Strided Difference Bound Matrices

Arjun Pitchanathan[1][0000−0002−7301−2307], Albert Cohen[2][0000−0002−8866−5343], Oleksandr Zinenko[2][0000−0003−1978−0222], and Tobias Grosser[3][0000−0003−3874−6003]

[1] University of Edinburgh, UK, arjun.pitchanathan@ed.ac.uk,
[2] Google DeepMind, Paris, France {albertcohen,zinenko}@google.com
[3] University of Cambridge, UK, tobias.grosser@cst.cam.ac.uk

Abstract. A wide range of symbolic analysis and optimization problems can be formalized using polyhedra. Sub-classes of polyhedra, also known as sub-polyhedral domains, are sought for their lower space and time complexity. We introduce the Strided Difference Bound Matrix (SDBM) domain, which represents a sweet spot in the context of optimizing compilers. Its expressiveness and efficient algorithms are particularly well suited to the construction of machine learning compilers. We present decision algorithms, abstract domain operators and computational complexity proofs for SDBM. We also conduct an empirical study with the MLIR compiler framework to validate the domain's practical applicability. We characterize a sub-class of SDBMs that frequently occurs in practice, and demonstrate even faster algorithms on this sub-class.

1 Introduction and Motivation

The analysis and verification of computing systems involves a variety of abstractions of the system semantics. Among these, numerical abstractions capture arithmetic properties of system variables, supporting mathematical models of systems such as timed and hybrid automata [3,2,25] and the static analysis of inductive definitions in loops and recursive programs [18]. Many of these abstractions implement special cases of Presburger arithmetic [50] where typical decision problems are NP-hard. The simplest special cases are non-relational, such as interval bounds $\pm x \leq c$ where x is a variable and c is a numeric constant. More expressive, relational cases include systems of inequalities of the form $\pm x \pm y \leq c$ known as Unit Two Variable Per Inequality (UTVPI) systems. They form the *octagon abstract domain* [38]. While being much cheaper to operate upon than convex polyhedra [18,4], UTVPI are sufficiently expressive to represent a wide range of multi-variable problems [39].

UTVPI algorithms rely on a Difference Bound Matrix (DBM) representation [7], with inequalities of the form $x - y \leq c$ or $\pm x \leq c$. DBMs are ubiquitous in formal verification [6] and static analysis [8]. Other abstractions such as congruences over linear combinations of integral variables [24] capture only the lattice structure of Presburger sets but not the inequalities. The special case of congruence equalities $x \equiv r \mod d$ where r, d are integral constants and $0 \leq r < d$ has low complexity [23] and is often used to enhance other abstract domains [14].

It has remained an open problem whether there are efficient algorithms for the conjunction of UTVPI and congruence constraints. Such a domain would have numerous applications in the analysis and optimization of machine learning (ML) models. Indeed, modern ML compilers [53,11,33,48] often use a form of Presburger representation for ML compute graphs and operations, e.g. to capture the data layout in memory or conversions such as reshaping and padding.

Affine expressions also arise in program transformations to leverage modern hardware, such as vectorization, fusion and thread-level parallelization. Most of these expressions represent hyper-rectangular shapes, with occasional cases of symmetric and triangular ones (Cholesky factorization and sequence models [35,12]), all of which can be expressed as UTVPI [49]. On the other hand, strides and block sizes resulting from (dilated) convolutions, pooling and normalization operations as well as the results of the tiling (block-wise decomposition) transformation require congruence constraints. While some of the most advanced compiler optimizations justify the efforts to implement full-fledged Presburger arithmetic packages such as isl [50] and FPL [44], the majority of simpler cases call for a definition of a relational abstract domain combining UTVPI and congruences with a low-degree polynomial complexity. We also expect such a domain to be applicable to verification efforts [13,46,5] that currently rely on Presburger arithmetic libraries and SMT solvers; we present early results in Section 6.3.

This paper considers the conjunction of inequalities represented as a DBM with single-variable congruences, a novel abstract domain we call *Strided Difference Bound Matrices* (SDBM). We also study a sub-case of these, *Harmonic SDBM* (HSDBM), where such congruences form a harmonic sorted chain, which is common in congruences produced by loop tiling in high-performance code.

Although the SDBM satisfiability problem turns out to be NP-hard, we are able to provide and algorithm that runs in $\mathcal{O}(nmD_{\text{lcm}})$ time, where n is the number of variables, m is the number of constraints and D_{lcm} is the least common multiple of all congruence divisors. This time complexity, which is pseudo-linear in D_{lcm}, is practical for program analysis applications. We also present an $\mathcal{O}(n^4)$ complexity algorithm for HSDBM satisfiability.

Finally, we define a normal form for SDBM constraint systems that is computable in at most $3m+m\log(nD_{\text{lcm}})+nD_{\text{lcm}}$ satisfiability checks in the general case, and $3m + m\log(nD_{\text{lcm}}) + n$ checks in the harmonic case. Given two systems in normal form, we show that it only takes linear time to perform the join operation, producing a constraint set admitting a union of solutions, common in abstract interpretation. Moreover, we can perform an equality check based on direct comparison of normal forms.

2 DBMs, SDBMs, and HSDBMs

We consider sets over the integers only, i.e., subsets of $\mathbb{Z}^n$ for some $n \in \mathbb{N}$. We first define some notation. For $m, n \in \mathbb{N}$, $[n]$ denotes the set $\{1, \ldots n\}$, $n\mathbb{Z}$ denotes the set of integer multiples of n, and $m \mid n$ denotes that m divides n. If G is a weighted graph with no negative cycles and u and v are vertices in it, then

$\delta_G(u, v)$ is the distance from u to v in G. $\lfloor x \rfloor_y$ refers to x rounded down to the nearest multiple of y smaller than or equal to x. If x, y are vectors and t a scalar, $x + t$ refers to element-wise addition.

Let us first formally recall the definition of Difference Bound Matrices (DBM) over integers and their properties [20,37] before presenting SDBM.

Definition 1. *A* Difference Bound Matrix (DBM) *is a constraint system over variables $x_1, \dots x_n \in \mathbb{Z}$ of the form*

$$-x_i + x_j \leq c_{ij} \qquad \ell_i \leq x_i \leq u_i \qquad (i, j, c_{ij}) \in E \text{ and } l_i, l_j \in \mathbb{Z}$$

where $E \subseteq [n] \times [n] \times \mathbb{Z}$ denotes the set of difference bound constraints. We will use $m = |E|$ to denote the number of such constraints.

Not all upper and lower variable bounds ℓ_i, u_i may be present. When no such variable bounds are present we call the system variable-bound-free (VBF)*; otherwise we say that the system has variable bounds.*

It is known that the satisfiability of DBM constraints can be determined in $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^2)$ space [7,39]. We now define two special cases of Presburger sets derived from DBMs by introducing additional congruence constraints.

Lemma 2 (DBM Shifting Lemma). *If x is a solution to a VBF DBM, then so is $x+t$ for $t \in \mathbb{Z}$, i.e., adding a constant to all variables preserves satisfiability.*

Proof. All constraints are bounds on differences of variables, and the differences don't change when adding a constant to all variables. □

Corollary 3. *If $S_{i,t}$ is the set of solutions to a VBF DBM such that $x_i = t$, then $S_{i,t} = \{x + t \mid x \in S_{i,0}\}$.*

Given a DBM with variable bounds, we can construct a new VBF system by adding a new variable x_0 and converting all variable bounds $\ell_i \leq x_i \leq u_i$ to difference bound constraints $\ell_i \leq x_i - x_0 \leq u_i$. Clearly $(x_1, \dots x_n)$ is a solution to the original system iff $(0, x_1, \dots x_n)$ is a solution to the new system. By the above lemma, the new system has a solution with $x_0 = 0$ iff it has any solution. Thus the original DBM with variable bounds is satisfiable iff the new VBF DBM is. VBF DBMs are best understood by analyzing their *potential graphs*.

Definition 4. *The* potential graph of a DBM *is a weighted directed graph over vertex set $[n]$ with an edge from i to j of weight c_{ij} for each $(i, j, c_{ij}) \in E$. The weights may be negative and the graph may contain negative cycles.*

Lemma 5. *Let $G = ([n], E)$ be the potential graph of a DBM. If G has a path from vertex u to v of total weight W, then $-x_u + x_v \leq W$ for every solution x to the DBM.*

Corollary 6. *If the graph has negative cycles, then no solution x exists.*

If the graph has no negative cycles, then for all $u, v \in [n]$, it holds that $-x_u + x_v \leq \delta_G(u, v)$. This is useful to define a normal form of the DBM.

Definition 7. *A* path-closed DBM *is one that is satisfiable and, for all u, v such that there exists a path from u to v in the potential graph G, the bound on $-x_u + x_v$ exists and is equal to $\delta_G(u, v)$.*

Clearly, any DBM can be brought to path-closed form by computing the distances in the potential graph, and by Corollary 6, doing so does not change the solution set. Moreover, the path-closed form has the following useful property.

