BullsEye: Scalable and Accurate Approximation Framework for Cache Miss Calculation

Nilesh Rajendra Shah, Ashitabh Misra, Antoine Miné, Rakesh Venkat, Ramakrishna Upadrasta

BullsEye: Scalable and Accurate Approximation Framework for Cache Miss Calculation

NILESH RAJENDRA SHAH, ASHITABH MISRA, ANTOINE MINÉ, RAKESH VENKAT, and RAMAKRISHNA UPADRASTA, Department of CSE, IIT Hyderabad, India and Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

For Affine Control Programs or Static Control Programs (SCoP), symbolic counting of reuse distances could induce polynomials for each reuse pair. These polynomials along with cache capacity constraints lead to non-affine (semi-algebraic) sets; and counting these sets is considered to be a hard problem. The state-of-the-art methods use various exact enumeration techniques relying on existing cardinality algorithms that can efficiently count affine sets.

We propose BullsEye, a novel, scalable, accurate, and problem-size independent approximation framework. It is an analytical cache model for fully associative caches with LRU replacement policy focusing on sampling and linearization of non-affine stack distance polynomials. Firstly, we propose a simple domain sampling method that can improve the scalability of exact enumeration. Secondly, we propose linearization techniques relying on *Handelman's theorem*, and *Bernstein's representation*. To improve the scalability of the *Handelman's theorem* linearization technique, we propose template (Interval or Octagon) sub-polyhedral approximations.

Our methods obtain significant compile-time improvements with high-accuracy when compared to HayStack on important polyhedral compilation kernels such as `nussinov`, `cholesky`, and `adi` from PolyBench, and `harris`, `gaussianblur` from LLVM-TestSuite. Overall, on PolyBench kernels, our methods show upto 3.31× (geomean) speedup with errors below ≈ 0.08% (geomean) for the octagon sub-polyhedral approximation.

CCS Concepts: • **Software and its engineering → Compilers**; **Software performance**.

Additional Key Words and Phrases: Static analysis, cache model, performance analysis

1 INTRODUCTION AND MOTIVATION

An important program analysis that reduces naturally to a counting problem is the Cache Miss Calculation (CMC) problem; namely, to estimate the number of cache misses in a given program loop. CMC estimation has large implications for program optimization. An efficient algorithm that can estimate the cache misses could be effectively used to find out which of the legal transformations of a given program loop lead to the most efficient code. To model the cache behaviour, some previous works use simulation [9], or instrumentation [20, 25, 68] of the input program on the hardware. These approaches are expensive as they are resource intensive. Therefore, analytical methods of cache modeling were proposed [10, 32], targeting affine programs.

Recently, there were major strides in this area: two *exact* and *sound* analytical cache modeling algorithms have been proposed: *PolyCache* by Bao et al. [2], and *HayStack* by Gysi et al. [35]. The above works rely on symbolic counting of parametric integer sets and maps. In particular, they rely on the Barvinok algorithm [3] to perform symbolic counting (cardinality).

For fully associative caches, the total cache misses include the compulsory/cold misses, in addition to the capacity misses. However, cold misses can be easily computed by counting the first accesses of each reference pair by a single call to an Integer Linear Programming (ILP) solver, relying on a *lexmin* computation. On the other hand, computing capacity cache misses involves a precise modeling of the subsequent accesses to the particular cache line using reuse distance, and counting integer sets/maps that affect the performance and scalability of the model.

Authors' address: Nilesh Rajendra Shah, cs19mtech11021@iith.ac.in; Ashitabh Misra, misra8@illinois.edu; Antoine Miné, antoine.mine@lip6.fr; Rakesh Venkat, rakesh@cse.iith.ac.in; Ramakrishna Upadrasta, ramakrishna@cse.iith.ac.in, Department of CSE, IIT Hyderabad, India, Sorbonne Université, CNRS, LIP6, F-75005 Paris, France .

To count the capacity misses for fully associative LRU caches, HayStack [35] calculates the *stack distance* [46] for each memory access. The stack distance (also referred to as reuse distance [6]) is defined to be the cardinality of the set of unique memory accesses between successive references to the same memory location. Beyls et al. [7] proposed techniques to analytically count these stack distances to obtain stack distance polynomials. In general, these stack distance polynomials are *non-affine* and are *Ehrhart polynomials* [21–23].

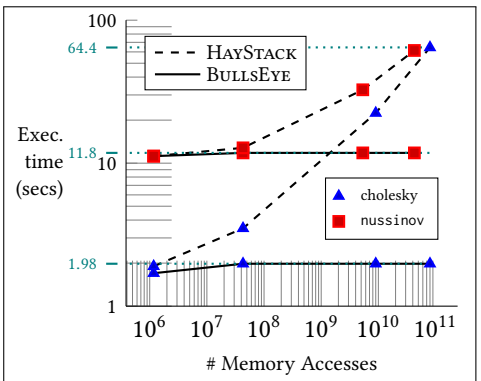


Fig. 1. Execution time of HayStack vs. BullsEye. (Handelman theorem Linearization using Octagon template.)

The classic Barvinok algorithm [3] can compute the cardinality of parametric affine sets by using Presburger arithmetic [36, 51], returning the symbolic count as Ehrhart polynomials. It is implemented in various libraries [1, 62], but faces the limitation that it can count only (parametric) *affine sets*. To overcome the above limitation, HayStack counts the non-affine stack distance polynomials using partial (or full) enumeration. This leads to the Barvinok library being called for each point in the non-affine dimensions. For some polyhedral kernels like `3mm` and `fdtd-2d` (from PolyBench [50]), where the non-affine polynomials are few and the domains are small, the above method works very well. However, for kernels like `nussinov` and `cholesky`, this technique is computationally expensive.

In Fig. 1, we show the execution time comparison of HayStack vs. our new approximate cache miss framework BullsEye. We obtain maximum speedups of 5.2× and 32.5×, and maximum errors of 0.82% and 0.55% for `nussinov` and `cholesky` respectively. It can be seen that our approximation provides good scalability and accuracy; it is also input problem-size independent.

Primary motivation: Here are some of our key motivations and insights:

(1) In their full generality, CMC formulations lead to non-affine (semi-algebraic) integer sets that need to be counted. The state-of-the-art cache models like HayStack use (partial or full) enumeration techniques on polynomials to iteratively count the exact cache misses.
(2) Counting integer polyhedra, involving Presburger arithmetic [30] is a theoretically unscalable problem as it involves worst-case exponential-time complexity algorithms.
(3) In general, exact CMC estimates are not *really* needed. Approximations should be sufficient, provided they are *empirically good*, and the CMC algorithm is *scalable*.

High-level summary of our approach. The following steps summarize our approach: (i) Obtain the stack distance (affine or non-affine) polynomials and their domains from an existing model (like HayStack). (ii) Apply heuristics to approximate the above (non-affine) polynomials. (iii) Count the number of integer points in the resultant polyhedron, using either exact or approximate means.

Contributions. In this paper, we make the following contributions:

- A *heuristic* based on sampling the non-affine dimensions in the domain, to approximate Cache Miss Count by reducing the number of explicit calls to Barvinok algorithm. (Sec. 4.2)
- An approximation framework based on *Handelman's theorem* [37] that linearizes a non-affine CMC polynomial constraint over a convex polytope. (Sec. 5.1)
- A new template (interval and octagon) sub-polyhedral approximation leading to a scalable and problem-size independent formulation, that does not rely on parametric simplex, and makes a *single call* to Barvinok. (Sec. 5.3)
- A linearization method based on *Bernstein* expansion over polytopes. (Sec. 6)

- An detailed experimental results of our methods on PolyBench as well as additional benchmarks. In particular, we show a comparison against HayStack [35], and Dinero [40] simulator, on which we obtain significant speedups along with high accuracy. (Sec. 7)

Organization of this paper. In Sec. 2, we introduce some basic mathematical background. In Sec. 3, we present an overview of HayStack algorithm and infrastructure. In Sec. 4, we present an overview of our framework for calculating approximate cache misses, and present a sampling-based heuristic. In Sec. 5, we present a new approximation framework that applies Handelman's theorem based linearizations using interval and octagon sub-polyhedra. In Sec. 6, we propose a Bernstein expansion based approximation. In Sec. 7, we show experimental results of our various methods. In Sec. 8, we discuss related works. In Sec. 9, we present our conclusions and directions for future work.

2 MATHEMATICAL BACKGROUND

In this section, we introduce the mathematical background for this paper. We broadly give the lemmas and the mathematical explanation.

2.1 Integer sets, Integer maps, Cardinality and Barvinok

To obtain the cache miss count, we use integer sets and maps. An integer set $\mathbb{Z}^d$ is a subset of real numbers $\mathbb{R}^d$. They define a set of d-dimensional integer tuples that satisfy a set of affine constraints. A integer set $\mathcal{S}$ in 2-dimensional integer space (i, j) (in isl-notation)

$$\mathcal{S} = [\eta] \rightarrow \{[i, j] : 0 \leq i \leq \eta, 0 \leq j \leq \eta\}, (i, j) \in \mathbb{Z}^2$$

is a set of integer tuples which satisfy affine constraints, with η as a parameter. These constraints are essentially Presburger formulas [36, 51] consisting of various operators and existential quantifiers. Integer sets support various operations like intersection, union, difference, projection, and cardinality.

Relations between pairs of integer tuples satisfying affine constraints are defined as integer maps:

$$\mathcal{M} = [\eta] \rightarrow \{[i, j] \rightarrow [i] : 0 \leq i < \eta, 0 \leq j < \eta\},$$

In addition to various set operations, these maps also support inversion, composition and domain intersection. They are provided by the ISL library [63] and can be used to define access relations. For an integer set $\mathcal{S}$, its cardinality is denoted as $\texttt{card}(\mathcal{S})$ (computed through call to the Barvinok library) and represents the number of integer points in $\mathcal{S}$. We also use a newly defined function $\texttt{affine}(g(x_1, \ldots, x_{m'}))$, which returns the affine terms of the multivariate polynomial $g(x_1, \ldots, x_{m'})$.

2.2 Bernstein representation of polynomials

We discuss Bernstein polynomials [4, 5, 26] that form a basis for the space of polynomials. This representation allows any type of polynomial to be expressed using the Bernstein coefficients. In addition, Bernstein expansion provides a way to bound polynomials over an interval or a convex set [14, 17]. For the range [0,1], a univariate Bernstein basis of degree m can be written as:

$$b_k^m(x) \triangleq \binom{m}{k}(1-x)^{m-k}x^k, \qquad k = 0, \ldots, m, \qquad \binom{m}{k} = \frac{(m)!}{(m-k)!(k)!}$$

We can express a given polynomial $g(x)$ of degree at most m as a linear combination of degree-m Bernstein base polynomials ($b_k^m(x)$) as shown in Eqn. 1. Meaning, for polynomials of degree at most m restricted to $x \in [0, 1]$, the Bernstein base polynomials of degree-m form a basis.

$$g(x) \triangleq \sum_{k=0}^{m} t_k b_k^m(x), \qquad t_k \in \mathbb{R} \tag{1}$$

Theorem 2.1. *Let $g(x)$ be a polynomial of degree m with real-valued coefficients, then*

$$\min(t_k : k = 0, \ldots, m) \leq g(x) \leq \max(t_k : k = 0, \ldots, m), \forall x \in [0, 1]$$

The lower (upper) bound is exact, if and only if it is equal to t_0 (t_m).

Let us see an example of a polynomial in its Bernstein form and apply the above Theorem 2.1.

Example 2.2 (Bernstein expansion). Let $g(x) = 4x^2 + 3x + 5 = 5b_0^2(x) + (\frac{13}{2})b_1^2(x) + 12b_2^2(x)$, where $b_0^2(x) = (1-x)^2, b_1^2(x) = 2x(1-x)$ and $b_2^2(x) = x^2$. On the interval [0,1], $g(x)$ is bounded by minimum and maximum Bernstein coefficients $t_0 = 5$ and $t_2 = 12$. Both bounds are exact.

Bernstein representation over a convex polytope. The Bernstein representation of a polynomial [14, 15] can be defined over a convex polytope $P \subset \mathbb{Q}^{m'}$ represented [57, 72] by a convex hull of its generators. To compute the bounds on a multivariate polynomial $g(x_1, \ldots, x_{m'})$ over P, we write x $(=[x_1, \ldots, x_{m'}])$ as the convex combination of vertices and substitute in $g(x_1, \ldots, x_{m'})$. Next, each term is made homogeneous, and the relevant generalized Bernstein coefficients [14] (of $b_k^m(x)$) are computed.

$$\min_{k \in S} t_k \leq g(x_1, \ldots, x_{m'}) \leq \max_{k \in S} t_k \quad \forall x_i \in [0,1], \{i = 1, \ldots, m'\}$$
$$\text{where: } S = \{(k_1, k_2, \ldots, k_{m'}) \in \mathbb{R}^{m'} \ : \ k_i \geq 0,\ k_1 + \ldots + k_{m'} = m\}$$

The ISL library [63] provides an interface to obtain lower/upper bounds [14] over a polynomial using Bernstein representation.

2.3 Positive polynomials over a Polytope

We apply Handelman's theorem [37] to obtain the linearization of a polynomial. It states that a polynomial g is strictly positive in a bounded polyhedron D iff it can be represented as a positive linear combination of monomials in D for a positive finite bound parameter K on the degree of the monomials. In practice, K is taken as a user defined parameter for computational reasons.

Theorem 2.3 (Handelman's theorem [37]). *Let D be a polytope in $\mathbb{R}^d$ defined by a system of n affine inequalities:*

$$D = \{x \mid p_i(x) \geq 0; i = 1, \ldots, n\}$$

and g, a polynomial in d variables that is strictly positive in D iff g can be expressed as a positive linear combination of products of monomials $p_i(x)$ in D as:

$$g(x) \equiv \sum_{I \in \mathcal{N}^n} \lambda_I p_1^{k_1}(x) \ldots p_n^{k_n}(x) \tag{2}$$

where $I = (k_1, k_2, \ldots, k_n)$, each λ_I is non-negative, and at least one λ_I is non-zero. Also, the sum of degrees of monomials is bounded by K, with $k_1 + k_2 + \cdots + k_n \leq K$.

