The van der Waerden Number $W(2,6)$ Is 1132

Michal Kouril and Jerome L. Paul

CONTENTS

1. Introduction
2. Preprocessing: Eliminating Redundancies
3. Finding Unavoidable Patterns
4. Preprocessing Patterns: Removing Redundant Branches
5. The Palindrome Heuristic
6. Putting the Preprocessing Together for $W(2,6)$
7. SAT Solver for van der Warden $K = 2$ Numbers
8. Architecture of the FPGA-Based SAT Solver
9. Performance Results
10. Extreme Partitions
11. Conclusion and Future Work
Acknowledgments
References

We have verified that the van der Waerden number $W(2,6)$ is 1132, that is, 1132 is the smallest integer $n = W(2,6)$ such that whenever the set of integers $\{1, 2, \ldots, n\}$ is 2-colored, there exists a monochromatic arithmetic progression of length 6. This was accomplished by applying special preprocessing techniques that drastically reduced the required search space. The exhaustive search showing that $W(2,6) = 1132$ was carried out by formulating the problem as a satisfiability (SAT) question for a Boolean formula in conjunctive normal form (CNF), and then using a SAT solver specifically designed for the problem. The parallel backtracking computation was run over multiple Beowulf clusters, and in the last phase, field programmable gate arrays (FPGAs) were used to speed up the search. The fact that $W(2,6) > 1131$ was shown previously by the first author.

1. INTRODUCTION

In 1926, B. L. van der Waerden proved the following result [van der Waerden 27]: given positive integers K, L, both at least 2 (the case of 1 being trivial), there is a smallest integer $n = W(K,L)$ such that every K-coloring of $\{1, 2, \ldots, n\}$ contains a monochromatic arithmetic progression of length L. Van der Waerden's proof gave little insight into the actual value of $W(K,L)$, and much interest has been shown in the combinatorial community in finding actual values for these numbers, or improving known lower bounds for them. In this paper, we restrict attention to $K = 2$, where the only nontrivial values heretofore known were $W(2,3) = 9$, $W(2,4) = 35$, and $W(2,5) = 178$. This last value was computed by Beeler and O'Neil in 1979 [Beeler and O'Neil 79].

In [Kouril and Franco 05] it was shown that $W(2,6) > 1131$ by exhibiting 2-colorings of $\{1, 2, \ldots, 1131\}$ not containing monochromatic arithmetic progressions of length 6 (see Section 8 of the current paper for a description of these 2-colorings). In this paper we show that $W(2,6) = 1132$. In order to carry out the massive computation, unavoidable patterns were utilized, and equivalent (and therefore redundant) branches in the search tree were pruned. For given integers K, L, and $n < W(K,L)$,

2000 AMS Subject Classification: Primary 68R05, 05D10

Keywords: Van der Waerden numbers, combinatorics, high-performance computing, Beowulf clusters, FPGAs

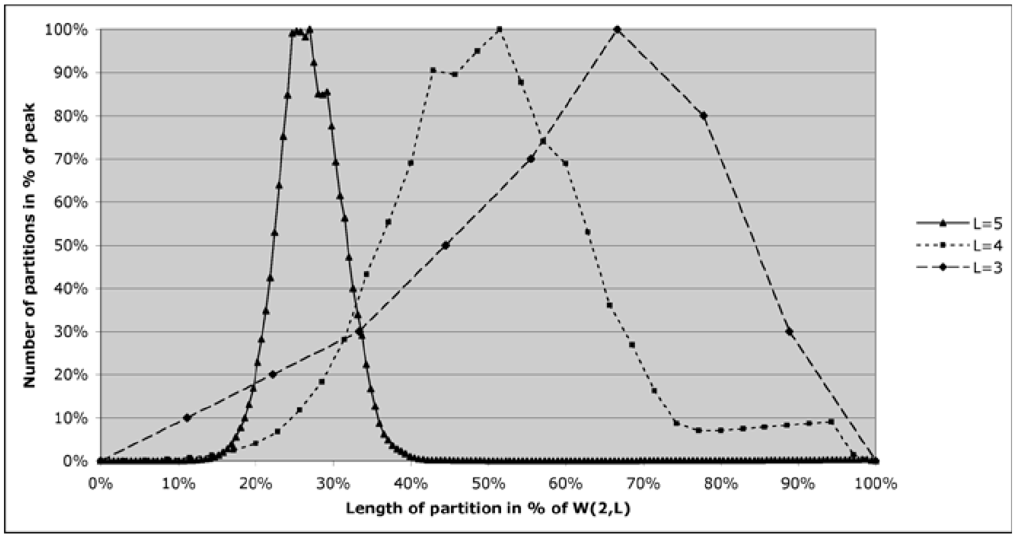

FIGURE 1. Number of partitions per partition length.

