A computation of D(9) using FPGA Supercomputing

Lennart Van Hirtum[1,2,3], Patrick De Causmaecker[1], Jens Goemaere[1], Tobias Kenter[2,3], Heinrich Riebler[2,3], Michael Lass[2,3], and Christian Plessl[2,3]

[1]KU Leuven, Department of Computer Science, KULAK
[2]Department of Computer Science, Paderborn University
[3]Paderborn Center for Parallel Computing, Paderborn University
e-mail adresses in footnote*

April 2023

Abstract

This paper reports on the first computation the 9^{th} Dedekind Number. This was done by building an efficient FPGA Accelerator for the core operation of the process, and parallelizing it on the Noctua 2 Supercluster at Paderborn University. The resulting value is

286386577668298411128469151667598498812366

This value can be verified in two steps. We have made the data file containing the 490M subresults available upon request, each of which can be verified separately on CPU, and the whole file sums to our proposed value.

1 Introduction

Let us consider the finite set $A = \{1, \ldots, n\}$, which we will call the *base set*, and let us denote the set of subsets of A by $\mathcal{P}(A)$. Dedekind numbers count the number of monotone Boolean functions on $\mathcal{P}(A)$. The set of monotone Boolean functions with respect to inclusion on $\mathcal{P}(A)$ is denoted by $\mathcal{D}_n$. The number of such monotone Boolean functions is denoted by $D(n)$ and this is called *the* n^{th} *Dedekind number*.

*lennart.vanhirtum@gmail.com, patrick.decausmaecker@kuleuven.be, jens.goemaere@kuleuven.be, kenter@uni-paderborn.de, heinrich.riebler@uni-paderborn.de, michael.lass@uni-paderborn.de, christian.plessl@uni-paderborn.de

The set of permutations of the elements of base set A generates an equivalence relation on $\mathcal{D}_n$. The set of equivalence classes of this relation are denoted by $\mathcal{R}_n$ and the number of such equivalence classes is denoted by $R(n)$.
Richard Dedekind first defined the numbers $D(n)$ in 1897 [1]. Over the previous century, Dedekind numbers have been a challenge for computational power in the evolving domain of computer science. Computing the numbers proved exceptionally hard, and so far only formula's with a double exponential time complexity are known. Until recently, the largest known Dedekind number was $D(8)$. In this paper, we report on a computation of $D(9)$. Table 1 shows the known numbers, including the result of our computation. As we explain below, some uncertainty about the correctness of the number existed at the time of the first computation and we planned a verification run. In the mean time however, results of an independent computation were reported [2], confirming our result. Since computational methods as well as hardware implementation differ significantly between the two computations, we can conclude that the result is correct with a probability very close to 1.
Table 2 shows the known numbers $R(n)$ of equivalence classes of monotone Boolean functions under permutation of the elements of the base set. Note that the last result dates from 2023.

D(0)	2	Dedekind (1897)
D(1)	3	Dedekind (1897)
D(2)	6	Dedekind (1897)
D(3)	20	Dedekind (1897)
D(4)	168	Dedekind (1897)
D(5)	7581	Church (1940)
D(6)	7828354	Ward (1946)
D(7)	2414682040998	Church (1965)
D(8)	56130437228687557907788	Wiedemann (1991)
D(9)	286386577668298411128469151667598498812366	Our result (2023)

Table 1: Known Dedekind Numbers [3] and our first result.