Here, a Handelman product is an element of a set of products $\mathbf{P}$: $p_1^{k_1}(x) \ldots p_n^{k_n}(x)$ for a given set of indices $I = (k_1, \ldots, k_n)$. Handelman's theorem is applicable to bounded rational non-parametric domains and can be used for constructing a *positivity certificate*. Handelman's theorem can be seen as a strict extension of the widely used Farkas Lemma [57, 72] that has been used in seminal polyhedral works like that by Feautrier [27].

Theorem 2.4 (Schweighofer's theorem [58]). *Let $\mathcal{T}$ be a semi-algebraic set in $\mathbb{R}^d$ with some polynomial constraints of degree greater than one and the set of affine inequalities in $\mathcal{T}$ defines a bounded polyhedron (polytope):*

$$\mathcal{T} = \{x \mid p_i(x) \geq 0; i = 1, n\}$$

Then, a polynomial g is strictly positive on $\mathcal{T}$ iff g can be expressed as shown in Eqn. 2.

Schweighofer [58] is an extension of Handelman's Theorem 2.3 for semi-algebraic sets $\mathcal{T}$ that contain polynomial constraints of degree ≥ 2 and the affine constraints of $\mathcal{T}$ define a polytope.

2.4 Sub-Polyhedra: Intervals and Octagons

Intervals. A d-dimensional interval polyhedron has only one variable per inequality. Its constraints are defined as follows: $\pm x_i \leq \alpha_i$.

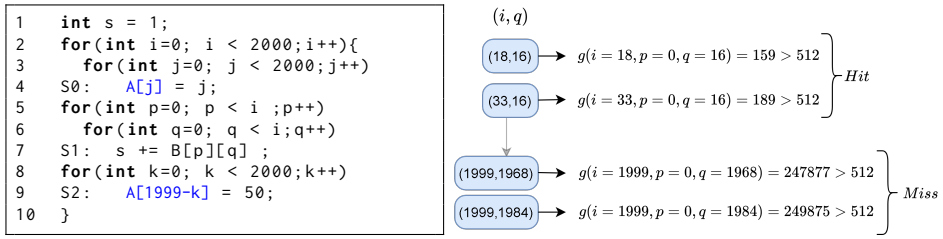


Fig. 2. (a) Example code, and (b) Illustration of full enumeration of HayStack (L1 cache, $c = 512$).

Table 1. Number of Affine (Af) and Non-affine (Naf) (with maximum degree=2) polynomials from PolyBench.

	3mm	adi	cholesky	correlation	covariance	deriche	durbin	fdtd-2d	gemver	lu	ludcmp	mvt	nussinov	trisolv	2mm	atax	bicg	doitgen	floyd-warshall	gemm	gesummv	gramschmidt	heat-3d	jacobi-1d	jacobi-2d	seidel-2d	symm	syr2k	syrk	trmm
Af	32	255	58	111	83	74	102	72	53	149	171	25	143	25	45	26	21	40	150	22	18	41	180	24	72	49	120	85	38	34
Naf	1	58	37	4	3	6	4	2	4	62	64	2	96	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Octagons (UTVPI). A d-dimensional Octagon, or Unit-Two-Variable-per-Inequality (UTVPI) polyhedron has at most two variables per inequality, with only ±1 as their coefficients. They can be defined as:

$$\pm x_i \pm x_j \leq \alpha_{ij},\ \forall\, (x_i, x_j) \in \mathrm{S}$$

where $\mathrm{S} = \{x_1, \ldots, x_d\}$ is the set of dimensions of the UTVPI polyhedron.

3 OVERVIEW OF HAYSTACK EXACT ENUMERATION ALGORITHM & INFRASTRUCTURE

Gysi et al. [35] proposed the HayStack tool that statically models fully associative caches with LRU replacement policy using exact enumeration. Here, we talk about its algorithm and infrastructure. For each pair of references in the input program, HayStack constructs a distance map $\mathcal{M}_d$ that relates every statement instance to the set of array references accessed after the last access of the same array element. Counting $\mathcal{M}_d$ results in a pair[1] $\langle g, D \rangle$, where g is a parametric stack distance polynomial, and D is called a *validity domain*, which is a subset of the input iteration domain. (If $\mathcal{D}_I$ is the input iteration domain, $D \subseteq \mathcal{D}_I$.)

HayStack *algorithm:* We summarize the HayStack algorithm based on the pair $\langle g, D \rangle$. We assume the cache has size C bytes (could be L1, or L2, or L3 cache), with the number of cache-lines c:

- If g is **affine**, then compute the cache miss set M, defined by $\{g > c\} \wedge D$. This set contains all the memory accesses with stack distance larger than c. Find the cardinality of M using Barvinok algorithm to count the capacity cache misses. (`card`(M))
- Else, if g is **non-affine**, then pre-process it using different simplification techniques to remove floor terms. Enumerate g over non-affine terms using:
  - *Partial Enumeration*: If g contains at least one affine dimension, then find the enumeration domain E of non-affine dimensions. To count the non-affine miss set M, for each integer point in E, get the affine miss set M by instantiating and perform symbolic counting.
  - *Full Enumeration*: If all dimensions in g are non-affine, for each point in D, evaluate the point in g and explicitly check the constraint $g > c$ for a capacity miss.

As noted earlier, the enumeration of non-affine cache miss sets is expensive. In Fig. 2, on an example code, we show the scalability issues of HayStack resulting from a full enumeration of M.

[1]In the HayStack paper, this pair is referred to as a *piece*.

*Example 3.1 (*HayStack *method of explicit (full) enumeration.).* Let $g(i,p,q)$ be a non-affine stack distance polynomial with domain D:

$$g(i,p,q) = \frac{1}{16}iq - \frac{1}{16}q + i + 124\,, D = \left\{ (i,p,q) \in \mathbb{Z}^2 \;\middle|\; \begin{array}{l} (q)mod(16) = 0\,, p = 0\,, i \leq 1999, \\ q \geq -17 + i\,, q \geq 16\,, q \leq -2 + i \end{array} \right\}$$

The result of the full enumeration to $g(i,p,q)\ \forall i,p,q \in D$ is shown in Fig. 2(b).

Here, in g, we have a non-affine term $i * q$. All the dimensions i.e., i,and q, are non-affine in the stack distance polynomial. As shown in Fig. 2, HayStack performs full enumeration. Testing for each point in the domain for polynomial inequality is expensive for larger iteration domains.

*Example 3.2 (*HayStack *method of converting non-affine to affine sets by partial enumeration.).* Let $g(i,j) = ai^2 + bj + 10$ be a non-affine stack distance polynomial and $E = \{1 \leq i \leq 1000\}$. Applying the partial enumeration to $g(i,j)\ \forall i \in E$ gives the following sets.

$$g(i=1,j) = a + bj + 10 \qquad g(i=2,j) = 4a + bj + 10 \qquad g(i=3,j) = \ldots$$

It is easy to see that each of the resulting polynomials are specializations for each value of i, and are indeed affine sets and amenable to be handled by Barvinok. HayStack exploits this insight and uses the partial and full enumeration techniques, along with other strategies, for effective enumeration to compute an exact cache miss count.

HayStack *infrastructure:* HayStack could also be seen as an infrastructure for computing cache misses using stack distance of a reuse pair for affine programs. It is implemented using the following tools: the Polyhedral extraction tool (PET) [64] to extract the polyhedral representation of the input program, the ISL library [63] to represent and manipulate integer sets and maps, and the *Barvinok* library [62] to count integer sets and maps.

4 OVERVIEW OF BullsEye FRAMEWORK AND APPROXIMATION USING SAMPLING

In this section, we discuss the polynomials obtained, give a brief overview of BullsEye and propose a simple method using statistical sampling to show the effectiveness of approximations.

Polynomial Analysis. Stack distance polynomials are parametric in the input dimensions. Some memory access patterns induce non-affine terms (like $i * j$, i^2) in the polynomial. We begin with showing an analysis of the stack distance polynomials in various kernels of PolyBench. It should be noted that 16/30 of the PolyBench kernels have stack distance polynomials that are already affine, and can be directly counted by using Barvinok without the need for any preprocessing. This includes kernels like `2mm`, `heat-3d`, `gemm`, `jacobi-1d`, etc. On the other hand, non-affine polynomials are present in important kernels like `3mm`, `nussinov`, `cholesky`, `adi`, etc. In PolyBench, 14/30 kernels induce non-affine stack distance polynomials. We term them as PolyBench-Non-Affine. The number of affine and non-affine polynomials in PolyBench is tabulated in Table 1.

4.1 Overview of our proposed system

In Fig. 3, we show the flow-diagram of HayStack, and our approximation framework BullsEye. The input to BullsEye is a SCoP and cache parameters. BullsEye uses, and builds from, the implementation of Beyls formulation available in the HayStack infrastructure. As discussed earlier, for each reuse pair, it computes a stack distance polynomial with a validity domain. Thereafter, standard methods suggested in HayStack are applied to obtain a simplified miss set M.

We propose various approximations of Miss set M in Sec. 4.2, Sec. 5, and Sec. 6. In Sec. 4.2, we propose a simple statistical approximation. In Sec. 5, our mathematical theory is based on Handelman's theorem [37]. We extend the earlier characterization of positive polynomials over a polytope by Feautrier [28], and polynomial linearizations by Maréchal et al. [45]. In Sec. 5.3, we propose to approximate M using sub-polyhedral (interval and octagon) template polyhedra [54–56] to provide a highly scalable linearization. In Sec. 6, our mathematical theory is based on Bernstein polynomials [4, 5, 26], and Bernstein expansion over convex polytope by Clauss et al. [14, 15].

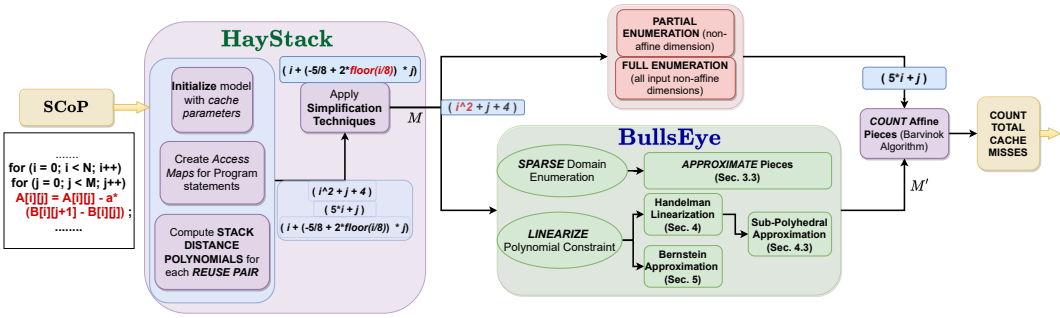


Fig. 3. Flow diagram of HayStack and (our proposed system) BullsEye.

4.2 Sparse Domain Enumeration (SparseEnum)

For arrays, it is known that the memory accesses are sequential. In most SCoPs, sequential array elements are stored in nearby cache blocks. This means that the reuse distance for a particular reuse pair—as a measure of spatial locality [67]—is a function of its block size [71]. We observe that reuse distances for nearby elements need *not all* be fully enumerated. Meaning that the cache miss induced by one particular access can be used as an approximation for its nearby accesses as well.

We use the above intuition to build a *sparse enumeration heuristic*. We show the working of our heuristic and its improvement over partial enumeration of HayStack using the following example.

Example 4.1 (SparseEnum). The example program in Fig. 2.(a) generates the following polynomial $g(i, j, k)$ with enumeration domain E:

$$g(i, j, k) = (i^2 + k + 1); \quad E = \{0 \leq i \leq 1999\}$$

In $g(i, j, k)$, the dimension i is non-affine; the term i^2 corresponds to the unique memory accesses from statement `S1` on line 7 between the reference pairs `(A[j], A[1999-k])`. The polynomial $g(i, j, k)$ is counted for each instance of reuse pair i.e., `(A[0], A[0])`, `(A[1], A[1])` and so on. These generate *similar* reuse distance values for the count. For example, reuse distance for reuse pairs `(A[0], A[0])`, `(A[1], A[1])` is almost same i.e., $g(i, j = 0, k = 1999) = g(i, j = 1, k = 1998) + 1$. This means that we can approximate the stack distance of an instance of reuse pair with its previous accesses defined by the same reference pair. This leads to a reduction in the number of calls to the exponential-complexity Barvinok algorithm.

Our SparseEnum can also be mathematically seen as an approximation of the piecewise polynomial sections by a step-function of a constant step size called *span*. For a reuse pair, we use uniform sampling on E to obtain the reuse distance of the sampled iterations, followed by counting the cache misses at *only these* sampled iterations. The same cache miss count is assigned to all the iteration points within the span. Thus, we reduce the number of points evaluated on g, and the number of Barvinok calls by a factor of *span* (to $\approx \texttt{card}(E)/span$). The first sampled iteration is selected by testing if the obtained affine domain is large enough to be counted by Barvinok.

For this example, setting $span = 100$ leads to counting the following pieces: $g(i = 0, k), g(i = 100, k), \ldots, g(i = 1900, k)$, followed by extrapolation of the cache miss count within the *span*. For example, the count obtained for $g(i = 0, k)$ is assigned to all the $1 \leq i \leq 99$ pieces within the *span*.

Example 4.2 (SparseEnum). Let $g_1(i, j) = (\frac{i^2}{16} + \frac{i}{2} + \frac{j}{8} + \frac{7}{4})$ be a non-affine stack distance polynomial with a domain D_1 as shown in Fig. 4:

$$D_1 = \left\{ (i, j) \in \mathbb{Z}^2 \;\middle|\; \begin{array}{l} (i-2)mod(8) = 0\,, (j)mod(8) = 0 \\ 18 \leq i \leq 3994\,, 8 \leq j \leq i - 10 \end{array} \right\} \xrightarrow{\Pi_j} E_1 = \left\{ \begin{array}{l} (i-2)\ mod\ 8 = 0, \\ 18 \leq i \leq 3994 \end{array} \right\}$$

Applying the SparseEnum to $g_1(i, j)$; $\forall i \in E$, gives the sets as shown in Fig. 4.