we say that a pattern P is (K, L, n)-*unavoidable* if P (or its various equivalent forms; see below) must exist in any K-coloring of $\{1, 2, \ldots, n\}$ not containing an arithmetic progression of length L. Note that if P is (K, L, n)-unavoidable, then it is (K, L, m)-unavoidable for all $n \leq m < W(K, L)$.

The following were the main steps in determining that $W(2, 6) = 1132$:

1. Show that the pattern 0000 (equivalently, 1111) is $(2, 6, 240)$-unavoidable. In other words, any 2-coloring of $\{1, 2, \ldots, 240\}$ not containing a monochromatic arithmetic progression of length 6 will contain either 0000 or 1111 (and hence, for example, must also contain either 00001 or 11110).

2. Start with an unavoidable pattern 00001 and grow the patterns while removing the redundant branches using a so-called palindrome heuristic. Do so while the set is manageable. In our case we stop at patterns of length at most 29, which yields 2,537,546 initial patterns.

3. Take the patterns generated in step 2, and convert (in the obvious way) the problem of extending them to patterns without monochromatic arithmetic progressions of length 6 to the problem of solving a set of 2,537,546 SAT problems. Here the associated SAT problem is satisfiable if the variables can be 2-colored without a monochromatic arithmetic progression of length 6 occurring in the corresponding pattern (we say that such a pattern is *satisfying*). The total number of variables in our set of SAT problems was taken as 240, and we placed the 2,537,546 patterns of length 28 or 29 found in step 2 (preset variables) in the middle portion of $\{1, 2, \ldots, 240\}$ for extending to satisfying patterns of length 240.

4. Solve each SAT problem from step 3. It turns out that only 111 out of 2,537,546 were found satisfiable. We used Beowulf clusters and later FPGA circuits to process all of the SAT problems.

5. Collect all solutions (satisfying patterns) for all satisfiable SAT problems from step 4.

6. To reduce the number of satisfying patterns further, we added 86 more unknown variables, bringing the total to 326 variables. Only 52 of the original 111 patterns found in step 4 can be extended to satisfying patterns of length 326, and there are 648,005 such patterns.

7. In order to re-create the original search space that was compressed in steps 1 and 2, shift all patterns from step 6 to all possible positions in $\{1, 2, \ldots, 1131\}$ and expand to satisfying patterns of length 1131. This yielded 3552 nonidentical satisfying patterns.

8. None of the resulting 3552 satisfying patterns of length 1131 from step 7 were extendable without introducing a monochromatic arithmetic progression of length 6, and therefore $W(2, 6) = 1132$.

