Parallel Solution of Covering Problems Super-Linear Speedup on a Small Set of Cores

Bernd Steinbach
Institute of Computer Science
Freiberg University of Mining and Technology
Freiberg, Germany
Email: steinb@informatik.tu-freiberg.de

Christian Posthoff
Department of Mathematics & Computer Science
The University of the West Indies
Trinidad & Tobago
Email: Christian.Posthoff@sta.uwi.edu

***Abstract*—This paper aims at better possibilities to solve problems of exponential complexity. Our special focus is the combination of the computational power of four cores of a standard PC with better approaches in the application domain. As the main example we selected the unate covering problem which must be solved, among others, in the process of circuit synthesis and for graph-covering (domination) problems.**

We introduce into the wide field of problems that can be solved using Boolean models. We explain the models and the classic solutions, and discuss the results of a selected model by using a benchmark set. Subsequently we study sources of parallelism in the application domain and explore improvements given by the parallel utilization of the available four cores of a PC. Starting with a uniform splitting of the problem, we suggest improvements by means of an adaptive division and an intelligent master. Our experimental results confirm that the combination of improvements of the application models and of the algorithmic domain leads to a remarkable speedup and an overall improvement factor of more than 35 millions in comparison with the improved basic approach.

***Keywords*-covering; XBOOLE; ternary vector; parallel; message passing interface; unate SAT problems; Boolean models**

I. Introduction

The recent improvements in computer hardware [3] lead to new challenges for software development. The increased computation power of modern PCs is realized by the multi-core architecture. However, many of the available programs use only a single core.

Distributed parallel computing is known from special large machines or clusters. We focus in this paper on the growing field of multi-core PCs. For our experiments we use a 4-core CPU. In order to point out possible restrictions of the memory we take into account a 32-bit architecture.

The Boolean space of n variables consists of 2^n Boolean vectors. Therefore many Boolean problems [4], [6], [7], [8], [13] have an exponential complexity [16]. Such Boolean problems must be solved, for instance, to develop more powerful prospective computers. Taking into account this particular challenge, we selected one Boolean problem as the main example, the *unate covering problem* [1]. Problems of this type must be solved, for instance, in the area of logic design in order to find minimal circuits. The unate covering problem is a special satisfiability problems (SAT) [14]. The characteristic property of such a problem: none of the variables is negated. In opposition to [1] we compute all exactly minimal solutions of a given unate covering problem.

II. Boolean Modeling

The complete representation of this topic requires a lot of space, therefore we restrict ourselves to some indications. We consider *finite sets* (mostly very large) of objects: $O = \{o_1, \dots, o_n\}$, the objects will be often enumerated by binary coding, and properties will be given by *Boolean functions*.

A. Properties of objects

The presence of a given property for an object o_i will be given by a logic function $\varphi(o_i)$:

$$\varphi(o_i) = \begin{cases} 1 & \textit{if the property is present} \\ 0 & \textit{otherwise} \end{cases}$$

Example. We want to describe properties of a chess board. There are 64 objects, i.e. 64 fields, and it must be chosen between a direct coding by one bit for the fields a1, a2 to h7, h8. This results in 64 bits altogether. The other possibility is a binary coding by six bits, from (000000) to (111111). If, for instance, the function φ is supposed to describe the black fields, then we have $\varphi(a1) = 1$, $\varphi(a2) = 0$ etc. The white fields will be given by $\overline{\varphi}$.

Example. The fields of a Sudoku game can be occupied by numbers $1, \dots, 9$, and the fields themselves can be enumerated according to 9 columns and 9 rows. This can be described and computationally handled by variables

$$x_{ijk} = \begin{cases} 1 & \textit{if the the value on the field } (i,j) = k \\ 0 & \textit{otherwise} \end{cases}$$

Here it is very easy to express the respective constraints of the game: if one field is occupied by one value, then this value must not be repeated in the same row, the same column and the same subsquare:

DOI: 10.5176_2010-2283_1.2.48

$$\begin{aligned} x_{111} = 1 \rightarrow \quad & x_{211} = x_{311} = \ldots = x_{911} = \\ & x_{121} = x_{131} = \ldots = x_{191} = \\ & x_{221} = x_{321} = \ldots = x_{331} = \quad 0. \end{aligned}$$

It is surprising that this coding works very well in spite of the huge number of 729 or even 4096 Boolean variables (for a Sudoku board of 16×16 with numbers $1, \ldots, 16$) [15].

Example. Let be given a grid with $n \times m$ nodes. Each node can be occupied by one of four colors. The first possibility is the use of four variables for each grid point:

$x_{ijr} = 1$ means that the point (i, j) shows the color *red*. Then the following four vectors represent the possibilities for one node including the constraints:

red	*green*	*blue*	*yellow*
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Another possibility would be the coding of the colors by two bits for each node.

x_{ij1}	x_{ij2}	
0	0	*red*
0	1	*green*
1	0	*blue*
1	1	*yellow*

These two representations can be used for the representation and solution of coloring problems. One popular challenging problem is, for instance, the distribution of the colors in such a way that there is no rectangle of any size where the four rectangles show the same color. We take, for instance, the four grid points x_{11}, x_{21}, x_{12} and x_{22}. At least one point must have a color which is different from the color of the other points. The following array shows the opposite (complementary) case:

x_{111}	x_{112}	x_{211}	x_{212}	x_{121}	x_{122}	x_{221}	x_{222}	φ
0	0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	0
1	1	1	1	1	1	1	1	0

The solution of the problem cannot include these vectors. In this way we write down the forbidden vectors for each rectangle. The intersection of the complements of the respective complements gives the final solution (all vectors with $\varphi = 1$) .