Here partial enumeration would have created $(3994 - 18)/8 + 1 = 498$ affine pieces for each point in E_1 of variable i, and each such affine piece will lead to a call to `card`. If we empirically set

```
1 for (i = N-1; i >= 0; i--) {
2  for (j=i+1; j<N; j++) {
3   if (j-1>=0)
4     T[i][j] = max_score(T[i][j], T[i][j-1]);
5   if (i+1<N)
6     T[i][j] = max_score(T[i][j], T[i+1][j]);
7   if (j-1>=0 && i+1<N) {
8    if (i<j-1)
9     T[i][j]=max_score(T[i][j],T[i+1][j-1]
10              + match(seq[i],seq[j]));
11   else
12      T[i][j] = max_score(T[i][j], T[i+1][j-1]);
13  }
14 for (k=i+1; k<j; k++)
15   T[i][j]=max_score(T[i][j], T[i][k]+T[k+1][j]);
16  }
17 }
```

(a)

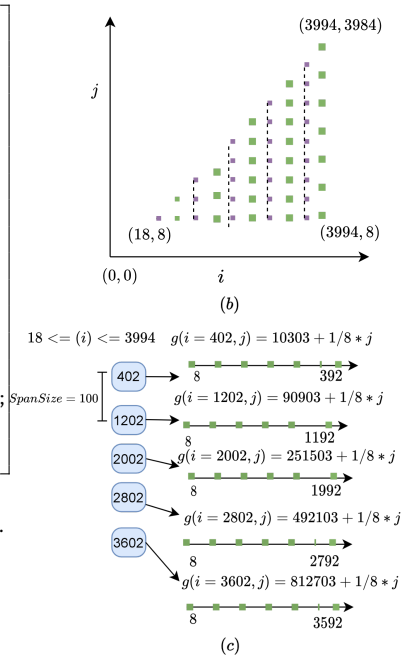


Fig. 4. In (a), we show a selection of nussinov from PolyBench. The highlighted reuse pairs in blue (red) are the ones which result in Ex. 4.2 (Ex. 6.1). In (b) and (c), we show SparseEnum. In (b) is Domain D_1 (of Ex. 4.2) with sparse enumeration in i dimension. In (c), are the pieces counted with $span = 100$.

$span = 20$, it reduces the number of calls to `card` to 25 ($\approx \lceil \frac{498}{20} \rceil$). This is an appreciable improvement from the partial enumeration (with 498 `card` calls). There is no significant loss of accuracy from this approximation; it yields $\approx \pm 2\%$ error, when compared to HaySTack exact enumeration.

5 LINEARIZING USING HANDELMAN'S THEOREM

In this section, we propose various techniques to linearize the non-affine terms in the stack distance polynomial g using *Handelman's theorem* [37], so that we obtain an *approximate integer polyhedron* M', which can be counted efficiently by Barvinok algorithm.

In Sec. 5.1, we propose our linearization framework and algorithm, and explain its working. We show how Handelman's theorem can be applied to CMC. This is an extension of the formulations proposed by Maréchal et al. [45] for linearizations, and Feautrier [28] for finding positive template polynomials over a polytope, though specialized for CMC. In Sec. 5.2, we illustrate the working of the above framework using an example (Ex. 5.1), with the Miss set extracted from the `nussinov` kernel from PolyBench [50]. In Sec. 5.3, we propose to over-approximate the Miss set using interval and octagon sub-polyhedra. Fixing approximate templates [54–56] of the over-approximation a priori to be intervals [18] or octagons [47, 48] has multiple advantages: (i) it leads to a smaller problem size for the parametric LP formulation, (ii) the generated system has almost no redundant constraints, and (iii) the cardinality on the approximate sub-polyhedron can be computed much faster. In Sec. 5.4, we illustrate the working of the approximations proposed in Sec. 5.3 on a Miss set extracted from the `correlation` kernel from PolyBench [50].

5.1 A framework for linearization using Handelman's theorem

In this section, we explain the details of our proposed framework and walk through the steps of Algorithm 1. The input to our algorithm is the *Miss set* M for a particular cache level. A miss set is a conjunction of a polynomial inequality $g > c$ and its validity domain D.

[Step 1.3-1.4] We use the method *RemoveDivs* to remove the integer divisions or modulo expressions from the domain D to get the approximate domain D' which does not contain divisions. Similarly, we use the method *RewriteFloorExpression* to rewrite the polynomial g without floor

terms (for instance, $\lfloor i/2 \rfloor^2$ is rewritten as $(i/2)^2$) to obtain f. Before proceeding forward, we see the applicability of Handelman/s theorem on the polynomial constraint $\mathcal{P} := \{g > c\}$ over domain D'.

Algorithm 1 Approximation using Linearization of Polynomial

```
1:  procedure LinearizePolynomial(M)
2:      // M ← {(g > c) ∧ D}
3:      D' ← RemoveDivs(D)
4:      f ← RewriteFloorExpression (g)
5:      // φ ← a_0 + a_1 x_1 + a_2 x_2 + ··· + a_d x_d
6:      P ← computeProducts(D')
7:      g' ← (f − c) + P · λ⃗
8:      // λ⃗ is the vector of multipliers
9:      θ ← min(affine(g'))
10:     // θ is a function of x_1, ..., x_d
11:     H ← constructHmatrix(g, P)
12:     A ← FindAffineForms(θ, H, D')
13:     // A is a set of affine forms
14:     M' ← AddConstraints(D)
15:     for each φ ∈ A do
16:         M' ← intersect(φ ≥ 0, M')
17:     vol ← card(M')
18:     return vol
```

Algorithm 2 Solve for Sub-polyhedral Approximate Affine forms

```
1:  procedure FindAffineForms(θ, H, D')
2:      A ← {}
3:      for each v ∈ V-form (D') do
4:          // Instantiate parameters with vertex v
5:          θ(v) ← InstParam(θ, v)
6:          switch (ApproxType) do
7:              case Interval:
8:                  for each i∈{1, ..., d} do
9:                      // φ ← a_0 + a_i x_i
10:                     b ← SolveLP1(θ(v), H, i)
11:                     A ←addAffineForm(b, A)
12:             case Octagon:
13:                 for each (i, j)∈{1, ..., d} do
14:                     // φ ← a_0 + a_i x_i + a_j x_j
15:                     b ← SolveLP2(θ(v), H, i, j)
16:                     A ←addAffineForm(b, A)
17:     // Return list of affine forms
18:     Return A
```

Given an input problem size, the iteration domain $\mathcal{D}_I$ is always bounded. The validity domain D' is *non-parametric*; it is also bounded, and a subset of $\mathcal{D}_I$. This means that g is strictly positive on D'. Therefore, by the application of Handelman's theorem, we are looking for all such affine forms which bound $\mathcal{P}$ on D'. Our goal is to find an affine form ϕ such that the polyhedral region $\{\phi > 0 \wedge D\}$ is an over-approximation of the target region M.

[Steps 1.5–1.7] We look to find an affine form ϕ which satisfies $\phi - (f - c) > 0$ on D'. This condition is sufficient to ensure that if $f - c > 0$ (on D') then $\phi > 0$ (on D').

So, we represent $\phi - (f - c)$ using its Handelman representation as

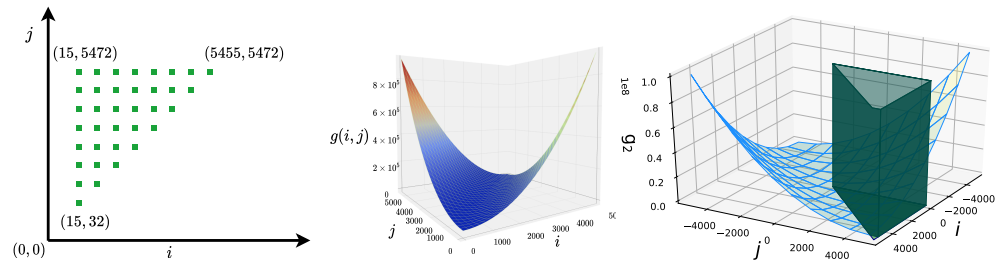
$$\phi - (f - c) = \left(\sum_{\mathbf{I} \in \mathbb{N}^n} \lambda_{\mathbf{I}} p_1^{k_1} \cdots p_n^{k_n} \right)$$

$$\phi = a_0 + \sum_{i=1}^{d} a_i x_i = \left(f - c + \sum_{\mathbf{I} \in \mathbb{N}^n} \lambda_I p_1^{k_1} \cdots p_n^{k_n} \right) = (f - c + \boldsymbol{\sigma}) = (f - c + \mathbf{P} \cdot \vec{\boldsymbol{\lambda}}^{\top}) \tag{3}$$

Here, $x_1, \ldots, x_d$ are the dimensions, and $a_0, \ldots, a_d$ are the coefficients of ϕ. We have $\boldsymbol{\sigma}$ as the Handelman sum of $\phi - (f - c)$. The problem is to find the tightest ϕ such that the non-affine terms in f are canceled out by the non-affine terms of $\mathbf{P}$, where $\mathbf{P}$ is the set of Handelman products.

[Steps 1.9 - 1.12] The above problem can be formulated (similarly to earlier works [28, 45]) as a parametric linear programming as:

$$\begin{aligned} \text{minimize} \quad & (\phi) = \texttt{affine}\left(f - c + \sum_{\mathbf{I} \in \mathbb{N}^n} \lambda_I p_1^{k_1} \cdots p_n^{k_n} \right) = \vec{\Lambda}^{\top} \cdot \Psi(x_1, \ldots, x_d) \\ \text{subject to} \quad & \boldsymbol{H} \cdot \vec{\Lambda}^{\top} = [a_0, a_1, \ldots, a_d, \vec{0}]^{\top} \\ & \lambda_I \geq 0 \end{aligned} \tag{4}$$



(a) (b) (c)

Fig. 5. nussinov kernel example (Ex. 6.1 and Ex. 5.1) (a) Domain integer Polytope D_2. (b) Stack distance polynomial constraint $\mathcal{P}_2 := \{g_2 > 512\}$ evaluated on points of domain D_2. (c) $\mathcal{P}_2$ shown in light-blue. Miss set M_2' shown in dark green. It is the intersection of the affine approximation $\mathcal{P}_2'$, with D_2.

In the Handelman matrix $\boldsymbol{H}$, the first column contains the coefficients for the affine and non-affine terms obtained for the polynomial f. And, the rest of the columns are the coefficients for each Handelman product. The column vector $\Lambda^\top = [1, \vec{\boldsymbol{\lambda}}]^\top$ contains the constant 1 that corresponds to the polynomial f, and the other Handelman multipliers ($\vec{\boldsymbol{\lambda}}^\top$) added for the set of products $\mathbf{P}$. The cost function is obtained from the `affine` part of the Handelman representation and the polynomial $(f - c)$ shown on line 9, which gives rise to the set of affine functions Ψ that are parametric in $x_1, \ldots, x_d$. We compute the Handelman matrix $\boldsymbol{H}$ using method ConstructHmatrix on line. 11.

Here, the first set of constraints $\boldsymbol{H} \cdot \vec{\Lambda}^\top = [a_0, a_1, \ldots, a_d, \vec{0}]^\top$ represent the cancellation of the non-affine terms, and they result in the affine coefficients of ϕ as $a_0 + a_1 x_1 + \ldots + a_d x_d$. Solving the above parametric LP yields a set of affine forms $\mathcal{A}$ for different values of the x-parameters.

It is to be noted that Eqn. 4 describes *a parametric LP formulation* the cost function of which is parametric; both Λ-vector[2] and Ψ-vector are unknown. The constraints however describe a normal (*non-parametric*) polyhedron in Λ. Such a formulation could be solved using a parametric rational[3] linear solver, like the MPT solver [39] to obtain ϕ. The contexts obtained will have multiple non-parametric polytope-regions; and for each of these non-parametric regions, the optimal solution is *naturally* obtained at one of the extremal points of the context polyhedron.

In Sec. 5.3, we propose a novel method, that avoids calling a parametric solver, resulting in a set of non-parametric cost functions, through the following steps: (i) restricting the shape of the affine form ϕ to a fixed template (intervals/octagons), and (ii) instantiating the parameter vector (Ψ) using vertices (of domain D') to obtain a non-parametric θ. The result—Algorithm 2, FindAffineForms, line 12—is a set of template affine forms $\mathcal{A}$ that define a sub-polyhedral system. It is obtained by a small number of (non-parametric) simplex calls, and has almost no redundant constraints.

[Steps 1.15-1.16] We next construct a miss set M' by adding the constraints of domain D and then intersect its constraints with the affine constraints $\phi \geq 0$ constructed from $\forall(\phi) \in \mathcal{A}$.

[Step 1.17] Next, make a call to the Barvinok cardinality function `card`(M') to get the count of Miss set M' as *vol* and return it as the capacity cache misses. The approximation that is obtained is guaranteed to be an over-approximation of the input semi-algebraic set; this is because, the linearization always contains the original input miss set.

[2]The Λ is the solution vector that is used to obtain ϕ.

[3]We found that relaxing the integer variables to rationals is sufficient as it returns a simple, fast, and reasonably accurate approximation.

5.2 Linearization of a polynomial from nussinov

Example 5.1 (Handelman Linearization). One of the polynomials g_2 and domain D_2 induced by the nussinov kernel (from PolyBench) are as follows (also shown in Fig. 5(a) and Fig. 5(b)):

$$g_2(i,j) = \frac{i^2}{32} + \frac{j^2}{32} - \frac{9i}{16} - \frac{j}{2} - \frac{ij}{16} + \frac{186605}{32}, D_2 = \left\{ (i,j) \in \mathbb{Z}^2 \;\middle|\; \begin{array}{l} (i+1) mod(16) = 0, i \geq 15, \\ (j) mod(16) = 0, 17 + i \leq j \leq 5472 \end{array} \right\} \quad (5)$$

Let the number of cache lines for L1 be c_{L1}, and L2 be c_{L2}. Their polynomial inequalities are $\mathcal{P}_2(L1) := \{g_2 > c_{L1}\}$ for $L1$ cache, and $\mathcal{P}_2(L2) := \{g_2 > c_{L2}\}$ for $L2$ cache. The intersection of the polynomial inequality and domain D_2 gives rise to a miss set M_2 that defines the capacity cache misses for a reuse pair. For cache with $c_{L1} = 512$, the miss set for $L1$ cache is: $M_2 = \{\mathcal{P}_2(L1) \wedge D_2\} = \{(g_2(i,j) > 512) \bigwedge D_2\}$. The capacity miss count of L1 cache is given by $\texttt{card}(M_2)$.