R(0)	2	
R(1)	3	
R(2)	5	
R(3)	10	
R(4)	30	
R(5)	210	
R(6)	16353	
R(7)	490013148	Tamon Stephen & Timothy Yusun (2014) [4]
R(8)	1392195548889993358	Bartłomiej Pawelski (2021) [5]
R(9)	7892046358420350405277408463002526 80	Bartłomiej Pawelski (2023) [6]

Table 2: Known Equivalence Class Counts

For clarity of this paragraph, let us assume that we consider monotonically decreasing Boolean functions. Note that a monotonically decreasing Boolean function is completely defined by the set of sets which are maximal among the sets for which the function value is true. For any monotone Boolean function, no two of its maximal sets include one another. Such a set of sets is called an anti-chain. A monotone Boolean function is completely determined by its associated anti-chain, and any anti-chain is completely determined by its associated monotone Boolean function. We will use any of the two representations whichever is more convenient. We will represent monotone Boolean functions or anti-chains by letters from the Greek alphabet. If we say that $X \in \alpha$, we mean that X is a maximal set among the sets for which α is $True$, in other words

$$\forall Y \subseteq X : \alpha(Y) = True \; and \; \forall Z \supsetneq X : \alpha(Z) = False$$

If we say that $\alpha = \{X, Y, Z\}$, we mean that the sets $X, Y, Z \subseteq A$ are the maximal sets among the sets for which α is $True$. For the set D_n of monotone Boolean functions on the base set a natural partial order $\leq$ is defined by

$$\forall \alpha, \beta \in D_n : \alpha \leq \beta \Leftrightarrow \forall X \subseteq A : \alpha(X) \Rightarrow \beta(X) \tag{1}$$

This partial ordering defines a complete lattice on D_n. We denote by $\bot$ and $\top$ the smallest, respectively the largest, element of D_n:

$$\forall X \subseteq A : \bot(X) = False, \top(X) = True \tag{2}$$

$$\bot(X) = \{\}, \top(X) = \{A\} \tag{3}$$

Intervals in D_n are denoted by

$$\forall \alpha, \beta \in D_n : [\alpha, \beta] = \{\chi \in D_n : \alpha \leq \chi \leq \beta\} \tag{4}$$

For $\alpha, \beta \in D_n$, the *join* $\alpha \vee \beta$ and the *meet* $\alpha \wedge \beta$ are the monotone Boolean functions defined by

$$\forall X \subseteq A : (\alpha \vee \beta)(X) = \alpha(X) \; or \; \beta(X) \tag{5}$$

$$\forall X \subseteq A : (\alpha \wedge \beta)(X) = \alpha(X) \; and \; \beta(X) \tag{6}$$

Finally, in the formulas below, a number defined for each pair $\alpha \leq \beta \in D_n$ plays an important role. We refer to this number as the *connector number* $C_{\alpha,\beta}$ of α and β. It counts the number of connected components of the anti-chain β with respect to α. Two such sets $X, Y \in \beta$ are connected if $\alpha(X \cap Y) = False$ or if there is a path $X, Z_1, ..., Z_n, Y$ of such subsets $X, Z_1, ..., Z_n, Y \subseteq A$ in which for every two subsequent sets $\alpha(X \cap Z_1) = \alpha(Z_1 \cap Z_2) = ... = \alpha(Z_n \cap Y) = False$. It turns out that the number of solutions of

$$\chi \vee \upsilon = \beta \tag{7}$$

$$\chi \wedge \upsilon = \alpha \tag{8}$$

for $\chi, \upsilon \in D_n$ is given by $2^{C_{\alpha,\beta}}$. This is called the *P-Coefficient* [7, 8].

2 Method, Theory

We start from the original P-Coefficient Formula as taken from [7].

$$D(n+2) = \sum_{\alpha,\beta \in D_n} |[\bot, \alpha]| 2^{C_{\alpha,\beta}} |[\beta, \top]| \tag{9}$$

In the master thesis of the first author of the current paper, Lennart Van Hirtum [9], the author reworked this formula to a form making use of equivalence classes to reduce the total number of terms.

$$D(n+2) = \sum_{\alpha \in R_n} |[\bot, \alpha]| D_\alpha \sum_{\substack{\beta \in R_n \\ \exists \delta \simeq \beta : \alpha \leq \delta}} |[\beta, \top]| \frac{D_\beta}{n!} \sum_{\substack{\gamma \in Permut_\beta \\ \alpha \leq \gamma}} 2^{C_{\alpha,\gamma}} \tag{10}$$