B. Graph Problems

The nodes can be again coded by binary vectors, according to the number of nodes. Sometimes it is already sufficient simply to enumerate the nodes (by natural numbers). Edges are mostly given by using the starting and the ending node in this order. This means that $(\mathbf{s}, \mathbf{e})$ would be an edge from node $\mathbf{s}$ to node $\mathbf{e}$. The set of edges of a given graph g can be introduced by the following definition:

$$g(\mathbf{s}, \mathbf{e}) = \begin{cases} 1 & \textit{if there is an edge from node } \mathbf{s} \textit{ to node } \mathbf{e} \\ 0 & \textit{otherwise} \end{cases}$$

If the first node is coded by, for instance, (010), the second by (111), then (010111) would be an edge from the node (010) to (111), the vector (111010) indicates an edge in the reverse direction. This model is an excellent approach to solve all kinds of path problems in graphs. The solution of finding all kinds of Hamiltonian paths or Hamiltonian circuits [12] mainly requires the input of the graphs which can be rather time-consuming for large graphs. The solution of the problem itself is straightforward.

C. Relations

Relations can be translated into the graph of the relation and treated as before.

As a tiny example we take the set $\{0, 1, 2, 3\}$ and represent the relation $<$ which can be represented by the following array:

$<$	0	1	2	3
$0 = (00)$		●	●	●
$1 = (01)$			●	●
$2 = (10)$				●
$3 = (11)$				

Using the coding given in brackets, a function φ precisely defines this relation:

x_{11}	x_{12}	x_{21}	x_{22}	φ
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	1	0	1
0	1	1	1	1
1	0	1	1	1

It can be expressed as

$$\varphi = \overline{x}_{11} x_{21} \vee \overline{x}_{12} x_{21} x_{22} \vee \overline{x}_{11} \overline{x}_{12} x_{22}.$$

D. Circuit Design

There is no doubt that the design of circuits and many problems related to this area are best-known. Therefore, we will only mention this area, there is a numberless set of explorations and investigations related to this area. One of them is the unate covering problem which we selected as example for detailed studies in this paper.

III. Unate Covering - the Problem

Based on the given assumptions, the equation to be solved is given as a *conjunctive form* [10], i.e. as a conjunction of clauses. The clauses are disjunctions of variables, and in the special case of a *unate covering problem* none of the variables is negated. Functions described in this way are called *Petrick functions* $P(\mathbf{x})$. The given expression (1) shows an example that has been used. The Petrick function defined by (1) depends on 8 variables and is given by eight clauses:

$$\begin{aligned}
&(x_4 \vee x_5 \vee x_6 \vee x_8) && \wedge (x_2 \vee x_3 \vee x_4 \vee x_7 \vee x_8)\\
&(x_1 \vee x_3 \vee x_4 \vee x_7 \vee x_8) && \wedge (x_1 \vee x_4 \vee x_5 \vee x_7 \vee x_8)\\
&(x_1 \vee x_2 \vee x_5 \vee x_6) && \wedge (x_4 \vee x_5 \vee x_6 \vee x_7 \vee x_8)\\
&(x_1 \vee x_4 \vee x_5 \vee x_6 \vee x_7 \vee x_8) && \wedge (x_4 \vee x_6 \vee x_7)\\
& && = 1 .
\end{aligned} \tag{1}$$

Equations of this type always have a solution: at least the assignment $x_i = 1, i = 1, \dots, n$ is a (trivial) solution of the equation. We are, however, interested in solutions with the smallest number of variables equal to 1. For the given example, the values $x_1 = 1$ and $x_4 = 1$, expressed by the ternary vector $(1--1----)$, are such a solution. At least one of these two variables appears in each clause. Due to the six dashes, the mentioned solution vector describes a total of $2^6 = 64$ solutions. The single exact minimal solution containing in the ternary vector $(1--1----)$ is described by the Boolean vector (10010000).

IV. Basic Approach

A. Theoretical Background

The classical approach to solve the unate covering problem is the application of the distributive law [4]

$$(a \vee b) \wedge (c \vee d) = a\,c \vee a\,d \vee b\,c \vee b\,d \tag{2}$$

from the left to the right for all clauses. The sign of the AND-operation '$\wedge$' will often be dropped. The emerging conjunctions will be longer and longer, the maximally possible length is equal to the number of clauses. However, if a variable appears more than once in a conjunction, the idempotence law [4]

$$a \wedge a = a \tag{3}$$

will be used.

The application of (2) and (3) during the solution process of equation (1) results in $180,000$ conjunctions. A time of 8.097 seconds has been measured on a 3 GHz PC for the computation of these $180,000$ conjunctions using a single processor core.

Both the runtime and the huge number of basic solutions make any improvement very desirable. The absorption law

```
    xxxx xxxx
    0000 0000
    1234 5678
    =========
    1--1 ----
    -1-1 ----
    ---1 1---
    ---1 -1--
    11-- -1--
    1-1- -1--
    --1- 11--
    ---- -11-
    ---- -1-1
    1--- --11
    ---- 1-1-
(a) -1-- --11
```

```
    xxxx xxxx
    0000 0000
    1234 5678
    =========
    1--1 ----
    -1-1 ----
    ---1 1---
    ---1 -1--
    ---- -11-
    ---- -1-1
(b) ---- 1-1-
```

Figure 1. Solutions of the simple covering problem (1): (a) all 12 minimal irredundant solutions; (b) all 7 wanted exact minimal solutions

[4] allows to remove conjunctions from the solution set which are covered by conjunctions of less variables:

$$a \vee a\,b = a \ . \tag{4}$$

The absorption is very powerful. Its application to the disjunction of the $180,000$ conjunctions leads to a disjunction of 12 conjunctions, see Figure 1 (a), that solve equation (1). This simplification needs 75.646 seconds on the same PC using a single processor core again.

These 12 solutions are minimal (irredundant) solutions with the following properties (see the example above):

- After assigning the value 1 to the variables of the solution conjunction, the remaining variables can take any value. Therefore, a solution with k variables of a Petrick function $P(\mathbf{x})$ of n variables describes 2^{n-k} solutions.
- If the value 0 is assigned to the variables that do not appear in the solution conjunction, and additionally the value 0 is assigned to one variable of the solution, then we get $P(\mathbf{x}) = 0$.
- The solution vectors describe solution sets which are not disjoint. Hence, the number of solutions of the equation cannot be calculated directly.