We show the working of our Algorithm 1 on the Miss Set M_2. We remove the existential quantifiers from D_2 and relax integers to rationals to obtain an approximate rational domain: $D_2' = \left\{(i,j) \in \mathbb{Q}^2 \;\middle|\; i \geq 15, 17 + i \leq j \leq 5472\right\}$.

As shown in Tab. 1, for PolyBench-Non-Affine, all the non-affine stack distance polynomials are of maximum degree 2. So, we can set a bound on the degree of monomials (K) with $K = 2$ (the higher exponent monomials are not needed). So, the number of products is a quadratic number ($O(n^2)$). Setting $K = 2$ results in the following set of products:

$$\mathbf{P} = [1, (i-15), (j-i-17), (5472-j), (i-15)(j-i-17), (i-15)(5472-j), \\ (j-i-17)(5472-j), (i-15)^2, (j-i-17)^2, (5472-j)^2]$$

The above set of products, along with λ-multipliers ($\vec{\Lambda}^\top = [1, \vec{\boldsymbol{\lambda}}]^\top = [1, \lambda_0, \ldots, \lambda_9]^\top$) define the following Handelman sum:

$$\boldsymbol{\sigma} = \mathbf{P} \cdot \vec{\boldsymbol{\lambda}}^\top = \lambda_0 + \lambda_1(i-15) + \lambda_2(j-i-17) + \lambda_3(5472-j) + \lambda_4(i-15)(j-i-17) + \\ \lambda_5(i-15)(5472-j) + \lambda_6(j-i-17)(5472-j) + \lambda_7(i-15)^2 + \lambda_8(j-i-17)^2 + \lambda_9(5472-j)^2$$

For the parametric LP, we compute the cost function $\theta = \texttt{affine}(g_2 - 512 + \boldsymbol{\sigma})$; this builds from the affine part of polynomial g_2 and the Handelman sum $\boldsymbol{\sigma}$:

$$\theta = (\ (186605) - 512 * 32 + \lambda_0 + \lambda_1(i-15) + \lambda_2(j-i-17) + \lambda_3(5472-j) + \lambda_4(-2i - 15j + 15 * 17) + \\ \lambda_5(5472i - 82080 + 15j) + \lambda_6(5489j - 5472i - 93024) + \lambda_7(225 - 30i) + \lambda_8(289 + 34i - 34j) + \\ \lambda_9(5472 * 5472 - 2 * 5472 * j))$$

We show the parametric LP formulation obtained from Eqn. 4 with the Handelman ($\boldsymbol{H}$) matrix:

$$\begin{array}{ll} \min & \theta = \sum_{q=1}^{10} \lambda_q \psi_q(i,j) \\ \text{s.t} & \boldsymbol{H}_{\text{naf}} \cdot \vec{\Lambda}^\top = \vec{0}, \\ & \lambda_q \geq 0;\ q = 0, \ldots, 9 \end{array} \quad (6)$$

$$\boldsymbol{H} = \begin{array}{c} \\ 1 \\ i \\ j \\ i*j \\ i^2 \\ j^2 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} g_2 & H_1 & H_2 & H_3 & H_4 & H_5 & H_6 & H_7 & H_8 & H_9 & H_{10} \end{array} \\ \left(\begin{array}{cccccccccccc} 170221 & 1 & -15 & -17 & 5472 & 255 & -82080 & -93024 & 225 & 289 & 5472^2 \\ -18 & 0 & 1 & -1 & 0 & -2 & 5472 & -5472 & -30 & 34 & 0 \\ -16 & 0 & 0 & 1 & -1 & -15 & 15 & 5489 & 0 & -34 & -10944 \\ -2 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & 0 & -2 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 1 \end{array} \right) \end{array} \begin{array}{l} \} \boldsymbol{H}_{\text{aff}} \\ \\ \\ \} \boldsymbol{H}_{\text{naf}} \end{array}$$

In Eqn. 6, $\psi_q(i,j)$ is an affine function: $(i,j) \rightarrow \mathbb{Q}$ in cost function θ. The matrix $\boldsymbol{H}$ contains the coefficients for affine and non-affine terms obtained for the polynomial in the first column, and for each Handelman product from next column to the last one. This means that $\boldsymbol{H}_{\text{aff}}$ is a sub-matrix of $\boldsymbol{H}$ with each row corresponding to coefficients of the affine terms. Similarly, $\boldsymbol{H}_{\text{naf}}$ is a sub-matrix of $\boldsymbol{H}$ with each row corresponding to coefficients of the non-affine terms. For this particular example, the affine part of the matrix $\boldsymbol{H}$ is $\boldsymbol{H}_{\text{aff}} = \boldsymbol{H}[0:2][:]$, and the non-affine part is $\boldsymbol{H}_{\text{naf}} = \boldsymbol{H}[3:5][:]$.

In Eqn. 6, the first set of constraints ($\boldsymbol{H}_{\text{naf}} \cdot \vec{\Lambda}^\top = \vec{0}$) are equations that enforce the cancellation of non-affine terms of g_2 with those of the matching products in $\boldsymbol{\sigma}$. We need to find the set of lambda

multipliers that result in the negative coefficients of the non-affine terms of $\boldsymbol{\sigma}$, such that these terms cancel out with the positive coefficients (with same magnitude) of the non-affine terms in g_2.

Here, we search for the constraints of a convex polyhedron such that ϕ is an affine form in d dimensions. To find the coefficients of the affine constraint $a_0 + a_1 i + a_2 j \geq 0$, we solve the equation $\boldsymbol{H}_{\text{aff}} \cdot \vec{\Lambda}^\top = [a_0, a_1, a_2]$. One such solution is for: $\lambda_1, \lambda_4 = 10^4$; $\lambda_5, \lambda_7 = 9999$; $\lambda_6 = 1$; $\lambda_0, \lambda_2, \lambda_3, \lambda_8, \lambda_9 = 0$; $\Rightarrow [a_0, a_1, a_2] = [-149503, 9966.85, -1] \Rightarrow \phi_1 = -149503 + 9966.85i - j \geq 0$.

We obtain the above set of affine constraints $\mathcal{P}'_2 = \{\phi \geq 0 \mid \forall \phi \in \mathcal{A}\}$ which define a polyhedral over-approximation of M_2. (In Fig. 5(c), we plot the domain D'_2, polynomial inequality $\mathcal{P}_2$, and the affine approximation $\mathcal{P}'_2$ to show M'_2, the approximation of miss set M_2.) We intersect $\mathcal{P}'_2$ with D_2 to obtain the approximate miss set M'_2, and call the Barvinok $\texttt{card}(M'_2)$ function, resulting in a cache miss count of 58311 for L1 cache with $c_{L1} = 512$. In comparison, HayStack will perform full enumeration $\forall (i, j) \in D_2$, with over 58×10^3 iteration points and return cache miss count as 58185. It is evident that both miss count numbers are very close to each other, with an error of +0.2%.

It can be seen that counting the miss sets for domains with large number of non-affine iteration points is expensive. An example is the `cholesky` from PolyBench, that has domains with 6.09×10^6 iteration points, and a corresponding number of `card` calls. Using our linearization technique, we count the approximated set using *a single call* to Barvinok with 0% precision loss.

Using parametric simplex (Maréchal et al.'s method [45]): The formulation suggested by Maréchal et al. [45] solves Eqn. 4 in the parametric space in d dimensions; this involves solving a series of parametric simplex problems resulting in a decision tree, the leaves of which are associated with (i) a parametric polyhedral region (the context), and (ii) a cost function θ. Each of these contexts needs to be solved further with a simplex call to obtain a (general) affine constraint $\phi \geq 0$.

For CMC, the above parametric simplex approach could be expensive: the number of contexts is exponential (in K), and the constraint systems have high redundancy. Furthermore, obtaining ϕ involves the additional overhead of creating a decision tree and traversing it.

For example, it can be seen from Table 1 that the number of non-affine stack distance polynomials induced by `nussinov` is 96, and for `cholesky` it is 37. For `nussinov`, solving the parametric LP results in a variable number of contexts: 23–34 across all polynomials. Similarly, for `cholesky`, the range of number of contexts is 27–56. Each of these contexts—after individually inducing a simplex call—results in a constraint system that is an approximation of M. All these approximations have large redundancy ($\approx$ 50–90%) as well, missing the opportunities for further optimizations.

In contrast, the method we propose in the next section will solve the formulation with fewer (non-parametric) simplex calls, leading to systems with no redundant constraints.

5.3 Linearization using sub-polyhedral approximations

The formulation described in Eqn. 4 solves for an affine form ϕ. In this section, we restrict the template of ϕ to be of Interval or Octagon sub-polyhedral type. Such a restriction will describe an (over-)approximation, and these sub-polyhedra have the advantage that they can be described by a finite (and small, respectively $O(d)$ and $O(d^2)$) number of possible affine constraints [48, 49]. In contrast, general convex polyhedra have no fixed "*template*" [54–56], and can lead to a constraint system with a potentially unbounded number of constraints.

Interval and octagon sub-polyhedral formulations In Eqn. 7 and Eqn. 8, we show the specialized parametric linear programming formulations for intervals and octagons respectively:

- *Interval Approximations:* To obtain the interval approximation $\textsc{Interval}(M)$, we look for an affine function ϕ of the ("*Interval*") form $a_0 + a_1 x_1$. We ensure this in Eqn. 7 by canceling the affine coefficients of all other dimensions other than x_1.
- *Octagon Approximations:* To obtain the octagon approximation $\textsc{Octagon}(M)$, we look for an affine function ϕ of the ("*Octagon*") form $a_0 + a_1 x_1 + a_2 x_2$, with $|a_1| = |a_2|$.

This means that the coefficients can only be equal $(+x_1, +x_2)$, or opposite $(+x_1, -x_2)$ (Eqn. 8).

For intervals we show the LP template for parametric dimension x_1, and for octagons we show the LP template for parametric dimensions x_1 and x_2.

$$\begin{aligned} \min \quad & \theta = \vec{\Lambda}^\top \cdot \Psi(x_1, \ldots, x_d) \\ \text{s.t.} \quad & \boldsymbol{H}_{\text{naf}} \cdot \vec{\Lambda}^\top = \vec{0}, \\ & \boldsymbol{H}_{\text{aff}}[x_i] \cdot \vec{\Lambda}^\top = \vec{0}, \ \forall\, x_i \in \mathrm{S} \backslash \{x_1\} \\ & \vec{\Lambda}^\top \geq \vec{0} \end{aligned} \tag{7}$$

$$\begin{aligned} \min \quad & \theta = \vec{\Lambda}^\top \cdot \Psi(x_1, \ldots, x_d) \\ \text{s.t.} \quad & \boldsymbol{H}_{\text{naf}} \cdot \vec{\Lambda}^\top = \vec{0}, \\ & \boldsymbol{H}_{\text{aff}}[x_1] \cdot \vec{\Lambda}^\top = \beta_{x_2}(\boldsymbol{H}_{\text{aff}}[x_2] \cdot \vec{\Lambda}^\top), \{x_1, x_2\} \in \mathrm{S} \\ & \boldsymbol{H}_{\text{aff}}[x_i] \cdot \vec{\Lambda}^\top = \vec{0}, \ \forall\, x_i \in \mathrm{S} \backslash \{x_1, x_2\} \\ & \beta_{x_2} \in \{1, -1\} \\ & \vec{\Lambda}^\top \geq \vec{0} \end{aligned} \tag{8}$$

As discussed earlier, the matrices $\boldsymbol{H}_{\text{naf}}$ and $\boldsymbol{H}_{\text{aff}}$ contain the set of coefficients for the non-affine and affine terms obtained for each Handelman product respectively. The first (equality) constraint $(\boldsymbol{H}_{\text{naf}} \cdot \vec{\Lambda}^\top = \vec{0})$ enforces cancellation of the non-affine part of the polynomial. Here, S is the set of dimensions of the vector-space where the input polynomial g is defined.

For Eqn. 7, the second constraint $\boldsymbol{H}_{\text{aff}}[x_i] \cdot \vec{\Lambda}^\top = \vec{0}$ enforces cancellation of the affine coefficients for dimensions other than x_1 to be zero. And in Eqn. 8, the second constraint $\boldsymbol{H}_{\text{aff}}[x_1] \cdot \vec{\Lambda}^\top = \beta_{x_2}(\boldsymbol{H}_{\text{aff}}[x_2] \cdot \vec{\Lambda}^\top)$ enforces the coefficients to be of equal magnitude for the affine dimensions involved in the affine function such that, for a pair of dimensions (x_1, x_2), we have $|a_1| = |a_2|$. The third constraint $\boldsymbol{H}_{\text{aff}}[x_i] \cdot \vec{\Lambda}^\top = \vec{0}$ sets the coefficient of affine terms other than x_1, x_2 to be zero.

Finding affine forms using instantiation Note that formulation (Eqn. 7 for intervals, and Eqn. 8 for octagons) *still* has a parametric cost function. As discussed in Sec. 5.2, using a parametric simplex solver for CMC is expensive. To avoid this scalability issue, we obtain a set of non-parametric LP problems, the cost functions of which are linear, and the constraints of which define (non-parametric) polyhedra. Solving these set of non-parametric LP problems will result in an interval or octagon polyhedron that we can use to obtain the resulting Interval(M) or Octagon(M).

Next, we explain the working of Algorithm 2 for the interval and octagon sub-polyhedra using LP1 shown in Eqn. 7 and LP2 shown in Eqn. 8:

[Step 2.3-2.5] It is possible to instantiate the cost function θ with any $x \in D$, i.e., the instantiating point for the parameters $x_1, \ldots, x_d$ is selected from the domain polyhedron D. We propose to improve the efficiency of the search for the optimal affine (interval or octagonal) form by instantiating using only the vertices (extremal points) of D. In method FindAffineForms, we iterate over each vertex v of domain D on line 5. Thereby, we instantiate the parameters with v using InstParam to obtain the instantiated cost function $\theta(v)$.