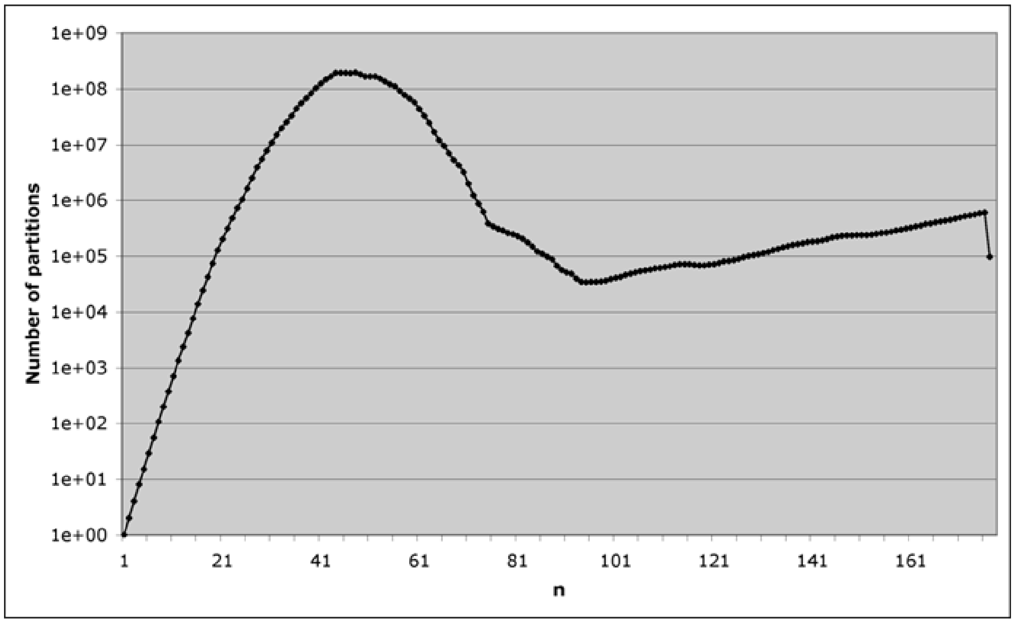

FIGURE 2. Number of 2-colored partitions without monochromatic arithmetic progressions of length 5.

Remark 1.1. During the analysis [Kouril 06] of $W(2,3)$, $W(2,4)$, and $W(2,5)$, the following property was observed: the number of all possible 2-colored patterns without an arithmetic progression of length L (for 3, 4, and 5) grows exponentially in the beginning, eventually hits a peak, and then falls exponentially (see Figure 2). It turns out that as L increases, the peak shifts to the left as a function of the length of the partition measured by a percentage of $W(2,L)$ (see Figure 1). The dip on the other side of the peak eventually levels and starts slowly growing again. The lowest point in the dip indicates that there is only a fairly limited number of satisfying patterns of length m even though m is much smaller than $W(2,L)$. Our approach was partially based on this observation. By collecting all possible patterns near but past the bottom of the dip (in our case of $W(2,6)$ we took $m = 240$, later improved to 326) and extending them to their maximum satisfying length, it became feasible to compute $W(2,6)$.

2. PREPROCESSING: ELIMINATING REDUNDANCIES

During the preprocessing, some of the branches will not be searched, because they are found to be equivalent to other branches. There are two operations on a pattern, namely negation and reversal, that yield equivalent patterns in the sense that the pattern can be extended to a satisfying pattern of a given length if and only if the patterns transformed by either one or both (or neither) of the operations can be so extended to a pattern of the same length. More formally, we consider the following functions:

- negation(p) returns the same pattern with the colors interchanged (negated).

  Example: negation(00001) = 11110.

- reverse(p) returns the same pattern but in reverse.

  Example: reverse(00001) = 10000.

Two patterns are said to be *equivalent* if one can be transformed to the other by applying one or both (or neither) of these two functions. For example, 001 is equivalent to 110, 100, and 011. For a pattern p, we define minimal(p) to be the equivalent pattern having the smallest binary value. For example, minimal(1011) = 0010. Obviously, a pattern p contains an arithmetic progression of length L if and only if minimal(p) contains such a progression. As part of our pattern-expansion technique, this will allow for significant pruning of the search tree. Note that two patterns p_1, p_2 are equivalent if and only if minimal(p_1) = minimal(p_2).

We recognize two types of pattern palindromes:

(a) positive palindromes (identical after reversal);

  Examples: 00100, 1001.

(b) negative palindromes (identical after reversal and negation; requires n to be even);

K	L	Pattern	m
2	L	01/10	$L-1$
2	L	00/11	$2L-1$
2	4	00/11	7
2	4	000/111	21
2	5	00/11	9
2	5	000/111	61
2	5	0000/1111	85
2	6	00/11	11
2	6	000/111	190
2	6	0000/1111	240

TABLE 1. Minimal m for various unavoidable patterns.

Examples: 1100, 101010.

Palindromes will play a critical role in eliminating redundancies.

