

Discovering Optimal Constant Matrix Multiplication Circuits with Boolean Satisfiability

Nicolai Fiege^{ID}, Peter Zipf^{ID}

Abstract—When designing arithmetic circuits, the multiplication of a variable by a constant number can be performed by a series of add/subtract, and shift operations instead of a multiplication (e.g., $7x = 2^3x - x = (x \lll 3) - x$). This allows for significant reductions in complexity of multiplication circuits, as adders are typically much more hardware-efficient than a generic multiplier, and shift operations can be performed without any overhead by appropriately connecting the concerned signals. This concept extends to the multiplication of a constant matrix by a vector of variables, known as constant matrix multiplication (CMM). So far, there exists no way of discovering CMM algorithms with provably minimal add/subtract count. Here we show how the CMM problem can be reduced to a series of Boolean Satisfiability (SAT) problems, enabling the use of powerful SAT solvers to determine optimal solutions for arbitrary matrices. Modeling the problem in a closed mathematical framework allows us to straightforwardly extend our algorithm towards secondary objectives, namely word-size reductions of the add/subtract operations and pipelining for throughput maximization. Compared to state-of-the-art heuristic methods we are consistently able to achieve improvements regarding the resulting implementation complexity, even for small but practical matrix sizes such as 2×2 or 3×3 . These results enable a reduction in implementation costs for a wide range of practical applications such as digital filters, convolutional cores in artificial neural networks, or the multiplication by complex constants in discrete transforms like the Fast Fourier Transform.

concept to the multiplication of a matrix of constants by a vector of variables

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_U \end{bmatrix} = \begin{bmatrix} C_{1,1} & C_{1,2} & \cdots & C_{1,V} \\ C_{2,1} & C_{2,2} & \cdots & C_{2,V} \\ \vdots & & \ddots & \vdots \\ C_{U,1} & C_{U,2} & \cdots & C_{U,V} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_V \end{bmatrix} \quad (1)$$

called constant matrix multiplication (CMM), contemporary algorithms run into numerous problems. Tabulating optimal solutions for all possible problems is infeasible due to the large problem space, even for a given word size. Specialized algorithms are utilized to compute circuits for the multiplication of a single input with a given set of constants [1]–[15], called Multiple Constant Multiplication (MCM). For the general case of CMM, however, only a few heuristic methods with unknown quality of results (QoR) exist [16]–[22].

To address these shortcomings, our contributions are the following:

- 1) We propose OatMeal², the first method to find shift-and-add-based CMM algorithms with minimum adder count using SAT solvers (Section II-C–II-E).
- 2) We extend the SAT model towards (i) pipelining for throughput maximization and (ii) bit-level costs for a minimization of adder/register word lengths (Section II-F–II-H).
- 3) We experimentally validate the proposed algorithm on realistic test instances and show superiority over the state-of-the-art (Section III–IV).

I. INTRODUCTION

IN this work we propose the first optimal method to discover multiplierless circuits for matrix-vector multiplications, where the matrix is made up of constants. For a given matrix of constants, the proposed approach repeatedly constructs and solves Boolean Satisfiability problems to determine a circuit and to also prove that no better circuit (in terms of circuit complexity) exists.

Implementing digital constant multiplication circuits solely with shift and add/subtract operations has the advantage that bit shifts can be realised by re-wiring and additions/subtractions are cheap to implement compared to costly multipliers¹. However, the problem of finding such circuits has been an active research topic for several decades.

The most basic case (i.e., multiplying a variable by a scalar) is solved for coefficient word sizes up to 32 bits [1], which suffices for most practical applications. When generalizing this

II. SAT-BASED CMM DISCOVERY

Formally, we seek to solve the following optimization problem:

Optimization problem 1 (CMM). *Given a matrix of constants $C \in \mathbb{N}^{U \times V}$. Find a shift-and-add realization with minimum adder count N^* .*

Instead of solving the CMM problem directly, we instead solve a series of related decision problems, called dCMM.

Decision problem 1 (dCMM). *Given a matrix of constants $C \in \mathbb{N}^{U \times V}$ and a candidate adder count $N \in \mathbb{N}$. Decide whether a shift-and-add realization exists with at most N adders.*

A realization with N^* adders is a solution to the CMM problem if and only if the dCMM problem is satisfiable for N^* and unsatisfiable for $N^* - 1$ adders.

²Optimal Constant Matrix Multiplication using Boolean Satisfiability

N. Fiege, and P. Zipf are with the University of Kassel, Germany. Manuscript received ??? ?? ???; revised ??? ?? ???.

¹In the following, we assume that implementation costs for adders and subtractors are roughly equal. Hence, we refer to both of them as “adder”.

83 We solve all dCMM problems by modeling the resulting
 84 circuit as a Boolean Satisfiability (SAT) problem, leaving
 85 all implementation-related parameters (e.g., shift values and
 86 connections between adders) as decision variables to be deter-
 87 mined by the solver.

88 **Decision problem 2 (SAT).** *Given a function f of Boolean*
 89 *variables $x_1 \dots x_n$ in conjunctive normal form (CNF). Deter-*
 90 *mine whether there exists a variable assignment ϕ such that*
 91 *f evaluates to true.*

92 If the corresponding SAT problem is satisfiable for N
 93 adders, we can reconstruct a solution to the dCMM problem
 94 from the variable assignment returned by the solver. If it is
 95 unsatisfiable for N adders, we know that a solution to the
 96 CMM problem has an adder count of at least $N + 1$. As a
 97 trivial search procedure, we could start with $N = 0$ adders,
 98 solve the corresponding dCMM problem, and increase N by
 99 one every time the SAT solver reports *unsatisfiable*. That way,
 100 the first solution encountered is guaranteed to be optimal.

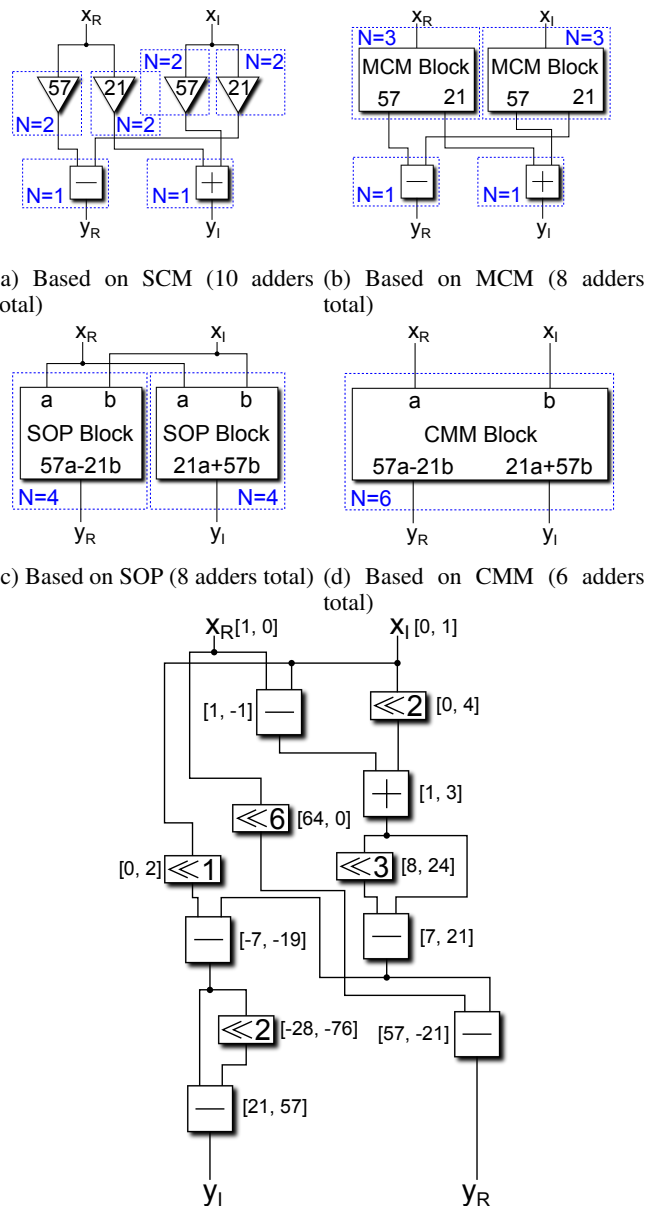
101 **A. Example**

102 As an example consider the complex-valued multiplication
 103 $y = (57 + j21) \cdot x$, where j represents the imaginary unit
 104 with $j^2 = -1$. The complex multiplication can be re-written
 105 in matrix form to

$$\begin{bmatrix} y_R \\ y_I \end{bmatrix} = \begin{bmatrix} 57 & -21 \\ 21 & 57 \end{bmatrix} \cdot \begin{bmatrix} x_R \\ x_I \end{bmatrix} \quad (2)$$

106 where x_R and y_R represent real parts and x_I and y_I represent
 107 imaginary parts. Fig. 1 shows various possible implementa-
 108 tions resulting from different ways of solving the underlying
 109 CMM problem.

110 A naive approach is based on single constant multiplication
 111 (SCM). The circuit computes all products separately using a
 112 total of ten adders; the multiplication circuits by 57 and 21 are
 113 shown in the appendix, in Fig. 10a and Fig. 10b, respectively.
 114 A more sophisticated approach groups the multiplications by
 115 57 and 21 into dedicated multiple constant multiplication
 116 (MCM) blocks. By doing so, only eight instead of ten adders
 117 are needed for an implementation, because the intermediate
 118 result $7x = (x \ll 3) - x$ can be used to compute both 57 and
 119 21 via $21x = (7x \ll 1) + 7x$ and $57x = (x \ll 6) - 7x$
 120 (cf. Fig. 11). Re-ordering the computation into a sum of
 121 products (SOP) form for y_R and y_I does not bring any benefit
 122 compared to MCM (cf. Fig. 12). Only when tackling the CMM
 123 problem as a whole, it is possible to reduce implementation
 124 costs to six adders. Next to each shifter and adder we give
 125 the linear combination of the inputs computed at the given
 126 position in the circuit. For example, $[1, -1]$ next to the top-
 127 most subtracter means that it computes $1 \cdot x_R - 1 \cdot x_I$. These
 128 lists can be interpreted as vectors, where inputs represent base
 129 vectors. We call these vectors “fundamental” in the following.
 130 In order for a CMM circuit to be valid, each row vector of
 131 the given matrix must be present as a fundamental within the
 132 circuit.



(e) CMM solution in detail. Next to each circuit element we give the linear combination of x_R and x_I computed at its output.

Fig. 1: Different solutions to $y = (57 + j21) \cdot x$, where $x = x_R + jx_I$ and $y = y_R + jy_I$. As a convention, subtracters always subtract the right input. Detailed adder graphs are given in the appendix.

TABLE I: Overview of special CMM cases

	$U = 1$	$U > 1$
$V = 1$	Single constant multiplication	Multiple constant multiplication
$V > 1$	Sum of products	Constant matrix multiplication

133 **B. Related work**

134 See Table I for an overview of special CMM cases regarding
 135 the matrix dimensions U and V . So far there only exist
 136 heuristic algorithms to solve the general CMM problem (i.e.,
 137 $U > 1$ and $V > 1$).