For polyhedral kernels (SCoPs in PolyBench), the validity domains[4] D are mostly[5] intervals or octagons, and so our linearization is simpler, natural, and cheaper, with the octagons being more accurate than the intervals. Also, in most cases D has less than 5 vertices. This is a very small number, and it reduces the number of parametric regions to at most the number of vertices. Therefore, finding affine forms does not get much affected by the exponentiality of vertices [12, 66, 72].

[Step 2.7-2.11] For Interval(M), we iterate over each dimension $i \in \mathrm{S}$ and use the same instantiated linear cost function $\theta(v)$. Then, on line 10, we call SolveLP1 to obtain the interval affine form as b and add b to the list of affine forms $\mathcal{A}$ on line 11.

[4]As explained in Sec. 3, D is a subset of the input iteration-domain $\mathcal{D}_I$, and the latter has been mostly established [60, 61] to be simple domains like Two-Variables-Per-Inequality (TVPI) polyhedra, octagons, or intervals.

[5]PolyBench-Non-Affine induces 368 validity domains, where 287 are octagons, and 77 are intervals. The rest 4 are TVPI.

[Step 2.12-2.16] For Octagon(M), we select a pair of dimensions $(i, j) \in$ S and use the same instantiated linear cost function $\theta(v)$. Then, on line 15, we call SolveLP2 to obtain the octagonal affine form as b, which we add to the list of affine forms $\mathcal{A}$ using method AddAffineForms.

[Step 2.18] Finally, we return the set of interval/octagonal affine forms as $\mathcal{A}$.

Complexity analysis: If the validity domain has $|V|$ vertices and d dimensions, we make $|V| \times d$ ($|V| \times d^2/2$) LP calls to find the interval (octagon) approximation. As mentioned earlier, this is a very small number (around 8 for intervals, and 10 for octagons[6] in most cases) of non-parametric calls. By comparison, the method by Maréchal et al. [45] makes around 30 parametric LP calls.

Also, as intervals and octagons are mostly simplices, this has two additional advantages: reducing the time taken to solve the LP formulation and the counting time by Barvinok `card` function for counting the resulting approximate sub-polyhedra. We will also show in Sec. 7, that fixing the over-approximation to be only intervals/octagons results in a scalable and accurate framework.

5.4 Linearization and counting of a polynomial from `correlation`

Example 5.2 (Interval and Octagon Approximations). The `correlation` kernel from PolyBench, induces the following polynomial and domain:

$$g_3(i,j) = \left(975650 + \left(\frac{-2999j}{8} + \frac{ij}{8}\right)\right); D_3 = \left\{(i,j) \in \mathbb{Z}^2 \;\middle|\; \begin{array}{l}(j) mod(8) = 0; 8 \le j \le 2584; \\ 1 \le i \le 2998\end{array}\right\}$$

The polynomial inequalities induced are $\mathcal{P}_3(L1) := \{g_3 > c_{L1}\}$ with $c_{L1} = 512$ for $L1$ cache. The conjunction of the polynomial inequality and domain D_3 gives rise to the following miss set M_3:

$$M_3 = \{\mathcal{P}_3 \wedge D_3\} = \left\{(g_3(i,j) > 512) \bigwedge D_3\right\}$$

We skip the steps of Algorithm 1, and show only the steps from Algorithm 2. We first compute the parametric objective function $\theta_3(i, j)$ (Eqn. 6):

$$\begin{aligned}\theta_3(i,j) = (9765650 + -2999/8 * j) - 512 + \lambda_1 * (2998 - i) + \lambda_2 * (2584 - j) + \lambda_4(2998i - 2998 + i) \\ + \lambda_3 + \lambda_5(8 - 8 * i - j) + \lambda_6(-2584 + 2584 * i + j) + \lambda_7(-23984 + 8 * i + 2998 * j) + \\ \lambda_8(7746832 - 2584 * i - 2998 * j) + \lambda_9(-20672 + 2592 * j) + \lambda_{14} * (i - 1) + \lambda_{15} * (j - 8) \\ + \lambda_{12}(64 - 16 * j) + \lambda_{13}(6677056 - 5168 * j) + \lambda_{10}(1 - 2 * i) + \lambda_{11}(8988004 - 5996 * i)\end{aligned}$$

We use vertex $v = (1, 8)$ of D_3 for instantiation, and obtain the following (non-parametric) cost function using InstParam method:

$$\begin{aligned}\min(\theta_3(v)) = \min(2997\lambda_1 + 2576\lambda_2 + \lambda_3 + \lambda_4 - 8\lambda_5 + 8\lambda_6 + 8\lambda_7 + 7720264\lambda_8 + 64\lambda_9 - \lambda_{10} \\ + 8982008\lambda_{11} - 64\lambda_{12} + 6635712\lambda_{13} + 7777112)\end{aligned} \tag{9}$$

Interval approximation: For an interval approximation, we solve the LP for each dimension of S with the cost function shown in Eqn. 9; for dimension i we obtain the following constraints:

$$\begin{aligned}(\lambda_5 - \lambda_6 - \lambda_7 + \lambda_8) = -1, (-\lambda_4 + \lambda_{10} + \lambda_{11}) = 0, (-\lambda_9 + \lambda_{12} + \lambda_{13}) = 0 \\ (-\lambda_2 - \lambda_5 + \lambda_6 + 2998\lambda_7 - 2998\lambda_8 + 2592\lambda_9 - 16\lambda_{12} - 5168\lambda_{13} + \lambda_{14}) = 2999\end{aligned} \tag{10}$$

Here, the first three constraints ensure cancellation of non-affine terms of the polynomial and the last equation adds a constraint to cancel affine coefficients for dimension $j \in$ S$\backslash\{i\}$. After solving Eqn. 10, we solve it similarly for dimension j. These two LP calls result in the following miss set: $M_3' = \{(i, j) \mid i \ge -3009, j \le 5186\}$ which is an interval polyhedron.

Octagon approximation: For an octagonal approximation, we solve the LP for each pair of dimensions of S with cost function shown in Eqn. 9 and the following constraints for the $(+i, +j) \in$ S template.

[6]It is possible that these can be further reduced by using properties of vertices of these specific sub-polyhedral varieties [49].

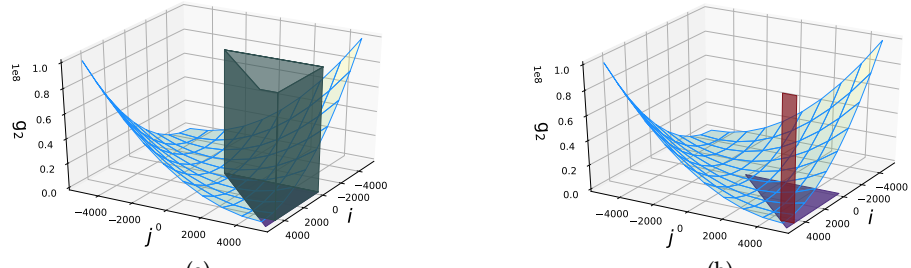


Fig. 6. Bernstein Illustration: nussinov kernel (Ex. 6.1): domain D_2 shown in purple and polynomial g_2 shown in blue. (a) Sub-domain D_i computed for upper bound on dimension i shown in dark green. (b) Sub-domain D_j computed for lower bound on dimension j shown in red.

$$\begin{aligned} &-\lambda_1 + \lambda_2 + 2999\lambda_4 - 7\lambda_5 + 2583\lambda_6 - 2990\lambda_7 + 414\lambda_8 - 2592\lambda_9 - 2\lambda_{10} \\ &- 5996\lambda_{11} + 16\lambda_{12} + 5168\lambda_{13} + \lambda_{14} - \lambda_{15} = -2999 \end{aligned} \tag{11}$$

We solve a LP, using the CPLEX LP solver, with Eqn. 9 as objective function and Eqn. 11 as constraints. Similarly, we solve another LP for opposite sign constraints. These two LPs result in the miss set: $M_3'' = \{(i,j) \mid -i - j \geq -7804102, i + j \geq -3001, i - j \geq -2602\}$, an octagon approximation.

For the interval approximation, we intersect M_3' with the input domain D_3, and then call $\texttt{card}(M_3' \wedge D_3)$ to obtain an approximate cache miss count of 968354. Similarly, for the octagon approximation, we intersect M_3'' with the input domain D_3, and then call $\texttt{card}(M_3'' \wedge D_3)$ to obtain an approximate count of 968351. HaySTack using full enumeration gives an exact count of 968354. In this case, the approximation is nearly exact for both intervals and octagons.

6 LINEARIZATION USING BERNSTEIN APPROXIMATION

In this section, we approximate the polynomial g using Bernstein polynomial linearization. Given a stack distance polynomial g, and a validity domain D, Bernstein linearization works on the vertex representation ($\mathcal{V}$-form) of D where g is represented using the Bernstein basis. The set of parametric Bernstein coefficients (t_k) can be obtained from the symbolic representation of the Bernstein basis [14, 15]. Using these coefficients, a polyhedron M' that bounds the intersection of D, and the polynomial constraint $\mathcal{P} := \{g > c\}$ can be obtained. Also, due to the convex hull property of Bernstein polynomials, g is bounded by the value of min-max Bernstein coefficients.

One way to linearize g is to find the range of each variable of D and obtain interval bounds. This range computed can be a linear approximation of the intersection of $\mathcal{P}$ and D. Now, we show the Bernstein approximation for a polynomial from nussinov:

Example 6.1 (Bernstein linearization). We reuse the Eqn. 5 for the nussinov kernel, with stack distance polynomial $g_2(i,j)$ and validity domain D_2 shown in Fig. 6.

As every point $(i,j) \in D_2$ which satisfies $g_2(i,j) > 512$ will produce a cache miss, the Bernstein approximation can be used to find the affine subdomain of D_2 where $g_2 > 512$. In Eqn. 12, we introduce a single parametric dimension u for obtaining an interval (lower/upper) bound over each dimension. As $g_2(i,j)$ is monotonically decreasing in i, we introduce u as an upper bound ($i \leq u$). For all upper bound constraints of i in D_2, we add additional constraints by replacing i with u.

$$\mathcal{S} = [u] \rightarrow \left\{ g_2(i,j) : \begin{matrix} (i+1) \bmod 16 = 0, i \geq 15, 17 + i \leq j \leq 5472, \\ j \bmod 16 = 0, i \leq u, u \leq j - 17 \end{matrix} \right\} \tag{12}$$

Next, we estimate the values of u for which the lower bound of g_2 is greater than 512, i.e., $g_2 - 512 > 0$. Using ISL, we obtain the lower bound of g_2 in terms of u (where the degree of the

Bernstein basis is $m = 2$):

$$lb(\mathcal{S}) = [u] \rightarrow \left\{\min((\frac{1}{32}u^2 - \frac{5481}{16}u + \frac{30041837}{32}), (\frac{93311}{16} - \frac{17}{16}u)) : 15 \leq u \leq 5455\right\} \quad (13)$$

We seek the minimum value of the coefficient in the validity domain of u ($D_u : 15 \leq u \leq 5455$). From Eqn. 13, let f_1 and f_2 be the Bernstein coefficients (t_k) obtained from the lower bound operation: $f_1 = (\frac{1}{32})u^2 - (\frac{5481}{16})u + (\frac{30041837}{32}) - 512$, $f_2 = (\frac{93311}{16}) - (\frac{17}{16})u - 512$. The roots for f_1 are $5354.87, 5607$, and f_2 is 5007. The second root of f_1 can be discarded since $5607 \notin D_u$. For the remaining two solutions, i.e, 5354.87 and 5007, we estimate the minimum value. As $\min(f_1(5007), f_1(5345.87), f_2(5007), f_2(5354.87)) = f_2(5007)$, the $\min(t_k)$ is obtained from f_2. We fix the value of parameter $u = 5007$ in Eqn. 12.

$$D_i = \{[i, j] : (i + 1) \bmod 16 = 0, j \bmod 16 = 0, i \geq 15, 17 + i \leq j \leq 5472, i < 5007, j \geq 5024\}$$

Similarly, we will estimate the bounds for j with $i = 5007$ such that $g_2(5007, j) = j^2/32 - 5015j/16 + 786454$. Since, $g_2(5007, j)$ is monotonically increasing in j we will estimate the lower bound of j. In this case l is the parametric dimension that is introduced.

$$lb(\mathcal{S}) = [l] \rightarrow \{(i, j) \rightarrow \frac{1}{32}i^2 + \frac{1}{32}j^2 - \frac{9}{16}i - \frac{1}{2}j - \frac{ij}{16} + \frac{186605}{32} : (i + 1) \bmod 16 = 0,$$
$$j \bmod 16 = 0, i >= 15, j \geq l, i = 5007, l \geq 17 + i, j \leq 5472\}, (i, j) \in \mathbb{Z}^2$$

We estimate the values of l for which the lower bound is greater than 512. The Bernstein basis is:

$$lb(\mathcal{S}) = [l] \rightarrow \{\min(\frac{1}{32}l^2 - \frac{5015}{16}l + 786454) : 5024 \leq l \leq 5472\}$$

Solving for the Bernstein coefficients in a similar fashion gives $l = 5024$. The final domain is:

$$D_j = \{[i, j] : (i + 1) \bmod 16 = 0, j \bmod 16 = 0, i >= 15, j > 5024, j \leq 5472, i = 5007\}$$

The total cache miss is $\texttt{card}(D_i) + \texttt{card}(D_j) = 57904$. D_i (D_j) is shown in Fig. 6(a) (Fig. 6(b)).

7 EXPERIMENTAL EVALUATION

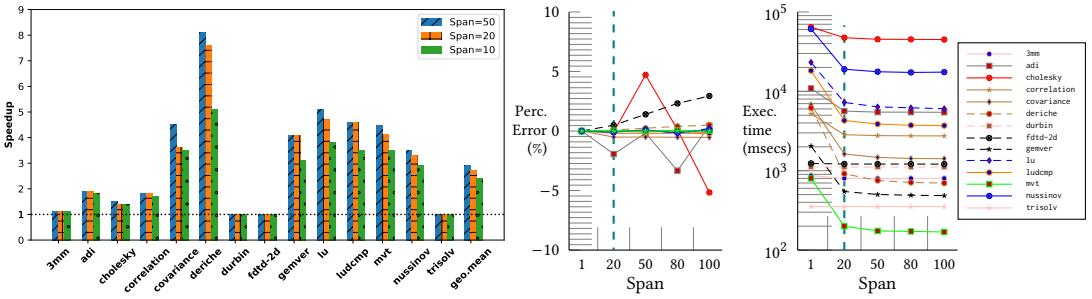
In this section, we show results on the performance (running time) and accuracy of our framework, BullsEye, and compare them with HayStack [35].