3. FINDING UNAVOIDABLE PATTERNS

It is sometimes easy to find (K, L, m)-unavoidable patterns. For example, 01 is a (K, L, L)-unavoidable pattern for $L > 2$. Also, 001 is a $(2, L, 2L-1)$-unavoidable pattern for $L > 2$. Finding m for longer patterns becomes increasingly difficult. Using a computer search we established m for selected patterns. The longest pattern found for $K = 2$ and $L = 6$ took several weeks on a single computer. We used a modified SAT solver that backtracked not only when it found a monochromatic arithmetic progression, but also when it found one of the unavoidable patterns. In Table 1, we show the minimal m for which $(2, L, m)$-unavoidable patterns exist.

4. PREPROCESSING PATTERNS: REMOVING REDUNDANT BRANCHES

In exploring the search space, the patterns resulting from the extension of a suitable set of minimal patterns of length m will cover all patterns of length $n = n(m)$ in the sense that all 2-colorings of $\{1, 2, \ldots, n\}$ will contain one of the patterns of length m or their equivalents. In our case the number of extensions to the initial pattern did not increase the covered n beyond the lower bound of $W(2, 6)$.

Given a pattern P, an *expansion* of P consists first in adding a zero or one to the right or left of P, resulting in what we call an *augmented* pattern P'. Then we add two unassigned *padding* variables to the left and to the right of this P'. The addition of the two padding variables allows the expanded pattern to be reversed and still contain the reversal of the augmented pattern. Note that each expansion adds three new variables (one assigned and two unassigned) to a given pattern. This notion of an extension allows for an efficient identification of equivalent branches in the search tree.

To demonstrate the identification of equivalent branches in the search space, we start with a pattern of a single element 0 and extend on the right, with x denoting an unassigned variable:

$$x00x$$
$$x01x$$

Extend on the right again:

$$xx000xx$$
$$xx001xx$$
$$xx010xx$$
$$xx011xx$$

Note that the fourth pattern is equivalent to the second by a reversal and negation, so it can be eliminated, and the three remaining patterns will cover the search space of all patterns of length 7. Here we are not implying that the resulting patterns will be exactly at the middle position. We are merely saying that they will be somewhere in the covered search space with $n \geq 7$. To recover the entire search space we need to shift these patterns and their variations over the entire $\{1, 2, \ldots, n\}$ using negation and reversal. If we consider patterns only in their minimal form and eliminate redundancies assuming an initial pattern length i and m expansions, the resulting set of patterns will cover all patterns of length $n = i + 3 \times m$.

Now assume that there is a sufficient number of unassigned variables on each side and focus only on the expanded patterns. Starting from a single 0, we have a choice to expand this pattern either on the right (a) or on the left (b), leading to the following patterns in minimal form:

(a) $0x \to 00,\ 01$

(b) $x0 \to 00,\ 10 \to 00,\ 01$

Notice that no matter whether you grow the patterns on the left or on the right, the resulting sets of patterns in minimal form are the same. Notice also that the two patterns generated are palindromes. It is easy to see that the same set of minimal patterns results whether a palindrome is expanded on the left or on the right. To illustrate, we go to the next step:

(a) $00x, 01x \to 000, 001, 010, 011 \to 000, 001, 010, 001$

(b) $00x, x01 \to 000, 001, 001, 101 \to 000, 001, 001, 010$

(c) $x00, 01x \to 000, 100, 010, 011 \to 000, 001, 010, 001$

(d) $x00, x01 \to 000, 100, 001, 101 \to 000, 001, 001, 010$

Each set consists of 000, 001, 010. Going yet another step yields

(a) $000x, 001x, 010x \to 0000, 0001, 0010, 0011, 0100, 0101$

(b) $000x, 001x, x010 \to 0000, 0001, 0010, 0011, 0010, 1010$

(c) $000x, x001, 010x \to 0000, 0001, 0001, 1001, 0100, 0101$

(d) $000x, x001, x010 \to 0000, 0001, 0001, 1001, 0010, 1010$

(e) $x000, 001x, 010x \to 0000, 1000, 0010, 0011, 0100, 0101$

(f) $x000, 001x, x010 \to 0000, 1000, 0010, 0011, 0010, 1010$

(g) $x000, x001, 010x \to 0000, 1000, 0001, 1001, 0100, 0101$

(h) $x000, x001, x010 \to 0000, 1000, 0001, 1001, 0010, 1010$

After the transformation we get two different sets of patterns:

(i) 0000, 0001, 0010, 0011, 0101

(ii) 0000, 0001, 0010, 0110, 0101

In general, for every existing pattern we have a choice of extending it either on the left or on the right. The number of patterns to consider can be greatly reduced by a careful choice of the direction in which they are extended, since some of them extend to identical patterns. Finding the optimal way to extend patterns is a hard problem, and looking for the best solution by enumerating all possibilities yielded results for only a few steps. Therefore we came up with a palindrome-based heuristic discussed in the next section.