SCM (i.e., $U = 1$ and $V = 1$) is solved optimally regarding the number of additions up to 32 bits [1]. A database for bit-level optimized SCM solutions up to 16 bits was created by Bierlee et al. for unsigned arithmetic with the goal of efficient SAT encodings for constant multiplications arising in constraint satisfaction problems such as the multi-dimensional knapsack problem [23]. However, this table might not be optimal for all coefficients since previous work has shown that allowing negative fundamentals can sometimes yield even cheaper solutions due to non-symmetric bit-level costs for subtractions [14].

Whenever a single input is multiplied with several constants (i.e., $V = 1$ and $U > 1$), intermediate results can be shared to reduce the computational overhead even further [2]. This occurs, for example, in digital filter implementations [5], [24]–[30]. Similar to SCM, state-of-the-art optimal algorithms exist for solving MCM problems of practical interest [4], [8], [10], [14], [31]. Hardware-aware extensions for bit-level optimizations via SAT [14] or ILP [10] or for pipelined realizations [31] are available in literature. Fiege et al. showed that SAT-based algorithms translate to larger coefficient word sizes where ILP-based algorithms run into numerical issues and the search space explodes for custom branch-and-bound methods [14].

Aside from good results for practical applications (cf. Section III), the reduction of constant multiplication problems to sequences of SAT instances is also theoretically justified. This is because the CMM, SOP, and MCM problems generalize the SCM problem, whose decision variant is known to be NP-complete [32], [33]. By choosing a solver from the same complexity class, we maintain the theoretical soundness of the proposed approach.

Contemporary algorithms which support multiple inputs (i.e., $V > 1$) for solving SOP ($U = 1$) or CMM ($U > 1$) [34] struggle when increasing the number of inputs [21]. This can, for example, be circumvented by transposing the matrix whenever $V > U$ and afterwards transposing the result to obtain a solution for the original problem [20], [21]. That way, MCM algorithms can be used for solving SOP instances and solving time for difficult CMM instances can be reduced. Still, only heuristic algorithms exist for the general case of CMM [17], [19], [22], [34], [35].

C. The base model for CMM

Voronenko and Püschel introduced the scalar \mathcal{A} operation

$$\mathcal{A}_q(u, v) = |2^{l_1}u + (-1)^s 2^{l_2}v|2^{-r} \quad (3)$$

to represent an adder in the resulting circuit [2]. Here we use a modified version of Kumm et al.'s extension to vector valued inputs \vec{u} and \vec{v} [34]

$$\vec{\mathcal{A}}_q(\vec{u}, \vec{v}) = (2^{l_1}\vec{u} + (-1)^s 2^{l_2}\vec{v})2^{-r} \quad (4)$$

with $l_1, l_2, r \in \mathbb{N}_0$, $l_1 = 0 \vee l_2 = 0$, $s \in \{0, 1\}$. Subscript $q = (l_1, l_2, r, s)$ represents the configuration of that adder. The parameters l_1 and l_2 are left shifts at one of the inputs, s is used to distinguish addition and subtraction and r is a right shift at the output of the adder. In order to preserve all

bits within the resulting circuit, r must be chosen such that 2^r divides each vector element in $(2^{l_1}\vec{u} + (-1)^s 2^{l_2}\vec{v})$.

In this work, we model the \mathcal{A} -operation via SAT variables and clauses. Additionally, we also add clauses to the SAT formula to ensure the following:

- Inputs $\vec{c}^{(0)} \dots \vec{c}^{(V-1)}$ are equal to the respective unit vectors.
- For each row vector in the input matrix $\vec{C}^{(u)}$, there exists at least one fundamental $\vec{c}^{(i)}$, computed by a respective \mathcal{A} operation with equal values.
- No overflows occur in left shifter or adder in order to preserve correctness of the computation.
- Only zeros are shifted out of the right shifter to preserve all bits in the resulting circuit.
- In case of pipelining, we require that all outputs arrive in the same pipeline stage (depending on the application, this constraint can also be omitted).

Refer to Table II for a complete overview of our notation.

D. Overall algorithm

Figure 2 shows an overview of the proposed algorithm. We assume that all input matrices C are properly normalized according to the procedure described by Kumm et al. [34]. OatMeal mainly consists of two loops. The first one determines the optimal adder count (i.e., a solution to the CMM problem), starting at a lower bound for the adder count, N_{LB} , which is incremented each time the SAT solver reports unsatisfiability. We use the trivial lower bound $N_{LB} = U$, but note that more sophisticated lower bounds can be chosen, for example the one proposed by Gustafsson [36]. That way, the first satisfiable instance is guaranteed to have the optimal adder count N^* . Then, the adder graph is extracted from the satisfying assignment given by the SAT solver and its bit-level costs \hat{B} are determined.

Afterwards, the second loop decreases \hat{B} until the first unsatisfiable instance is reached, at which point the last satisfiable solution was optimal regarding bit-level costs for the optimal adder count. Note that the \hat{B} does not have to be decreased by one in each iteration. In some cases, the new adder graph decreases bit-level costs by more than one compared to the previous solution, hence, \hat{B} should be modified accordingly.

The dCMM solved in each iteration consists of checking whether the configurable circuit shown in Fig. 3 is able to compute the requested constant multiplication. The configuration for each adder node is determined by the corresponding $\vec{q}^{(V)} \dots \vec{q}^{(V+N-1)}$ input. This can be seen as a sort of equivalence checking problem, where the inputs $\vec{c}^{(0)} \dots \vec{c}^{(V-1)}$ are constrained to the respective unit vectors and it is required that each row vector $\vec{C}^{(u)}$ in the input matrix is produced by at least one output $\vec{c}^{(V)} \dots \vec{c}^{(V+N-1)}$. Formally, a configuration for a given adder count is valid whenever the following condition holds:

$$\forall u \in \llbracket 0, U - 1 \rrbracket \exists n \in \llbracket 0, N - 1 \rrbracket : \vec{c}^{(V+n)} = \vec{C}^{(u)}. \quad (5)$$

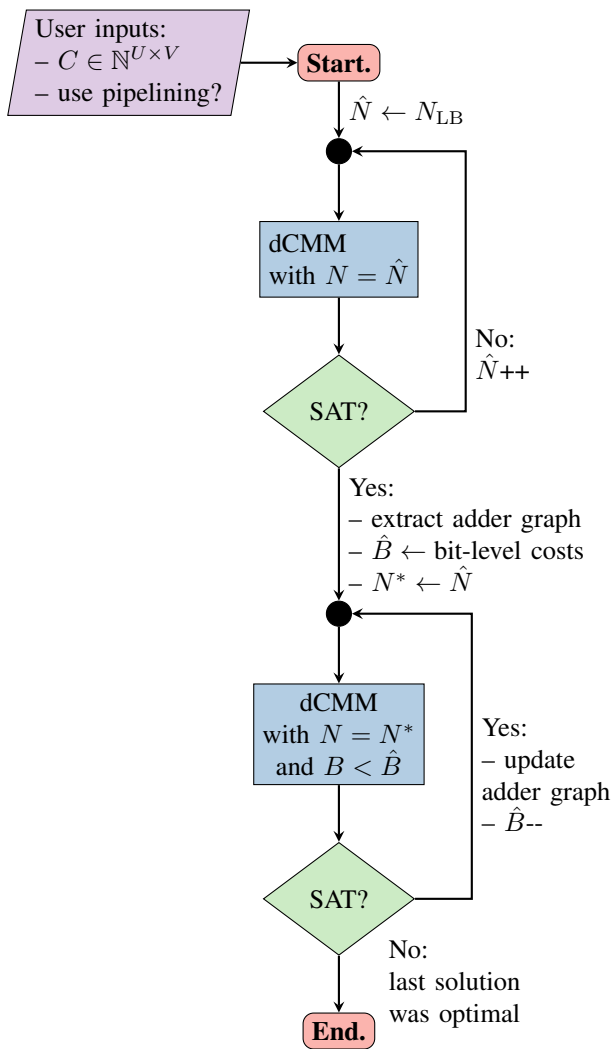


Fig. 2: Flowchart for OatMeal

242 E. SAT model for the \mathcal{A} -operation

243 Fig. 4 shows the corresponding circuit model to represent
 244 the i^{th} $\vec{\mathcal{A}}$ operation, which we then model as a CNF formula
 245 as an input to the SAT solver. Our model is based on the
 246 SAT model from previous work [14]. It is extended towards
 247 CMM-related intricacies (such as vector-valued coefficients)
 248 as explained in the following.

249 In order to prohibit cyclic dependencies, we only allow
 250 inputs connected to any adder with a lower index (i.e., adder i
 251 can only be connected to adders $0 \dots i-1$). The value $\vec{c}^{(i)}$ is a
 252 vector-valued representation of the fundamental and it depends
 253 on the fundamentals from preceding nodes and the values
 254 chosen for $\alpha^{(i)}$, $\beta^{(i)}$, $\gamma^{(i)}$, $\delta^{(i)}$, $\varepsilon^{(i)}$, and $\zeta^{(i)}$ (constituting
 255 the \vec{q} inputs in Fig. 3). All values are represented as two's
 256 complement integers. Hence, the XOR gate at adder input
 257 B , together with bit $\varepsilon^{(i)}$, controls whether an addition or a
 258 subtraction is performed. Multi-input multiplexers to select $\vec{l}^{(i)}$
 259 and $\vec{r}^{(i)}$ are built as trees made up of two-input multiplexers.
 260 The left- and right-shifters are also MUX-based and the
 261 adder/subtractor is modeled as a ripple carry adder made up
 262 of full and half adders. Each circuit element is modeled using

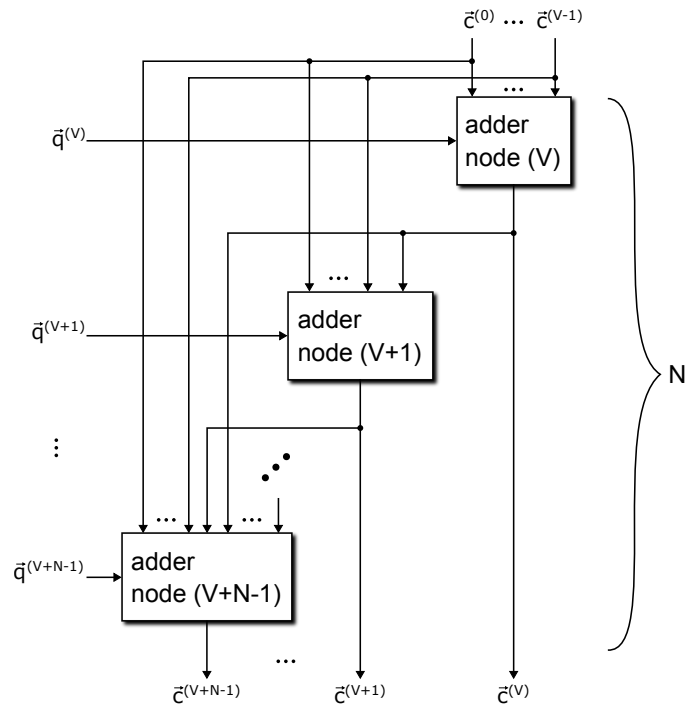


Fig. 3: Overall circuit model

the clauses described by Fiege et al. in previous work [14].

F. Extension to adder depth

265 Adder depth is a common secondary optimization goal
 266 when designing constant multiplication circuits [34], [37],
 267 [38]. It is a metric of how many addition operations are
 268 performed in series within the circuit. For example, the circuit
 269 shown in Fig. 1e has an adder depth of five because five
 270 consecutive addition operations must be performed to compute
 271 $y_l: [1, -1] \rightarrow [1, 3] \rightarrow [7, 21] \rightarrow [-7, -19] \rightarrow [21, 57]$. Large
 272 adder depths generally come with high power consumption
 273 (due to a high number of glitches) and a low maximum clock
 274 frequency (due to long combinational path delays).