Lemma 8 (DBM Projection Lemma). *If a DBM is path-closed, then the projection of its solution set onto a subset of variables is equal to the solution set of the constraints involving only those variables.*

It follows that the path-closed form is the tightest constraint system with the same solution set as the original system, i.e., in a path-closed DBM there exist solutions on the surface of every inequality, so no inequality can be further tightened without changing the solution set. Moreover, if there is no constraint on some $-x_i + x_j$ then adding any upper bound on this changes the solution set. Finally, the following is useful to compute a complete explicit solution.

Lemma 9. *For any vertex u in the potential graph from which all other vertices are reachable, the assignment $x_v = \delta_G(u, v)$ satisfies the DBM.*

Note that in this solution, $x_u = 0$. We now define the new abstract domains.

Definition 10. *A* Strided DBM (SDBM) *is a DBM with additional constraints*

$$x_i \equiv r_i \mod d_i \qquad\qquad i \in [n]$$

where all d_i, r_i are in $\mathbb{Z}$. When referring to such a system, D_{lcm} will denote $\operatorname{lcm}(d_1, \ldots d_n)$. *Given an SDBM, we define the* underlying DBM *as the constraint system without these congruence constraints.*

Note that one may encode the lack of a congruence constraint as $x_i \equiv 0 \mod 1$.

Definition 11. *A* Harmonic SDBM (HSDBM) *constraint system is an SDBM where the congruence divisors are sorted and each one divides the next, i.e., $d_1 \mid d_2 \mid \cdots \mid d_n$.*

3 Satisfiability

We start by reducing the SDBM satisfiability problem to a simpler form. Firstly, let $y_i = x_i - r_i$. Then we can see that $x_i \equiv r_i \mod d_i$ iff $y_i \equiv 0 \mod d_i$. Furthermore, $-x_i + x_j \leq c_{ij}$ iff $-y_i + y_j \leq c_{ij} + r_i - r_j$. Thus the original SDBM

$$x_i \equiv r_i \mod d_i \qquad -x_i + x_j \leq c_{ij} \qquad \ell_i \leq x_i \leq u_i$$

is satisfiable iff the following system is:

$$y_i \equiv 0 \mod d_i \qquad -y_i + y_j \leq c_{ij} + r_i - r_j \qquad \ell_i - r_i \leq y_i \leq u_i - r_i.$$

Thus we reduce satisfiability of any SDBM to the satisfiability of another SDBM where all congruence constraints have remainder zero. We can further reduce satisfiability of SDBMs with variable bounds to satisfiability of VBF SDBMs. To do this, we generalize the DBM shifting lemma to SDBMs.

Lemma 12 (SDBM Shifting Lemma). *If x is a solution to a VBF SDBM, then so is $x + tD_{lcm}$ for $t \in \mathbb{Z}$.*

Proof. By the DBM shifting lemma (Lemma 2), the inequality constraints continue to be satisfied. Since the scalar being added is a multiple of all the congruence divisors, the congruence constraints also continue to be satisfied. □

Corollary 13. *For a given VBF SDBM with the congruence constraint on x_n being $x_n \equiv 0 \mod D_{lcm}$, let S_t be the set of solutions such that $x_n = t$. Then $S_t = \{x + t \mid x \in S_0\}$ for $t \in D_{lcm}\mathbb{Z}$. (Of course, $S_t = \varnothing$ for non-congruent t).*

We convert SDBMs to VBF form similarly to the procedure for DBMs. Let C be an SDBM with variable bounds and all remainders zero. Now create a VBF SDBM C' by adding a variable x_0 and replacing the constant bounds $\ell_i \le x_i \le u_i$ of C with inequalities $\ell_i \le x_i - x_0 \le u_i$. Then the set of solutions of C is equal to the set of solutions of C' such that $x_0 = 0$. Now by the above corollary, if we add the constraint that $x_0 \equiv 0 \mod D_{\text{lcm}}$, then C' is satisfiable iff C is satisfiable. Thus satisfiability of SDBMs with variable bounds can be efficiently reduced to satisfiability of the following simpler class of SDBMs.

Definition 14. *A constraint system of the form*

$$x_i \in d_i\mathbb{Z} \qquad -x_i + x_j \le c_{ij} \qquad (i, j, c_{ij}) \in E$$

is called a simple SDBM. *We sometimes refer to d_i as the* stride *of the variable x_i. When the system satisfies $d_1 \mid \cdots \mid d_n$, we call it a* simple HSDBM.

3.1 GCD-Tightening constraints

If a DBM is unsatisfiable, repeatedly applying the following inference rule will produce a contradiction eventually.

$$-x_i + x_j \le c_{ij} \wedge -x_j + x_k \le c_{jk} \Rightarrow -x_i + x_k \le c_{ij} + c_{jk} \quad \text{(path inference rule)}$$

This is because if the DBM is unsatisfiable then a negative cycle exists, and in that case, repeatedly applying the above leads to an inequality of the form $0 \le c$ for some negative c. In an SDBM, if the underlying DBM is unsatisfiable then the above is true. However, it is possible for an unsatisfiable SDBM to have its underlying DBM be satisfiable. Consider the following example:

$$x, y \in 2\mathbb{Z} \qquad 1 \le x - y \le 1$$

The inequalities on their own are clearly satisfiable over the integers. However, because both x and y are even, $x - y$ cannot be 1 as required by the above

system. Due to the congruence constraints, $x - y \leq 1$ implies $x - y \leq 0$ and similarly $1 \leq x - y$ implies $2 \leq x - y$, so the system is unsatisfiable. In general, by Bézout's lemma, when $x \in a\mathbb{Z}$, $y \in b\mathbb{Z}$, then $x - y \in \gcd(a, b)\mathbb{Z}$. Thus we can always tighten bounds to a multiple of the GCD, leading to a new inference rule:

$$-x_i + x_j \leq c_{ij} \implies -x_i + x_j \leq \lfloor c_{ij} \rfloor_{\gcd(d_i, d_j)} \qquad \text{(GCD-tightening rule)}$$

We use the above to define a GCD-tight SDBM.

Definition 15. *A* GCD-tight SDBM *is one where, for all* $i, j \in [n]$*, we have* $c_{ij} \mid \gcd(d_i, d_j)$*.*

These two rules are still not sufficient to determine if an SDBM is satisfiable. The following system is GCD-tight, path-closed, and the inequalities are satisfiable over integers, but the system as a whole is unsatisfiable.

$$\begin{aligned} x &\equiv 0 \mod 4 \cdot 5 & 0 \leq y - x \leq 5 \\ y &\equiv 0 \mod 5 \cdot 7 & 20 \leq x - z \leq 24 \\ z &\equiv 0 \mod 4 \cdot 7 & 21 \leq y - z \leq 28 \end{aligned} \tag{1}$$

To see that it is unsatisfiable, reparameterize the solution as $(c+a, c+b, c)$; this vector is a solution to the congruences iff

$$c \equiv a \mod 4 \cdot 5 \qquad c \equiv b \mod 5 \cdot 7 \qquad c \equiv 0 \mod 4 \cdot 7$$

which by the general Chinese remainder theorem [42] has solutions iff

$$a \equiv b \mod 5 \qquad a \equiv 0 \mod 4 \qquad b \equiv 0 \mod 7.$$

Since the solution is of the form $(a, b, 0) + c$, it satisfies the inequalities iff $(a, b, 0)$ does, by Lemma 2. Thus the inequalities hold iff

$$0 \leq b - a \leq 5 \qquad 20 \leq a \leq 24 \qquad 21 \leq b \leq 28.$$

Due to the congruence constraints we have $a \in \{20, 24\}$, $b \in \{21, 28\}$, and $b - a \in \{0, 5\}$, which cannot be satisfied simultaneously, so the SDBM is unsatisfiable. For the case of HSDBMs however, path-closure and GCD-tightening suffice.

3.2 Satisfiability for HSDBMs in $O(n^4)$ time

By the earlier discussion, we can assume that the given HSDBM is simple. In this case, path-closure and GCD-tightening are sufficient to determine satisfiability. To show this, we prove a projection lemma for HSDBMs; while the general projection lemma for DBMs (Lemma 8) does not apply to HSDBMs, it does hold when the subset of variables chosen forms a suffix. We will call an HSDBM path-closed when its underlying DBM is path-closed.

Lemma 16. *Let* H *be a path-closed, GCD-tight VBF HSDBM. If* $S_{k:n}$ *is the projection of the solution set of* H *onto* $x_k, \ldots x_n$*, then* $S_{k:n}$ *is equal to the set of solutions to the inequalities and congruence constraints involving only* $x_k, \ldots x_n$*.*

Proof. Suppose $(p_{k+1}, \dots p_n) \in S_{k+1:n}$. We show that there exists a p_k such that $(p_k, \dots p_n) \in S_{k:n}$. By substituting $p_{k+1}, \dots p_n$ into the system, we obtain bounds of the form $p_i - c_{ki} \le x_k \le p_i + c_{ik}$ for $k < i \le n$ on x_k when the corresponding inequalities exist. If none of the lower bounds exist or none of the upper bounds exist, then we can definitely find a multiple of d_k satisfying these bounds to assign to x_k.