Figures 3 and 4 show pattern-expansion trees. In Figure 3 the only pattern in which the choice of the direction of expansion matters is 001. It expands either to two patterns to the lower left or to two patterns to the lower right. The other patterns are palindromes, and it never matters for such patterns whether they are expanded on the left or on the right.

Highlighted patterns in Figure 4 indicate which patterns have a choice of expansion on the left or on the right. Unhighlighted patterns are palindromes, which yield the same patterns if expanded on the left or on the right. It is obvious that the number of patterns for which the choice matters increases significantly as the portion of palindromes drops.

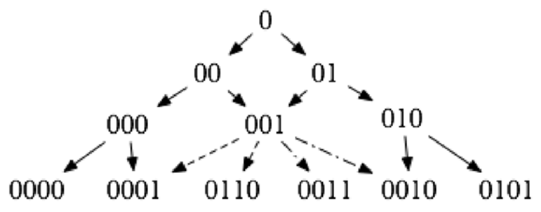

FIGURE 3. Tree of minimal patterns for four numbers.

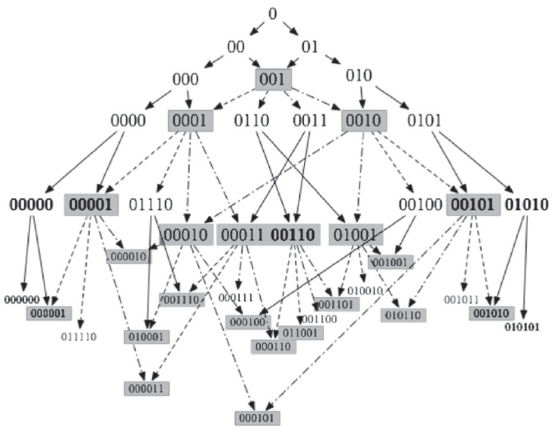

FIGURE 4. Tree of minimal patterns for six numbers.

5. THE PALINDROME HEURISTIC

The best approximation to pattern extension we have seen so far is based on palindrome patterns, whereby an extension on each end of such a pattern yields three new patterns instead of four.

The following is an example of the expansion process:

$$\begin{gathered} x010x \\ 00100 \to 00100 \\ 10101 \to 01010 \\ 00101 \to 00101 \\ 10100 \to 00101 \end{gathered}$$

Since the last two patterns are equivalent, one of them can be eliminated.

Here is another example:

$$\begin{gathered} x0011x \\ 000111 \to 000111 \\ 100110 \to 011001 \\ 000110 \to 000110 \\ 100111 \to 000110 \end{gathered}$$

Again, last two patterns are equivalent, and one of them can be eliminated. Another advantage is that two of the newly formed patterns are again palindromes.

Iteration	Palindromes	Nonpalindromes	Maximum Length	Total Count
0	0	1	5	1
1	2	0	7	2
2	4	3	9	7
3	8	18	11	26
4	14	78	13	92
5	28	319	15	347
6	46	1252	17	1298
7	97	4581	19	4678
8	176	17034	21	17210
9	363	62092	23	62455
10	722	224113	25	224835
11	1218	763984	27	765202
12	2429	2535117	29	2537546

TABLE 2. Counts of patterns during palindrome-based pattern growing.

The nonpalindromes are extended so that they reach a palindrome in the shortest number of extension steps. For example, 0001 is not a palindrome, but if extended on the left it yields a palindrome (10001) and a nonpalindrome (00001).

As demonstrated earlier, when starting with 00001, after one expansion we have safely covered pattern length $n \geq 8$; after the second expansion we have safely covered pattern length $n \geq 11$; and so forth. Since in our case we are doing at most 24 expansions with the initial pattern of length 5, we would safely cover pattern length $n \geq 77$.