275 The depth $d^{(i)}$ for a given adder depends on the depth of
 276 its inputs $d^{(u)}$ and $d^{(v)}$ and is calculated via

$$d^{(i)}(u, v) = \max\{d^{(u)}, d^{(v)}\} + 1. \quad (6)$$

277 We can also express this computation as additional clauses in
 278 the SAT formula, and, optionally, limit it to a given value or
 279 minimize it for the optimal adder count as the second objective
 280 (e.g., by inserting an additional loop at the end of the flow
 281 chart shown in Fig. 2, that decreases the adder depth until it
 282 encounters the first unsatisfiable result, indicating that the last
 283 solution had optimal adder depth).

284 Figure 5 shows a circuit model to implement a max-
 285 operation via clauses for an exemplary bit width of four
 286 bits. Each block titled “clauses” represents an identical set
 287 of clauses (corresponding to a bit-slice in a circuit implementa-
 288 tion). The carry signal $y_w^{(i)}$ denotes whether it was possible
 289 to determine the larger input at any bit position greater than
 290 w , while $x_w^{(i)}$ denotes which of the two inputs is greater (only
 291 relevant in case $y_w^{(i)} = 1$).

TABLE II: Describing the CMM problem (adapted from previous work [14, TABLE I] for CMM): Constants (first part); node input/output decision variables (second part); node internal decision variables for minimum adder count (third part); helper variables for bit optimization (fourth part); helper variables for pipelining (fifth part)

Constant/variable	Explanation
$U \times V$	$\in \mathbb{N} \times \mathbb{N}$ matrix size
U	$\in \mathbb{N}$ number of matrix rows (i.e., outputs of the CMM circuit)
V	$\in \mathbb{N}$ number of matrix columns (i.e., inputs into the CMM circuit)
$C^{(u,v)}$	$\in \mathbb{N}_0$ matrix element in row u , column v
$\vec{C}^{(u)}$	$\in \mathbb{N}_0^V$ u^{th} row vector
N	$\in \mathbb{N}_0$ number of adders to implement the CMM circuit
N_{LB}	$\in \mathbb{N}_0$ lower bound for the number of adders
S	$= \max_{u,v} \lceil \log_2 C^{(u,v)} \rceil$ max. allowed shift
W	$= S + 2$ internal word size within the SAT formulation
W_{in}	$\in \mathbb{N}$ input word size
σ	$= \lceil \log_2 W \rceil$ shift word size
B	$\in \mathbb{N}_0$ upper limit for bit-level costs
W_B	$\in \mathbb{N}$ bit-level cost word size
$\vec{c}^{(i)}$	$\in \mathbb{N}_0^V$ output of node i
$c^{(i,v)}$	$\in \mathbb{N}$ v^{th} output of node i
$t^{(i,u,v)}$	$\in \{0, 1\}$ indicates whether $c^{(i,v)} = C^{(u,v)}$
$\alpha^{(i)}$	$\in \mathbb{N}_0$ select input of left input MUX of node i
$\beta^{(i)}$	$\in \mathbb{N}_0$ select input of right input MUX of node i
$\gamma^{(i)}$	$\in \mathbb{N}_0$ pre-add shift input value of node i
$\delta^{(i)}$	$\in \{0, 1\}$ negate input MUX select bit of node i
$\varepsilon^{(i)}$	$\in \{0, 1\}$ add/sub select bit of node i
$\zeta^{(i)}$	$\in \mathbb{N}_0$ post-add shift input value of node i
$\vec{r}^{(i)}$	$\in \mathbb{N}_0$ right input MUX output of node i
$\vec{l}^{(i)}$	$\in \mathbb{N}_0$ left input MUX output of node i
$\vec{s}^{(i)}$	$\in \mathbb{N}_0$ pre-add shift output of node i
$\vec{o}^{(i)}$	$\in \mathbb{N}_0$ right negate select MUX output of node i
$\vec{p}^{(i)}$	$\in \mathbb{N}_0$ left negate select MUX output of node i
$\vec{q}^{(i)}$	$\in \mathbb{N}_0$ negate output of node i
$\vec{z}^{(i)}$	$\in \mathbb{N}_0$ adder output of node i
$a^{(i)}$	$= \sum_{v=0}^{V-1} z^{(i,v)} $ sum over absolute value of the v^{th} adder output vector
$b^{(i)}$	$= a^{(i)} - f^{(i)}$ correction for $a^{(i)}$ to handle negative powers of two
$f^{(i)}$	$\in \{0, 1\}$ tie-breaker for bitlevel costs for negative powers of two
$\Omega^{(i,v)}$	$\in \{0, 1\}$ indicates whether $c^{(i,v)} > 0$; used to determine $f^{(i)}$
$k^{(i)}$	$= \lceil \log_2(b^{(i)}) \rceil$ word size of node i 's adder output
$m^{(i)}$	$\in \{0, 1\}$ whether the bit for computing node i 's MSB can be omitted (only relevant for SCM/MCM)
$g^{(i)}$	$\in \mathbb{N}_0$ the number of bits omitted on the LSB side due to the shift operation
$d^{(i)}$	$\in \mathbb{N}_0$ adder depth of node i

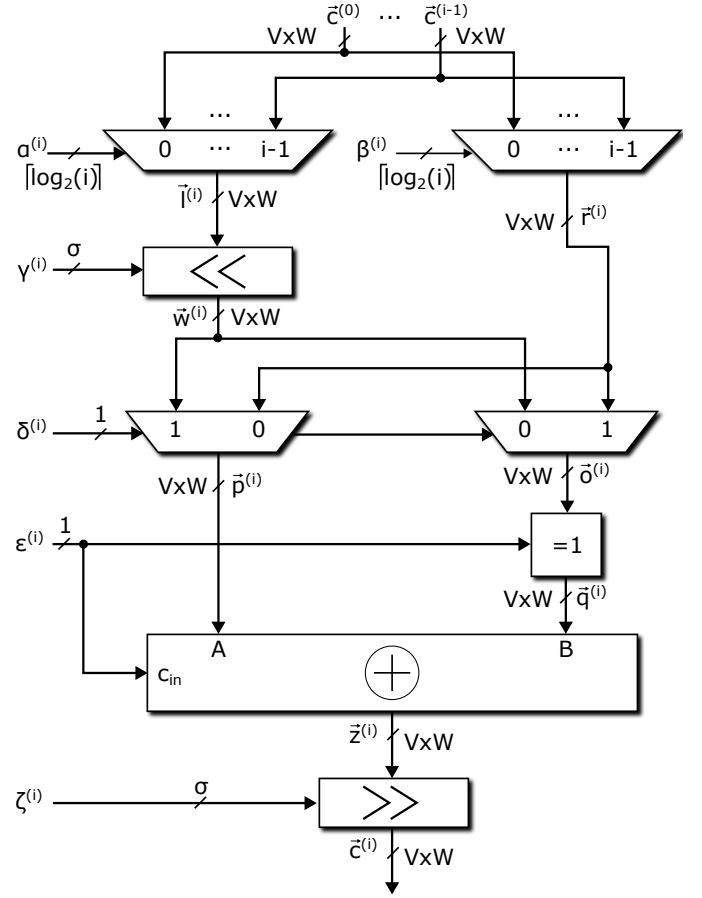


Fig. 4: Adder node model: [14, Fig. 3] adapted for vector-valued inputs & outputs

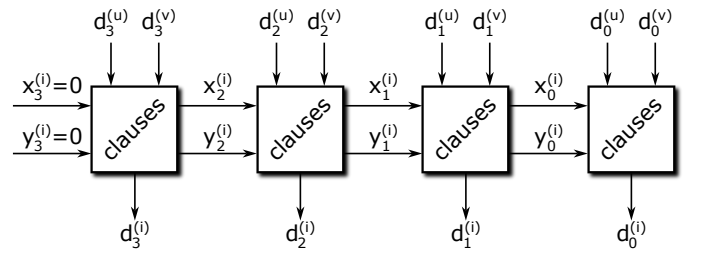


Fig. 5: Computing the maximum of two unsigned integers via SAT clauses: example for 4 bits word size

This translates to the following clauses in order to model 292

293 the computation of bit $d_w^{(i)}$:

$$d_w^{(u)} \vee d_w^{(v)} \vee \bar{d}_w^{(i)} \quad (7)$$

$$d_w^{(u)} \vee \bar{x}_w^{(i)} \vee \bar{y}_w^{(i)} \vee \bar{d}_w^{(i)} \quad (8)$$

$$d_w^{(v)} \vee x_w^{(i)} \vee \bar{y}_w^{(i)} \vee \bar{d}_w^{(i)} \quad (9)$$

$$\bar{d}_w^{(v)} \vee x_w^{(i)} \vee d_w^{(i)} \quad (10)$$

$$\bar{d}_w^{(v)} \vee y_w^{(i)} \vee d_w^{(i)} \quad (11)$$

$$\bar{d}_w^{(u)} \vee \bar{x}_w^{(i)} \vee d_w^{(i)} \quad (12)$$

$$\bar{d}_w^{(u)} \vee y_w^{(i)} \vee d_w^{(i)} \quad (13)$$

$$\bar{d}_w^{(u)} \vee \bar{d}_w^{(v)} \vee d_w^{(i)}. \quad (14)$$

294 Note that (14) is redundant but might help solvers during unit
295 propagation, speeding up the solving process.

296 Carry bits $x_w^{(i)}$ and $y_w^{(i)}$ are modeled via clauses

$$\bar{x}_w^{(i)} \vee \bar{y}_w^{(i)} \vee x_{w-1}^{(i)} \quad (15)$$

$$d_w^{(v)} \vee y_w^{(i)} \vee x_{w-1}^{(i)} \quad (16)$$

$$x_w^{(i)} \vee \bar{y}_w^{(i)} \vee \bar{x}_{w-1}^{(i)} \quad (17)$$

$$\bar{d}_w^{(v)} \vee y_w^{(i)} \vee x_{w-1}^{(i)} \quad (18)$$

297 and

$$\bar{y}_w^{(i)} \vee y_{w-1}^{(i)} \quad (19)$$

$$d_w^{(u)} \vee \bar{d}_w^{(v)} \vee y_{w-1}^{(i)} \quad (20)$$

$$d_w^{(v)} \vee \bar{d}_w^{(u)} \vee y_{w-1}^{(i)} \quad (21)$$

$$\bar{d}_w^{(u)} \vee \bar{d}_w^{(v)} \vee y_w^{(i)} \vee \bar{y}_{w-1}^{(i)} \quad (22)$$

$$d_w^{(u)} \vee d_w^{(v)} \vee y_w^{(i)} \vee \bar{y}_{w-1}^{(i)} \quad (23)$$

298 respectively.