The $Permut_\beta$ term is the collection of all $n!$ equivalents of β under permutation of the base set. D_β is the number of different equivalents, and hence, $Permut_\beta$ contains duplicates iff $D_\beta < n!$. These duplicates are divided out by the $\frac{D_\beta}{n!}$ factor.
For $D(9)$, this means iterating through D_7. That would require iterating over an estimated $4.59 * 10^{16}$ α, β pairs. The total number of P-Coefficients ($C_{\alpha,\gamma}$) that needed to be computed was $1.148 * 10^{19}$. However we were able to improve on this further using the process of 'deduplication', where we can halve the total amount of work again, by noticing that pairs of α, β give identical results to their dual pair $\overline{\beta}, \overline{\alpha}$. As per Equation 11. This allowed us to halve the total amount of work to $5.574 * 10^{18}$ P-Coefficients. [1]

$$|[\bot, \alpha]| 2^{C_{\alpha,\beta}} |[\beta, \top]| = |[\overline{\alpha}, \top]| 2^{C_{\overline{\beta},\overline{\alpha}}} |[\bot, \overline{\beta}]| \tag{11}$$

3 Computing P-Coefficients on FPGA

Computing P-Coefficients is uniquely well-suited for hardware implementation. Computing these terms requires solving the problem of counting the number of distinct connected components within a standard graph structure. An example of such a graph with its distinct connected components colored is shown in Figure 1. The standard depth first search algorithm for this problem is linear in the sum of the number of vertices (sets) and the number of edges between these vertices. Given the number of P-coefficients to be evaluated, it is clear that traditional instruction-based computing methods, particularly Single Instruction Multiple Data (SIMD), fare poorly on it. Since counting connected components in such fixed-sizes graphs (in this case 128-node 7-d hypercubes) consists almost purely of plain Boolean operations, it translates very well to a hardware implementation and provides a highly efficient implementation of the

[1] We made sure not to deduplicate pairs that were their own dual, ie when $\beta = \overline{\alpha}$

algorithm. A simple schematic implementation is shown in Figure 2. A detailed explanation of how it works is provided in the first author's master thesis [9]. In this thesis, some optimizations are derived that bring the average number of iterations down to 4.061. This corresponds to the number of cycles in the hardware design.

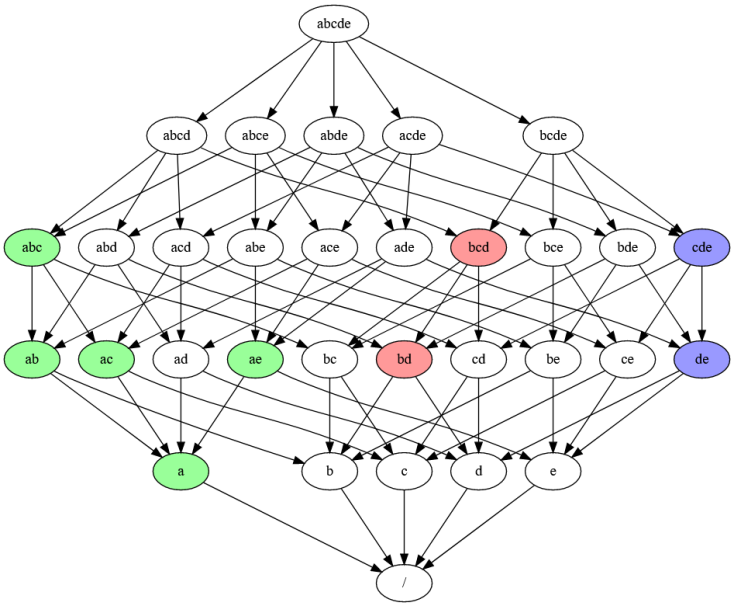

Figure 1: Connected components of an example graph. In this case there are 3 connected components.