Only a subset of this set of irredundant solutions is the wanted solution with the minimal number of variables. The set of 12 solution vectors consists of 7 solutions with 2 variables, see Figure 1 (b), and 5 solutions with 3 variables. The wanted minimal solutions can simply be found by counting the number of variables of the solution conjunctions (i.e. the values 1 in the ternary solution vectors).

B. Practical Algorithm

A practical algorithm that solves the unate covering problem can be implemented such that the idempotency law (3) is applied implicitly. The application of the distributive law (2) to two sets of r and s conjunctions results in a

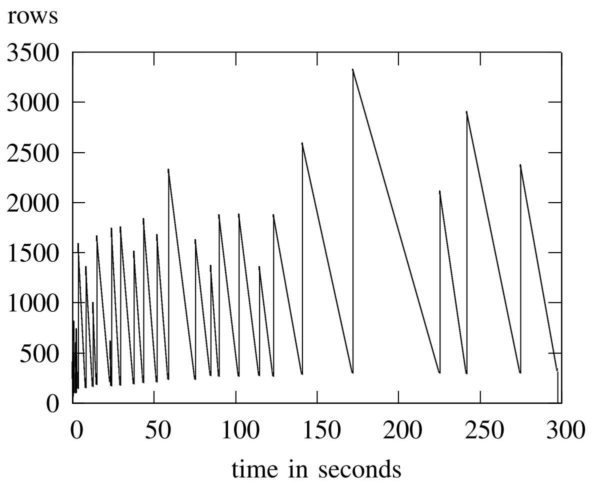

Figure 2. Change of the number of row applying the distributive and absorption law consecutively in the loop of the computation of the solution for the covering problem of 16 variables and 32 clauses

set of $r \cdot s$ conjunctions. The complexity of this part of the algorithm for a Petrick function of n variables that consists of c clauses is equal to $O(n^c)$. The absorption (4) strongly reduces the number of conjunctions. Hence, the absorption has been used after each application of the distributive law. This change in the algorithm reduced the runtime for the above example by a factor of 13,957. The same 7 solution vectors with 2 variables were found within 6 milliseconds instead of 83.743 seconds.

Figure 2 visualizes the solution process for the largest solvable example using the suggested algorithm within a time limit of 10 minutes. Figure 2 shows on the one hand that the execution of the distributive law for one clause result quickly in a much larger number of rows, which represent intermediated solutions. On the other hand this Figure shows also that it is time consuming to exclude such intermediated solutions which are absorbed by other intermediated solutions. The different local maximal values of this process depend on both the last reduced number of rows and the number of literals in the used clause.

Table I shows the experimental results of the suggested basic approach. This approach is acceptable for Petrick functions of 8 variables. It is too much time consuming to compensate the complexity of $O(n^c)$ for the distributive law by the application of the absorption law for larger functions – as can be seen already for Petrick functions of 16 variables.

The practical requirement to solve such covering problems for larger numbers of variables on the one hand and the extremely growing of runtime on the other hand forces to search for alternative approaches which utilize the complete parallel computation power of modern PCs.

In order to increase the number of variables and to reduce the runtime, respectively, we now consider parallel approaches to solve the given problem. It is our aim to present approaches that can be widely used for the detailed studied covering problem as well as for similar SAT problems on typical modern PCs.

Table I
BENCHMARK RESULTS FOR THE BASIC APPROACH

Literals	Clauses	Solutions	Minimal Number of Variables	Minimal Number of Solutions	Total Time (ms)
8	8	12	2	7	6
8	16	13	2	4	13
8	32	8	4	3	13
8	64	5	5	4	86
8	128	4	5	3	68
8	256	2	6	2	16
16	8	110	2	8	1,761
16	16	199	2	1	11,218
16	32	320	3	9	299,928

V. PARALLELISM IN THE APPLICATION DOMAIN

Having n_{core} cores in the PC, it is an obvious approach to use the known techniques [2], [5] to adapt the previous algorithm to several cores that can contribute simultaneously to the solution of the problem. Due to the exponential complexity of the problem, the additionally possible number of variables n_+ is restricted to $n_+ = log_2\, n_{core}$.

Furthermore, due to Amdahl's law, the reachable speedup will be in most cases less than the number of cores working in parallel. Taking into account that we concentrate on a small number of cores and already achieved a speedup of more than 13,000, we will study first the application domain for alternative faster approaches.

As mentioned above, a solution vector including k of n variables describes 2^{n-k} solutions. This shows that for n Boolean variables the exponential size 2^n of the search space will be easier manageable when 2^{n-k} Boolean vectors are represented by a single ternary vector and, generally, ternary vectors are used as the respective data structure. XBOOLE [4], [9], [11], [13] is a library that utilizes this approach consistently for both the representation of Boolean functions and the respective computations.

The solution set of a characteristic equation with a *disjunctive form* [10] on the left hand side can be found in constant time. Hence, an alternative approach to solve the equation $P(\mathbf{x}) = 1$ is the transformation of the Petrick function $P(\mathbf{x})$ given in conjunctive form [10] into an equivalent disjunctive form $AS(\mathbf{x})$ that describes all solutions of this equation as well. We use the following two properties:

1) Two successive negations do not change a Boolean function: $f(\mathbf{x}) = \overline{\overline{f(\mathbf{x})}}$.
2) The negation using *de Morgan's law* alternates between conjunctive and disjunctive form.