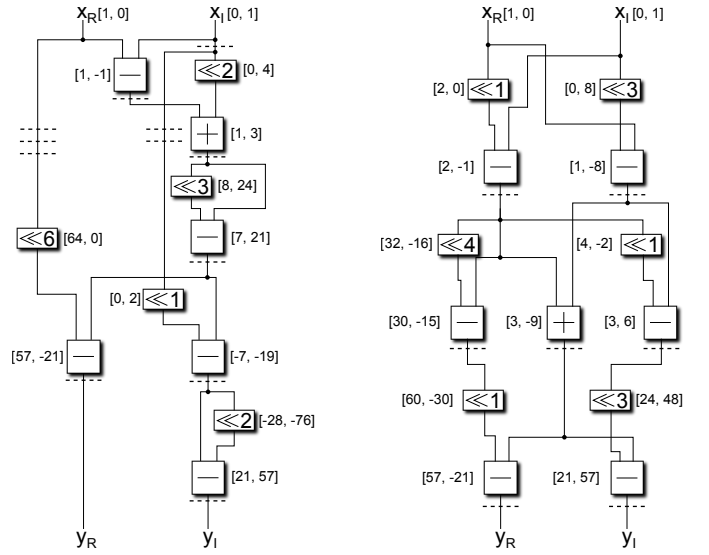
299 G. Extension to pipelining

300 If limiting the adder depth is insufficient to achieve high
301 clock frequencies, pipelining can be applied to the generated
302 circuit to break up the critical path [31], [34], [39].

303 The incorporation of pipelining into the CMM problem is
304 closely related to the adder depth described in Section II-F.
305 When fully pipelining the generated circuit, the pipeline stage
306 of an adder is equal to its adder depth. Hence, both of its
307 inputs must come from within the same (previous) pipeline
308 stage. Otherwise, there would exist two paths from inputs to
309 outputs, which pass a different number of registers, making
310 the pipeline invalid.

311 Fully pipelining the example circuit from Fig. 1e for maxi-
312 mum throughput could, for example, yield the circuit shown in
313 Fig. 6a. This would require twelve registers, where six of them
314 are only used for pipeline balancing. An alternative solution
315 is shown in Fig. 6b. It needs to implement one more adder
316 but only uses seven registers instead of twelve. Due to the
317 increased adder count, the computation can be ordered such
318 that no additional register is needed for pipeline balancing.

319 Previous work for pipelined MCM (pMCM) and pipelined
320 CMM (pCMM) has shown that register costs typically domi-
321 nate adder costs [31], [34]. Hence, the optimization goal



(a) Pipelined version of the
adder graph shown in Fig. 1e
(12 registers)

(b) Adder graph with minimum
register cost (7 registers)

Fig. 6: Pipelined adder graphs for $y = (57+j21) \cdot x$, where $x = x_R + jx_I$ and $y = y_R + jy_I$. As a convention, subtractors always subtract the right input. Dashed lines represent registers.

322 for pMCM and pCMM is to find a circuit with minimum
323 register count under the constraint that at least one register
324 is put at each adder output in order to achieve the shortest
325 critical path possible. This is particularly true for FPGA-based
326 implementations, where a slice is occupied whether an adder
327 is implemented before the register or not. Hence, adders are
328 usually assumed to be “free” [31], [39].

329 Previous work for ILP-based pMCM modeled adder nodes
330 and registers separately [31]. On one hand, this has the
331 advantage that the register count after each adder for pipeline
332 balancing can be represented using a single integer variable
333 per adder node. On the other hand, solutions with optimal
334 register count might have non-minimal adder count [31].

335 We support pipelining in OatMeal by modeling adder
336 depth for each adder in addition to the constraint that adder
337 depths for both adder inputs must always be equal. Explicitly
338 modeling pure registers in order to pass a value from one
339 pipeline stage to the next one (as done by Garcia and Volkova
340 [31]) is not necessary, since, for example, setting $\vec{u} = \vec{v}$
341 and $\hat{q} = (\hat{l}_1, \hat{l}_2, \hat{r}, \hat{s}) = (1, 0, 0, 1)$, or $\hat{q} = (0, 0, 1, 0)$ yields
342 $\vec{A}_{\hat{q}}(\vec{u}, \vec{u}) = \vec{u}$. Such operations are identified during post-
343 processing and replaced by pure registers for implementation.
344 Additionally, this ensures that the first solution encountered is
345 always optimal with respect to the register count.

346 H. Extension to bit-level costs

347 Instead of expressing each bit-adder individually in the SAT
348 formula, we only express the word size *in addition* to the
349 input word size for each adder. This has the advantage that
350 computations are easier to model in SAT and, hence, require
351 fewer variables and clauses [14].

352 The word size to implement an addition depends on the
 353 input word size W_{in} , the fundamental $\bar{c}^{(i)}$, the left shift value l
 354 and the operation performed (i.e., $2^l \cdot u + v$, $2^l \cdot u - v$ or $u - 2^l \cdot v$).
 355 Without pipelining, we have the following possibilities to save
 356 hardware bit-adders:

- 357 • Small fundamental values result in a small word size
 358 increase compared to the input word size. For example,
 359 $y = x \cdot 123$, with $W_{\text{in}} = 8$ has a word size of $W_{\text{out}} = 15$
 360 and $y = x \cdot 2$, with $W_{\text{in}} = 8$ has a word size of $W_{\text{out}} = 9$.
- 361 • For the computations $2^l \cdot u + v$ and $u - 2^l \cdot v$ we do not
 362 need to explicitly compute the lowest l bits, since they
 363 are identical to v 's lowest l bits.

364 Note that, for the case $V = 1$ we can omit computing the
 365 MSB if the result has the same sign as one of its inputs. This
 366 is, in general, not possible for CMM, since each fundamental
 367 consists of a linear combination of multiple values.

368 Including pipelining, savings due to the choice of operation
 369 or omitting the MSB are not possible since each output
 370 bit must be stored in a register, independent whether an
 371 addition was performed on it or not. Hence, pipelined CMM
 372 circuits only offer saving potential when the solver chooses to
 373 implement non-output fundamentals with low word size.

374 Due to its practical relevance, we focus on circuits that
 375 operate using two's complement number representations. This
 376 requires sign extensions when adding two bit vectors of
 377 varying sizes. Therefore, bit-level cost savings due to non-
 378 overlapping operands as reported by Garcia and Volkova [10]
 379 are not possible.

380 Another peculiarity arises due to the number format's asym-
 381 metry around zero. Previous work [14] has computed the word
 382 size at the output of a given adder i via

$$W_{\text{out}}^{(i)} = W_{\text{in}} + \lceil \log_2(|c^{(i)}|) \rceil \quad (24)$$

383 where W_{in} is the circuit's input word size and $c^{(i)}$ is the
 384 fundamental computed by that adder. This formula applies
 385 under the following assumptions:

- 386 1) $V = 1$ (SCM/MCM)
- 387 2) $\lceil \log_2(|c^{(i)}|) \rceil < W_{\text{in}}$
- 388 3) $c^{(i)}$ is *not* a negative power of two.

389 While Assumption 2 generally holds for practical applications,
 390 Assumptions 1 & 3 are regularly violated when targeting CMM
 391 and/or pipelined circuits.

392 As an example, consider an input word size $W_{\text{in}} = 8$ for
 393 $V = 1$. This means that the input signal x is a scalar that takes
 394 integer values in the interval $x \in [-128, 127]$. Multiplying x
 395 by a negative power of two such as $y = -2x$ yields values in
 396 the interval $y \in [-254, 256]$. Hence, the word size to represent
 397 y must be 10 bits, where the tenth bit is only relevant for
 398 $y = 256 = 0b0100000000$.

399 Applying (24) to a given adder node i yields

$$W_{\text{out}}^{(i)} = W_{\text{in}} + \left\lceil \log_2 \left(\left(\sum_{v=0}^{V-1} |c^{(i,v)}| \right) - f^{(i)} \right) \right\rceil \quad (25)$$

400 with $f^{(i)}$ defined as

$$f^{(i)} = \begin{cases} 1 & \text{if } \bar{c}^{(i)} \text{ contains any positive entry} \\ 0 & \text{else.} \end{cases} \quad (26)$$

Here, $f^{(i)}$ works as a tie-breaker to detect adder nodes that
 need an additional bit to represent their output. If $\bar{c}^{(i)}$ contains
 any positive entry, then the additional output bit can be
 omitted. This is detected for each adder node i separately
 by the introduction of additional variables and clauses as
 described in the following.

For the computation $\lceil \log_2 \left(\left(\sum_{v=0}^{V-1} |c^{(i,v)}| \right) - f^{(i)} \right) \rceil$ we
 can modify the SAT model for $\lceil \log_2(|x|) \rceil$ described in
 previous work [14] to perform the absolute value computation
 for each vector element and include a summation as well as
 a 1-bit-subtraction before computing $\lceil \log_2(\dots) \rceil$.

Computing $f^{(i)}$ for each adder node works by allocating a
 Boolean variable $\Omega^{(i,v)}$ for each adder node and each of its
 vector elements, where $\Omega^{(i,v)}$ indicates $c^{(i,v)} > 0$. Formally,

$$\Omega^{(i,v)} \implies \left(\bar{z}_{W-1}^{(i,v)} \wedge \bigvee_{w=0}^{W-2} z_w^{(i,v)} \right). \quad (27)$$

This is realized using the following two clauses for each adder
 node i and each of its vector elements v :

$$\bar{\Omega}^{(i,v)} \vee \bar{z}_{W-1}^{(i,v)} \quad (28)$$

$$\bar{\Omega}^{(i,v)} \vee \bigvee_{w=0}^{W-2} z_w^{(i,v)}. \quad (29)$$

With that, $f^{(i)}$ can be modeled in SAT via the relationship

$$f^{(i)} \equiv \bigvee_{v=0}^{V-1} \Omega^{(i,v)} \quad (30)$$

which is realized by adding the following clauses to the solver
 for each adder node i :

$$\begin{aligned} & \bar{\Omega}^{(i,0)} \vee f^{(i)} \\ & \bar{\Omega}^{(i,1)} \vee f^{(i)} \\ & \dots \\ & \bar{\Omega}^{(i,V-1)} \vee f^{(i)} \\ & f^{(i)} \vee \bigvee_{v=0}^{V-1} \Omega^{(i,v)} \end{aligned} \quad (31)$$

Note that (28) and (29) can be satisfied trivially by setting
 $\Omega^{(i,v)} = 0$. This means that the solver is always permitted to
 set $f^{(i)} = 0$ leading to an overestimation of the actual word
 size, and, hence, bit-level costs. This is no problem in our
 context since we aim for a minimization of bit-level costs. For
 the optimal word size, finally, the solver is forced to represent
 the word size accurately.

III. EXPERIMENTAL EVALUATION

In order to validate practical usefulness of the proposed
 algorithm, we run a wide array of benchmarks and compare
 OatMeal to several algorithms from the state-of-the-art using
 Kissat [40], [41] with a custom C++ API as our SAT solver.
 Our C++ implementation of OatMeal including all results is
 publicly available³.

³https://gitlab.uni-kassel.de/uk025743/sat_cmm

434 Usually, SCM and MCM optimization algorithms only focus
 435 on positive fundamentals [4], [7], [8], [10], since negative
 436 coefficients can be generated from positive values by changing,
 437 for example, an adder to a subtracter in the surrounding system
 438 where the constant multiplier is embedded (e.g., in digital
 439 filter implementations). Only recently, it was discovered that
 440 allowing negative fundamentals possibly reduces implementa-
 441 tion costs compared to optimal implementations consisting of
 442 all-positive coefficients [14].

