the elements of N and an element g with $g^p := h$ and $n^g := n^\alpha$ for each $n \in N$. Define the set of all possible group extensions $\mathrm{Ext}(N, p)$ to be $\{\mathrm{Ext}(N, p, h, \alpha) \mid h \in N, \alpha \in \mathrm{Aut}(N), h \text{ and } \alpha \text{ admissible for } N \text{ and } p\}$. If $\mathcal{G}$ is a set of groups one defines $\mathrm{Ext}(\mathcal{G}, p) := \cup_{G \in \mathcal{G}} \mathrm{Ext}(G, p)$.

Clearly, the construction of such group extensions needs a good knowledge of the automorphism group.

Because subgroups of solvable groups are solvable too, it is possible to construct solvable groups by iteration of the procedure just indicated: Assume one wants to find all solvable groups of a given order n. Because the order of subgroups always divides the order n, the lattice of divisors of n comes into play. By induction, one can assume that all strict subgroups have already been determined. Then, one has to construct all group extensions for all possible subgroups of prime index (in the group of order n). Therefore, consider the prime factorization of n. Any rearrangement of the primes might possibly occur as a sequence of orders of factor groups in the normal chain of a solvable group of order n. So, one starts at the bottom (the trivial group) constructing all groups of prime order. Then one goes up and determines all groups with order a product of two primes, three primes and so on. Note that the lattice of subgroups of a group of order n can be divided into layers according to the number of primes of the corresponding group orders (counting multiplicities).

Denoting the set of solvable groups of order n by $\mathcal{AG}_n$ (German: “auflösbare Gruppe”) one has

$$\mathcal{AG}_n = \bigcup_{p \mid n, p \,\mathrm{prime}} \mathrm{Ext}(\mathcal{AG}_{n/p}, p). \tag{1}$$

In the prime power case this reduces to the formula $\mathcal{AG}_{p^k} = \mathrm{Ext}(\mathcal{AG}_{p^{k-1}}, p)$. Because groups of prime power order are solvable, any group of prime power order will be obtained by this way.

In the following we will specialize our notation to the case of solvable groups $\mathcal{AG}_n$. Let p and q be primes dividing n. Then it might happen that extensions $G_1 \in \mathrm{Ext}(\mathcal{AG}_{n/p}, p)$ and $G_2 \in \mathrm{Ext}(\mathcal{AG}_{n/q}, q)$ define isomorphic groups. So, the task is to *find* all the extensions for a given order and afterwards to *reduce* the set of groups up to isomorphism.

As example, consider the two groups of order 4 and their extensions to groups of order 8:

$4\#1 \simeq Z_2 \times Z_2$:

$$A^2 = id$$
$$B^2 = id,\ A^B = B^{-1}AB = A$$

id	A	B	BA
A	id	BA	B
B	BA	id	A
BA	B	A	id

$4\#2 \simeq Z_4$:

$$A^2 = id$$
$$B^2 = A,\ A^B = B^{-1}AB = A$$

id	A	B	BA
A	id	BA	B
B	BA	A	id
BA	B	id	A

Next, we show the groups together with their subgroup lattice (where only selected cover-relations are drawn). In the text below one can find generators and relations, the column to the right gives a label for the group, the isoclinism class, the order of the first central factor group, the order of the derived subgroup, the order of the automorphism group, their factorization, the Sylow type and the number of conjugacy classes of subgroups with respect to conjugation by the full automorphism group (first line), by the group of only inner automorphisms (second line) and by the trivial group (third line). So, this line also gives the number of groups in any layer of the lattice. To the right of the closing braces the sums of the entries are given. Here is not enough space to explain this in details, the interested reader may have a look at:

`http://btm2xd.mat.uni-bayreuth.de/home/research.html`

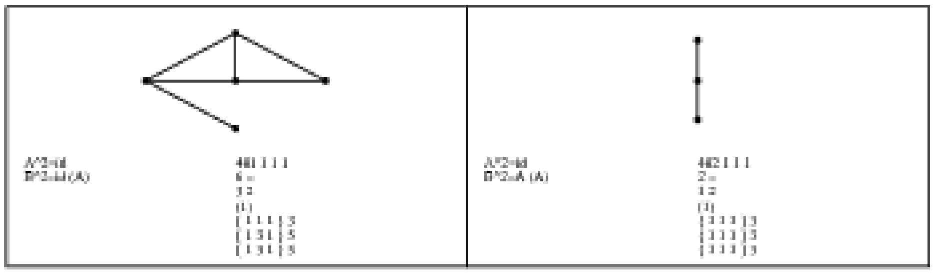

Fig. 1. The two groups of order 4

To define group extensions, it is necessary to study the automorphism groups of 4#1 and 4#2. The group $Z_2 \times Z_2$ admits any permutation of its non-trivial elements as an automorphism. So, $\mathrm{Aut}(4\#1) \simeq S_3$. In the other case, there is only one non-trivial automorphism, namely the map $B \mapsto BA = B^{-1}$ (mapping A onto itself). In this case $\mathrm{Aut}(Z_4) \simeq Z_2$.

In the following, we often substitute the elements of the groups by their lexicographic numbers, always counting from 0 on for convenience. Thus we have $0 = id$, $1 = A$, $2 = B$, $3 = BA$ (because $B^2 = id$). As permutation groups, $\mathrm{Aut}(4\#1) = \langle (1\,2), (2\,3) \rangle$ and $\mathrm{Aut}(4\#2) = \langle (2\,3) \rangle$.

In order to compute admissible pairs $\alpha \in \mathrm{Aut}(N)$ and $h \in N$ (where $N \in \{4\#1, 4\#2\}$) we define the *extension matrix* of a group N:

$$\mathrm{E}(N,p) = (e_{\alpha,h})_{\alpha \in \mathrm{Aut}(N),\, h \in N} = \begin{cases} 1 \Leftrightarrow h^\alpha = h \wedge \alpha^p = \mathrm{inn}_h \\ 0 \quad \text{otherwise} \end{cases}, \tag{2}$$

i.e. $e_{\alpha,h}$ is 1 iff (α, h) is an admissible pair (for N and p). The elements of $\mathrm{Aut}(4\#1)$ (left column) and $\mathrm{Aut}(4\#2)$ (automorphisms listed by their images on the generators and as permutations of the elements) are:

no.	$\alpha \in \mathrm{Aut}(4\#1)$	ord(α)
0	[1, 2] id	1
1	[1, 3] $(2\,3)$	2
2	[2, 1] $(1\,2)$	2
3	[2, 3] $(1\,2\,3)$	3
4	[3, 1] $(1\,3\,2)$	3
5	[3, 2] $(1\,3)$	2

no.	$\alpha \in \mathrm{Aut}(4\#2)$	ord(α)
0	[1, 2] id	1
1	[1, 3] $(2\,3)$	2

(3)

The extension matrices are:

$$E(4\#1,2) = \begin{pmatrix} X X X X \\ X X\ .\ . \\ X\ .\ .\ X \\ .\ .\ .\ . \\ .\ .\ .\ . \\ X\ .\ X\ . \end{pmatrix} \quad E(4\#2,2) = \begin{pmatrix} X X X X \\ X X\ .\ . \end{pmatrix} \tag{4}$$

So, there are 10 possible extensions of 4#1 and 6 extensions of 4#2. As noted before, one cannot expect 16 different groups of order 8 since some of the candidates may be isomorphic.

By computing Ext(4#1) one gets three non-isomorphic groups: the first is $Z_2 \times Z_2 \times Z_2$ and will be called 8#1. The second is $Z_4 \times Z_2$ (8#2), the third 8#3 is non-abelian: $A^2 = id$, $B^2 = id$, $C^2 = id$ with relations $A^B = A$, $A^C = A$ and $B^C = AB$. This defines a dihedral group and is also an example that computers may give other presentations of a group than one would expect.

In computing Ext(4#2) one obtains the groups 8#2 and 8#3 again. But there are two new groups: 8#4 is cyclic of order 8 and 8#5 is the (non-abelian) quaternionic group: $A^2 = id$, $B^2 = A$, $C^2 = A$, $A^B = A$, $A^C = A$ and $B^C = AB$. It is clear that – for example – one cannot get 8#1 as an extension of $4\#2 \simeq Z_4$: the elementary abelian group has no subgroup isomorphic to Z_4.