In addition, our initial pattern 00001 is unavoidable for $n > 239$. We assume that it can appear anywhere in patterns of length 240 or more. To account for all possible positions we will have to shift the resulting patterns (after crossing the peak; see Remark 1.1) within certain pattern lengths. A safe estimation of this pattern length is based on the fact that we can place the initial pattern at either side of a pattern of length 240 and add the covered search space from the subsequent expansion. Therefore $n > 239 + 77 + 77 = 393$. This can undoubtedly be tightened, but with the lower bound for $W(2, 6)$ being 1131, this will suffice.

To recover the original search space we have only to shift and extend the final set of patterns throughout $\{1, 2, \ldots, n\}$. The shifting–extending phase of the patterns is computationally feasible, since the length of the patterns in our set is sufficient past the peak [Kouril and Franco 05].

6. PUTTING THE PREPROCESSING TOGETHER FOR $W(2,6)$

The longest unavoidable pattern we have verified for $W(2, 6)$ is 0000. This pattern can even be extended to 00001 without loss of generality. Growing this pattern when the lower bound is 1131 gives us plenty of space for a number of iterations of pattern expansion. We grow the initial pattern while the number of patterns is manageable, stopping after the twelfth iteration when the patterns are of length 28 or 29, and result in a total count of 2,537,546 patterns (see Table 2):

1. 00001
2. $100001, 000001$
3. $01000010, 010000010, 00111100, 001111100,$
   $001111010, 001111011, 001111101$
4. ...

To show the dramatic effect of the palindrome heuristic, there is a total of 155,896,884 patterns of length 29 without a monochromatic arithmetic progression of length 6, versus the 2,537,546 found using the heuristic.

7. SAT SOLVER FOR VAN DER WAERDEN $K = 2$ NUMBERS

For convenience we transform the problem of evaluating van der Waerden numbers into an equivalent problem of satisfiability of a certain Boolean expression in conjunctive normal form (CNF). With each integer i in the vector $\{1, 2, \ldots, n\}$ we associate a Boolean variable x_i, and to each 2-coloring of $\{1, 2, \ldots, n\}$ we assign x_i true or false, one value for each color. The CNF associated with a 2-coloring simply enumerates all possible progressions in order to cause a contradiction whenever it occurs. For example, $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5 \vee x_6)$ will cause a contradiction whenever the variables x_1, x_2, x_3, x_4, x_5, and x_6 are assigned false, which is also an arithmetic progression of length 6 on positions 1 to 6. Our solver does not store the problem explicitly in CNF, but implicitly does checking for progressions as though it were stored in CNF.

We tried various algorithms to find the fastest SAT solver that would answer the question whether a given partial assignment can be extended to an assignment of the remaining unknown variables such that there is no monochromatic arithmetic progression of length L. It turned out that the fastest algorithm we found was a DPLL SAT solver restricted to checking for inferences and contradictions. The single function that dominates all other computations checks whether the newly assigned variable is part of a progression (in which case it triggers a contradiction) or is part of an $L-1$ progression with an unknown variable present (in which case it triggers an inference). We eventually used FPGAs to do this checking, which resulted in considerable speedup.

Similar attention for the sake of efficiency was given to finding the best branching heuristic. None of the tested special heuristics yielded any faster result than the static one that began selecting variables on which to branch from the middle outward. If the selected variable was already assigned, it was skipped, and the next one was tried.

8. ARCHITECTURE OF THE FPGA-BASED SAT SOLVER

As mentioned above, our solver searched the search space of 2-colorings of $\{1,2,\ldots,n\}$ for a coloring not containing a monochromatic arithmetic progression of a certain length. Each number has a color, for which we have two registers of length n: C_0 and C_1, one for each color. At the start of the computation, 0 is assigned to both registers for all numbers except those that are part of the initial assignment. An assignment of 1 to either register constitutes a coloring of that number with the respective color. Assigning both registers the value 1 is not a possible state.