443 For CMM it is obvious that negative vector entries must be
 444 allowed if the input matrix still contains negative elements
 445 after normalization. But what about all-positive matrices?
 446 To the knowledge of the authors, recent work has neither
 447 proven nor disproven whether negative numbers may appear in
 448 optimal solutions for CMM problems with all-positive matrix
 449 entries.

450 In our tests we encountered the CMM problem

$$y = \begin{bmatrix} 123 & 321 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad (32)$$

451 which has an optimal adder count of 4 when allowing negative
 452 intermediate results and needs 5 adders when only allowing
 453 positive intermediate results (results generated with OatMeal,
 454 adder graphs given in the appendix). This shows that, even
 455 for CMM problems with all-positive matrices, it is necessary
 456 to account for negative coefficients when aiming for optimal
 457 solutions. Hence, in all following tests, all vectors appearing
 458 in Fig. 4 are represented in two's complement representation,
 459 even though an unsigned binary representation in case of
 460 all-positive coefficient matrices would have utilized fewer
 461 variables/clauses leading to faster solving times.

462 A. Complex Constant Multiplication

463 The first test involves finding shift-and-add realizations for
 464 complex coefficient multiplications in the form $y = (C_R +$
 465 $jC_I) \cdot x$, represented in matrix form as

$$\begin{bmatrix} y_R \\ y_I \end{bmatrix} = \begin{bmatrix} C_R & -C_I \\ C_I & C_R \end{bmatrix} \cdot \begin{bmatrix} x_R \\ x_I \end{bmatrix}. \quad (33)$$

466 Coefficients are taken from a recent FFT implementation by
 467 Garrido and Malagón, which was optimized for efficient use of
 468 shift-and-add circuits for rotator implementation [42]. Adder
 469 graphs generated with OatMeal are given in the appendix.
 470 Table III shows results when using the proposed approach to
 471 find CMM circuits with or without the use of pipelining. It
 472 can be seen that OatMeal is able to optimize the constant
 473 multipliers by Garrido and Malagón even further, yielding re-
 474 ductions in adders/registers by 4.35%/21.21% on average and
 475 up to 12.50%/30.00%. This translates to bit-level cost savings
 476 by 24.70%/21.45% on average and up to 27.63%/24.71%.
 477 Even in cases with identical adder count (i.e., $181 - j181$
 478 and $473 - j196$), OatMeal is able to find adder graphs with
 479 significantly reduced bit-level cost savings.

480 B. Pipelined Multiple Constant Multiplication

481 The next evaluation targets pipelined MCM (pMCM) imple-
 482 mentations. For our evaluation we choose digital filter kernels

TABLE III: Improving optimized FFT rotations

Complex coeff. (Without pipelining)	Garrido and Malagón [42]		OatMeal	
	#Add	#Bit	#Add	#Bit
$181 - j181$	8	152	8	110
$251 - j50$	8	136	7	104
$473 - j196$	7	133	7	103
(With pipelining)	#Reg	#Bit	#Reg	#Bit
$181 - j181$	12	206	10	170
$251 - j50$	10	170	7	128
$473 - j196$	11	202	9	156

483 for image processing, already used for benchmarks in previous
 484 work [4], [10], [14], [31]. We compare the proposed approach
 485 to a recent ILP-based algorithm for optimal design of pipelined
 486 MCM implementations [31] and use the original open-source
 487 implementation [43] written in Julia, called jMCM, using
 488 Gurobi [44] to solve ILP instances.

489 Results are shown in Table IV. It can be seen that OatMeal
 490 either matches the quality of results or finds cheaper imple-
 491 mentations, compared to the ILP-based pMCM algorithm by
 492 Garcia and Volkova [31].

493 Theoretically, jMCM should find implementations with
 494 equal QoR compared to OatMeal. Differences are caused by
 495 jMCM's search procedure. It takes the number of adders
 496 computed by a heuristic MCM algorithm as a baseline, and
 497 afterwards minimizes register costs. While Garcia and Volkova
 498 showed that this approach yields good solutions for practical
 499 applications, non-optimality can be caused by the following:

- The heuristic approach fails to determine the optimal adder count.
- Even if the heuristic finds a solution for the optimal adder count, it might be possible to increase the adder count slightly in favor of register savings.

505 In order to find optimal solutions, manual user intervention
 506 could be used to explicitly limit the adder count for a design-
 507 space exploration as described in the original publication [31],
 508 where the authors manually fine-tune the solving procedure in
 509 order to arrive at the optimal solutions computed by OatMeal
 510 in Table IV. The upside of utilizing a heuristic for an initial
 511 solution is that the solver does not have to repeatedly prove
 512 infeasibility for some adder counts, leading to quicker solving
 513 times. For example, for Lowpass 9×9 , OatMeal needs 11.3
 514 hours to arrive at the solution with 17 registers, whereas jMCM
 515 only utilizes the pre-defined timeout of two hours to obtain the
 516 solution with 18 registers, since the first adder count attempt—
 517 by definition—is already feasible.

518 C. Constant Matrix Multiplication

519 This test is about examples for matrix multiplications occur-
 520 ing in practice. The following matrices were already used as
 521 benchmarks in previous work [34] for pipelined CMM. Results
 522 are shown in Table V. For a time limit of one day per SAT
 523 instance, OatMeal is able to either match or improve QoR for
 524 all but one instance. For the one instance where OatMeal fails
 525 to match the best result reported in literature, it times out and
 526 yields a solution with one more adder. However, by increasing

TABLE IV: Pipelined MCM including bit optimization for selected image processing filter kernels with $W_{in} = 8$: ILP (jMCM [31]) vs. SAT (OatMeal)

Idx	Instance	U	W	#Reg		#Bit	
				ILP	SAT	ILP	SAT
0	Gaussian 3×3	3	8	5	5	58	58
1	Laplacian 3×3	3	8	5	5	61	61
2	Unsharp 3×3	3	8	5	5	56	56
3	Unsharp 3×3	3	12	7	7	91	91
4	Gaussian 5×5	3	12	9	9	111	111
5	Highpass 5×5	5	7	7	7	74	74
6	Lowpass 5×5	5	11	8	8	98	98
7	Highpass 9×9	10	9	8	8	85	85
8	Lowpass 9×9	13	12	18	17	234	221
9	Highpass 15×15	19	11	16	16	186	186
10	Lowpass 15×15	31	14	36	34	488	467

TABLE V: Comparison against state-of-the-art heuristics for CMM (timeout: 1 day per SAT instance)

Matrix	Source	#Add	
		Ref.	OatMeal
$\begin{bmatrix} 77 & 150 & 29 \\ -44 & -87 & 131 \\ 131 & -110 & -21 \end{bmatrix}$	[34]	12	13
$\begin{bmatrix} 256 & 0 & 351 \\ 256 & -86 & -179 \\ 256 & 444 & 0 \end{bmatrix}$	[35], [45]	11	11
$\begin{bmatrix} 7 & 8 & 2 & 13 \\ 12 & 11 & 7 & 13 \\ 5 & 8 & 2 & 15 \\ 7 & 11 & 7 & 11 \end{bmatrix}$	[35]	14	12
$\begin{bmatrix} 21 & 39 \\ 11 & 5 \end{bmatrix}$	[17], [34]	6	5
$\begin{bmatrix} 7 & 24 \\ 9 & 3 \end{bmatrix}$	[16], [34]	5	4
$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -2 & 2 & -1 \end{bmatrix}$	[46]	8	8

527 the timeout, OatMeal successfully computed a solution for 12
 528 adders within 24.8 hours and even a solution for 11 adders in
 529 79.3 hours.

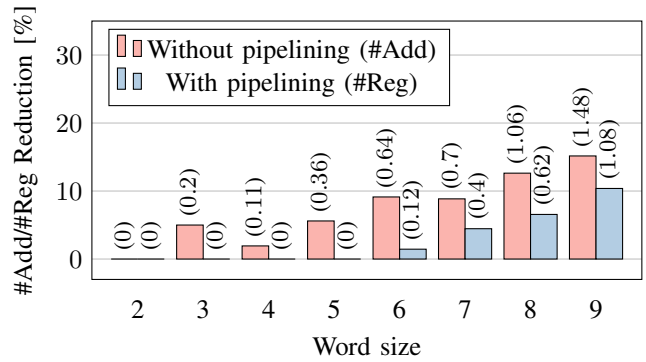
530 *D. Random matrices for practical applications*

531 Finally, we compare OatMeal to RPAG [34], targeting
 532 randomly generated complex coefficient multiplications and
 533 randomly generated 3×3 convolutional cores with different
 534 sparsity settings.

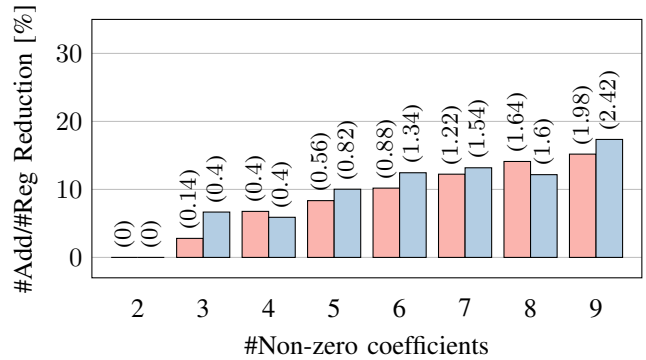
535 Complex multiplications are again represented via (33), and
 536 3×3 convolutional cores can be represented as SOP operations
 537 via

$$y = \sum_{m=0}^2 \sum_{n=0}^2 C_{m,n} \cdot x_{m,n}. \quad (34)$$

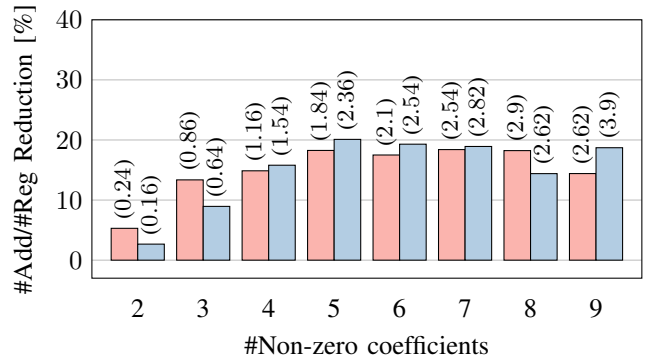
538 For each setting (word size/sparsity) we generate up to 50
 539 input matrices with uniformly distributed coefficients, where
 540 at least one coefficient per matrix utilizes the given word size.
 541 Matrix sets for small word sizes may consist of less than 50
 542 matrices in cases where all possible coefficient combinations
 543 are enumerated.



(a) Complex multiplications



(b) Convolutional cores with coefficient word sizes up to $W = 4$



(c) Convolutional cores with coefficient word sizes up to $W = 6$

Fig. 7: Average adder/register count reduction over RPAG; absolute numbers are given in parentheses

544 Figure 7 shows results for a comparison regarding the
 545 number of adders/registers in the generated implementations.
 546 It is clear that the optimal approach via OatMeal is able to
 547 significantly reduce the implementation overhead compared to
 548 the state-of-the-art heuristic approach by Kumm et al. [34].
 549 As seen, savings increase when scaling up the difficulty of
 550 the underlying CMM problem, indicating that RPAG's QoR
 551 decreases for harder problems. For very hard problems (e.g.,
 552 convolutional cores with $W = 6$ and nine non-zero coeffi-
 553 cients), however, OatMeal fails to provide optimal solutions
 554 due to timeouts, effectively acting as a heuristic algorithm,
 555 and presumably causing its QoR advantage over RPAG to
 556 diminish.