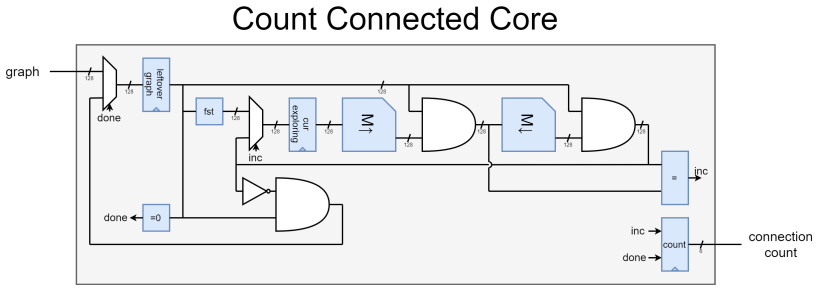

Figure 2: Register Transfer Level Design of the CountConnected Core

4 Computation on Noctua 2

We implemented this hardware accelerator on the Intel Stratix 10 GX 2800 cards found in Paderborn University's Noctua 2 supercomputer. We were able to fit 300 of these CountConnected Cores on a single field-programmable gate array (FPGA) die. These CountConnected Cores run at 450MHz. This gives us a throughput of about 33 Billion CountConnected operations per second. At this rate, a single FPGA processes about 5.2 α values per second, taking 47'000 FPGA hours to compute D(9) on Noctua 2, or about 3 months real-time.
The computation is split across the system along the lines of Equation 10. α values (also named tops) are divided on the job level. There are 490M tops to be processed for D(9). We split these into 15000 jobs of 30000 tops each. The β values per top (also named bottoms) are placed in large buffers of 46M bots on average, and sent over PCIe (Peripheral Component Interconnect Express) to the FPGA. The FPGA then computes all 5040 permutations (γ) of each bottom, computes and adds up their P-Coefficients. This result is stored in an output buffer of the same size.
The artifact of this computation is a dataset with an intermediary result for each of the 490M α values. Each of these can be checked separately[2], and the whole file sums to 28638657766829841112846915166759849881236 6.

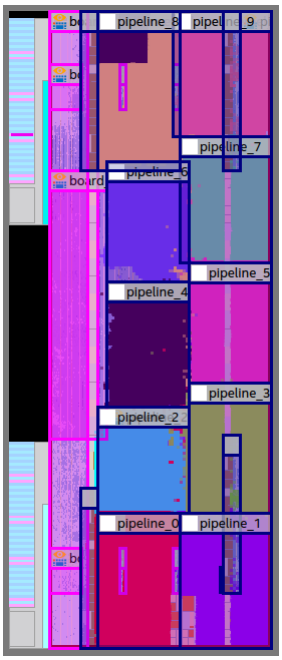

Figure 3: The FPGA Accelerator Die

[2]It takes about 10-200s to compute a single α result on 128 AMD Epyc CPU cores

5 Correctness

As much of the code as possible is written generically. This means the same system is used for computing D(3) - D(8). All of these yield the correct results. Of course, the FPGA kernel is written specifically for D(9) computation, so its correctness was verified by comparing its results with the CPU results for a small sample. In effect, both methods verified each other's correctness.
We did apply a number of additional checks to increase our confidence in the result:

- The most direct is the $D(9) \equiv 6 \ mod \ 210$ check provided by Pawelski & Szepietowski [10]. Our result passes this check. Sadly, due to the structure of our computation, nearly all terms are divisible by 210, which strongly hampers the usefulness of this check. One thing that this check does give us is that no integer overflow has occurred, which was an important concern given we were working with integers of 128 and 192 bits wide.