Otherwise, if at least one upper bound and one lower bound is produced, then the set of x_k satisfying the inequalities is $[\max_i(p_i - c_{ki}), \min_i(p_i + c_{ik})]$, which is of the form $[p_i - c_{ki}, p_j + c_{jk}]$ for some $i, j \in \{k+1, \dots n\}$. This interval is non-empty by the DBM projection lemma (Lemma 8).

Now note that $p_i \in d_i\mathbb{Z} \subseteq d_k\mathbb{Z}$ by the harmonic property and similarly $p_j \in d_k\mathbb{Z}$. Also, $c_{ki} \in \gcd(d_k, d_i)\mathbb{Z} = d_k\mathbb{Z}$ by path-closure and the harmonic property; similarly, $c_{jk} \in d_k\mathbb{Z}$. So both the endpoints lie in $d_k\mathbb{Z}$ and therefore it certainly contains a multiple of d_k. Repeating this, we can extend any point in $S_{k:n}$ into a point in $S_{1:n}$, a full solution to the whole system. □

Corollary 17. *A path-closed GCD-tight HSDBM is satisfiable.*

Proof. $S_{n:n} = d_n\mathbb{Z} \neq \varnothing$ is the projection of the solution set onto x_n. □

This forms the basis of the SolveHSDBM algorithm in Figure 1 to decide the satisfiability of HSDBMs: first, obtain the path-closure of the inequalities by running the Floyd-Warshall algorithm [16] on the potential graph, then GCD-tighten all inequalities, and repeat these two steps until a fixpoint or contradiction is reached, at which point we know whether the system is satisfiable.

```
1: function SolveHSDBM(E, d)
2:     Path-close inequalities E
3:     while (E, d) not a fixpoint do
4:         Set every c_ij in E to
5:                                  ⌊c_ij⌋_gcd(d_i,d_j)
6:         if negative cycles in E then
7:             return ⊥
8:         Compute all pairs of distances
9:         Set every c_uv to δ_E(u, v)
10:    return SAT
```

```
1: function SolveSDBM(E, d)
2:     if no integral solution to E then
3:         return ⊥
4:     p ← an integral solution to E
5:     D_lcm ← lcm(d_1, ... d_n)
6:     ℓ ← p − nD_lcm
7:     u ← p + nD_lcm
8:     for i ∈ [n] do u_i ← ⌊u_i⌋_{d_i}
9:     while fixpoint not reached do
10:        for (i, j, c_ij) ∈ E do
11:            if u_j < ⌊u_i + c_ij⌋_{d_j} then
12:                u_j ← ⌊u_i + c_ij⌋_{d_j}
13:                if u_j < ℓ_j then return ⊥
14:    return u
```

Fig. 1. HSDBM and SDBM satisfiability.

Lemma 18. *Let $G = (V, E)$ be a transitively closed graph with no negative cycles, i.e. whenever there is a path from u to v, there is an edge from u to v*

of weight $\delta_G(u, v)$. Let $U \subseteq V$. Now let F be a copy of E in which we have decreased the weights of some edges that go from one vertex in U to another in U. Finally, let $H = (V, F)$.

Suppose that H has no negative cycles. Then for any vertices u and v in U with a path from u to v, there is a shortest path from u to v that never leaves U.

Proof. Let $p = (p_1, \ldots p_k)$ be the vertices of a path starting and ending in U and with all the intermediate vertices lying outside U. Let W be the weight of p in H and let c be the weight of the edge from p_1 to p_k in H. Then $W \geq \delta_G(p_1, p_k)$ because only edges that stay within U decreased, and $\delta_G(p_1, p_k) \geq c$ because the edge in G had weight equal to $\delta_G(p_1, p_k)$ and it can only have decreased or stayed the same in H. Thus the path p cannot have weight less than the weight of the direct edge in H.

For a general path that goes in and out of U repeatedly, we can always replace all sections of the path that go outside and come back in with the direct edges that stay in U, to obtain a path within U whose weight is at most that of the original path. Thus for any start and end point in U, the shortest path that stays in U has weight equal to the shortest path in the entire graph H. □

Theorem 19. SolveHSDBM *in Figure 1 terminates in $O(n^4)$ time.*

Proof. We will view the algorithm as operating on the potential graph; all modifications to c_{ij} then become modifications to the edge weights. We will show that at most $n-1$ repetitions are needed for fixpoint. We prove that after the ith application of GCD tightening, all edges between vertices in $\{v_i, \ldots v_n\}$ will stay multiples of d_i for the rest of the algorithm. We prove this by induction. The base case for $i = 1$ is true since when all edges are multiples of d_1, path-closure cannot change this divisibility, and GCD-tightening will not change this either.

Now assume it to be true for i; we will show it for $i + 1$. The $i + 1$-th application of GCD tightening only decreases edge weights between vertices in $U = \{v_{i+1}, \ldots v_n\}$, by the induction hypothesis. Now we want to analyze how path closure affects the edge weights in the subgraph induced by U. After tightening, all edges in the subgraph are multiples of d_{i+1}, so distances between nodes in the subgraph are also multiples of d_{i+1} by Lemma 18. Thus path closure does not affect divisibility at this step. Therefore, subsequent GCD-tightening does not affect it either. Repeated applications of these preserve the property.

Thus the nth application of GCD tightening does nothing since there are no edges in the graph induced on the single vertex v_n for $i = n$. Therefore, neither does the subsequent application of path-closure. Thus, fixpoint is achieved after $n - 1$ runs of GCD-tightening and path-closure. □

3.3 Satisfiability for SDBMs in $O(nmD_{\mathbf{lcm}})$ time

Extending work by Lagarias [32], it can be shown that the SDBM satisfiability problem is NP-hard (see the appendix of the extended paper [43]) so no polynomial-time algorithm is likely to exist. In program analysis applications,

the inequality coefficients can be large, so we would like an algorithm that runs in polynomial time in the representation size of these coefficients. On the other hand, in these applications, the congruence divisors are typically small, so we are willing to let the algorithm be polynomial in the *values* of these, i.e., pseudo-polynomial in these. In fact, these divisors typically share many common factors, so that their LCM is not much bigger than the divisors. We present an algorithm that is pseudo-linear in the LCM.

The intuition for the algorithm comes from the following extensions of our inference rules to upper bounds $x_i \leq u_i$ on the variables.

$$x_i \leq u_i \implies x_i \leq \lfloor u_i \rfloor_{d_i}$$
$$x_i \leq u_i \wedge -x_i + x_j \leq c_{ij} \implies x_j \leq u_i + c_{ij}$$

Suppose we have an SDBM with all variables bounds present and we keep applying these rules. Then we either obtain a contradiction $u_i < \ell_i$, or a fixpoint. At the fixpoint it holds that $u_i \in d_i\mathbb{Z}$ and moreover $u_j \leq u_i + c_{ij}$. So in fact, u becomes a solution to the SDBM. Each successful application of an inference rule reduces the gap $u_i - \ell_i$ between some upper bound and lower bound. If this difference becomes negative, a contradiction is obtained and the algorithm halts.

So the worst-case runtime of this method depends on the sum of the gaps $u_i - \ell_i$ between the upper bounds and the lower bounds, which could naively be exponential in the representation size of the constraint system. To avoid this worst-case scenario, we reduce the satisfiability of SDBMs to the satisfiability of SDBMs with variable bounds where the gap between the upper and lower bound is at most $2nD_{\text{lcm}}$ for each variable.

For a matrix A, let $\text{MASD}(A)$ be the maximum absolute determinant among all square submatrices of A. A standard fact [15] in the theory of integer programming is that if $P \subseteq \mathbb{R}^n$ is a polyhedron, $P \cap \mathbb{Z}^n$ is non-empty, and x is in P, then there exists a point y in $P \cap \mathbb{Z}^n$ such that $\|x - y\| \leq n\,\text{MASD}(A)$. We slightly generalize this to obtain the following lemma.

Lemma 20. *Let $S = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ be a non-empty polyhedron. Let L be the set of solutions to some single-variable congruence constraints such that $S \cap L \neq \varnothing$, and let D_{lcm} be the LCM of the congruence divisors of L. Let $y \in S$. Then there exists a solution $x \in S \cap L$ such that $\|x - y\|_\infty \leq nD_{lcm}\,\text{MASD}(A)$.*

Moreover, we show that $\text{MASD}(A) = 1$ for DBMs, making the bound nD_{lcm}.

Lemma 21. *Let A be an $m \times n$ matrix where each row has exactly one $+1$ and one -1. Then $\text{MASD}(A) = 1$.*

We defer the proofs to the appendix of the paper's extended version [43]. These lemmas allow to solve an SDBM by first finding any integral solution p to the inequalities and adding constant bounds on the variables to lie within a box of side length $2nD_{\text{lcm}}$ centered at p, then applying the above inference rules until reaching a contradiction or fixpoint.

In SolveSDBM in Figure 1, we first GCD-tighten all the upper bounds and then look for opportunities to apply the path-closure inference rule to the upper

bounds, by checking each difference bound. Whenever an upper bound decreases due to path-closure, we immediately apply the GCD-tightening rule to it. It takes $O(m)$ time to look over all edges. Since each variable's upper and lower bounds differ by $2nD_{\text{lcm}}$, there can be at most $2n^2D_{\text{lcm}}$ steps of such tightening, for an overall runtime of $\mathcal{O}(n^2mD_{\text{lcm}})$.