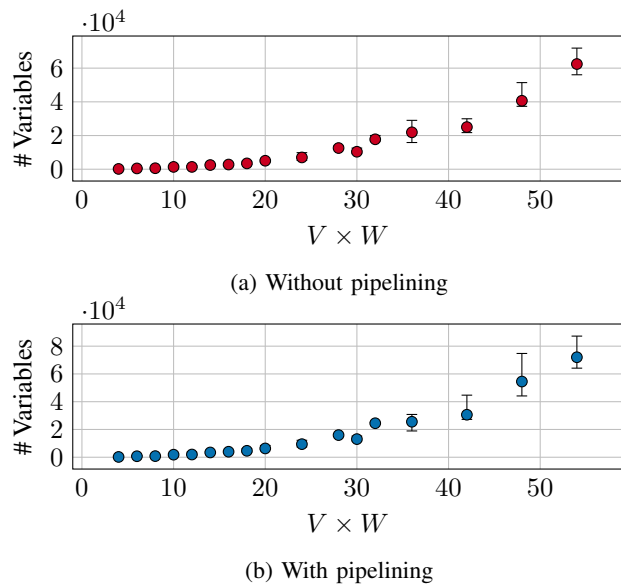


Fig. 8: SAT instance size analysis

557 *E. Scalability analysis*

558 In order to quantify the scalability of the proposed algo-
 559 rithm, we show (i) how W and V influence the resulting SAT
 560 instance size, and (ii) how W and V influence CPU time as
 561 well as optimality for the benchmarks performed.

562 In Fig. 4 each internal signal is represented by $V \times W$ bits.
 563 This suggests that the sizes of the resulting SAT instances
 564 strongly correlate with the product $V \times W$. However, the
 565 number of SAT variables should not directly be proportional
 566 to $V \times W$, since, for example, the input multiplexer sizes not
 567 only depend on V but also on the number of preceding adder
 568 nodes, and the left and right shifters are modeled internally
 569 by $\sigma \times V \times W$ 2:1 multiplexers, where $\sigma = \lceil \log_2 W \rceil$.

570 Figure 8 shows a scatter plot containing each one of the
 571 randomly generated SAT instances from Section III-D. Mean
 572 values and the 1-sigma confidence interval ($\approx 68.27\%$) are
 573 shown for each value of $V \times W$. Although the variance grows
 574 for larger problem instances, we can clearly see a correlation
 575 between the SAT instance size and $V \times W$. Hence, $V \times W$ is
 576 a strong indicator for the SAT instance's complexity.

577 The main measures for scalability of the proposed approach
 578 are (i) optimality regarding the two objectives N and B , and
 579 (ii) CPU time. An analysis is shown in Fig. 9. Figures 9a
 580 and 9b show how many problems could be optimally solved
 581 w.r.t. the difficulty indicator $V \times W$ and Fig. 9c shows the
 582 connection between the average runtime and $V \times W$.

583 Regarding optimality we can clearly see a turning point
 584 between $20 \leq V \times W \leq 30$, beyond which OatMeal fails to
 585 guarantee optimality and acts as a heuristic. For $V \times W \leq 12$
 586 OatMeal yields its solutions nearly instantly. For $V \times W >$
 587 12, the run time grows approximately linear caused by the
 588 specified timeout and the number of failed solving attempts
 589 before reaching a valid solution.

590 Additionally, we cannot see a clear distinction between an
 591 optimization with or without pipelining. This indicates that

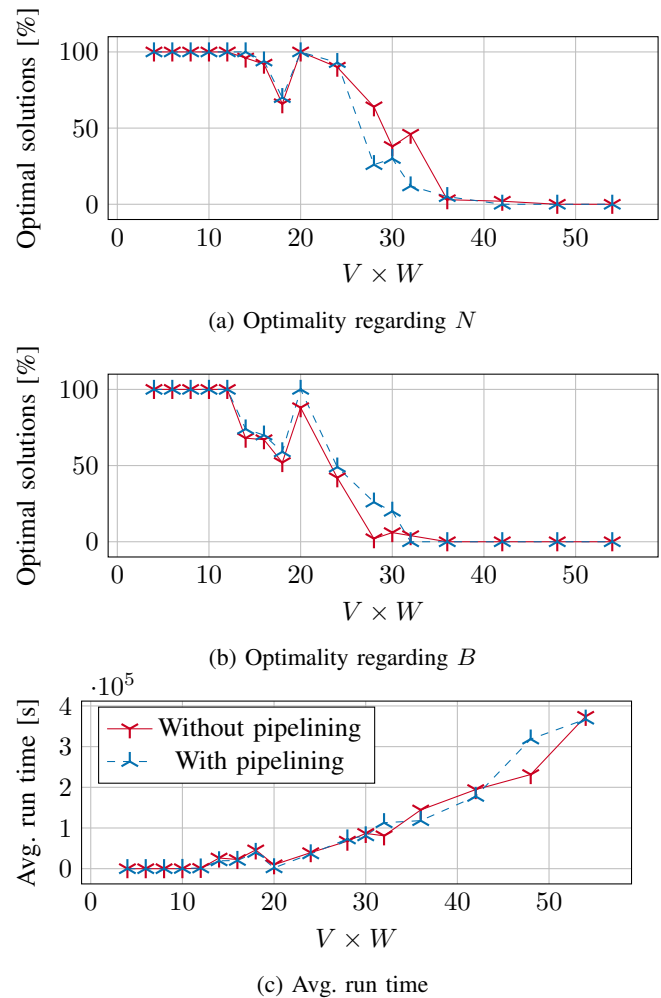


Fig. 9: Problem difficulty analysis

592 pipelining has no explicit impact on the difficulty of the
 593 underlying optimization problem.

594 A total of approximately five CPU years was spent solving
 595 SAT instances for the randomly generated problems in Sec-
 596 tion III-D.

597 **IV. SYNTHESIS RESULTS**

598 In order to show practical relevance of our approach, we
 599 generate FPGA and ASIC implementations for the complex
 600 coefficient multiplications from Table III for a low-cost Zynq-
 601 7000 FPGA (*xc7z020c1g400-1*) using Vivado [47] for
 602 synthesis and the open-source SkyWater-130 nm technology
 603 using OpenLane [48]. FPGA and ASIC circuits use an input
 604 word size of twelve bits. VHDL code was obtained using
 605 FloPoCo [49] and converted to Verilog using GHDL [50].

606 ASIC area is constrained to $32,400 \mu\text{m}^2$ for all designs in
 607 order to have enough space for I/O pins. Instead of directly
 608 reporting area, we therefore report core area utilization [%] to
 609 get an estimate of the actual area usage when embedding the
 610 constant multipliers into a larger system. Slices are the main
 611 resource for logic implementations on FPGAs. They comprise
 612 look-up-tables and flip-flops. For each system, the FPGA op-
 613 erating frequency was varied between 125 MHz and 500 MHz

614 and we report results for the designs with the shortest critical
 615 paths. This ensures that we account for cases where Vivado's
 616 heuristic timing optimization yields sub-optimal results due to
 617 a mismatched frequency setting. Note, however, that constant
 618 multipliers only rarely lie on the critical path in a complex
 619 system.

620 Synthesis results are summarised in Table VI. It can be
 621 seen that circuits generated with OatMeal always use less
 622 area compared to original implementations by Garrido and
 623 Malagón. The area reduction sometimes comes at the price of
 624 a slightly longer critical path. This effect is most extreme for
 625 $251 - j50$ without pipelining, where the critical path using
 626 OatMeal increases by 1.32 ns (23.57 %) for the FPGA and by
 627 0.43 ns (12.36 %) for the ASIC implementation, caused by an
 628 increased adder depth. This clearly shows that adder/register
 629 reductions in the adder graph translate to cheaper solutions
 630 in hardware implementations for both FPGAs and ASICs.
 631 Furthermore, pipelining proves to be an effective method for
 632 critical path reduction. Due to the routing delay on FPGAs,
 633 this performance increase is more pronounced on ASICs.

634 V. CONCLUSION

635 This work presented OatMeal, the first algorithm to discover
 636 optimal constant matrix multiplication (CMM) algorithms via
 637 a reduction to the well-known Boolean Satisfiability (SAT)
 638 problem, leveraging the speed of contemporary SAT solvers.
 639 The proposed SAT model is equipped with several (optional)
 640 extensions for hardware design such as bit-level costs, pipelin-
 641 ing and adder depth.

642 Extensive testing for multiple flavors of the CMM problem
 643 showed the practical relevance of the proposed approach
 644 and superiority over state-of-the-art heuristics, and optimal
 645 algorithms for sub-problems of CMM (i.e., multiple constant
 646 multiplication). Even for small problem sizes (e.g., 2×2
 647 matrices and word sizes less than 10 bits), OatMeal exceeds
 648 or matches the quality of heuristic methods for matrices
 649 occurring in practical applications such as complex coefficient
 650 multiplications in FFTs, or convolutions in artificial neural
 651 networks or image processing filters.

652 Future work should investigate an extension to truncated
 653 multiplications for applications that do not require the full out-
 654 put precision similar to previous work for truncated MCM [10]
 655 or the heuristic approximation of CMM [18]. Additionally,
 656 it would be interesting to derive a more detailed bit-level
 657 cost model, differentiating costs for inverters, half and full
 658 adders, which might enable even higher area savings for ASIC
 659 implementations. A possible implementation could weigh the
 660 specific circuit elements by their transistor count and decide—
 661 based on the computation by each adder node—how many
 662 transistors are necessary to implement each adder node. Aside
 663 from backend-specific transistor counts, this approach would
 664 need information about the input word size in order to deter-
 665 mine the correct amount of inverters/half adders/full adders.

666 In order to improve the scalability of the proposed al-
 667 gorithm, it would be interesting to investigate whether it
 668 is possible to combine heuristic and optimal methods (e.g.,
 669 similar to compressor tree synthesis [51]). A heuristic could

670 for example generate an incomplete adder tree and a SAT-
 671 based method constructs an optimal completion based on the
 672 initial solution generated by the heuristic.

673 ACKNOWLEDGEMENT

674 The authors are thankful for Rémi Garcia's support regard-
 675 ing the reproduction of pMCM results [31] using the jMCM
 676 tool [43], as well as for Martin Kumm's support with the
 677 VHDL code generation using FloPoCo.