Using the XBOOLE-operators `NDM(`$f(\mathbf{x})$`)` for the negation according to de Morgan's law and `CPL(`$f(\mathbf{x})$`)` for

Table II
BENCHMARK RESULTS FOR THE ALGORITHM: CPL(NDM(f))

Benchmark		All	TV	Solution		Time
L	C	Solutions	Count	L	C	ms
8	8	198	16	2	7	0
8	16	183	24	2	4	0
8	32	43	12	4	3	0
8	64	19	12	5	4	1
8	128	16	7	5	3	1
8	256	6	6	6	2	1
16	8	62,260	189	2	8	1
16	16	57,727	469	2	1	1
16	32	51,899	1,552	3	9	3
16	64	45,710	2,403	4	15	5
16	128	42,624	2,854	4	4	8
16	256	22010	2,933	5	2	12
24	8	16,748,287	1,198	2	38	3
24	16	16,500,394	8,231	3	147	22
24	32	16,186,932	16,508	3	19	66
24	64	15,950,753	33,909	4	128	225
24	128	14,894,860	95,316	4	1	1,737
24	256	14,495,195	200,368	5	3	7,849
32	8	4,292,102,982	5,206	2	61	14
32	16	4,289,047,266	20,718	2	8	96
32	32	4,286,710,172	91,058	3	79	1,556
32	64	4,261,521,507	334,172	4	398	23,594
32	128	4,249,596,098	1,001,493	4	22	220,184
32	256	3,186,686,031	1,649,871	5	38	734,171

the calculation of the complement, we get the following algorithm:

$$AS(\mathbf{x}) = \texttt{CPL}(\texttt{NDM}(P(\mathbf{x}))\ . \qquad (5)$$

The XBOOLE-operator NDM($f(\mathbf{x})$) has a complexity of $O(1)$. The main computational effort is required by the XBOOLE-operator CPL. The benefits of this approach in comparison to the application of the distributive law are as follows:

1) The solution is represented by an orthogonal set of ternary vectors [4].
2) Each ternary vector that includes d dash elements represents 2^d solutions.
3) Due to the orthogonal representation, the wanted exact minimal solutions can be selected by counting the 1-elements in the solution vectors.

Table II shows the experimental results for the approach of using formula (5). L indicates the number of variables (literals), C the number of clauses. We used the same benchmark set for these experiments. Due to the higher power of the CPL(NDM(f))-approach, we could solve significantly larger covering problems. Due to the orthogonal representation of the solution it is possible to calculate the number of all solutions of the solved equation. These values are given in the third column of Table II. The number of ternary vectors which represent all solutions is given in the forth column of Table II. The benefit of the applied ternary representation becomes visible by comparing column 3 and 4 of Table

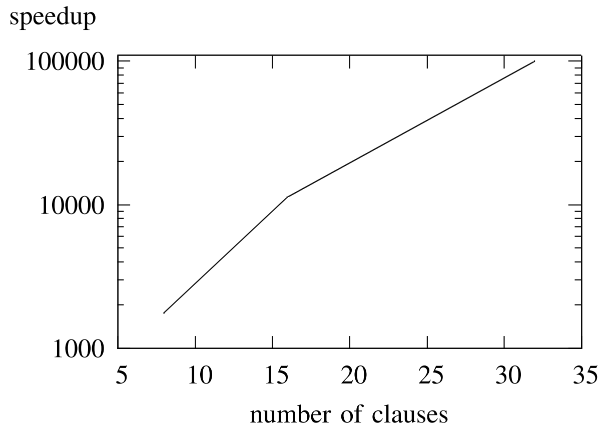

Figure 3. Speedup reached by the XBOOLE-approach CPL(NDM(f)) (5) in comparison to the already strongly improved iterative basic algorithm using the distributive and absorption law for covering problems of 16 variables

II. Based on the largest benchmark solved with the basic approach (16×32), the CPL(NDM(f))-approach reached a speedup of a factor of 99,976. The same nine solution vectors with 3 variables were found in 3 milliseconds instead of 299.928 seconds using a single processor core.

Figure 3 show the serious speedup in the range of more than 1,000 up to nearly 100,000 reached by the XBOOLE-approach CPL(NDM(f))-approach (5) for identical covering examples of 16 variables and 8 up to 32 clauses. This strong improvement confirms that *the utilization of properties given by the application domain should be combined with power of the common use of several processors working in parallel.*

VI. PARALLEL SOLUTION USING A 4-CORE PC

A. Uniform Distribution

A concurrent processing on several cores can be implemented by threads of a single process or a set of processes. Due to the common use of the same program functions and some local control variables in these functions, threads cannot be used when the same XBOOLE operation must be applied concurrently. Hence, we must use the message-passing interface (MPI) [2] for the concurrent solution of the covering problem on the available 4 cores.

The main task of the unate covering problem is solved by the CPL-operation. After the execution of $h(\mathbf{x}) = \texttt{CPL}(g(\mathbf{x}))$, the function $h(\mathbf{x})$ is equal to 1 for such patterns $\mathbf{x}$ of the Boolean space B^n for which the function $g(\mathbf{x})$ is equal to 0. Hence, the CPL-operation calculates the difference between the whole Boolean space and the 1-patterns of the given function $g(\mathbf{x})$. The XBOOLE-operation DIF(f, g) calculates $f(\mathbf{x}) \wedge \overline{g(\mathbf{x})}$.

An obvious approach for the parallel solution of the covering problem is the partition of the Boolean space into subspaces of comparable size and the concurrent execution

Table III
BENCHMARK RESULTS FOR THE CONCURRENT ALGORITHM: $\mathtt{DIF}(f_{ss}^1(2,r,\mathbf{x}_0)\mathtt{NDM}(P(\mathbf{x}))$ USING 4 CORES

Bench-	Solution	Time in Milliseconds				
mark	L/C	T0	T1	T2	T3	Total
24x8	2/38	3	1	0	0	3
24x16	3/147	29	12	3	1	29
24x32	3/19	61	19	28	11	61
24x64	4/128	138	44	41	10	139
24x128	4/1	679	127	293	67	679
24x256	5/3	1,873	691	874	175	1,873
32x8	2/61	22	11	11	7	22
32x16	2/8	99	16	37	4	100
32x32	3/79	956	104	128	32	956
32x64	4/398	9,674	568	1,623	178	9,674
32x128	4/22	78,106	3,785	8,004	284	78,107
32x256	5/38	241	31,931	25,952	4,960	31,931

of the DIF-operation for these subspaces. For a compact representation of the problems we use

- $f^1(\mathbf{x})$ for the function that is equal to 1 for any $\mathbf{x}$, and
- $f_{ss}^1(k,i,\mathbf{x}_0)$ for the function that is equal to 1 for any $\mathbf{x}_0$ of the i-th of 2^k subspaces.