Our design consists of the following major blocks (see Figure 5):

1. IB: Inference block
2. CD: Contradiction-detection block (Figure 6)
3. CP: Choice-point-selection Block
4. LB: Logic block
5. RAM: Backtrack memory

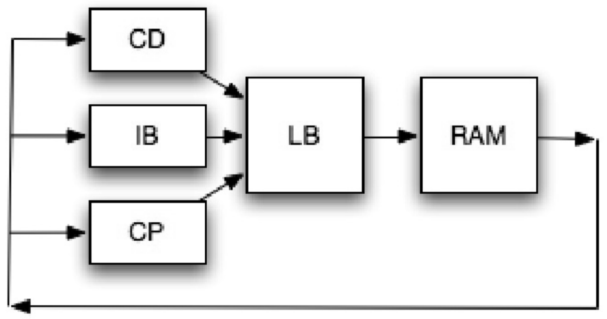

FIGURE 5. The FPGA solver block diagram.

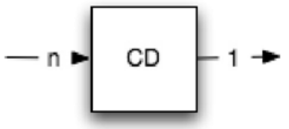

FIGURE 6. Contradiction-detection block.

The CD (contradiction-detection block) has a color vector as input, and the output is true or false depending on whether the input vector contains an arithmetic progression.

For example, if the input is 01111110001, the output will be 1 (true), since positions 2 to 7 contain an arithmetic progression of length 6.

The IB (inference block; see Figure 7) has a color vector as input, and the output is a list of inferences, i.e., a list of colorations forced to occur in the other color.

For example, if the input vector is 111101, it is obvious that we cannot color the fifth position using the same color, and therefore the opposite color is inferred. The output of the IB is 000010.

The CP (choice-point-selection block; see Figure 8) has an input consisting of both color vectors and an output length n. The output will have only one bit set, which corresponds to the next unassigned number. Our solver uses a simple selection of the next choice point by selecting the numbers to color starting from the middle and working its way out. The CP selects the next unassigned number that is closest to the middle ($n/2$).

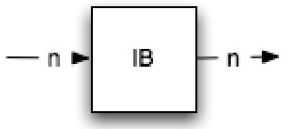

FIGURE 7. Inference block.

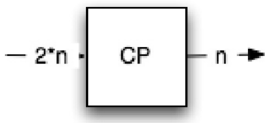

FIGURE 8. The choice-point-selection block.

The LB (logic block) takes the output of all three previously described blocks and does the following. If the assignment contains a contradiction (as would be indicated by the CD block), then backtracking will be triggered, and the previously saved assignment will be retrieved from RAM to be expanded. If there is no assignment in memory, the computation is done, and no assignment without an arithmetic progression is possible.

If the assignment does not contain a contradiction but a new coloring was inferred by the IB block, then the inference is made and set as the new input to all three previously described blocks to check for contradictions and possible additional inferences.

Finally, if the assignment contains neither a contradiction nor new inferences, a new choice point is selected. If all numbers are assigned a color, then the solution has been found. Otherwise, the selected number, colored using the first color, is the current new assignment, and the selected number, colored using the second color, is saved into RAM for future backtracks.

This design simplifies our fastest SAT-solver version by removing the delayed literal evaluation logic while still maintaining excellent speedup over the sequential version and taking advantage of the fine-grained parallelism in an FPGA.

9. PERFORMANCE RESULTS

For illustration, our implementation in C of the solver can prove $W(2,5) = 178$ in 2.613 seconds, provide a partition of length 177 for $W(2,5)$ in 0.091 seconds, and provide all satisfiable assignments for $W(2,5)$ for $n = 177$ in 2.683 seconds on an Intel Pentium 4 running at 3 GHz.

For the $W(2,6)$ search using parallel backtracking we ran the solver on the following clusters:

- 66x AMD Opteron cores running at 1800 MHz;
- 34x AMD Athlon running at 1533 MHz;
- 64x Intel PIII running at 450 MHz;
- around 50 other Intel and Sun Sparc-based processors not exceeding 500 MHz each.

Given the above computational resources, it is our estimate that it would take more than a year to complete the search of all 2,537,546 preprocessed patterns using only cluster-based computation.

The wall processing time (time from start to finish) was about 253 days on clusters that were time-shared with other students and faculty. We had over 200 processors working on the problem at various times. The first half of the search was done using time-shared clusters only (and took about six months), and the second half was done by a combination of time-shared clusters and four dedicated Xilinx Virtex-4 based FPGAs. The estimated time for an FPGA-only computation is three months, which could even be shortened with an improved design or an increased number of FPGAs.