678 REFERENCES

- 679 [1] J. Thong and N. Nicolici, "A novel optimal single constant multiplication
 680 algorithm," in *Design Automation Conference*, Jun. 2010, pp. 613–616,
 681 iSSN: 0738-100X.
- 682 [2] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multi-
 683 plication," *ACM Transactions on Algorithms*, vol. 3, no. 2, pp. 11–es,
 684 2007.
- 685 [3] K. Johansson, O. Gustafsson, and L. Wanhammar, "A detailed com-
 686 plexity model for multiple constant multiplication and an algorithm
 687 to minimize the complexity," in *Proceedings of the 2005 European
 688 Conference on Circuit Theory and Design, 2005.*, vol. 3, 2005, pp.
 689 III/465–III/468 vol. 3.
- 690 [4] M. Kumm, "Optimal Constant Multiplication Using Integer Linear
 691 Programming," *IEEE Transactions on Circuits and Systems II: Express
 692 Briefs*, vol. 65, no. 5, pp. 567–571, May 2018.
- 693 [5] L. Aksoy, E. Costa, P. Flores, and J. Monteiro, "Optimization of area in
 694 digital FIR filters using gate-level metrics," in *Proceedings of the 44th
 695 annual Design Automation Conference*, ser. DAC '07. New York, NY,
 696 USA: Association for Computing Machinery, Jun. 2007, pp. 420–423.
 697 [Online]. Available: <https://dl.acm.org/doi/10.1145/1278480.1278588>
- 698 [6] M. Kumm, O. Gustafsson, M. Garrido, and P. Zipf, "Optimal Single
 699 Constant Multiplication Using Ternary Adders," *IEEE Transactions on
 700 Circuits and Systems II: Express Briefs*, vol. 65, no. 7, pp. 928–932, Jul.
 701 2018.
- 702 [7] V. Lagoon and A. Metodi, "Deriving Optimal Multiplication-by-
 703 Constant Circuits With A SAT-based Constraint Engine," in *The 19th
 704 workshop on Constraint Modelling and Reformulation*, 2020, pp. 1–6.
- 705 [8] L. Aksoy, E. O. Güneş, and P. Flores, "Search algorithms for the
 706 multiple constant multiplications problem: Exact and approximate,"
 707 *Microprocessors and Microsystems*, vol. 34, no. 5, pp. 151–162, Aug.
 708 2010.
- 709 [9] R. Garcia, A. Volkova, and M. Kumm, "Truncated Multiple Constant
 710 Multiplication with Minimal Number of Full Adders," in *IEEE Interna-
 711 tional Symposium on Circuits and Systems (ISCAS)*, 2022.
- 712 [10] R. Garcia and A. Volkova, "Toward the Multiple Constant Multiplication
 713 at Minimal Hardware Cost," *IEEE Transactions on Circuits and Systems
 714 I: Regular Papers*, vol. 70, no. 5, pp. 1976–1988, May 2023.
- 715 [11] A. G. Dempster and M. D. Macleod, "Constant integer multiplication
 716 using minimum adders," *IEE Proceedings - Circuits, Devices and
 717 Systems*, vol. 141, no. 5, pp. 407–413, Oct. 1994, publisher: IET Digital
 718 Library.
- 719 [12] O. Gustafsson, "Towards Optimal Multiple Constant Multiplication: A
 720 Hypergraph Approach," in *2008 42nd Asilomar Conference on Signals,
 721 Systems and Computers*, ser. Asilomar Conference on Signals, Systems
 722 and Computers (ACSSC), 10 2008, pp. 1805 – 1809.
- 723 [13] O. Gustafsson, A. G. Dempster, M. D. Johansson, K. and Macleod, and
 724 L. Wanhammar, "Simplified design of constant coefficient multipliers,"
 725 *Circuits, Systems, and Signal Processing*, vol. 25, no. 2, pp. 225 – 251,
 726 2006.
- 727 [14] N. Fiege, M. Kumm, and P. Zipf, "Bit-Level Optimized Constant
 728 Multiplication Using Boolean Satisfiability," *IEEE Transactions on
 729 Circuits and Systems I: Regular Papers*, vol. 71, no. 1, pp. 249–261,
 730 Jan. 2024. [Online]. Available: [https://ieeexplore.ieee.org/document/
 731 10320216](https://ieeexplore.ieee.org/document/10320216)
- 732 [15] M. Potkonjak, M. Srivastava, and A. Chandrakasan, "Multiple constant
 733 multiplications: efficient and versatile framework and algorithms for
 734 exploring common subexpression elimination," *IEEE Transactions on
 735 Computer-Aided Design of Integrated Circuits and Systems*, vol. 15,
 736 no. 2, pp. 151–165, Feb. 1996.

TABLE VI: Synthesis results for FFT rotations

Complex coeff. (Without pipelining)	FPGA				ASIC			
	Garrido and Malagón [42]		OatMeal		Garrido and Malagón [42]		OatMeal	
	LUTs	t_{cp} [ns]	LUTs	t_{cp} [ns]	Area util. [%]	t_{cp} [ns]	Area util. [%]	t_{cp} [ns]
181 – j 181	148	6.94	106	6.90	29.30	4.19	25.90	3.94
251 – j 50	132	5.60	102	6.92	26.41	3.48	25.99	3.91
473 – j 196	127	6.79	100	6.84	28.53	4.07	25.14	3.99
(With pipelining)	LUTs/Registers	t_{cp} [ns]	LUTs/Registers	t_{cp} [ns]	Area util. [%]	t_{cp} [ns]	Area util. [%]	t_{cp} [ns]
181 – j 181	150/227	2.48	135/192	2.40	44.99	0.87	39.04	0.79
251 – j 50	132/192	2.34	115/151	2.40	36.75	0.78	32.36	0.76
473 – j 196	129/221	2.47	111/175	2.33	43.61	0.83	35.04	0.75

[16] S. Ghisconi, E. Costa, J. Monteiro, and R. Reis, "Combination of constant matrix multiplication and gate-level approaches for area and power efficient hybrid radix-2 DIT FFT realization," in *2011 18th IEEE International Conference on Electronics, Circuits, and Systems*, Dec. 2011, pp. 567–570.

[17] A. G. Dempster, O. Gustafsson, and J. O. Coleman, "Towards an Algorithm for Matrix Multiplier Blocks." Kraków: European Circuit Society, 2003.

[18] L. Aksoy, P. Flores, and J. Monteiro, "A Novel Method for the Approximation of Multiplierless Constant Matrix Vector Multiplication," in *2015 IEEE 13th International Conference on Embedded and Ubiquitous Computing*, Oct. 2015, pp. 98–105.

[19] M. Hardieck, M. Kumm, P. Sittel, and P. Zipf, "Constant Matrix Multiplication with Ternary Adders," in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Dec. 2018, pp. 85–88.

[20] N. M. Sarband, O. Gustafsson, and M. Garrido, "Obtaining Minimum Depth Sum of Products from Multiple Constant Multiplication," in *2018 IEEE International Workshop on Signal Processing Systems (SiPS)*, Oct. 2018, pp. 1–6, iSSN: 2374-7390.

[21] —, "Using Transposition to Efficiently Solve Constant Matrix-Vector Multiplication and Sum of Product Problems," *Journal of Signal Processing Systems*, vol. 92, no. 10, pp. 1075–1089, Oct. 2020. [Online]. Available: <https://doi.org/10.1007/s11265-020-01560-z>

[22] A. Kinane, V. Muresan, and N. O'Connor, "Towards an optimised VLSI design algorithm for the constant matrix multiplication problem," in *2006 IEEE International Symposium on Circuits and Systems*. Island of Kos, Greece: IEEE, 2006, p. 4. [Online]. Available: <http://ieeexplore.ieee.org/document/1693782/>

[23] H. Bierlee, J. J. Dekker, V. Lagoon, P. J. Stuckey, and G. Tack, "Single Constant Multiplication for SAT," in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, B. Dilkina, Ed. Cham: Springer Nature Switzerland, 2024, pp. 84–98.

[24] O. Gustafsson and A. Dempster, "On the use of multiple constant multiplication in polyphase FIR filters and filter banks," in *Proceedings of the 6th Nordic Signal Processing Symposium, 2004. NOR SIG 2004.*, Jun. 2004, pp. 53–56.

[25] R. Garcia, A. Volkova, M. Kumm, A. Goldsztejn, and J. Kühle, "Hardware-Aware Design of Multiplierless Second-Order IIR Filters With Minimum Adders," *IEEE Transactions on Signal Processing*, vol. 70, pp. 1673–1686, 2022.

[26] M. Kumm, A. Volkova, and S.-I. Filip, "Design of Optimal Multiplierless FIR Filters With Minimal Number of Adders," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 2, pp. 658–671, Feb. 2023.

[27] A. Dempster and M. Macleod, "IIR digital filter design using minimum adder multiplier blocks," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 45, no. 6, pp. 761–763, Jun. 1998. [Online]. Available: <https://ieeexplore.ieee.org/document/686699/?arnumber=686699>

[28] A. Volkova, R. Garcia, F. De Dinechin, and M. Kumm, "Hardware-Optimal Digital FIR Filters: One ILP to Rule Them all and in Faithfulness Bind Them," in *2023 57th Asilomar Conference on Signals, Systems, and Computers*, Oct. 2023, pp. 1574–1578, iSSN: 2576-2303. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10476962>

[29] A. Dempster and M. Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 42, no. 9, pp. 569–577, Sep. 1995. [Online]. Available: <http://ieeexplore.ieee.org/document/466647/>

[30] M. Macleod and A. Dempster, "Multiplierless FIR filter design algorithms," *IEEE Signal Processing Letters*, vol. 12, no. 3, pp. 186–189, Mar. 2005. [Online]. Available: <https://ieeexplore.ieee.org/document/1395936/?arnumber=1395936>

[31] R. Garcia and A. Volkova, "Multiple Constant Multiplication: From Target Constants to Optimized Pipelined Adder Graphs," in *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*. New York: IEEE, Sep. 2023, pp. 137–143. [Online]. Available: <https://ieeexplore.ieee.org/document/10296462/>

[32] P. Cappello and K. Steiglitz, "Some complexity issues in digital signal processing," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 5, pp. 1037–1041, Oct. 1984.

[33] M. R. Garey, *Computers and intractability : a guide to the theory of NP-completeness*, 27th ed., ser. A series of books in the mathematical sciences. New York, NY: Freeman, 2005.

[34] M. Kumm, M. Hardieck, and P. Zipf, "Optimization of Constant Matrix Multiplication with Low Power and High Throughput," *IEEE Transactions on Computers*, vol. 66, no. 12, pp. 2072–2080, Dec. 2017.

[35] O. Gustafsson, H. Ohlsson, and L. Wanhammar, "Low-complexity constant coefficient matrix multiplication using a minimum spanning tree approach," in *Proceedings of the 6th Nordic Signal Processing Symposium, 2004. NOR SIG 2004.*, 2004, pp. 141–144.

[36] O. Gustafsson, "Lower bounds for constant multiplication problems," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 11, pp. 974 – 978, 11 2007.

[37] A. Dempster, S. Dimirsoy, and I. Kale, "Designing multiplier blocks with low logic depth," in *2002 IEEE International Symposium on Circuits and Systems. Proceedings (Cat. No.02CH37353)*, vol. 5, May 2002, pp. V–V. [Online]. Available: <https://ieeexplore.ieee.org/document/1010818/?arnumber=1010818>

[38] M. Faust and C.-H. Chang, "Minimal Logic Depth adder tree optimization for Multiple Constant Multiplication," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, May 2010, pp. 457–460, iSSN: 2158-1525. [Online]. Available: <https://ieeexplore.ieee.org/document/5537658>

[39] M. Kumm, *Multiple Constant Multiplication Optimizations for Field Programmable Gate Arrays*. Wiesbaden: Springer Fachmedien, 2016. [Online]. Available: <http://link.springer.com/10.1007/978-3-658-13323-8>

[40] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt, "CaDiCaL, GimsatL, IsaSAT and Kissat Entering the SAT Competition 2024," in *Proc. of SAT Competition 2024 - Solver, Benchmark and Proof Checker Descriptions*, ser. Department of Computer Science Report Series B, vol. B-2024-1. Helsinki: University of Helsinki, 2024, pp. 8–10.