Generally, it is desirable to compute a list of groups which is both *complete* and *irredundant* – that is, a representative of each isomorphism type is present and no two groups are of the same isomorphism type. Completeness is guaranteed by the introductory remarks of this section; irredundancy involves solving the *isomorphism problem* for groups. This is yet another story which definitively cannot be solved in this paper. One would have to look at invariants of groups and talk about canonical forms to solve the isomorphism problem.

The amount of computational effort needed for constructing all groups of given order mainly depends on the number of candidates of groups to be tested. There are methods which reduce the number of candidates drastically. For instance when considering conjugacy classes in Aut(G) it is easily seen that it suffices to compute extensions using just a system of representatives of the classes. Another reduction can be made by letting the centralizer of each fixed automorphism act on the entries 1 in the extension matrix. We do not notice further details. Let us move straightforward to parallelization.

2 Parallelization

We are going to parallelize the computation of Ext($\mathcal{N}, p$). Just before starting, one has to look at the program in order to discover the possible approaches for parallelization. In our case two attempts were made to parallelize at different positions: the crucial point is to know where in the program most of the work is located. One has to recognize subproblems in the algorithm that can be computed widely independent, i.e. without the need of too much communication between the parts. An important concept is the notion of *task granularity* which is the ratio of computation to network activity of a task. This parameter can decide on success or failure of a parallelization in many cases. So, keeping this value in mind is very important.

The serial version of the program acts in the following way (compare figure 2): For any group $N \in \mathcal{N}$ all possible p-extensions are computed by the generator (using reduced extension matrices). These are the candidates which have to be filtered up to isomorphism. An important tool for parallelism is a so called

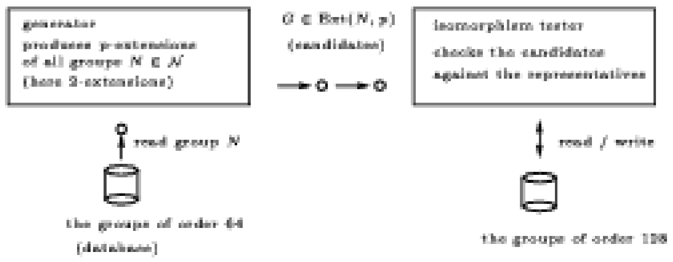

Fig. 2. The serial program

"pool of tasks". Here one collects all the subproblems which shall be processed in parallel. A one-to-many relationship between the involved programs is also included in the model: one certain program acts as a controller and administrates the pool (it is therefore called "master"). He also controls the subordinate part, namely those programs doing the parallel work (therefore called "slaves") (compare figure 3). The "pool of tasks" model can be compared with the situ-

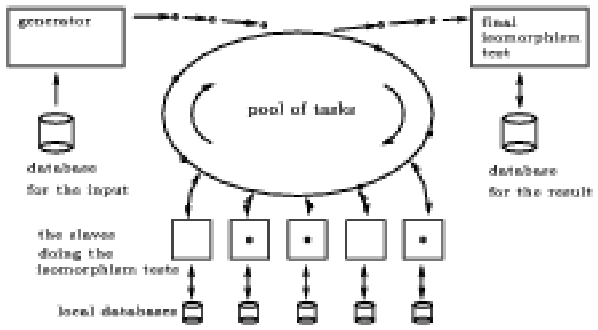

Fig. 3. The pool of tasks model

ation at the airport: The distribution of the luggage to the travellers is solved by circulation; the suitcases (which can be compared with the tasks here) run upon a cyclic band until they are fetched by some traveller. In our language of distributed computing, the task is assigned to a particular slave for processing. But what is a task and what shall the slaves do with it?

There are two approaches. Figure 3 just shows the first one by the labels (the model is independent and will also be used in the second algorithm).