We have to process the edge (i, j, c_{ij}) once in the beginning. After that, we only have to process it again when the RHS of the if-condition on line 11 changes, i.e., only when u_i decreases. So we can replace lines 9-13 with the following.

1: `dirty` $\leftarrow [n]$
2: **while** `dirty` $\neq \varnothing$ **do**
3: Pick $i \in$ `dirty`
4: Remove i from `dirty`
5: **for** $(i, j, c_{ij}) \in E$ **do**
6: **if** $u_j < \lfloor u_i + c_{ij} \rfloor_{d_j}$ **then** $u_j \leftarrow \lfloor u_i + c_{ij} \rfloor_{d_j}$
7: **if** $u_j < \ell_j$ **then return** $\perp$
8: Add j to `dirty`

Here u_i can decrease at most $2nD_{\text{lcm}}$ times since after that it will go below the lower bound ℓ_i and produce a contradiction. Each time u_i decreases, we check all edges that go out from i, as these are the edges that might use the reduced value of u_i. Thus if o_i is the number of edges leaving i, then the time complexity of this more careful implementation is $\sum_i \mathcal{O}(nD_{\text{lcm}}o_i) = \mathcal{O}(nD_{\text{lcm}}m)$ since $\sum_i o_i = m$.

4 HSDBM Normalization

We consider normalization for satisfiable systems; if a system is unsatisfiable we normalize it by setting it to some canonical unsatisfiable system. We first normalize the inequalities and then the congruence constraints.

Definition 22. *An* inequality-normalized (H)SDBM *is one where for any bound $-x_i + x_j \leq c$, if it holds in the solution set that $-x_i + x_j \leq d$, then $c \leq d$, i.e. the bound in the system is the tightest valid bound.*

Note that any two SDBMs with the same solution sets will have the same normalized inequalities, since this depends only on the solution set and not on the form of the initial constraint system. The above definition is equivalent to saying that every bound $-x_i + x_j \leq c$ has a solution that makes it tight, and whenever a bound does not exist that expression can take arbitrarily large values in the solution set. Also, an inequality-normalized system is always path-closed and GCD-tight since no such tightening inference rules can decrease any bound.

We previously showed that path-closure and GCD-tightening are not sufficient to check satisfiability of SDBMs. Thus, we do not expect these to be sufficient for inequality normalization either. One might hope that it is enough for HSDBMs, but in fact it is not the case either. Consider the following example.

$$\begin{array}{llll} -1 \leq x - y \leq 1 & -1 \leq x - w \leq 0 & 0 \leq x - z \leq 1 & x, y \in \mathbb{Z} \\ 0 \leq w - z \leq 2 & -1 \leq y - w \leq 0 & 0 \leq y - z \leq 1 & z, w \in 2\mathbb{Z} \end{array}$$

It is obviously GCD-tight and path-closed. But all constraints are not as tight as possible. Note that w is either z or $z+2$. If $w = z$ then $x-z = x-w = 0$, and similarly $y-z = 0$, implying $x-y = 0$. Otherwise $w = z+2$, then $x-w = x-z = 1$ and $y - z = 1$, so $x - y = 0$ again, yielding tighter inequalities $0 \leq x - y \leq 0$. Therefore, we need to do more for inequality normalization.

First, let us consider variable-bound-free systems. Suppose the system has an inequality $-x_i + x_j \leq c$ and we want to check if replacing it by $-x_i + x_j \leq b$ for $b < c$ excludes any solutions. This is equivalent to asking if there are any solutions with $b+1 \leq -x_i + x_j \leq c$, which is a single satisfiability check. We can thus binary search over the values of b to find the minimum valid one, to obtain the tightest form of the inequality. We now establish a bound on the range of such b values over which we have to search.

By the projection lemma (Lemma 8), path-closing the underlying DBM brings it to normal form. Therefore every difference bound $-x_i+x_j \leq c_{ij}$ has an integral point y satisfying all the inequalities and such that $-y_i + y_j = c_{ij}$. By Lemma 20, there exists a solution z to the whole system with $z_j \geq c_{ij} - nD_{\text{lcm}}$ and $z_i \leq c_{ij} + nD_{\text{lcm}}$, so that $-z_i + z_j \geq c_{ij} - 2nD_{\text{lcm}}$. Therefore, the tightest version of the inequality has a bound that is tighter by at most $2nD_{\text{lcm}}$. The binary search then takes at most $3 + \log(nD_{\text{lcm}})$ steps. Inequality normalization thus takes at most $m(3 + \log(nD_{\text{lcm}}))$ emptiness checks.

Now consider systems with variable bounds, still with remainder zero congruence constraints. Path-closure and GCD-tightening are sufficient to normalize these, by converting them to VBF form and applying the following lemma.

Lemma 23. *If an HSDBM is path-closed and GCD-tight then all inequalities involving x_n are tight. If a bound on $-x_i + x_n$ is missing then $-x_i + x_n$ is unbounded in that direction, and similarly for bounds on $-x_n + x_i$.*

Proof. First, we show it for bounds of the form $-x_n + x_i \leq c_{ni}$. Suppose all such bounds exist. Then the point with $x_n = 0$ and $x_i = c_{ni}$ for $i < n$ is a solution. By GCD-tightening and the harmonic property, it satisfies the congruences. By path-closure, we have $c_{nj} \leq c_{ni} + c_{ij}$, so it satisfies the inequalities.

Now consider the case where all bounds do not exist. Let R be the set of variables that have a bound on $-x_n + x_i$ and let $\overline{R}$ be its complement. Set $x_i = c_{ni}$ as before, for variables in R, except x_n which we set to zero. Now we find a way to fill in the values of x_j in $\overline{R}$. Note that there can be no bound of the form $-x_i + x_j \leq c_{ij}$ for $x_i \in R, x_j \in \overline{R}$ because then by path-closure we would have a bound $-x_n + x_j \leq c_{ni} + c_{ij}$ which contradicts $x_j \in \overline{R}$.

Thus, assigning values to variables in R can impose lower bounds on variables in $\overline{R}$, but not upper bounds. Since the whole HSDBM is non-empty we can find a solution y to the subsystem of constraints that only involve variables in $\overline{R}$. Moreover, $t + y$ is a solution for any real $t \in D_{\overline{R}}\mathbb{Z}$ where $D_{\overline{R}}$ is the LCM of the congruence divisors of variables in $\overline{R}$. By making t sufficiently large, $t + y$ satisfies the lower bounds imposed by substituting values for R variables. Thus we have a solution making all the bounds c_{ni} tight. Also, by increasing t we can make the variables in $\overline{R}$ arbitrarily large so these are unbounded above.

To prove the case of bounds on $-x_i + x_n$, negate all the variables so that bounds on $-x_n + x_i$ become bounds on $-x_i + x_n$ and vice versa. Now we can apply the same proof as above. □

Therefore, to normalize a satisfiable HSDBM with variable bounds, we:

1. Convert the system into VBF form,
2. Bring the converted system into path-closed and GCD-tightened form,
3. Convert the system back to a form with variable bounds, and
4. Binary search on the remaining inequalities to normalize them.

If the system was not satisfiable, we would find out at step 2, at which point we can normalize the system by setting it to some canonical unsatisfiable HSDBM.

Let us now consider how to congruence-normalize simple HSDBMs; in this setting we require that the normal form's congruence constraints have remainder zero.

Definition 24. *A* congruence-normalized VBF HSDBM *where the congruence constraint system implies all other valid congruence constraint systems for that solution set, i.e., an HSDBM with congruence divisors $d_1^*, \dots d_n^*$ is congruence-normalized if for all HSDBMs with the same solution set having congruence divisors say $d_1, \dots d_n$, it holds that each $d_i \mid d_i^*$.*

Note that the above definition depends only on the solution set of a system, and so the normalized congruence system of any two systems having the same solution set will be the same. In a simple HSDBM, x_n can always take all values in $d_n\mathbb{Z}$ by the shifting lemma (Lemma 12), so any system with the same solution set will have the same congruence d_n for x_n. Therefore, $d_n^* = d_n$. We now normalize the remaining congruences iteratively, starting from x_{n-1} and going downwards. Suppose that we already computed $d_{i+1}^*, \dots d_n^*$ and we want to compute d_i^*.

Note that for any valid congruence system it holds that $d_i \mid d_{i+1} \mid d_{i+1}^*$ by the harmonic property and congruence normalization. Thus d_i^* is the maximum of all $d_i \mid d_{i+1}^*$ such that $x_i \in d_i\mathbb{Z}$ holds in the solution set. By the projection lemma (Lemma 16), we can reduce this to finding the largest possible divisor for x_1 in a given constraint system with divisors $d_1, \dots d_n$. As shown above we only need to consider divisors $m \mid d_2$. For it to be a valid divisor, it also needs to not be so dense as to allow additional solutions; we ensure this by mandating that $d_1 \mid m$. Note that the greatest divisor cannot be a non-multiple of d_1 anyway, since if m is a valid congruence for x_1 then so is $\operatorname{lcm}(d_1, m)$.