Experimental setup. We use the following system setup for evaluation: Intel Xeon W-2133 processor, 32GB RAM with inclusive 384KB L1 (192KB data cache, 192KB instruction cache each with 32KB per core), 6MB L2 caches, and a shared 8.25MB L3 cache.

We test our heuristics on PolyBench [50] kernels with `Large (L)` or `Extra large (XL)` input size (referred to as PolyBench-L and PolyBench-XL), where each result is a median of 10 evaluations. As discussed in Sec. 3, we focus mainly on PolyBench-Non-Affine kernels. We also test on Additional-Benchmarks which will mean 3 kernels (`harris`, `bilateralfiltering`, and `gaussianblur`) from LLVM-TestSuite, 2 kernels (`minver`, and `libud`) from Embench, and 2 kernels (`regdetect`, and `dynprog`) from PolyBench-3.2-XL. In LLVM-TestSuite, for `bilateralfiltering`, the input problem size is 128×128. For `harris` and `gaussianblur`, the image size is 2048×2048; in Embench, the input size for matrices is 3000×3000 for `minver`, and 1024 for `libud`.

The following are the (exact/approximate) analytical tools/algorithms that we compare, along with Cache Simulator (Dinero [40]) and Cache Performance Counter (PAPI [59]).

- **[HayStack [35]]** (Sec. 3) After the Miss sets are constructed, the time taken by HayStack is the time for partial/full enumeration and time for (multiple) calls to Barvinok.
- **[SparseEnum]** (Sec. 4.2) We show the results of SparseEnum, also focusing on the best span obtained. We also illustrate the performance-accuracy trade-off with the span selection.



(a) (b) (c)

Fig. 7. Performance of SparseEnum (Sec. 4.2) on PolyBench-Non-Affine-XL with different span sizes: (a) Speedups over HayStack, (b) Percentage Error with respect to HayStack, and (c) Execution time.

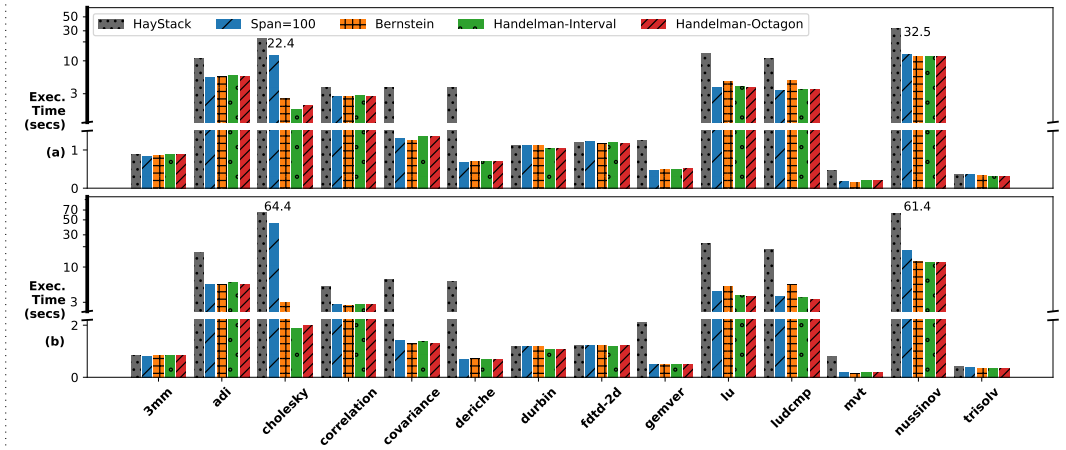


Fig. 8. Execution time on PolyBench-Non-Affine. For (a) PolyBench-L, and (b) PolyBench-XL. Note that for Execution time, plots use partly linear (shown with thin line), and partly log (shown with thick line) scales.

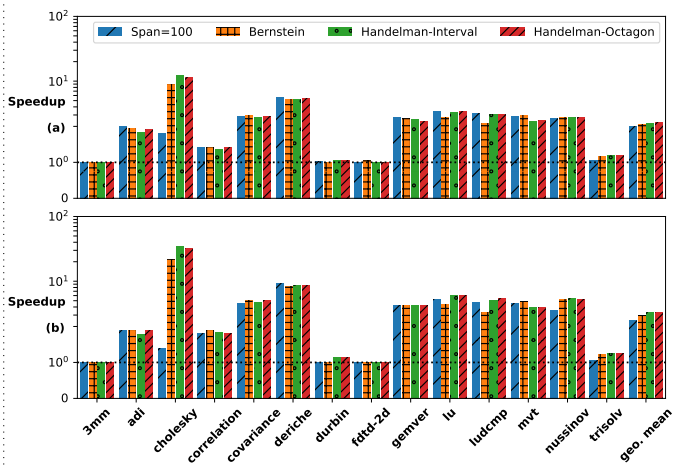
- **[Bernstein Linearization]** (Sec. 6) We linearize M by applying Bernstein basis. We obtain the affine approximation using Bernstein lower/upper bound operations.
- **[Handelman and Interval/Octagon Linearizations]** (Sec. 5.3) We linearize M using Handelman's theorem by adapting the parametric LP to search for interval and octagonal affine approximations.

The resultant M' set(s) are counted using the `card` function of Barvinok. The cumulative time taken for counting is also added to the execution time.

7.1 Execution time results

For performance (speedup) improvements, we compare the total execution time of SparseEnum technique on PolyBench-Non-Affine-XL with HayStack as baseline in Fig. 7(a). Using different span sizes, we obtain a geomean speedup of $(2.1\times, 2.36\times, 2.5\times)$ for $span = (10, 20, 50)$ respectively. It can be seen that, as the span size is increased, the number of `card` calls are reduced by a *span*.

In Fig. 8, we show the comparison of the execution time, and in Fig. 9(a) and (b), we show the speedup of the various proposed methods on PolyBench-Non-Affine in comparison with HayStack. For PolyBench-L, we obtain geomean speedups of $2.15\times$, $2.25\times$, and $2.27\times$ for Bernstein, Handelman-Interval, and Handelman-Octagon, respectively. For PolyBench-XL, the speedups



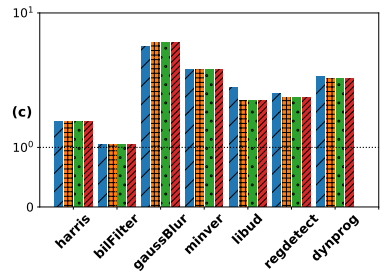
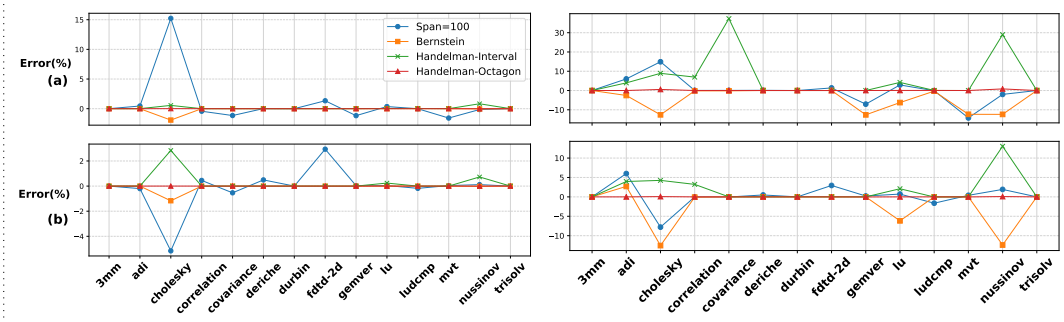


Fig. 9. Speedups obtained by SparseEnum (*span* = 100), Bernstein, Handelman-Interval, and Handelman-Octagon over HayStack exact enumeration. For (a) and (b), the input is PolyBench-Non-Affine, with (a) PolyBench-L, and (b) PolyBench-XL. (c) Input is Additional-Benchmarks.



(1) L1 Cache (2) L2 Cache

Fig. 10. Accuracy on PolyBench-Non-Affine with HayStack as baseline. For (a) PolyBench-Non-Affine-L, and (b) PolyBench-Non-Affine-XL with (1) L1 Cache and (2) L2 Cache.

obtained are 3×, 3.28×, and 3.31× respectively. In Fig. 9(c), we show the results on Additional-Benchmarks. It can be seen that for all these kernels taken from different benchmark suites, our methods show consistently better results.

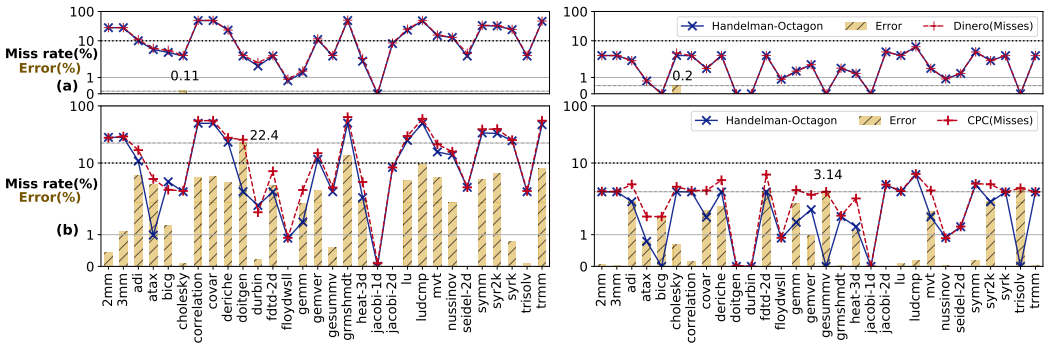
7.2 Accuracy results

Comparison with HayStack. In Fig. 10, we show the accuracy results on PolyBench-Non-Affine for PolyBench-L and PolyBench-XL on L1 and L2 caches. Here, we compare the capacity misses approximated by our various proposed approximation methods SparseEnum, Bernstein, Handelman-Interval and Handelman-Octagon, with the *exact enumeration* of HayStack.

We have observed that the speedup gains after a certain span size (≈ 60) saturate; there is a 2–7% reduction in the accuracy as well. In Figs. 7(b) and 7(c) we show that when *span* = 20, the maximum performance gains, along with minimal accuracy drop are obtained for all the benchmarks.

For PolyBench-L, we show the percentage errors for the L1 cache in Fig. 10.1.a, and for the L2 cache in Fig. 10.2.a. Similarly, for `Extra-Large` input size, we show the percentage errors for the L1 cache in Fig. 10.1.b, and for the L2 cache in Fig. 10.2.b. Here, we obtain a geometric mean error of $\approx$ 1% for the L1 cache, and $\approx$ 2% for the L2 cache for all of our proposed methods.

The max error for Bernstein (Handelman-Interval) is −12.6% (37.3%) on `cholesky` (`covariance`). This large error for Handelman-Interval on `covariance` is only an outlier, due to a single polynomial having a large domain; removal of this polynomial makes the error on `covariance` close to 7×10^{-2}%. On `covariance`, Handelman-Octagon however gives a negligible error of 10^{-4}% error.



(1) L1 Cache (2) L2 Cache

Fig. 11. Accuracy on PolyBench-L comparing Handelman-Octagon vs. (a) Dinero (fully-associative setting), and (b) Cache Performance Counters (CPC) for (1) L1 Cache, and (2) L2 Cache. Line plots show the miss rates. Bar plot is the (%) error between the two line plots. The max error is highlighted with its exact numeric value.

Comparison with Dinero and PAPI. We obtain cache miss measurements from simulation by using Dinero IV [40], and real hardware by using PAPI [59]. For simulation, we use Dinero IV uniprocessor cache simulator with fully associative setting. For real hardware numbers, we obtain two Cache (Hardware) Performance Counters (CPC), PAPI_L1_DCM and PAPI_L2_DCM for L1 and L2 caches respectively; these sum all the data cache misses (including compulsory, conflict, and capacity). For CPC miss rate, we disable the hardware prefetchers.

In Fig. 11.a, we compare Handelman-Octagon vs. Dinero, focusing on the miss rates for L1 and L2 caches. Similarly, in Fig. 11.b, we compare Handelman-Octagon vs. CPC.

For Handelman-Octagon vs. Dinero for L1 and L2 caches, we obtain maximum errors of 0.11% and 0.2%. Also, for Handelman-Octagon vs. CPC, the error is low for almost all the benchmarks; with 1.09% and 0.16% geomean error for L1 and L2 caches respectively. The exceptions are `doitgen` and `gramschmidt` with errors of 22.4% and 13.7% respectively. These errors can be attributed to the following approximations: (i) cache replacement policy (LRU), (ii) differences in the associativity, and (iii) imprecision of the approximate counting itself.

7.3 Discussion

Execution time. In Fig. 9, it is evident that BullsEye scales better than the exact enumeration of HayStack. For `cholesky`, the speedup obtained by Handelman-Interval it is 34.2×, by Handelman-Octagon it is 32.5×, and by Bernstein it is 21.6×. It can be observed that Handelman-Octagon is the fastest, and most precise among all. These impressive speedups could be attributed to the *large non-affine domains* in this benchmark, which need to be fully enumerated by HayStack. Moreover, the execution time of our linearization frameworks (either Handelman or Bernstein based) are independent of the size of the input problem; this can be attributed to the fact that the number of polynomials obtained for a PolyBench kernel are independent of the input-size.

Fig. 12. Accuracy comparison (geomean)

Method	Large		Extra Large	
	L1	*L2*	*L1*	*L2*
Span=100	0.41	0.78	0.27	0.57
Bernstein	0.30	2.48	0.15	1.02
Handelman-Interval	0.02	0.92	0.21	1.88
Handelman-Octagon	0.002	0.029	0.03	0.08

Accuracy. For PolyBench-XL on L2 cache, using Handelman-Interval, we obtain good accuracy for all benchmarks, except in the case of `nussinov` and `cholesky`, where we obtain 13% and 4.25% errors respectively. However, Handelman-Octagon results in much better approximations resulting in just 0.09%, and 0.074% errors, respectively. These high accuracies could be attributed to the fact that the polynomials are convex or nearly convex. In Tab. 12, we show that BullsEye obtains good accuracy for the entire PolyBench-Non-Affine benchmark.