[41] A. Biere, "The Kissat SAT Solver," <https://github.com/arminbiere/kissat>, 2024, accessed: 19 Nov. 2024.

[42] M. Garrido and P. Malagón, "The Constant Multiplier FFT," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 1, pp. 322–335, Jan. 2021. [Online]. Available: <https://ieeexplore.ieee.org/document/9241023/?arnumber=9241023>

[43] R. Garcia, "jMCM," <https://github.com/remi-garcia/jMCM>, 2024, commit: ff3b874d0e85a5ea2c44aabd33c57e877f84bc4f.

[44] Gurobi, "Gurobi Optimizer," 2022.

[45] K. Holm and O. Gustafsson, "Low-Complexity and Low-Power Color Space Conversion for Digital Video," in *2006 NORCHIP*.

853 Linköping, Sweden: IEEE, Nov. 2006, pp. 179–182. [Online].
 854 Available: <http://ieeexplore.ieee.org/document/4126976/>
 855 [46] H. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, “Low-
 856 complexity transform and quantization in h.264/AVC,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp.
 857 598–603, 2003.
 858 [47] AMD, “Vivado Design Suite,” [https://www.amd.com/de/products/](https://www.amd.com/de/products/software/adaptive-socs-and-fpgas/vivado.html)
 859 [software/adaptive-socs-and-fpgas/vivado.html](https://www.amd.com/de/products/software/adaptive-socs-and-fpgas/vivado.html), 2024, accessed: 18 Nov.
 860 2024.
 861 [48] M. Shalan and T. Edwards, “Building OpenLANE: A 130nm
 862 OpenROAD-based Tapeout- Proven Flow : Invited Paper,” in *2020 IEEE/ACM International Conference On Computer Aided Design*
 863 *(ICCAD)*, Nov. 2020, pp. 1–6, ISSN: 1558-2434. [Online]. Available:
 864 <https://ieeexplore.ieee.org/document/9256623>
 865 [49] F. de Dinechin and B. Pasca, “Designing Custom Arithmetic Data Paths with FloPoCo,” *IEEE Design & Test of Computers*, vol. 28, no. 4, pp.
 866 18–27, Jul. 2011.
 867 [50] T. Gingold, “GHDl,” <https://github.com/ghdl/ghdl>, 2024.
 870 [51] M. Kumm and J. Kappauf, “Advanced Compressor Tree Synthesis for
 871 FPGAs,” *IEEE Transactions on Computers*, vol. 67, no. 8, pp. 1078–
 872 1091, Aug. 2018.
 873

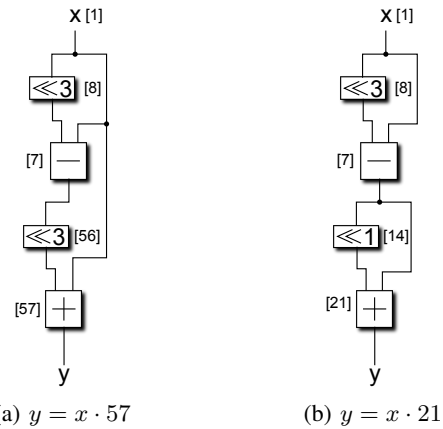


Fig. 10: Adder graphs for $y = x \cdot 57$ and $y = x \cdot 21$



874 **Nicolai Fiege** received the B.Sc. and M.Sc. degrees
 875 in Electrical Engineering from the University of
 876 Kassel, Germany, in 2018 and 2021, respectively,
 877 where he is currently working toward his Ph.D. in
 878 Electrical Engineering.
 879 His current research interests include the design
 880 of optimal digital circuits as well as reconfigurable
 881 computing.

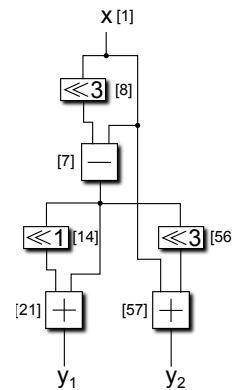


Fig. 11: Adder graph for $y_1 = x \cdot 57$ and $y_2 = x \cdot 21$



883 **Peter Zipf** (M'05) received the Ph.D. (Dr.-Ing.)
 884 degree from the University of Siegen, Germany, in
 885 2002.
 886 He was a Postdoctoral Researcher at the De-
 887 partment of Electrical Engineering and Information
 888 Technology, Darmstadt University of Technology,
 889 Darmstadt, Germany, until 2009. He is currently the
 890 chair of Digital Technology at the University of Kas-
 891 sel, Germany. His current research interests include
 892 reconfigurable computing, embedded systems and
 893 CAD algorithms for circuit optimization.

VI. APPENDIX

894 Figures 10–12 show adder graphs for the motivational
 895 example shown in Fig. 1.

896 The optimal adder graph for $y = 123 \cdot x_1 + 321 \cdot x_2$ with
 897 unsigned coefficients (Fig. 13a) needs one adder more than
 898 the one with signed coefficients (Fig. 13b).
 899

900 Adder graphs for the FFT rotations from Table III and
 901 Table VI are given in Fig. 14–16.

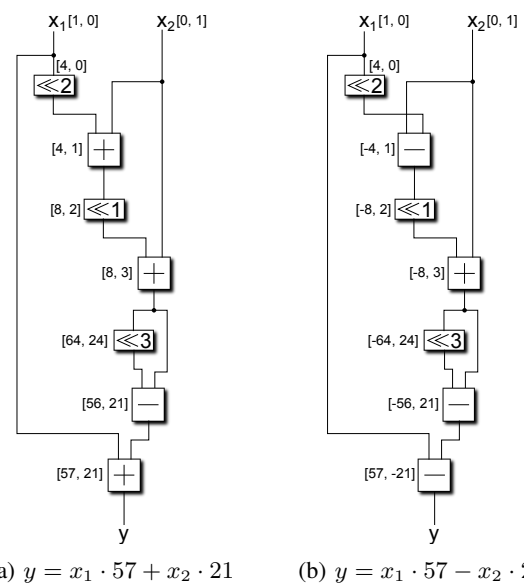


Fig. 12: Adder graphs for $y = x_1 \cdot 57 + x_2 \cdot 21$ and $y = x_1 \cdot 57 - x_2 \cdot 21$

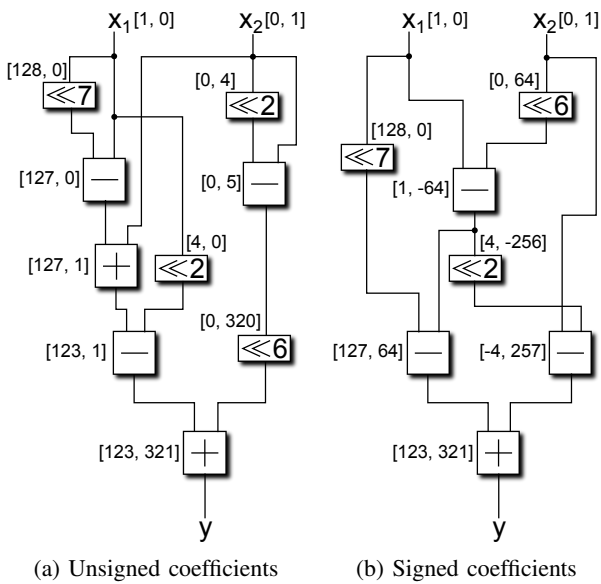


Fig. 13: Adder graphs for $y = x_1 \cdot 123 + x_2 \cdot 321$ with unsigned and signed coefficients

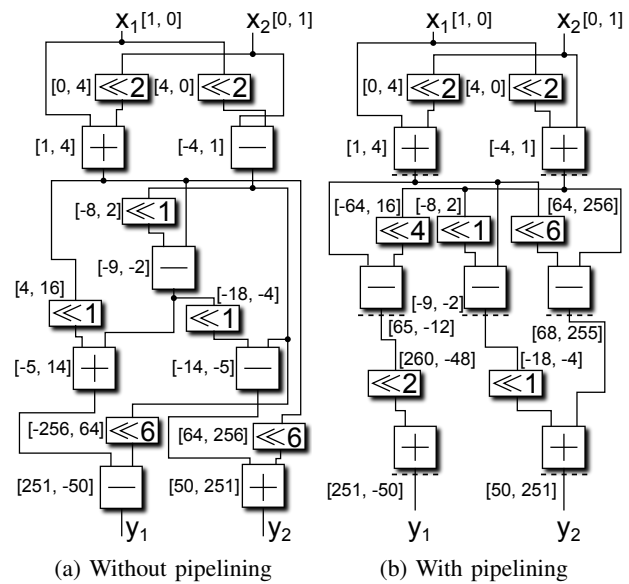


Fig. 15: Adder graphs for $y = (251 + j \cdot 50)x$ with and without pipelining

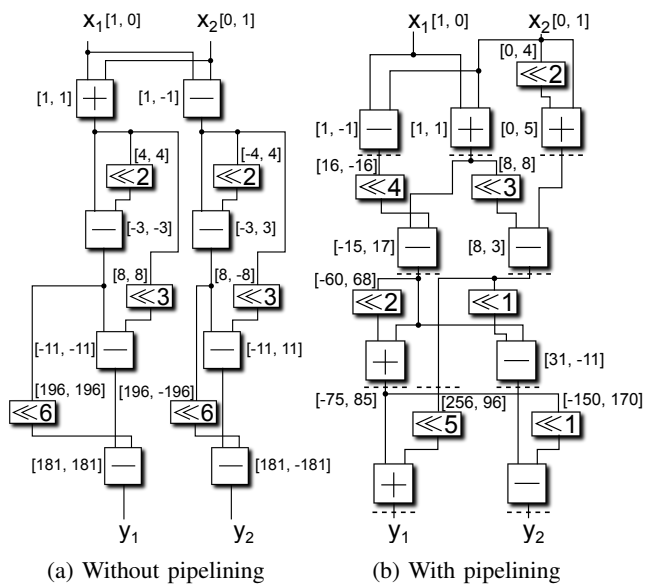


Fig. 14: Adder graphs for $y = (181 + j \cdot 181)x$ with and without pipelining

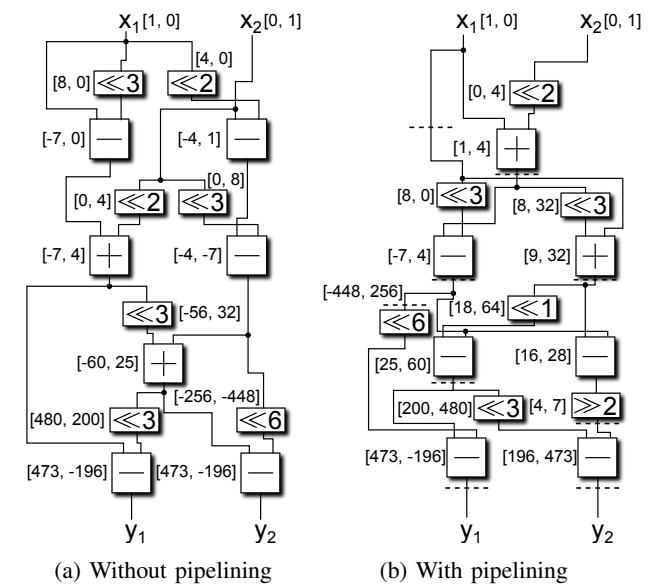


Fig. 16: Adder graphs for $y = (473 + j \cdot 196)x$ with and without pipelining