10. EXTREME PARTITIONS

There are exactly 3552 extreme partitions for $W(2,6)$ of length 1131. Each of these partitions involves variations of the basic pattern B of length 56, where

$$B = 01111001000001011110111001111101101110100011101001010011.$$

Letting $A_1 = BB_{\text{rev,neg}}$ and $A_2 = B_{\text{neg}}B_{\text{rev}}$, half of these 3552 patterns are of the form

$$xA_1xA_2xA_1xA_2xA_1xA_2xA_1xA_2xA_1xA_2x,$$

where the 11 "glue" variables x can be assigned $0,1$ values independent of their position in the partition. The only constraint on these glue variables is that the $0,1$ assignment to them not create an arithmetic progression of length 6 (among themselves). One of these extreme partitions is illustrated in Figure 9. The other 1776 extreme partitions for $W(2,6)$ are obtained by negating the above 1776 extreme partitions.

11. CONCLUSION AND FUTURE WORK

Finding the value of van der Waerden numbers presents a challenging problem, since the underlying principle behind their computation is still unknown. Using preprocessing resulted in a significant reduction of the search space, which together with optimized SAT solvers and (eventually) hardware support in the form of FPGAs allowed for the computation of the sixth known van der Waerden number $W(2,6) = 1132$.

We have tested the same approach with $W(2,7)$, and the preprocessing does not reduce the size of the search

```
001111001000001011110111001111101101110100011101001010011
 00110101101000111010001001000001100010000101111110110000 1
010000110111111010000100011000000100100010111000101101011 00
 110010100101110001011101101111100111011110100000010011110
001111001000001011110111001111101101110100011101001010011
 001101011010001110100010010000011000100001011111101100001
01000011011111101000010001100000010010001011100010110101100
 110010100101110001011101101111100111011110100000010011110
001111001000001011110111001111101101110100011101001010011
 001101011010001110100010010000011000100001011111101100001
11000011011111101000010001100000010010001011100010110101100
 110010100101110001011101101111100111011110100000010011110
001111001000001011110111001111101101110100011101001010011
 001101011010001110100010010000011000100001011111101100001
01000011011111101000010001100000010010001011100010110101100
 110010100101110001011101101111100111011110100000010011110
001111001000001011110111001111101101110100011101001010011
 001101011010001110100010010000011000100001011111101100001
01000011011111101000010001100000010010001011100010110101100
 110010100101110001011101101111100111011110100000010011110
1
```

FIGURE 9. An extreme partition for $W(2,6)$.

space sufficiently to be computable at this time. We have also applied the same preprocessing technique to computing multicolor van der Waerden numbers, and in this case the results look promising. It may also be possible to prove longer unavoidable patterns. We believe that with more optimization and clever theoretical work more numbers can be verified.

12. ACKNOWLEDGMENTS

This work has been supported by NSF Grants 9871345 and 0521189, and also by a fellowship grant to Michal Kouril by the Ohio Board of Regents.

REFERENCES

[van der Waerden 27] B. L. van der Waerden. “Beweis einer Baudetschen Vermutung.” *Nieuw Archief voor Wiskunde* 15 (1927), 212–216.

[Beeler and O’Neil 79] M. D. Beeler and P. E. O’Neil. “Some New van der Waerden Numbers.” *Discrete Mathematics* 28 (1979), 135–146.

[Kouril and Franco 05] M. Kouril and J. Franco. “Resolution Tunnels for Improved SAT Solver Performance.” In *Eighth International Conference on Theory and Applications of Satisfiability Testing, St. Andrews, Scotland*, pp. 143–157. Berlin / Heidelberg: Springer, 2005

[Kouril 06] M. Kouril. “A Backtracking Framework for Beowulf Clusters with an Extension to Multi-cluster Computation and SAT Benchmark Problem Implementation.” PhD thesis, University of Cincinnati, 2006.

Michal Kouril, Computer Science Department, 814B Rhodes Hall, University of Cincinnati, Cincinnati, OH 45221-0030 (mkouril@ececs.uc.edu)

Jerome L. Paul, Computer Science Department, 814B Rhodes Hall, University of Cincinnati, Cincinnati, OH 45221-0030 (jerry.paul@uc.edu)

Received May 5, 2007; accepted November 3, 2007.