Theorem 25. *Let H be an inequality-normalized simple HSDBM. Let L be the set of $m \in \mathbb{N}$ such that $d_1 \mid m \mid d_2$ and for any solution x of H, it holds that $x_1 \in m\mathbb{Z}$. We are interested in the sparsest possible congruence divisor,* $\max L$. *Let g be the GCD of all c_{i1} and c_{1i}, and let $q = \gcd(g, d_2)$.*

Then $\max L$ *is either d_1 or q. Moreover, it is q iff a specific other HSDBM H' is unsatisfiable, where the constraint system H' can be computed in linear time from the system H.*

Proof. We first show that for any $r \in L$, $r \mid g$. Suppose not, then without loss of generality, r does not divide some c_{i1}. Since the system is inequality normalized, it has some solution satisfying $x_1 = x_i + c_{i1}$. But since $x_i \in d_i\mathbb{Z} \subseteq r\mathbb{Z}$ and $c_{1i} \notin r\mathbb{Z}$, we have $x_1 \notin r\mathbb{Z}$, so $r \notin L$ which is a contradiction. So this case is impossible and we have $r \mid g$. Since $r \mid d_2$, we have $r \mid \gcd(g, d_2) = q$. Thus, $\max L \mid q$. If $q = d_1$, we are done and $\max L = d_1$.

Otherwise, let $q \neq d_1$. We now show that either $q \mid \max L$, implying $\max L = q$, or $\max L = d_1$. Let $S_{2:n}$ be the projection of the solution set onto $x_2, \ldots x_n$. For now, assume that all constraints in the system exist. Then every assignment $(p_2, \ldots p_n) \in S_{2:n}$ implies constraints of the form $p_i - c_{1i} \leq x_1 \leq p_i + c_{i1}$. So the set of possible x_1 values for this assignment is $\bigcap_{i=2}^{n} [p_i - c_{1i}, p_i + c_{i1}] \cap d_1\mathbb{Z}$. This set is non-empty by the definition of $S_{2:n}$. Since $q \mid d_2 \mid p_i$ for all $i \geq 2$ and q divides all the coefficients c_{1i} and c_{i1}, all interval endpoints are multiples of q. Therefore the endpoints of the intersection are also multiples of q. Since $d_1 \mid q$, if the intersection contains more than one element then it definitely contains two adjacent multiples of d_1, implying $\max L = d_1$. Otherwise, if the intersection contains exactly one element, that element is surely a multiple of q.

Thus, $q \mid \max L$ if for all points in $S_{2:n}$, the intersection of the intervals is a singleton. Otherwise, $\max L = d_1$. The intersection of some intervals is a singleton iff the right endpoint of some interval equals the left endpoint of some interval, possibly the same one. So we have to check whether, for every valid assignment in $S_{2:n}$, some two intervals $[x_i - c_{1i}, x_i + c_{i1}]$ and $[x_j - c_{1j}, x_j + c_{j1}]$ intersect only at their endpoints, i.e., there always exist some $i, j \in \{2, \ldots n\}$ such that $x_i + c_{i1} = x_j - c_{1j}$, i.e., $-x_j + x_i = c_{i1} + c_{1j}$. Note that by path closure, if $x_2, \ldots x_n \in S_{2:n}$, then it already holds that $-x_j + x_i \leq c_{ji} \leq c_{j1} + c_{1i}$. So it is only left to check whether $\forall x_2, \ldots x_n \in S_{2:n},\ \exists i, j \in \{2, \ldots n\},\ -x_j + x_i \geq c_{j1} + c_{1i}$. This is equivalent to $\nexists x_2, \ldots x_n \in S_{2:n},\ \forall i, j \in \{2, \ldots n\},\ -x_j + x_i < c_{j1} + c_{1i}$, by logically negating twice. The strict inequality is equivalent to the constraint that $-x_j + x_i \leq c_{j1} + c_{1i} - 1$ since all variables are integers. By the HSDBM projection lemma (Lemma 16), a vector belongs to $S_{2:n}$ iff it satisfies the constraints on those variables in the HSDBM. Thus the condition above can be checked using a single HSDBM satisfiability check.

If some of the c_{1i} or c_{i1} bounds did not exist then the corresponding intervals in the intersection would have ranged till infinity on that side. Still, the same conclusion holds: $\max L \neq d_1$ iff the intersection is a singleton, meaning that some two finite endpoints have to coincide, and the rest of the proof proceeds the same way. Whenever some c_{1i} or c_{i1} does not exist we simply do not add any of the bounds in the constructed system that depend on that bound. □

Generalizing to HSDBMs with variable bounds. When variable bounds exist, it is possible for a variable to take only a single value, in which case any congruence divisor is valid and the sparsest congruence constraint is not well-defined. In this case, in inequality-normalized form, the variable will have upper and lower bounds equal, so we can immediately detect this case by looking at the variable bounds. When this happens, we first eliminate these variables

by substituting in the single value that they can take. We then compute the congruence normalization of the resulting system, then add back the eliminated variables and give them some canonical congruence constraint that preserves the harmonic property. For example, use $x_1 \equiv 0 \mod 1$ if it is the first variable and use the divisor of the previous variable otherwise.

We now consider congruence normalization of systems with variable bounds where every variable takes at least two values.

Lemma 26. *Let C be an SDBM with variable bounds, where each variable takes at least two values. Let C' be the system converted into VBF form with the added variable x_{n+1} having divisor D, with $D_{lcm} \mid D$. Let $e_1, \dots e_n$ be the normalized congruence divisors of the converted system, and let $d_1^*, \dots d_n^*$ be the sparsest congruences for the original system. Then $\forall i, e_i = \gcd(D, d_i^*)$.*

Proof. Let S be the set of values x_i takes in C and let T be the set of values it takes in C'. Then $T = \{x+tD \mid x \in S, t \in \mathbb{Z}\}$ by the shifting lemma (Lemma 12). The sparsest congruence divisor for S is the GCD of all elements in S, which we call g. Similarly, the sparsest congruence divisor for T is the GCD of all elements in T, which is equal to $\gcd(g, D)$ since for any a, $\gcd_{t\in\mathbb{Z}}(a+tD) = \gcd(a, D)$. □

Lemma 27. *In an HSDBM with variable bounds where x_n takes at least two possible values, the sparsest possible congruence divisor for x_n is d_n.*

Proof. Convert the system to VBF form. Let x_{n+1} be the variable added for the conversion. By the projection lemma (Lemma 16), the set of valid values of these two variables is the set of constraints involving only them. The set of valid values of x_n in the original system is the set of values of x_n in the converted system with $x_{n+1} = 0$, and is therefore the set of multiples of d_n within the variable bounds of x_n. Thus the sparsest congruence divisor for x_n is still d_n. □

We now show how to compute the sparsest congruence constraints.

Theorem 28. *Given an HSDBM with variable bounds where every variable takes at least two values, the sparsest congruence constraints are equal to sparsest constraints for the system after converting to VBF form.*

Proof. We convert the system to VBF form by adding a variable x_{n+1} with congruence divisor $d_{n+1} := d_n$. We then compute its congruence normalization to obtain divisors $e_1 \mid \cdots \mid e_n \mid e_{n+1}$. Let $d_1^* \mid \cdots \mid d_n^*$ be the true sparsest congruences for the input system. Then $e_i = \gcd(d_{n+1}, d_i^*)$ by Lemma 26 and $d_{n+1} = d_n = d_n^*$ by Lemma 27. Hence $e_i = \gcd(d_n^*, d_i^*) = d_i^*$ since $d_i^* \mid d_n^*$. □

Generalizing to arbitrary congruence constraints. For HSDBMs with arbitrary congruence constraints, we can find any solution and shift the system so that the origin becomes a solution. Then all valid congruence constraints have remainder zero since there is a solution at the origin. Computing the sparsest possible congruence for this system and performing the inverse shift therefore gives us the sparsest possible congruence for the original system.

5 Operations for Abstract Interpretation

We introduce intersection, equality, inclusion, and join operations for (H)SDBMs, completing the set of operations typically required for abstract interpretation.

Intersection. To intersect, we just take the tighter of the bounds on each $-x_i + x_j$ and of the variable bounds.

Equality. We check if two HSDBMs have equal solution sets by checking if their normal forms are equal. For simple SDBMs, we first check if their normalized inequalities are equal, then compare congruences: given an SDBM C, $D \in \mathbb{N}$ such that all $d_i \mid D$, and $r \in \{1, \dots D-1\}$, there exists a solution with $x_i \equiv r \mod D$ iff there exists one with $x_i = r$, by the shifting lemma (Lemma 12).

Now given two VBF SDBMs, let D be the LCM of their congruence divisors. Checking which values modulo D each variable can take in each system takes $2nD$ satisfiability checks. If both are equal and the normalized inequalities are also equal then both systems have equal solution sets. Otherwise, they do not.

Now given two SDBMs with variable bounds, we again set D to be the LCM of the congruence divisors and inequality normalize both, then convert them to VBF form using the same congruence divisor D for the added variable. The two original systems are equivalent iff the converted systems are, and we know how to check equality of solution sets for VBF SDBMs.