Using r as an index of the respective process, on each core the subtask

$$AS[r](\mathbf{x}) = \mathtt{DIF}(f_{ss}^1(2,r,\mathbf{x}_0)\mathtt{NDM}(P(\mathbf{x})) \qquad (6)$$

must be solved. The final solution for the special case of 4 cores can be calculated by

$$AS(\mathbf{x}) = \bigvee_{r=0}^{3} AS[r](\mathbf{x}) \ . \qquad (7)$$

Due to the orthogonality of $f_{ss}^1(2,r,\mathbf{x}_0)$ the partial solution set $AS[r](\mathbf{x})$ of (7) are orthogonal too. For that reason the disjunctions in (7) can be realized by concatenation of partial solution sets, which can be done in constant time.

Table III shows the results for the application of (6) and (7) using 4 cores. Due to the very short runtime we did not include the benchmark results for 8 and 16 variables.

Columns T0, ..., T3 show the runtime for the four different subspaces. Despite the same size of the subproblems, these runtimes vary in a wide range:

1) Depending on the given Petrick function, the representation of the partial solution in the subspace can require different numbers of ternary vectors.
2) The computation with different numbers of ternary vectors requires different time intervals.
3) The final evaluation of the solution set with regard to the wanted minimal solution requires different time intervals depending on both the number of ternary vectors and the iteration when the first minimal solution is found.

Therefore the runtime for a subspace can even be higher than the runtime for the whole covering problem, especially

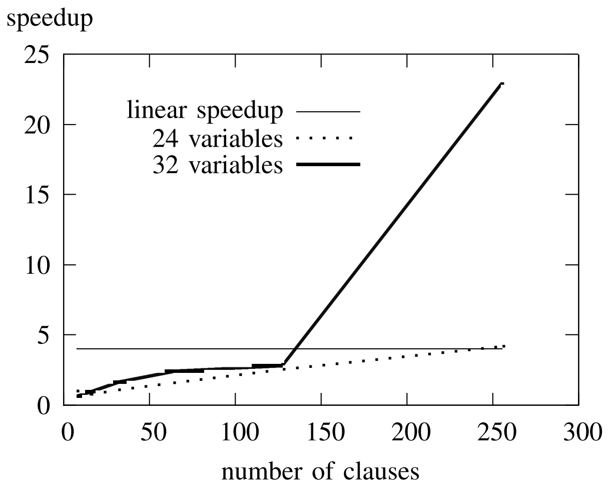

Figure 4. Comparison of the XBOOLE-approaches $\mathtt{UNI}(\mathtt{DIF}(\mathtt{NDM}(f[r])))$ (6), (7) and $\mathtt{CPL}(\mathtt{NDM}(f))$ (5) for covering problems of 24 and 32 variables

for relatively small problems. Figure 4 shows in the left half this weakness of a uniform division of the Boolean space. On the other hand, the same reasons lead to a super-linear speedup of 22.99 for the solution of the unate covering problem of 32 literals and 256 clauses using 4 cores. Figure 4 shows on the right half that for larger numbers of clauses the simple XBOOLE-approaches $\mathtt{UNI}(\mathtt{DIF}(\mathtt{NDM}(f[r])))$ reaches linear or even super-linear speedups.

The observed bad load balancing of the uniform division of the Boolean space forced ourselves to suggest an improved parallel solution that will be discussed in the next subsection.

B. Adaptive Distribution

Basically the Boolean space B^n can be maximally divided into 2^n subspaces. For the concurrent computation on n_{core} cores we need at least n_{core} subspaces. This minimal number of subspaces was used in the previous approach and caused a bad load balancing. Hence we decided to split the covering problem into subproblems for a larger number of subspaces.

The larger the number of subspaces, the better the load balancing that can be achieved. However, the creation of too many subspaces contradicts the very valuable improvements by means of ternary vectors. Each ternary vector itself represents a subspace that is directly defined by the context of the problem to be solved. As a compromise we use $n_{ss} = 2^6 = 64$ subspaces for our PC with 4 cores.

In order to improve the load balance, the subspaces must be assigned to the processes in such a way that all working processes will finish approximately at the same time. Hence, one of the processes must control the assignment of the subtasks. Therefore we implemented a master-worker architecture (see Figure 5). The *master* process controls the assignment of the subtasks to the *worker* processes. Hence,

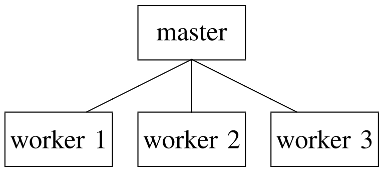

Figure 5. Master - Worker Architecture of a PC with 4 Cores

Table IV
BENCHMARK RESULTS FOR THE ADAPTIVE CONCURRENT ALGORITHM: $\texttt{DIF}(f^1_{ss}(6,i,\mathbf{x}_0)\texttt{NDM}(P(\mathbf{x}))$ USING 4 CORES

Bench-	Solution	Time in Milliseconds				Speedup
mark	L/C	T1	T2	T3	Total	
24x8	2/38	3	3	3	3	1.0
24x16	3/147	8	10	8	10	2.2
24x32	3/19	24	25	25	25	2.6
24x64	4/128	52	53	52	53	4.2
24x128	4/1	168	168	167	168	10.3
24x256	5/3	335	335	333	335	23.4
32x8	2/61	13	13	13	13	1.1
32x16	2/8	34	34	34	34	2.8
32x32	3/79	172	174	171	174	8.9
32x64	4/398	684	750	689	750	31.5
32x128	4/22	3,028	3,856	3,307	3,856	57.1
32x256	5/38	3,096	3,099	3,097	3,099	236.9

we lost one core for the direct problem solution.