- Our computation was plagued by one issue in particular. Namely that there is a bug in the vendor library for communication over PCIe, wherein, occasionally and at a low incidence rate, full 4K pages of FPGA data are not copied properly from FPGA memory to host memory. This results in large blocks of incorrect bottoms for some tops. We encountered this issue in about 2300 tops. We were able to mitigate this issue by including extra data from the FPGA to host memory, namely the 'valid permutation count'. By checking these values, we could determine if a bottom buffer had been corrupted. Additionally, adding all of these counts yields the value for D(8), which shows that the correct number of terms have been added.

- Finally, there is an estimation formula, which gives us an estimation which is relatively close to our result. The Korshunov estimation formula estimates $D(9) = 1.15 * 10^{41}$ which is off by about a factor 2. [3]

6 The danger of SEU events

The one way our result could have still been wrong was due to a Single Event Upset (SEU), such as a bitflip in the FPGA fabric during processing, or a bitflip during data transfer from FPGA DDR memory to Main Memory.
It is difficult to characterise the odds of these SEU events. The expected number of occurrences for the FPGAs we used are not available to the best of our knowledge. But example values shown on Intel's website pin the error rate at around 5000 SEU events per billion FPGA hours. In that case, given our 47000 FPGA hours, we expected to see 0.235 errors Poisson distributed, giving us a

[3] This isn't too unusual though, as the results for odd values are off by quite a lot. Estimation for D(3) overestimates by a factor 2, D(5) also overestimates by a factor 2, and D(7) overestimates roughly 10%

chance of 20% of a hit. Of course, this is just an example and the real odds might be have been higher than that.
But, given that we have Jäkel reaching an identical result [2], the odds of a stochastic error affecting both implementations in exactly the same way are so astronomically small, that we can rule them out.

7 Conclusion

In conclusion, our method for computing D(9) works, our implementation should theoretically give the correct result. All that remains is: Have any bit errors occurred during this first computation? Our plan was to start up a second run. Each subresult would have been computed a second time, and any values that differ could be recomputed a third time as a tiebreaker. On April 4^{th} however, a preprint claiming D(9) was published, right before the present publication by Christian Jäkel [2]. This paper confirmed our result as we obtained it on the 8^{th} of March. So, the 9^{th} Dedekind Number was found on the 8^{th} of March, 2023 using the Noctua 2 supercluster at Paderborn University. This value was registered in the corresponding github commit: https://github.com/VonTum/Dedekind/commit/1cf7b019afca655586e8210f97fbb5399d61e842 All code is available at https://github.com/VonTum/Dedekind.

References

[1] R. Dedekind. Über Zerlegungen von Zahlen Durch Ihre Grössten Gemeinsamen Theiler, pages 1–40. Vieweg+Teubner Verlag, Wiesbaden, 1897.

[2] Christian Jäkel. A computation of the ninth dedekind number, 2023.

[3] Doug Wiedemann. A computation of the eighth dedekind number. https://link.springer.com/article/10.1007%2FBF00385808, 1991.

[4] Tamon Stephen and Timothy Yusun. Counting inequivalent monotone boolean functions. Discrete Applied Mathematics, 167:15–24, 2014.

[5] Bartłomiej Pawelski. On the number of inequivalent monotone boolean functions of 8 variables, 2021.

[6] Bartłomiej Pawelski. On the number of inequivalent monotone boolean functions of 9 variables, 2023.

[7] Patrick De Causmaecker and Stefan De Wannemacker. On the number of antichains of sets in a finite universe, 2014.

[8] Patrick De Causmaecker, Stefan De Wannemacker, and Jay Yellen. Intervals of antichains and their decompositions, 2016.

[9] Lennart Van Hirtum. A path to compute the 9th dedekind number using fpga supercomputing. https://hirtum.com/thesis.pdf, 2021. KU Leuven, Masters Thesis.

[10] Bartlomiej Pawelski and Andrzej Szepietowski. Divisibility properties of dedekind numbers, 2023.