Inclusion. We can check for inclusion using intersection and equality checks since for sets A and B, we have $A \subseteq B$ iff $A \cap B = A$.

Join. Given two SDBMs in normal form, the system with the smallest solution set that encompasses both the inputs' solution sets is the system that takes the looser of the two bounds on each $-x_i + x_j$. When one of the systems has no bound, the result should have no bound either. This follows from Definition 22.

For the congruences of the joined system, we compute the congruence normalization of both the input systems and for each variable, take the sparsest congruence constraints that encompass both. Say the two constraints are $x \equiv r_1 \mod q_1$ and $x \equiv r_2 \mod q_2$. Let p be any solution to these two constraints, then the two constraints are equivalent to $x - p \equiv 0 \mod q_1$ and $x - p \equiv 0 \mod q_2$ respectively. The sparsest constraint that holds for $x - p$ satisfying either one of these constraints is $x - p \equiv 0 \mod \gcd(q_1, q_2)$, i.e., $x \equiv p \mod \gcd(q_1, q_2)$.

Finding an equivalent simple representation of an SDBM. It may sometimes be useful to find a simple representation of an SDBM, if one exists. A satisfiable SDBM C with variable bounds can never have the same solution set as a VBF SDBM C', because by the shifting lemma (Lemma 12), for any solution x of C', there exists a constant D such that $x + kD$ is also a solution for any

integer k. Hence the solution set of C' does not satisfy any variable bounds and so is different from that of C.

A VBF SDBM C with non-zero remainders admits a simple SDBM representation if there is a way to replace its congruence constraints with zero-remainder constraints while preserving the same solution set. This can be determined by computing the possible remainders of all variables modulo D_{lcm}. By the shifting lemma, a remainder $x_i \equiv r_i \mod D_{\text{lcm}}$ is possible iff there is a solution with $x_i = r_i$, which amounts to a satisfiability check.

Let g_i be the GCD of all possible remainders obtained above and D_{lcm}. By the shifting lemma, g_i is the GCD of all valid values of x_i. To ensure that our new congruence constraint for x_i does not invalidate any solutions of the original system, it is necessary and sufficient that the new divisor be a divisor of g_i.

To disallow any extraneous solutions, we make the congruence constraint as sparse as possible. Consider the system C' with congruence constraints $x_i \equiv 0 \mod g_i$ and the inequality constraints of C. C can be represented by a simple SDBM with the same solution set iff C and C' have the same solution set, which we can check as described above.

6 Empirical Study

The goal of this study is to demonstrate the *suitability* of SDBM for program representation and analysis. To this end, we instrumented several optimizing compilers that use polyhedral domains internally and analyzed those domains. Evaluating the compilation time or the run time of the compiled program is beyond the scope of the study as it requires additional engineering to compete with highly-optimized Presburger arithmetic libraries [50,44].

6.1 Methodology

We instrumented the following compilation and analysis projects.

- The MLIR compiler infrastructure [33], widely used in production to support domains ranging from machine learning compilers to hardware synthesis. We used MLIR version `llvmorg-18-init-16246-g4daea501c4fc` (Jan 5, 2024) and compiled the test suite provided with the project using `ninja check-mlir`. We collected statistics from 2176 compiler invocations. Some tests feature multiple invocations.
- The Polygeist CUDA-to-OpenMP cross-compiler [40] based on the archived artifact [41]. We compiled 17 benchmarks from the CUDA subset of the Rodinia suite [10] accepted by Polygeist with the same 7 configurations as [41].
- The PPCG polyhedral compiler [51] version `0.09.1` (Apr 2, 2023, most recent release). We compiled 30 benchmarks from the Polybench/C benchmark suite version `4.2.1` [45] using `ppcg -target=c -openmp -tile` to enable autoscheduling, parallelization and tiling.

MLIR and MLIR-based Polygeist were instrumented to intercept the creation of affine expressions and sets bounded by such expressions as well as (integer) emptiness checks of these sets. For each expression and set, we verified whether it can be expressed as a (H)SDBM. We say that an expression can be In MLIR, unique expressions are reused so that the collected statistics reflect unique SDBM objects that existed throughout the execution of the test. PPCG, and its underlying isl library [50], were instrumented to check if the following objects fit (H)SDBM: affine constraints, convex sets, unions thereof and unions of non-convex sets in multiple vector spaces. We collected all such objects at several moments in the compilation process: after constructing the initial representation, after performing dependence analysis, before and after scheduling, and just before final code generation.

6.2 Prevalence of SDBMs

MLIR. Out of 2176 test cases, 1264 (58.1%) construct affine expressions throughout their lifetime. The following analysis focuses only on those. Overall, 96.3% of affine expressions and 95.6% of integer sets (we consider MLIR multidimensional affine maps as such) can be represented using SDBM. 714 (56.5%) of the cases use only SDBM expressions. In the remaining cases, $90.3\% \pm 15.9$[4] of expressions and $88.2\% \pm 17.5$ sets can be represented using SDBM.

45 of the test cases perform a total of 7695 emptiness checks. 6262 (81.4%) of these are performed on HSDBM integer sets, and none on more general SDBMs. 22 (48.9%) test cases perform emptiness checks only on HSDBM. In the remaining cases, $73.5\% \pm 37.7$ of the checks are performed on HSDBM sets.

These results suggest that SDBM is sufficient to represent a large fraction of affine constructs appearing in a compiler infrastructure supporting polyhedral compilation [31], machine learning compilers [34], high-level synthesis [54] and other hardware design [21]. It is worth noting that the test suite covers rare representational edge cases, so practical applications may have better coverage. For example, many non-SDBM expressions are found in Affine dialect tests, which exercise the full expressive power of (quasi-)affine expressions, including divisions by parameters, huge coefficients, or expressions with hundreds of terms.

Some of the 17 compiled benchmarks consist of multiple translation units processed separately, for a total of 39. Each one was compiled with 7 different configurations, leading to the total of 273 test cases. Out of these, 266 (97.4%) construct affine expressions and 50 (18.3%) perform emptiness checks.

Polygeist. 96.3% of the affine expressions and 95.6% of the integer sets fit the SDBM domain. 185 (69.5%) cases use only SDBM constructs. The remaining cases have $95\% \pm 5.1$ and $93.8\% \pm 6.4$ SDBM expressions and sets, respectively.

These test cases perform a total of 540 emptiness checks all of which can be expressed using HSDBM. In Polygeist, emptiness checks are performed during

[4] The $\mu \pm \sigma$ notation indicates the mean and standard deviation.

dependence analysis. Since the benchmarks are originally written in CUDA, they use only simple single-variable subscript expressions, leading to compatible $i - j$ expressions in dependence relations.

These results indicate that SDBM is suitable for end-to-end compilation, even if a more expressive representation may be occasionally required. Note also the higher ratio compared to the MLIR test suite.

PPCG. While none of the benchmarks can be completely processed using exclusively SDBM, most steps of the compilation process are largely compatible. Specifically, the initial representation of the program uses only SDBM for 25 (83.3%) programs, and the result of dependence analysis is representable for 26 (86.7%) programs. ILP-based affine scheduling does not match SDBM requirements for any of the programs since it extensively uses multi-variable expressions through its use of the Farkas lemma [49]. On the other hand, the resulting schedule can be expressed as a union of SDBM integer sets for 21 (70%) programs. Using the hierarchical form of the schedule [52] instead of a flat union brings this number up to 24 (80%). When applying loop tiling on a hierarchical schedule, 23 (76.7%) programs still use only SDBM with divisibility constraints associated with tile sizes. Finally, code generation is expressible only for the one program, `durbin.c`, as it produces linearized expressions of the form $C \cdot i + ii$ to recombine loop indexes after tiling (such linearization was previously avoided in the hierarchical schedule); `durbin.c` does not contain a tileable loop nest and only accesses single-dimensional arrays with subscripts of the form `i` and `i - j - C`, which are SDBM. We could also confirm our intuition that *all SDBM expressions are also HSDBM*. This is due to congruences being introduced by tiling, which uses the fixed factor of 32 by default. We verified this by disabling tiling, which brought the number of supported test cases for flat schedule and code generation to 21 (70%). Tile factors are typically chosen as powers of two or fractions of the problem sizes, so they are likely to remain divisible.

Overall, across all stages and benchmarks, $85.6\% \pm 21.6$ of affine constraints and $78.1\% \pm 37$ sets are SDBM. This number ranges from $41.5\% \pm 14.3$ constraints for the ILP set to $99.8\% \pm 0.5$ for dependency analysis, and from $10.8\% \pm 24.2$ sets for code generation to $99.6\% \pm 1.1$ for dependency analysis. These results suggest that SDBM combined with structured affine representations such as schedule trees may power a large part of a polyhedral compiler, for all stages except ILP-based affine scheduling.

6.3 Applications to Translation Validation

We additionally used our instrumented version of MLIR[5] to process three end-to-end machine learning models as described in [5]. Specifically, we took the following models (fetched on January 19, 2024).

- `text_classification_v2` obtained from `https://www.tensorflow.org/lite/examples/text_classification/overview`.