The concurrent algorithm follows the previous approach. Using i as index of the Boolean subspace, the three worker processes solve the assigned subtask

$$AS[i](\mathbf{x}) = \texttt{DIF}(f^1_{ss}(6,i,\mathbf{x}_0)\texttt{NDM}(P(\mathbf{x})) \ . \tag{8}$$

The aggregate solutions for the special case of 4 cores can be calculated by

$$AS(\mathbf{x}) = \bigvee_{i=0}^{n_{ss}} = AS[i](\mathbf{x}) \ . \tag{9}$$

This architecture is very profitable because the master can assign the next unsolved subtask to the worker immediately on request of the worker.

Table IV shows the experimental results for the adaptive approach of formulas (8) and (9) using 4 cores. The columns T1, T2, and T3 of Table IV show the runtime of the three workers. No speedup is reached for the small benchmarks which need in all improved approaches only few milliseconds.

Despite the restriction to three workers, the highest speedup in comparison to the single core solution is 236.9. The reason of this impressive speedup is based on a special utilization of the implemented concurrent approach. Each worker sends the results of the unate covering problem for the assigned subspace to the master process. These results include both the minimal number of values 1 in this partial solution and the number of such minimal solutions. The

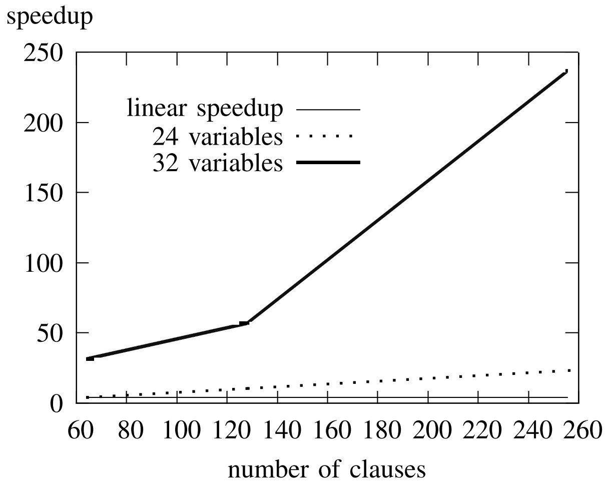

Figure 6. Speedup reached by the adaptive XBOOLE-approache $\texttt{UNI}$ ($\texttt{DIF}(\texttt{NDM}(f[r])))$ (8), (9) in comparison to $\texttt{CPL}(\texttt{NDM}(f))$ (5) for covering problems of 24 and 32 variables and 64 and more clauses

master process handles the partial solutions in the following way:

- If the master process already knows solutions with a smaller number of values 1, the received solutions with a larger number of values 1 are omitted immediately.
- If the master process already knows solutions with the same number of values 1, the received solutions are accumulated.
- If the master process knows so far only solutions with more values 1, the stored accumulated solution is replaced by the new better solution.

Using this simple algorithm, the master process knows the smallest number of values 1 found so far by the concurrent worker processes. On each request of a worker for the next subtask, the master process sends both the number of the next subspace and the smallest solution found so far. This information helps the worker process to simplify the evaluation algorithm because large solutions must not be taken into account anymore.

Small tasks, where the $\texttt{CPL}(\texttt{NDM}(f))$ approach (5) needs a couple of milliseconds only, should solved by single processor core. Convenient thresholds for the covering problems are 24 variables and 64 clauses. Figure 6 shows the reached speedups for the solved tasks above these thresholds using the approach of adaptive distribution. In all these cases we reached a super-linear speedup. It is notable that the super-linear speedup rises even strongly for larger tasks. This observation leads to an important general conclusion: *the available set of processor cores should not be use simply for computation of the assigned subtasks but mainly as source of knowledge that restricts effort for the subsequent subtasks.*

Table V
BENCHMARK RESULTS FOR THE APPROACH THAT COMBINED THE ADAPTIVE CONCURRENT ALGORITHM $\mathtt{DIF}(f^1_{ss}(6, i, \mathbf{x}_0)\mathtt{NDM}(P(\mathbf{x}))$ AND THE INTELLIGENT MASTER USING 4 CORES

Bench-mark	Solution L/C	Time in Milliseconds	Speedup	Efficiency
32x8	2/61	38	0.4	0.1
32x16	2/8	75	1.3	0.3
32x32	3/79	234	6.6	1.6
32x64	4/398	535	44.1	11.0
32x128	4/22	1,295	170.0	42.5
32x256	5/38	2,061	356.2	89.0

C. Intelligent Master

In the previous approach a master process is required to control the adaptive assignment of the subtask to the worker. The master process waits most of the time for the requests of the workers. An additional worker thread inside the master process can cause delays to answer the worker requests. Hence, the chosen master-worker architecture of Figure 5 should not change.

In the previous adaptive approach we saw the positive effect of distributing the smallest number of values 1 known so far. Up to now this knowledge has been used only by the workers. In a further approach the same knowledge is used by the master itself. Such an *intelligent master* process allows the additional reduction of the runtime for large problems.

This approach of an intelligent master relies on the following property: we are searching only for solution vectors with the smallest number of values 1. A subspace is defined by fixing a certain number of variables; $(x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1, x_5 = 0)$ defines, for instance, one of 32 subspaces where the first 5 variables are fixed. If a solution of two values 1 is already known, it can be concluded that no solution results from this subspace since already three variables have the value set to 1.

Such an evaluation can be realized by the master when it is waiting for a request of a worker. Subspaces in which no solution can exist are excluded by the *intelligent master* without detailed calculations in the subspace itself.

We implemented this *intelligent master* approach too. This approach should be applied especially for large benchmarks. Due to this intelligent behavior of the master it is possible to extend the division of the Boolean space into more subspaces. As a compromise we used $n_{ss} = 2^{10} = 1024$ subspaces for our PC with 4 cores. In that way, we get a much better load balancing – we measured exactly the same runtime for all 3 concurrent processes. Another positive effect is the reduction of the required memory space. The number of necessary ternary vectors is reduced by a factor of $n_{ss} = 2^{10} = 1024$.