Comparison between sub-polyhedra. The octagon linearizations are superior to the (straightforward) interval-based ones, justifying their utility with better accuracy. Also, both intervals and octagons are comparable in their LP overhead—meaning, instantiation and solving time. This is because of the low-dimensionality, and the *nearly octagon nature* of most of the domains.

We show in Fig. 7.[b,c] that it is possible to empirically set the span size using cache parameters (element size and cache line size) to obtain a trade-off between performance and accuracy.

For SparseEnum, we obtain both positive and negative errors. However, for Handelman linearizations, we obtain only over-approximations as the approximation always contains the non-affine set M. As mentioned earlier, CMC does not need *only* over-approximations; both over and under approximations, corresponding to positive and negative errors respectively, are acceptable.

Limitations. The largest errors by our SparseEnum approximation—in benchmarks such as `choleksy`—are when (i) the validity domain of the reuse distance polynomial increases sharply in the non-affine dimensions, and/or (ii) when the iteration domain involves division/mod operations.

Our Handelman-based linearizations work only when D is non-parametric [70]; while the Bernstein-based linearization can work on a parametric domain. For CMC, D is always non-parametric, as the parameters in the miss sets are substituted before linearization.

For higher degree polynomials (degree of g is ≥ 3), our Handelman (Bernstein) linearizations could lead to an explosion in the number of terms in the Handelman parameter K (Bernstein basis m). In such cases, we think that our SparseEnum may be more scalable but less precise than linearization-based methods.

8 RELATED WORK

We discuss various related works based on algorithms, tools, and methodologies for CMC:

Cache Miss Calculation by Simulators: There are many tools [9] that perform (dynamic) cache simulation and give accurate results. Dinero IV [40] is one such popular uniprocessor cache simulator that handles hierarchical, fully, and set associative caches, along with various replacement and write policies. Static analysis tools [2, 35] compare themselves against it. It is well known that dynamic tools consume large amounts of time and memory.

Cache Miss Calculation as Static Analysis: It has been well recognized that the CMC problem reduces to a counting problem on Presburger formulae. Several researchers have attempted [10, 31] to estimate the number of Cache Misses using static analysis and Presburger arithmetic based tools. For example, Chatterjee et al. [10] modeled it using the Omega library [52].

Beyls et al. [7], building from their previous work [6], proposed two methods—based on profiling and analytical computation—to generate cache hints for runtime improvement. They proposed an analytical formulation to count capacity cache misses of fully associative caches using stack (reuse) distances to result in non-affine Ehrhart polynomials.

Bao et al. [2] proposed PolyCache, an analytical model for set-associative caches with LRU replacement policy using symbolic counting, with detailed application to real-world hardware. They also model multi-level caches and various write policies, though the complexity of their algorithm increases with associativity. Their tool is based on the ISL library [63] and Barvinok [62, 65].

Gysi et al. [35] proposed HayStack, a technique for analytical modeling of Static Control (polyhedral) Programs on fully-associative caches. It computes the exact cache miss count, using an implementation of the model derived from the analytical computation of stack distance by Beyls et al. [7]. They also propose various novel and efficient techniques to count non-affine sets, using division and mod operations, by a thorough evaluation on PolyBench benchmarks [50].

BullsEye directly builds from Barvinok library and HayStack infrastructure. On affine benchmarks, HayStack as well as BullsEye quickly and precisely result in the exact cache miss calculations; they both *directly rely* on Barvinok. HayStack solves the problem of counting non-affine miss sets

using exact enumeration. BullsEye proposes various novel statistical and mathematical techniques that approximate the cache misses, with scalability and accuracy as the twin goals. On PolyBench-Non-Affine, the performance of BullsEye is better than HayStack. On the rest, the performance of BullsEye is *exactly the same* as HayStack.

Chen et al. [11] proposed Static Parallel Sampling (SPS) to estimate the cache miss ratio by modeling LRU fully associative caches. They analyze the program structure by LLVM framework [41], while using sampling techniques to obtain *reuse time*. SPS can even handle irregular loop nests.

Our SparseEnum is based on the uniform sampling of iterations in non-affine dimensions. In contrast to Chen et al. [11], our method approximates reuse distance for nearby iteration points, focusing on polyhedral kernels, and using Barvinok cardinality for counting.

Ehrhart polynomials: For an integer polytope $P \in \mathbb{R}^d$, the number of lattice points in P with dilation factor n can be represented using a polynomial expression, referred to as Ehrhart polynomials [21–24]. The French mathematician Eugène Ehrhart first proposed Ehrhart polynomials. Later they were extended [13, 16] to represent the number of integer points of a parameterized polytope.

Algorithms for Ehrhart polynomial computation: Clauss [13, 16] first proposed different techniques for counting integer solutions of a parametric polyhedron for program analysis. This involves counting parametric polyhedra using a set of Ehrhart polynomials with different validity domains. The worst-case computation of Ehrhart polynomials for fixed dimensions using their method is exponential in the input. Moreover, an implementation computing Ehrhart polynomials could give rise to degenerate domains and larger periods, resulting in high execution times.

Verdoolaege et al. [65] proposed a polynomial time approach for counting integer solutions (Ehrhart polynomial) of parametric polyhedra using a Barvinok based decomposition of validity domains. They apply parametric counting [19] along with decomposition of validity domains of parameter space [44]. To count Presburger formulas, Pugh et al. [53] gave various techniques based on rewriting summation formulae, based on the Omega library [52].

Ehrhart Polynomials and program analysis: Ehrhart polynomials have been used for representing counts of integer sets in several applications. Clauss [13] was the first to use Ehrhart polynomials for program analysis. Analytical counting of such polynomials in polynomial time [65] enables scalable program analysis on integer polyhedra such as symbolically counting the reuse distance [7], the number of operations performed by a loop [43], the number of cache lines touched by a loop [29], allocating parallel processing elements in a FPGA to execute a loop [38].

Barvinok Algorithm: Barvinok [3] gave a polynomial time algorithm for counting integer points in a polyhedron, when the number of dimensions is fixed. This algorithm with later improvements is implemented in *LattE integrale* [1], and *Barvinok library* [62] with an isl [63] interface. Barvinok symbolically counts parametric polytopes as Ehrhart (quasi-)polynomials [21, 24], that are parametric in input dimensions. Problems like CMC for a LRU fully associative cache reduce to counting parameterized polytopes [6, 7]. We use the Barvinok algorithm [65] implemented [62] in ISL library.

Parametric Solvers: Hang et al. [69] proposed to solve a parametric LP by applying parallelization and using a parametric simplex, or a set of LP problems using instantiation. In Sec. 5.3, instead of a parametric solver, we use LP instantiations to find the sub-polyhedral affine forms for scalability.

Linearizations using Handelman's theorem: Maréchal et al. [45] proposed a linearization method using Handelman's theorem on positive polynomials along various heuristics to reduce the complexity of selecting the products. For static analysis (abstract interpretation) applications and SMT solving, they show that programs containing nonlinear expressions can be linearized. They formulate a parametric LP problem and solve it using a *decision tree* method which results in a general polyhedral over-approximation, and implement it in Verified Polyhedra Library [8].

Our framework (Sec. 5.1) is similar to Maréchal et al.'s formulation [45], though we show in Sec. 5.2, that it is particularly suited to CMC. In Sec. 5.3 we show improvements based on sub-polyhedral approximations resulting in almost non-redundant systems. We also propose other simplifications (Sec. 4.2), and use Bernstein based [14, 15, 17] linearization techniques (Sec. 6).

Handelman's theorem and Polyhedral Compilation: Feautrier [28] was the first to propose going beyond polyhedra by identifying Handelman's theorem [37] as a strict extension of Farkas' lemma [57], and proposed to apply it to various polyhedral compiler applications.

Recently, Yuki [70] proposed applying Handelman's theorem for polynomial scheduling problems of Affine Control Loops, more precisely by performing Index Set Splitting (ISS) [33]. This attempt led to a negative result because: not all polynomials have a Handelman representation when the domain is parametric, and for ISS problem, the global minimizers occur in the interior of the domain.

As far as we are aware of, our method is the first one to propose using Handelman's theorem for linearizations–and approximations based on the above theorem—for CMC.

Interval [18] and octagon sub-polyhedra [48] have been successfully used in various abstract interpretation problems [47, 48], as well as in polyhedral compilation [60, 61]. It is well known [47–49] that for a fixed dimension, the number of constraints is fixed for the intervals and octagonal sub-polyhedra. In contrast, the number of constraints is unbounded for general (convex) polyhedra. Sankaranarayanan et al. [54–56] proposed template polyhedra that generalize intervals and octagons, and can be used to limit the forms of approximation.

In this paper, we use template sub-polyhedral approximations for better scalability.

9 CONCLUSIONS AND FUTURE WORK

We have proposed a new framework, BullsEye, for the approximation of non-affine (semi-algebraic) stack distance polynomials to count capacity misses. We believe that ours is the first work that proposes scalable, accurate, and problem-size independent approximations based on static-analysis for CMC. We propose a variety of techniques: statistical sampling techniques, and from within the polyhedral model either relying on Bernstein's theorem or Handelman's theorem based mathematical linearizations, embellished with sub-polyhedral (interval/octagon) approximations.

We believe that ours is the first method to propose applying Handelman's theorem (a strict extension of Farkas Lemma) for CMC. We have implemented our methods, and the results show good speedups (geomean 3.31×), as well as accuracy (geomean 0.08%) for octagons over the state-of-the-art technique HayStack that uses exact enumeration for counting non-affine sets. Also, our comparison with the Dinero simulator shows that our results are relevant for realistic cache policies, beyond our current LRU fully associative model.

Our methods are already integrated with HayStack, and we plan to release the source code of BullsEye. We also plan to implement BullsEye in the standard LLVM [41] (inside or outside the Polly [34] polyhedral loop-optimization pass), or the latest MLIR [42] compiler infrastructures.

Our sampling and linearization methods can potentially be selectively applied in a complementary fashion for different varieties of miss sets: based on the degree of the polynomial, or on the shape, size, dimension, and type of the validity domain. Deciding which method to apply in a particular scenario also depends on the application where CMC is used. These aspects are left for future work.

We also plan to extend our current formulation to set-associative caches by considering an additional parameter that encodes associativity, followed by sub-polyhedral algorithms for scalable approximate counting. Considering arbitrary template polyhedra provided as a set of linear forms is also left for future work. Our source code and relevant material are available at https://compilers.cse.iith.ac.in/projects/bullseye.

ACKNOWLEDGEMENTS

We are thankful to Govindarajan Ramaswamy, Albert Cohen, Rajesh Kedia, Jyothi Vedurada, Utpal Bora, and S. VenkataKeerthy for their valuable feedback on our work at various stages. We would like to thank the anonymous reviewers of IMPACT-2021 workshop, and ACM TACO for their insightful and detailed comments which helped in improving the article. This work has been partly supported by the funding received from DST, Govt of India, through the Data Science cluster of the ICPS program (DST/ICPS/CLUSTER/Data Science/2018/General), and an NSM research grant (MeitY/R&D/HPC/2(1)/2014).

References

[1] V. Baldoni, N. Berline, J.A. De Loera, B. Dutra, M. Koppe, S. Moreinis, G. Pinto, M. Vergne, and J. Wu. 2013. *A User's Guide for LattE integrale v1.7.2.* http://www.math.ucdavis.edu/~latte/

[2] Wenlei Bao, Sriram Krishnamoorthy, Louis-Noel Pouchet, and P. Sadayappan. 2017. Analytical Modeling of Cache Behavior for Affine Programs. *Proc. ACM Program. Lang.* 2, POPL, Article 32 (Dec. 2017), 26 pages.

[3] Alexander I. Barvinok. 1994. A Polynomial Time Algorithm for Counting Integral Points in Polyhedra When the Dimension Is Fixed. *Mathematics of Operations Research* 19, 4 (1994), 769–779. http://www.jstor.org/stable/3690312

[4] S. Bernstein. 1952. Collected Works, vol. 1. *USSR Academy of Sciences* (1952).

[5] S. Bernstein. 1954. Collected Works, vol. 2. *USSR Academy of Sciences* (1954).

[6] Kristof Beyls and Erik D'Hollander. 2001. Reuse Distance as a Metric for Cache Behavior. In *Proc. of the IASTED Int. Conference on Parallel and Distributed Computing and Systems, IASTED, Anaheim, California, USA, 2001.* 617–622.

[7] Kristof Beyls and Erik H. D'Hollander. 2005. Generating Cache Hints for Improved Program Efficiency. *J. Syst. Archit.* 51, 4 (April 2005), 223–250. https://doi.org/10.1016/j.sysarc.2004.09.004

[8] S. Boulmé, A. Maréchaly, D. Monniaux, M. Périn, and H. Yu. 2018. The Verified Polyhedron Library: an Overview. In *2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC).* 9–17.

[9] Hadi Brais, Rajshekar Kalayappan, and Preeti Ranjan Panda. 2020. A Survey of Cache Simulators. *ACM Comput. Surv.* 53, 1, Article 19 (Feb. 2020), 32 pages. https://doi.org/10.1145/3372393

[10] Siddhartha Chatterjee, Erin Parker, Philip J. Hanlon, and Alvin R. Lebeck. 2001. Exact Analysis of the Cache Behavior of Nested Loops. In *Proceedings of the ACM SIGPLAN 2001 Conference on Programming Language Design and Implementation* (Snowbird, Utah, USA) *(PLDI '01).* ACM, New York, NY, USA, 286–297. https://doi.org/10.1145/378795.378859

[11] Dong Chen, Fangzhou Liu, Chen Ding, and Sreepathi Pai. 2018. Locality Analysis through Static Parallel Sampling. *SIGPLAN Not.* 53, 4 (jun 2018), 557–570. https://doi.org/10.1145/3296979.3192402

[12] N.V Chernikova. 1965. An algorithm for finding a general formula for the non-negative solutions of linear inequalities. *U.S.S.R. Computational Mathematics and Mathematical Physics* 5, 2 (1965), 228–233.