[5] `llvmorg-18-init-16246-g4daea501c4fc(Jan 5,2024)`, same for MLIR test suite.

- MobileNet v3, variation "large-075-224-classification" obtained from `https://www.kaggle.com/models/google/mobilenet-v3/frameworks/tfLite`.
- SqueezeNet: `https://www.kaggle.com/models/tensorflow/squeezenet`.

We further converted these models from the original TFLite format into the MLIR TOSA dialect using the TensorFlow tool `flatbuffer_translate -tflite-flatbuffer-to-mlir` to yield a TFLite MLIR representation, as well as `tf-opt -tfl-to-tosa-pipeline` to obtain TOSA.[6] We do not run the models but (partially) compile them along the lines of [5] using the command: [7]

```
mlir-opt --pass-pipeline='builtin.module(func.func(tosa-optional-decompositions),
  canonicalize, func.func(tosa-infer-shapes, tosa-make-broadcastable, tosa-to-linalg-named),
  canonicalize, func.func(tosa-layerwise-constant-fold, tosa-make-broadcastable),
  tosa-validate, func.func(tosa-to-linalg, tosa-to-arith, tosa-to-tensor),
  linalg-fuse-elementwise-ops, one-shot-bufferize)'
```

We collected SDBM-related statistics from all three cases in Table 1. None of the models required an emptiness check.

Table 1. SDBM is sufficient to represent most affine sets and expressions during the partial compilation pipeline from TOSA to the bufferized Linalg dialect in MLIR.

Model	Sets		Expressions	
	Total	SDBM	Total	SDBM
Text Classification	4099	4099 (100%)	9148	9148 (100%)
MobileNet	58876	52596 (89.3%)	208840	202110 (96.8%)
SqueezeNet	28131	27806 (98.8%)	96140	95490 (99.3%)

7 Related Work

The relevance of weakly relational domains for loop parallelization and optimization is well established [1]. More recently, UTVPI approximations enabling complex affine transformations (such as those enabled by PPCG in the empirical evaluation) have also been identified [49]. But these techniques remain unaware of congruence properties, missing optimization opportunities as a result [49].

[6] Both were compiled from source: `https://github.com/tensorflow/tensorflow` version `ae7eb0931d2973095`, which depends on a different version of MLIR, but the textual representation of TOSA in both is compatible.

[7] We noticed the existing flag `tosa-to-linalg-pipeline` does not produce any code, so we reconstructed the MLIR pass pipeline from its source code in `mlir/lib/Conversion/TosaToLinalg/TosaToLinalgPass.cpp`. Notable differences with the previously reported pipeline include additional TOSA normalization passes and the decomposition of the Standard MLIR dialect into the Arith and Tensor dialects, as well as the recomposition of bufferization passes into a single one.

There is a rich literature on sub-polyhedral domains [22]. APRON[8] [30] provides a reference implementation for many of these. See also ELINA[9] [47] for advanced algorithms and optimizations. Combinations of abstract domains are popular in static analysis [17,14]. These aim at increasing precision by "cross-fertilization" of analyses without the need for new abstract domains. Yet actual intersections of sub-polyhedral domains received much less attention. Bygde surveys some of these [9], the most closely related being the trapezoidal domain [36] which combines lattices with intervals, forming a non-relational domain.

Considering SDBM algorithms themselves, our iterative approach to the satisfiability problem is reminiscent of the dynamic all-pairs shortest paths [19] and incremental closure algorithms [29]. Complexity results in this space relate to the cubic upper bound of the Floyd-Warshall algorithm and do not contribute to improving the complexity of the GCD tightening iterations.

The weak NP-completeness of TVPI has been established by Hochbaum and Naor [28,26,27], together with a (pseudo-polynomial) integer linear programming algorithm that is quadratic in the largest bound of the inequalities. Our SDBM algorithm has lower complexity and also makes it pseudo-polynomial in the congruence divisors instead. In compilation problems of interest, congruences correspond to tile and vector sizes dictated by hardware parameters; they are much smaller than bounds of the iteration spaces and arrays.

8 Conclusion

We introduced the Strided Difference Bound Matrix (SDBM) abstraction combining two-variable inequalities with congruence constraints. We demonstrated the prevalence of these across the compiler test suites of MLIR, Polygeist and PPCG. We showed that the satisfiability of SDBM is NP-hard but also admits an algorithm pseudo-linear in the LCM of the congruence divisors. We identified the Harmonic SDBM (HSDBM) sub-case that commonly arises in compilation problems for deep learning and other areas. HSDBM satisfiability has a worst-case complexity of $\mathcal{O}(n^4)$, which is practical for uses in compilers and has the potential to accelerate verification tools based on more general Presburger arithmetic. We gave an $\mathcal{O}(mn^4 \log(nD_{\text{lcm}}))$ algorithm for HSDBM normalization. Finally, given a pair of normalized HSDBM, we showed linear-time algorithms to check for equality and to perform the join operation. The design of a widening operator, also necessary for abstract interpretation, is left for future work.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Allen, R., Kennedy, K.: Optimizing Compilers for Modern Architectures: A Dependence-based Approach. Morgan Kaufmann (2001)

[8] `https://antoinemine.github.io/Apron/doc/api/c`
[9] `https://elina.ethz.ch`

2. Alur, R.: Timed automata. In: Computer Aided Verification: 11th International Conference, CAV'99 Trento, Italy, July 6–10, 1999 Proceedings 11. pp. 8–22. Springer (1999)
3. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical computer science **126**(2), 183–235 (1994)
4. Bagnara, R., Hill, P.M., Ricci, E., Zaffanella, E.: Precise widening operators for convex polyhedra. Sci. Comput. Program. **58**(1–2), 28–56 (oct 2005). https://doi.org/10.1016/j.scico.2005.02.003, https://doi.org/10.1016/j.scico.2005.02.003
5. Bang, S., Nam, S., Chun, I., Jhoo, H.Y., Lee, J.: Smt-based translation validation for machine learning compiler. In: Shoham, S., Vizel, Y. (eds.) Computer Aided Verification. pp. 386–407. Springer International Publishing, Cham (2022)
6. Bengtsson, J., Larsen, K., Larsson, F., Pettersson, P., Yi, W.: Uppaal — a tool suite for automatic verification of real-time systems. In: Alur, R., Henzinger, T.A., Sontag, E.D. (eds.) Hybrid Systems III. pp. 232–243. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
7. Berthomieu, B., Menasche, M.: An enumerative approach for analyzing time petri nets. In: Mason, R.E.A. (ed.) Information Processing 83, Proceedings of the IFIP 9th World Computer Congress, Paris, France, September 19-23, 1983. pp. 41–46. North-Holland/IFIP (1983)
8. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: A static analyzer for large safety-critical software. In: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation. pp. 196–207 (2003)
9. Bygde, S.: Abstract Interpretation and Abstract Domains. Master's thesis, Mälardalen University (June 2006), http://www.es.mdu.se/publications/948-
10. Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J.W., Lee, S.H., Skadron, K.: Rodinia: A benchmark suite for heterogeneous computing. In: 2009 IEEE International Symposium on Workload Characterization (IISWC). pp. 44–54 (2009). https://doi.org/10.1109/IISWC.2009.5306797
11. Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., Cowan, M., Wang, L., Hu, Y., Ceze, L., Guestrin, C., Krishnamurthy, A.: TVM: An automated End-to-End optimizing compiler for deep learning. In: 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). pp. 578–594. USENIX Association, Carlsbad, CA (Oct 2018), https://www.usenix.org/conference/osdi18/presentation/chen
12. Child, R., Gray, S., Radford, A., Sutskever, I.: Generating long sequences with sparse transformers. CoRR **abs/1904.10509** (2019), http://arxiv.org/abs/1904.10509
13. Clément, B., Cohen, A.: End-to-end translation validation for the halide language. Proc. ACM Program. Lang. **6**(OOPSLA1) (apr 2022). https://doi.org/10.1145/3527328, https://doi.org/10.1145/3527328
14. Codish, M., Mulkers, A., Bruynooghe, M., de la Banda, M.G., Hermenegildo, M.: Improving abstract interpretations by combining domains. In: Proceedings of the 1993 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation. p. 194–205. PEPM '93, Association for Computing Machinery, New York, NY, USA (1993). https://doi.org/10.1145/154630.154650, https://doi.org/10.1145/154630.154650
15. Cook, W., Gerards, A.M.H., Schrijver, A., Tardos, É.: Sensitivity theorems in integer linear programming. Mathematical Programming **34**(3), 251–264 (Apr 1986). https://doi.org/10.1007/BF01582230, https://doi.org/10.1007/BF01582230

16. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, Third Edition. The MIT Press, 3rd edn. (2009)
17. Cousot, P., Cousot, R., Mauborgne, L.: The reduced product of abstract domains and the combination of decision procedures. In: Hofmann, M. (ed.) Foundations of Software Science and Computational Structures. pp. 456–472. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
18. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: Proceedings of the 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. p. 84–96. POPL '78, Association for Computing Machinery, New York, NY, USA (1978). https://doi.org/10.1145/512760.512770, https://doi.org/10.1145/512760.512770
19. Demetrescu, C., Italiano, G.F.: A new approach to dynamic all pairs shortest paths. In: Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing. p. 159–166. STOC '03, Association for Computing Machinery, New York, NY, USA (2003). https://doi.org/10.1145/780542.780567, https://doi.org/10.1145/780542.780567
20. Dill, D.L.: Timing assumptions and verification of finite-state concurrent systems. In: Sifakis, J. (ed.) Automatic Verification Methods for Finite State Systems. pp. 197–212. Springer Berlin Heidelberg, Berlin, Heidelberg (1990)
21. Eldridge, S., Barua, P., Chapyzhenka, A., Izraelevitz, A., Koenig, J., Lattner, C., Lenharth, A., Leontiev, G., Schuiki, F., Sunder, R., et al.: Mlir as hardware compiler infrastructure. In: Workshop on Open-Source EDA Technology (WOSET) (2021)
22. Gange, G., Ma, Z., Navas, J.A., Schachte, P., Søndergaard, H., Stuckey, P.J.: A fresh look at zones and octagons. ACM Trans. Program. Lang. Syst. **43**(3) (sep 2021). https://doi.org/10.1145/3457885, https://doi.org/10.1145/3457885
23. Granger, P.: Static analysis of arithmetical congruences. International Journal of Computer Mathematics **30**(3-4), 165–190 (1989). https://doi.org/10.1080/00207168908803778, https://doi.org/10.1080/00207168908803778
24. Granger, P.: Static analysis of linear congruence equalities among variables of a program. In: Proceedings of the International Joint Conference on Theory and Practice of Software Development on Colloquium on Trees in Algebra and Programming (CAAP '91): Vol 1. p. 169–192. TAPSOFT '91, Springer-Verlag, Berlin, Heidelberg (1991)
25. Henzinger, T.: The theory of hybrid automata. In: Proceedings 11th Annual IEEE Symposium on Logic in Computer Science. pp. 278–292 (1996). https://doi.org/10.1109/LICS.1996.561342
26. Hochbaum, D.S.: Monotonizing linear programs with up to two nonzeroes per column. Oper. Res. Lett. **32**(1), 49–58 (jan 2004). https://doi.org/10.1016/S0167-6377(03)00074-9, https://doi.org/10.1016/S0167-6377(03)00074-9
27. Hochbaum, D.S.: Applications and efficient algorithms for integer programming problems on monotone constraints. Networks **77**(1), 21–49 (2021). https://doi.org/10.1002/NET.21983, https://doi.org/10.1002/net.21983
28. Hochbaum, D.S., Naor, J.S.: Simple and fast algorithms for linear and integer programs with two variables per inequality. SIAM Journal on Computing **23**(6), 1179–1192 (1994). https://doi.org/10.1137/S0097539793251876, https://doi.org/10.1137/S0097539793251876
29. Howe, J.M., King, A., Simon, A.: Incremental closure for systems of two variables per inequality. Theoretical Computer Science **768**, 1–42 (2019). https://doi.org/https://doi.org/10.1016/j.tcs.2018.12.001, https://www.sciencedirect.com/science/article/pii/S0304397518307199

30. Jeannet, B., Miné, A.: Apron: A library of numerical abstract domains for static analysis. In: Bouajjani, A., Maler, O. (eds.) Computer Aided Verification. pp. 661–667. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
31. Katel, N., Khandelwal, V., Bondhugula, U.: Mlir-based code generation for gpu tensor cores. In: Proceedings of the 31st ACM SIGPLAN International Conference on Compiler Construction. p. 117–128. CC 2022, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3497776.3517770, https://doi.org/10.1145/3497776.3517770
32. Lagarias, J.C.: The computational complexity of simultaneous diophantine approximation problems. SIAM Journal on Computing **14**(1), 196–209 (1985). https://doi.org/10.1137/0214016, https://doi.org/10.1137/0214016
33. Lattner, C., Amini, M., Bondhugula, U., Cohen, A., Davis, A., Pienaar, J., Riddle, R., Shpeisman, T., Vasilache, N., Zinenko, O.: Mlir: Scaling compiler infrastructure for domain specific computation. In: 2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). pp. 2–14 (2021). https://doi.org/10.1109/CGO51591.2021.9370308
34. Liu, H.I.C., Brehler, M., Ravishankar, M., Vasilache, N., Vanik, B., Laurenzo, S.: Tinyiree: An ml execution environment for embedded systems from compilation to deployment. IEEE Micro **42**(5), 9–16 (2022). https://doi.org/10.1109/MM.2022.3178068
35. Martens, J., Grosse, R.: Optimizing neural networks with kronecker-factored approximate curvature. In: Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37. p. 2408–2417. ICML'15, JMLR.org (2015)
36. Masdupuy, F.: Array abstractions using semantic analysis of trapezoid congruences. In: Kennedy, K., Polychronopoulos, C.D. (eds.) Proceedings of the 6th international conference on Supercomputing, ICS 1992, Washington, DC, USA, July 19-24, 1992. pp. 226–235. ACM (1992). https://doi.org/10.1145/143369.143414, https://doi.org/10.1145/143369.143414
37. Miné, A.: A new numerical abstract domain based on difference-bound matrices. In: Danvy, O., Filinski, A. (eds.) Programs as Data Objects. pp. 155–172. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
38. Miné, A.: The octagon abstract domain. In: Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE'01). p. 310. WCRE '01, IEEE Computer Society, USA (2001)
39. Miné, A.: The octagon abstract domain. CoRR **abs/cs/0703084** (2007), http://arxiv.org/abs/cs/0703084
40. Moses, W.S., Chelini, L., Zhao, R., Zinenko, O.: Polygeist: Raising c to polyhedral mlir. In: 2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT). pp. 45–59 (2021). https://doi.org/10.1109/PACT52795.2021.00011
41. Moses, W.S., Ivanov, I.R., Domke, J., Endo, T., Doerfert, J., Zinenko, O.: High-performance gpu-to-cpu transpilation and optimization via high-level parallel constructs. In: PPoPP '23. p. 119–134. Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3572848.3577475, https://doi.org/10.1145/3572848.3577475
42. Ore, O.: The general chinese remainder theorem. The American Mathematical Monthly **59**(6), 365–370 (1952). https://doi.org/10.1080/00029890.1952.11988142, https://doi.org/10.1080/00029890.1952.11988142

43. Pitchanathan, A., Cohen, A., Zinenko, O., Grosser, T.: Strided difference bound matrices. CoRR **abs/2405.11244** (2024). https://doi.org/10.48550/ARXIV.2405.11244, https://doi.org/10.48550/arXiv.2405.11244
44. Pitchanathan, A., Ulmann, C., Weber, M., Hoefler, T., Grosser, T.: FPL: Fast presburger arithmetic through transprecision. Proc. ACM Program. Lang. **5**(OOPSLA) (oct 2021). https://doi.org/10.1145/3485539, https://doi.org/10.1145/3485539
45. Pouchet, L.N., Yuki, T.: Polybench/c 4.2.1, online: https://sourceforge.net/projects/polybench/
46. Reinking, A., Bernstein, G.L., Ragan-Kelley, J.: Formal semantics for the halide language. CoRR **abs/2210.15740** (2022). https://doi.org/10.48550/ARXIV.2210.15740, https://doi.org/10.48550/arXiv.2210.15740
47. Singh, G., Püschel, M., Vechev, M.: Fast polyhedra abstract domain. In: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages. p. 46–59. POPL '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3009837.3009885, https://doi.org/10.1145/3009837.3009885
48. Tillet, P., Kung, H.T., Cox, D.: Triton: an intermediate language and compiler for tiled neural network computations. In: Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages. p. 10–19. MAPL 2019, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3315508.3329973, https://doi.org/10.1145/3315508.3329973
49. Upadrasta, R., Cohen, A.: Sub-polyhedral scheduling using (unit-)two-variable-per-inequality polyhedra. In: Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. p. 483–496. POPL '13, Association for Computing Machinery, New York, NY, USA (2013). https://doi.org/10.1145/2429069.2429127, https://doi.org/10.1145/2429069.2429127
50. Verdoolaege, S.: isl: An integer set library for the polyhedral model. In: Fukuda, K., van der Hoeven, J., Joswig, M., Takayama, N. (eds.) Mathematical Software – ICMS 2010. pp. 299–302. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
51. Verdoolaege, S., Carlos Juega, J., Cohen, A., Ignacio Gómez, J., Tenllado, C., Catthoor, F.: Polyhedral parallel code generation for cuda. ACM Trans. Archit. Code Optim. **9**(4) (jan 2013). https://doi.org/10.1145/2400682.2400713, https://doi.org/10.1145/2400682.2400713
52. Verdoolaege, S., Guelton, S., Grosser, T., Cohen, A.: Schedule trees. In: International Workshop on Polyhedral Compilation Techniques, Date: 2014/01/20-2014/01/20, Location: Vienna, Austria (2014)
53. XLA: Accelerated linear algebra, online: https://www.tensorflow.org/xla and https://github.com/openxla/xla
54. Zhao, R., Cheng, J.: Phism: Polyhedral high-level synthesis in MLIR. CoRR **abs/2103.15103** (2021), https://arxiv.org/abs/2103.15103