Table V shows that this approach should be used for large unate covering problems. We suggest an implementation where the applied approach is selected depending on the known number of variables. The speedup in column 4 is the ratio between the improved approach of Table II on a single core and the approach of Table V using 4 cores. Due to Amdahl's law the speedup is typically less than the number of used cores. This is true for the two smallest evaluation benchmarks. Such small tasks can be solved in a couple of milliseconds on a single core; no parallel approach must be applied. The speedup for the largest executed benchmark is equal to 356.2. Hence, without substantial computations the intelligent master approach reduces the required runtime by one third.

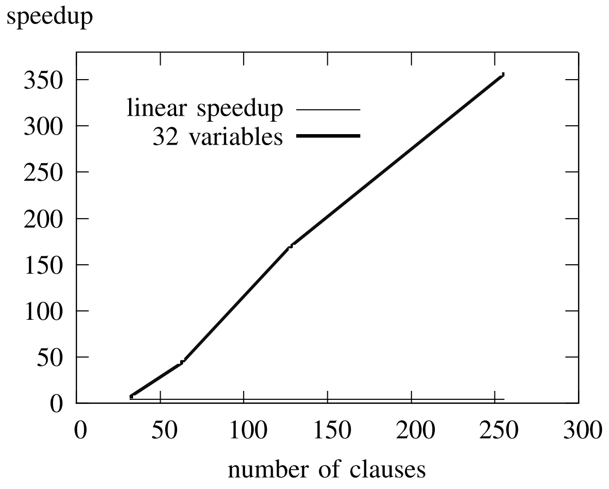

Figure 7. Sspeedup reached by the intelligent master XBOOLE-approache based on $\mathtt{UNI}(\mathtt{DIF}(\mathtt{NDM}(f[r])))$ (8), (9) in comparison to $\mathtt{CPL}(\mathtt{NDM}(f))$ (5) for covering problems of 32 variables and 32 and more clauses

It is an important property of the suggested *intelligent master* approach that the speedup grows with larger tasks. A second evaluation parameter is the *efficiency*. The efficiency is defined as quotient between the speedup and the number of used cores. An ideal implementation is reached if the value of the efficiency is equal to 1. For benchmarks larger than 32 variables and 32 clauses the approach of the intelligent master holds this ideal value. It is a remarkable result that the efficiency grows strongly with the size of the problem. The efficiency for the largest benchmark is 89.

Figure 7 shows in comparison to Figure 6 the further improvement of the *intelligent master* approach over the approach of adaptive distribution. The super-linear speedup is reached already for the covering problem of 32 variables and 32 clauses. The suggested enhancement of the simple master to an intelligent one improved the reached speedup by further 50%.

VII. CONCLUSIONS

We selected for our studies in this paper one task belonging to the set of the most complex problems, the unate covering problem. These problems have an exponential complexity. Our approaches utilize special properties of this task, but these approaches can be mapped to other Boolean problems of the same high complexity.

The main instrument to deal with the exponential increase of the time required to solve such tasks is the *parallelism*. Thinking about parallelism associates recently to supercomputers of thousands of processor cores. Of course, such expensive computers can be the final way to solve extreme tasks. We showed that the utilization of both much cheaper computers and well analyzed properties of the task to be solved lead to unbelievable improvements.

In detail we confirmed the following facts.

- The order of operations has a strong influence on the runtime. Replacing a single absorption after the application of the distributive law for all clauses by a repeated absorption after the application of the distributive law for each single clause reduced the runtime for the simplest 8×8 benchmark by a factor of 13,957.
- The utilization of the parallelism in the application domain is the crucial factor to solve larger Boolean problems of exponential complexity. The application of operations of the XBOOLE library reduces the runtime to solve the largest benchmark solvable by the improved basic approach (16×32) by another factor of 99,976.
- The division of the Boolean space into uniform subspaces for the available cores in a parallel implementation does not lead to a similar runtime of the subtasks. The runtime of the cores in such a solution differs by a factor of more than 10.
- An adaptive approach improves the load balancing. Therefore a larger number of subspaces is necessary. Fixing a larger number of variables contradicts the parallel approach in the application domain. The division into 64 subspaces was found as a good compromise for a 4-core PC.
- *The distributed solution itself is a source for an additional speedup, even when a small number of cores is used*. The exchange of the simplest solution found so far between the worker by the help of the master allows to restrict the analysis task of the workers and leads to a speedup of 236.9 using only 4 cores.
- The additional utilization of the simplest solution found so far by the intelligent master on a higher level improves the speedup by another 50%.
- An implementation should combine several approaches and select the most suitable approach depending on the size of the task to be solved.
- In comparison to the improved basic approach which was 13,957 times faster than the basic approach with a single absorption we reached an overall improvement factor of more than 35 million where nearly hundred thousand comes from utilization of properties in the application domain and more than 350 by a super-linear speedup of four processor cores only.

The improvement factors were calculated from the largest comparable benchmark examples. More detailed comparisons confirm that these factors are not constant but functions which increase their values for larger tasks. In future research we will explore unate covering problems depending on even more variables and clauses. We see the key for further improvements in the combined utilization of approaches of both the application domain and the parallelism of the used computer equipment.

Acknowledgment

The authors would like to thank Marek Perkowski from the Portland Sated University, Oregon (USA) who introduced us to this topic. We also thank his master student Eric Paul for providing the used benchmark generator.

References

[1] Cordone, R., Ferrandi, F., Sciuto, D. and Wolfler Calvo, R. *An Efficient Heuristic Approach to Solve the Unate Covering Problem*. Proceedings of the Conference on Design, Automation and Test in Europe, Paris, France, 2000, pp. 364–371.

[2] Gropp, W., Thakur, R. and Lusk, E. *Using MPI-2: Advanced Features of the Message Passing Interface*. MIT Press, Cambridge, MA, USA, 1999.

[3] Patterson, D. A. and John L. Hennessy, J. L. *Computer Organization and Design: the Hardware/Software Interface*. Morgan Kaufmann Publishers, Burlington, USA, 2009.