The first way is to parallelize the isomorphism part of the problem, i.e. the right hand side of figure 2. For any newly generated group it has to be tested whether the group is an isomorphic copy of another group already computed or not. This work is done by the slaves, each of them holding a local database of all groups which already have been computed. The newly generated groups are deposited at the pool of tasks where they reside until some slave is idle (and willing to do the job, just to say it in human language). At this moment such a group is transferred to the slave. If it proves to be isomorphic to one of the groups already in the list it can be skipped (this means a message to the master). In the other case, the group comes back to the master (or only a tag which group is meant because the group is already held at the master). The slave is marked to be idle again. It will get another task if there is any. But what about the group sent back to the master with the information: "non-isomorphic to groups $1, \ldots, l$" where l is the number of groups at the slave's list? Now we have a difficulty: In the meantime, the master might have defined new groups and so further tests are necessary. But maybe there will not be too many new groups, so one can do this instantly. One has to be careful at this point to avoid possible bottlenecks in the computation. Experience shows that the ratio of definition of new groups is relatively low (compared to the number of candidates which have to be tested). So, estimatedly, there will not be too many late isomorphism tests needed at the master. Finally, if the group passes these tests too it is in fact new: The master adds it to his own list of groups and distributes it to all isomorphism slaves so that they can complete their lists.

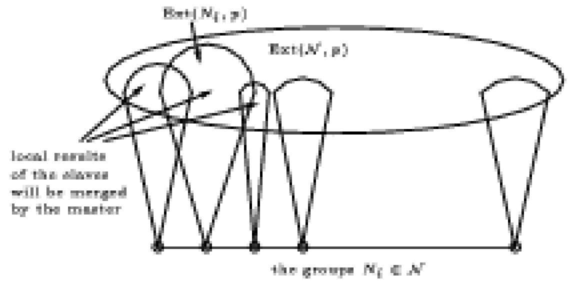
$\mathrm{Ext}(N_i, p)$
$\mathrm{Ext}(\mathcal{N}, p)$
local results of the slaves will be merged by the master
the groups $N_i \in \mathcal{N}$

Fig. 4. The parallel group extension model

A second approach is parallelization according to equation (1) (see figure 4). Again, one uses a pool of tasks for the distribution of the groups $N \in \mathcal{N}$. Each slave gets his own group for calculating $\mathrm{Ext}(N, p)$. The results of the slaves will be sent back to the master. There they will get merged together. One might suppose a parallel merging in form of a binary tree but one has to admit that the results of the slaves will come back in no predictable order: the computation of $\mathrm{Ext}(N)$ may sometimes be very difficult, sometimes very easy (the amount of work depends for instance on the size – or better the number of entries **1** – of the extension matrix). Thus a linear merge was preferred in the actual implementation.

Some remarks should be added concerning the aspect of holding the data in the algorithm. As the number of groups can be quite large for higher n, the amount of storage needed by the slaves should not be underestimated. At the moment (in the current implementation) each slave keeps his own database for ease of access (no collisions during write). One has to admit that the access to common disk space via NFS can cause another bottleneck. Local disks will help here. But since the list of groups is the same for all slaves, an approach of a commonly shared database seems to be a good idea. The *PIOUS* [5] system for parallel file IO could support this aspect of the problem. PIOUS is able to use network wide distributed disk space for its files. The total amount of storage would be widely reduced. Maybe the network traffic increases a little. This question is not yet tested but PIOUS seems to be an interesting approach.

3 Results: Serial vs. Parallel Version

At first, let us try to compare the speed of different architectures (see table 1). This seems to be necessary because results of general benchmarks cannot easily be transferred to the specific program which is run here. For the test, we just run a very small example. Note that this program uses only integers (no floating points) and is very special in its kind because it merely does not "compute" in the narrow sense. It is more a collection of lots of (deeply nested) loops. Moreover, the program does not take advantage of DECs 64 bit processors. This might help to explain why hardware based on Intel Pentium is so well suited for the program (considering also prices !). On each platform, a high grade of optimization was tried (`cxx` supports optimization only up to `-O2`).