[4] Posthoff, Ch. and Steinbach, B. *Logic Functions and Equations - Binary Models for Computer Science*. Springer, Dordrecht, The Netherlands, 2004.

[5] Quinn, M. J. *Parallel Programming in C with MPI and OpenMP*. Mc Graw Hill, New York (NY), USA, 2004.

[6] Sasao, T. and Butler, J. T. *Progress in Applications of Boolean Functions - Synthesis Lecturers on Digital Circuits and Systems*. Morgan & Claypool Publishers, San Rafael (CA), USA, 2010, pp 1–153.

[7] Steinbach, B. (Ed.): *Boolean Problems, Proceedings of the 8th International Workshops on Boolean Problems*. Freiberg University of Mining and Technology, 2008, pp. 1–250.

[8] Steinbach, B. (Ed.): *Boolean Problems, Proceedings of the 9th International Workshops on Boolean Problems*. Freiberg University of Mining and Technology, 2010, pp. 1–238.

[9] Steinbach, B. *XBOOLE - A Toolbox for Modelling, Simulation, and Analysis of Large Digital Systems*. System Analysis and Modeling Simulation, Gordon & Breach Science Publishers, Volume 9, Number 4, 1992 - pp. 297–312.

[10] Steinbach, B. and Posthoff, Ch. *An Extended Theory of Boolean Normal Forms*. in: Proceedings of the 6th Annual Hawaii International Conference on Statistics, Mathematics and Related Fields, Honolulu, Hawaii, 2007, pp. 1124–1139.

[11] Steinbach, B. and Posthoff, Ch. *Boolean Differential Calculus*. in: Sasao, T. and Butler, J. T. Progress in Applications of Boolean Functions Synthesis Lecturers on Digital Circuits and Systems. Morgan & Claypool Publishers, San Rafael, CA USA, 2010, pp. 55-78 and 121-126.

[12] Steinbach, B. and Posthoff, Ch. *Complete Sets of Hamiltonian Circuits for Classification of Documents*. in: Moreno-Diaz, R., Pichler, F. and Quesada-Arencibia, A. Computer Aided System Theory - EUROCAST 2009. Springer (LNCS 5717), Berlin, Heidelberg, New York, 2009, pp. 526 - 533.

[13] Steinbach, B. and Posthoff, Ch. *Logic Functions and Equations - Examples and Exercises*. Springer Science + Business Media B.V., 2009.

[14] Steinbach, B and Posthoff, Ch. *Set-Based SAT-Solving*. in: FACTA UNIVERSITATIS (NIŠ), Series: Electronics And Energetics. Volume 20, Issue No. 3,), December 2007, pp. 395–414.

[15] Steinbach, B and Posthoff, Ch. *Sudoku Solutions Using Logic Equations*. in: Steinbach, B. (Ed.) Boolean Problems, Proceedings of the 8th International Workshops on Boolean Problems, Freiberg University of Mining and Technology, Freiberg, 2008, pp. 49 - 57.

[16] Wegener, I. *Complexity Theory. Exploring the Limits of Efficient Algorithms: Exploring the Limits of Efficient Algorithms*. Springer, 2004.

Bernd Steinbach is a Full Professor of Computer Science (Software Engineering and Programming) in the Department of Computer Science at the Freiberg University of Mining and Technology since 1992.

Dr. Steinbach got his Ph.D. with a thesis about the Solution of Boolean Differential Equations and graduated with a Dr. sc. techn. (Doctor scientiae technicarum) for his second doctoral thesis from the Faculty of Electrical Engineering of the Chemnitz University of Technology in 1981 and 1984, respectively. In 1991 he obtained the Habilitation (Dr.-Ing. habil.) from the same Faculty.

He has served as Head of the Department of Computer Science and Vice-Dean of the Faculty of Mathematics and Computer Science. He is the head of a group that developed the XBOOLE software system. He published three books in logic synthesis. The first one (together with D. Bochmann) covers Logic Design using XBOOLE (in German). The following two (together with Christian Posthoff) are ”Logic Functions and Equations Binary Models for Computer Science” and ”Logic Functions and Equations Examples and Exercises”, Springer 2004, and 2009, respectively. He published more than 180 chapters in books, complete issues of journals, and papers in journals and proceedings.

He has served as Program Chairman for the IEEE International Symposium on Multiple-Valued Logic (ISMVL), and as guest editor of the Journal of Multiple-Valued Logic and Soft Computing. He is the initiator and general chair of a biennial series of International Workshops on Boolean Problems (IWSBP) which started in 1994, with 9 workshops until now.

He received the Barkhausen Award from the University of Technology Dresden in 1983.

Christian Posthoff is a Full Professor of Computer Science at the University of The West Indies, St. Augustine, Trinidad & Tobago since 1994. From 1996 to 2002 he was Head of the Department of Mathematics & Computer science. In 2010 his is retired.

Dr. Posthoff got his Ph.D. in 1975 with the thesis ”Application of Mathematical Methods in Communicative Psychotherapy” from the University of Leipzig.

Important results of his research activities have been algorithms and programs for solving Boolean equations with a high number of variables and the Boolean Differential Calculus for the analytical treatment of different problems in the field of logic design. These results have been collected in a monograph *Binary Dynamic Systems* published simultaneously in Akademie-Verlag Berlin, Oldenbourg Munich-Vienna and in the Soviet-Union, and allowed the habilitation (Dr.-Ing. habil.) at the Faculty of Electrical Engineering in 1979 and the promotion to Associate Professor.

In 1983, he started as Full Professor of Computer Science in the Department of Computer Science at the same university, with the aim of starting the program in Computer Science. Since 1984, he was Head of the Institute of Theoretical Computer Science and Artificial Intelligence and Research Director of the Department of Computer Science.

He is the author or co-author of 15 books and many publications in journals and conference proceedings.

He received the Scientific Award of the Chemnitz University of Technology four times. In 2001, he received the Vice-Chancellor’s Award of Excellence.