machine type	cmplr.	hh:mm:ss	P90 speed
PentiumPro 200 MHz	`g++ -O3`	10:52	299 %
DEC AlphaStation 600, 333 MHz	`cxx -O2`	11:06	293 %
SGI PowerChallenge chip: R10000, 190 MHz	`g++ -O3`	12:21	263 %
Intel Pentium 90 MHz	`g++ -O3`	32:30	100 %
DEC Alpha 3000 / 600, chip 21064, 175 MHz	`cxx -O2`	34:55	93 %
Silicon Graphics Indy	`g++ -O3`	38:29	84 %
DEC Alpha 3000 / 400, chip 21064, 130 MHz	`cxx -O2`	49:49	65 %
Intel 486 DX2/50 MHz VL	`gcc -O3`	1:36:08	34 %

Table 1. The serial version

Testing the PVM version of the program involves some difficulties. It only makes sense to measure the real time of the run, i.e. the life-time of the master. There is no way of measuring "user-time" as it was done in the serial version. Thus, the load of the computer imposed by other users has a negative impact on the evaluation. The values presented here were obtained during a week-end's night with no other processes running on the machines. It is also desirable to test the PVM program on a pool of homogeneous machines so that one can study the effect of succesively increasing the number of processors. The test runs presented here were made on a pool of (equal) SGI workstations at the computing center of Bayreuth (see figure 5).

For optimal behaviour of the parallelized version, the test was limited to the computation of the sets of extensions $\mathrm{Ext}(G,p)$ (which was the primary aim

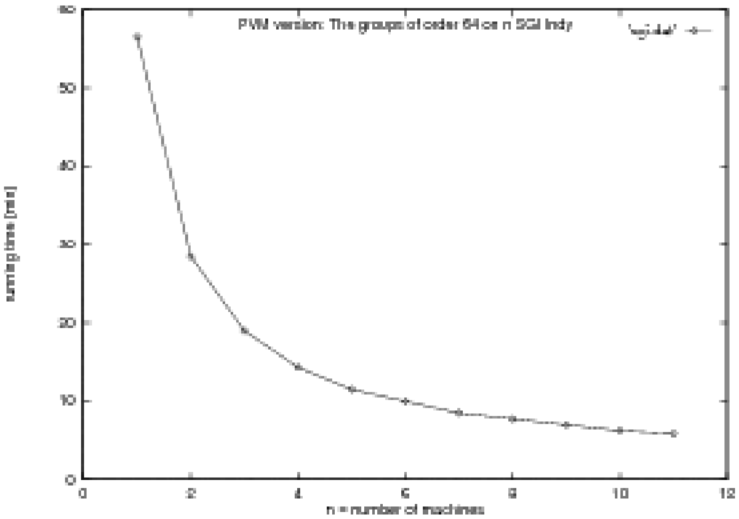

Fig. 5. Testing the parallel version

of the parallelized version of the program); the second step (merging these lists together) was left out for the test. Another possible source of friction comes from task granularity. It has already been discussed that the amount of computation should not be too low compared with the time spent for sending messages and for administrating the distributed system. Here we have the other side of the medal: if one task is computing for a very long time, in the meantime many (or all) other tasks might have terminated. So in this case the computation is unnecessarily lengthened at the end with only few working tasks. The problem chosen for this particular test was the computation of all 267 groups of order 64 realized as extensions of the 51 groups of order 32.

References

1. Bettina Eick: Charakterisierung und Konstruktion von Frattinigruppen mit Anwendung in der Konstruktion endlicher Gruppen. Thesis, RWTH Aachen, 1996.
2. Marshall Hall, James K. Senior: The groups of order 2^n ($n \leq 6$). MacMillan Company, New York, London 1964.
3. Bertram Huppert: Endliche Gruppen I. Springer Verlag, Berlin, Heidelberg, New York, 1967.
4. Reinhard Laue: Zur Konstruktion und Klassifikation endlicher auflösbarer Gruppen. *Bayreuther Math. Schr.* **9** (1982).
5. Steven A. Moyer, V. S. Sunderam: PIOUS for PVM, Version 1.2, User's Guide and Reference Manual. `http://www.mathcs.emory.edu/Research/Pious.html`
6. Joachim Neubüser: Die Untergruppenverbände der Gruppen der Ordnungen ≤ 100 mit Ausnahme der Ordnungen 64 und 96. Thesis, Kiel 1967.
7. Edward Anthony O'Brien: The groups of order 256. *J. Algebra*, **143** (1991), 219-235.

This article was processed using the LaTeX macro package with LLNCS style