[13] Philippe Clauss. 1996. Counting Solutions to Linear and Nonlinear Constraints through Ehrhart Polynomials: Applications to Analyze and Transform Scientific Programs. In *ACM International Conference on Supercomputing 25th Anniversary Volume* (Munich, Germany). Association for Computing Machinery, New York, NY, USA, 237–244.

[14] P. Clauss, F J Fernández, D. Garbervetsky, and S. Verdoolaege. 2009. Symbolic Polynomial Maximization over Convex Sets and Its Application to Memory Requirement Estimation. *IEEE Trans. VLSI Syst.* 17, 8 (Aug. 2009), 983–996.

[15] P. Clauss, D. Garbervetsky, V. Loechner, and S. Verdoolaege. 2011. Polyhedral Techniques for Parametric Memory Requirement Estimation. In *Energy-Aware Memory Management for Embedded Multimedia Systems: A Computer-Aided Design Approach.* Taylor and Francis.

[16] Philippe Clauss and Vincent Loechner. 1998. Parametric analysis of polyhedral iteration spaces. *Journal of VLSI signal processing systems for signal, image and video technology* 19, 2 (1998), 179–194.

[17] P. Clauss and I. Tchoupaeva. 2004. A Symbolic Approach to Bernstein Expansion for Program Analysis and Optimization. In *Compiler Construction, 13th Int. Conference, CC 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, Mar 29 - Apr 2, 2004, Proceedings (LNCS, Vol. 2985).* Springer, 120–133.

[18] P. Cousot and R. Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* ACM Press, New York, NY, Los Angeles, California, 238–252.

[19] Jesús A. De Loera, Raymond Hemmecke, Jeremiah Tauzer, and Ruriko Yoshida. 2004. Effective lattice point counting in rational convex polytopes. *Journal of Symbolic Computation* 38, 4 (2004), 1273–1302. https://doi.org/10.1016/j.jsc.2003.04.003 Symbolic Computation in Algebra and Geometry.

[20] Chen Ding and Yutao Zhong. 2003. Predicting Whole-Program Locality through Reuse Distance Analysis. *SIGPLAN Not.* 38, 5 (May 2003), 245–257. https://doi.org/10.1145/780822.781159

[21] Eugéne Ehrhart. 1962. Sur les polyédres rationnels homothétiques à n dimensions. *Comptes rendus de l'Académie des Sciences* 254 (1962), 616–618.

[22] Eugéne Ehrhart. 1967. Sur un probléme de géométrie diophantienne linéaire. I. *Polyédres et réseaux, J. Reine Angew. Math* 226 (1967), 1–29.

[23] Eugéne Ehrhart. 1967. Sur un probléme de géométrie diophantienne linéaire. II. *Systémes diophantiens linéaires, J. Reine Angew. Math* 227 (1967), 25–49.

[24] Eugéne Ehrhart. 1977. Polynômes arithmétiques et méthode des polyédres en combinatoire. *International Series of Numerical Mathematics* 35 (1977), 165.

[25] David Eklov and Erik Hagersten. 2010. StatStack: Efficient modeling of LRU caches. In *2010 IEEE International Symposium on Performance Analysis of Systems Software (ISPASS)*. 55–65. https://doi.org/10.1109/ISPASS.2010.5452069

[26] Rida Farouki. 2012. The Bernstein polynomial basis: A centennial retrospective. *Computer Aided Geometric Design* 29 (08 2012), 379–419. https://doi.org/10.1016/j.cagd.2012.03.001

[27] Paul Feautrier. 1992. Some efficient solutions to the affine scheduling problem: I. One-dimensional time. *Int. J. Parallel Program.* 21 (October 1992), 313–348. Issue 5. https://doi.org/10.1007/BF01407835

[28] Paul Feautrier. 2015. The Power of Polynomials. In *Fifth Int. Workshop on Polyhedral Compilation Techniques (IMPACT'15), in conjunction with HiPEAC'15*. Amsterdam, The Netherlands. https://acohen.gitlabpages.inria.fr/impact/impact2015/

[29] Jeanne Ferrante, Vivek Sarkar, and W. Thrash. 1991. On Estimating and Enhancing Cache Effectiveness. In *Proc. of the 4th Int. Workshop on Languages and Compilers for Parallel Computing*. Springer-Verlag, Berlin, Heidelberg, 328–343.

[30] Michael J. Fischer and Michael O. Rabin. 1998. Super-Exponential Complexity of Presburger Arithmetic. In *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Springer, 122–135.

[31] Somnath Ghosh, Margaret Martonosi, and Sharad Malik. 1998. Precise Miss Analysis for Program Transformations with Caches of Arbitrary Associativity. In *Proc. of the 8th Int. Conference on Architectural Support for Programming Languages and Operating Systems* (USA) *(ASPLOS VIII)*. ACM, New York, NY, USA, 228–239.

[32] Somnath Ghosh, Margaret Martonosi, and Sharad Malik. 1999. Cache Miss Equations: A Compiler Framework for Analyzing and Tuning Memory Behavior. *ACM Trans. Program. Lang. Syst.* 21, 4 (July 1999), 703–746.

[33] M. Griebl, P. Feautrier, and C. Lengauer. 2000. Index Set Splitting. *IJPP* 28 (2000), 607–631.

[34] Tobias Grosser, Armin Groesslinger, and Christian Lengauer. 2012. Polly—performing polyhedral optimizations on a low-level intermediate representation. *Parallel Processing Letters* 22, 04 (2012), 1250010.

[35] Tobias Gysi, Tobias Grosser, Laurin Brandner, and Torsten Hoefler. 2019. A Fast Analytical Model of Fully Associative Caches. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Phoenix, AZ, USA) *(PLDI 2019)*. ACM, New York, NY, USA, 816–829. https://doi.org/10.1145/3314221.3314606

[36] Christoph Haase. 2018. A Survival Guide to Presburger Arithmetic. *ACM SIGLOG News* 5, 3 (July 2018), 67–82. https://doi.org/10.1145/3242953.3242964

[37] David Handelman. 1988. Representing polynomials by positive linear functions on compact convex polyhedra. *Pacific J. Math.* 132, 1 (1988), 35–62. https://projecteuclid.org:443/euclid.pjm/1102689794

[38] Frank Hannig and Jürgen Teich. 2001. Design Space Exploration for Massively Parallel Processor Arrays. In *Parallel Computing Technologies*, Victor Malyshkin (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 51–65.

[39] M. Herceg, M. Kvasnica, C. N. Jones, and M. Morari. 2013. Multi-Parametric Toolbox 3.0. In *2013 European Control Conference (ECC)*. 502–510. https://doi.org/10.23919/ECC.2013.6669862

[40] Jan Edler, and Mark D. Hill. 1999. Dinero IV Trace-Driven Uniprocessor Cache Simulator. http://pages.cs.wisc.edu/ markhill/DineroIV/.

[41] Chris Lattner and Vikram S. Adve. 2004. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *2nd IEEE / ACM International Symposium on Code Generation and Optimization (CGO 2004), 20-24 March 2004, San Jose, CA, USA*. IEEE Computer Society, 75–88. https://doi.org/10.1109/CGO.2004.1281665

[42] C Lattner, M Amini, U Bondhugula, A Cohen, A Davis, J. A. Pienaar, R Riddle, T Shpeisman, N Vasilache, and O Zinenko. 2021. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In *IEEE/ACM Int. Symp. on Code Generation and Optimization, CGO 2021, Seoul, South Korea, Feb. 27 - Mar 3, 2021*. IEEE, 2–14.

[43] B Lisper. 2003. Fully Automatic, Parametric Worst-Case Execution Time Analysis. In *Proc. of the 3rd Int. Workshop on Worst-Case Execution Time Analysis, WCET 2003*, Vol. MDH-MRTC-116/2003-1-SE. 99–102.

[44] Vincent Loechner and Doran K. Wilde. 1997. Parameterized Polyhedra and Their Vertices. *Int. J. Parallel Program.* 25, 6 (Dec. 1997), 525–549. https://doi.org/10.1023/A:1025117523902

[45] Alexandre Maréchal, Alexis Fouilhé, Tim King, David Monniaux, and Michaël Périn. 2016. Polyhedral Approximation of Multivariate Polynomials using Handelman's Theorem. In *International Conference on Verification, Model Checking, and Abstract Interpretation 2016*. Barbara Jobstmann and Rustan Leino, St. Petersburg, United States.

[46] R. L. Mattson, J. Gecsei, D. Slutz, and I. Traiger. 1970. Evaluation Techniques for Storage Hierarchies. *IBM Syst. J.* 9 (1970), 78–117.

[47] A. Miné. 2004. *Weakly Relational Numerical Abstract Domains*. Ph. D. Dissertation. École Polytechnique, Palaiseau, France. http://www.di.ens.fr/~mine/these/these-color.pdf.

[48] Antoine Miné. 2006. The Octagon Abstract Domain. *Higher-Order and Symbolic Computation* 19, 1 (2006), 31–100.
[49] Abhishek Patwardhan and Ramakrishna Upadrasta. 2019. Some Efficient Algorithms for the Tightest U-TVPI Polyhedral Over-Approximation problem. In *Ninth International Workshop on Polyhedral Compilation Techniques (IMPACT'19), in conjunction with HiPEAC'19*. Valencia, Spain. https://acohen.gitlabpages.inria.fr/impact/impact2019/
[50] Louis-Noël Pouchet, Tomofumi Yuki, et al. 2018. PolyBench 4.2 Benchmarks. http://sourceforge.net/projects/polybench/.
[51] Mojżesz Presburger. 1929. Über die Vollstandigkeit eines gewissen systems der Arithmetik ganzer Zahlen, in Welchem die Addition als einzige Operation hervortritt. Comptes Rendus du I congres de Mathematiciens des Pays Slaves, 92–101.
[52] William Pugh. 1991. The Omega test: a fast and practical integer programming algorithm for dependence analysis. In *Proceedings of the 1991 ACM/IEEE conference on Supercomputing* (Albuquerque, New Mexico, United States) *(Supercomputing '91)*. ACM, New York, NY, USA, 4–13. https://doi.org/10.1145/125826.125848
[53] William Pugh. 1994. Counting Solutions to Presburger Formulas: How and Why. In *Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation* (Orlando, Florida, USA) *(PLDI '94)*. ACM, New York, NY, USA, 121–134. https://doi.org/10.1145/178243.178254
[54] Sriram Sankaranarayanan, Michael A. Colón, Henny Sipma, and Zohar Manna. 2006. Efficient Strongly Relational Polyhedral Analysis. In *Verification, Model Checking, and Abstract Interpretation*, E. Allen Emerson and Kedar S. Namjoshi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 111–125.
[55] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. 2004. Constraint-Based Linear-Relations Analysis. In *SAS (Lecture Notes in Computer Science, Vol. 3148)*, Roberto Giacobazzi (Ed.). Springer, 53–68.
[56] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. 2005. Scalable Analysis of Linear Systems Using Mathematical Programming. In *Verification, Model Checking, and Abstract Interpretation*, Radhia Cousot (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 25–41.
[57] Alexander Schrijver. 1986. *Theory of linear and integer programming.* John Wiley & Sons, Inc., New York, NY, USA.
[58] Markus Schweighofer. 2002. An algorithmic approach to Schmüdgen's Positivstellensatz. *Journal of Pure and Applied Algebra* 166, 3 (2002), 307 – 319. https://doi.org/10.1016/S0022-4049(01)00041-X
[59] D Terpstra, H Jagode, H You, and J Dongarra. 2010. Collecting Performance Data with PAPI-C. In *Tools for High Performance Computing 2009*. Springer, 157–173.
[60] Ramakrishna Upadrasta. 2013. *Sub-Polyhedral Compilation Using (Unit-)Two-Variable-Per-Inequality Polyhedra or Scalability Challenges in the Polyhedral Model: An Algorithmic Approach using (Unit-)Two-variable Per Inequality Sub-Polyhedra.* Ph. D. Dissertation. Université Paris-Sud (11), Orsay, France. http://tel.archives-ouvertes.fr/tel-00818764.
[61] Ramakrishna Upadrasta and Albert Cohen. 2013. Sub-Polyhedral Scheduling Using (Unit-)Two-Variable-Per-Inequality Polyhedra. In *40th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2013)*. Rome, Italy.
[62] Sven Verdoolaege. 2007. Barvinok, a library for counting the integer points in parametric and non-parametric polytopes. See https://repo.or.cz/barvinok.git.
[63] Sven Verdoolaege. 2010. Isl: An Integer Set Library for the Polyhedral Model. In *Proc. of the 3rd Int. Congress Conference on Mathematical Software* (Kobe, Japan) *(ICMS'10)*. Springer-Verlag, 299–302. https://repo.or.cz/isl.git.
[64] Sven Verdoolaege and Tobias Grosser. 2012. Polyhedral Extraction Tool. Second International Workshop on Polyhedral Compilation Techniques (IMPACT'12), Paris, France.
[65] S Verdoolaege, R Seghir, K Beyls, V Loechner, and M Bruynooghe. 2004. Analytical Computation of Ehrhart Polynomials: Enabling More Compiler Analyses and Optimizations. In *Proc. of the 2004 Int. Conf. on Compilers, Architecture, and Synthesis for Embedded Systems* (Washington DC, USA) *(CASES '04)*. ACM, USA, 248–258.
[66] H. Le Verge. 1992. *A Note on Chernikova's Algorithm.* Technical Report 635. IRISA, Rennes, France.
[67] Michael E. Wolf and Monica S. Lam. 1991. A Data Locality Optimizing Algorithm. In *Proceedings of the ACM SIGPLAN 1991 Conference on Programming Language Design and Implementation* (Toronto, Ontario, Canada) *(PLDI '91)*. Association for Computing Machinery, New York, NY, USA, 30–44. https://doi.org/10.1145/113445.113449
[68] X Xiang, C Ding, H Luo, and B Bao. 2013. HOTL: A Higher Order Theory of Locality. In *Proc. of the 18th Int. Conf. on Architectural Support for Programming Languages and Operating Systems* (USA) *(ASPLOS '13)*. ACM, USA, 343–356.
[69] Hang Yu. 2019. *Towards an Efficient Parallel Parametric Linear Programming Solver.* Ph. D. Dissertation. Université Grenoble Alpes.
[70] Tomofumi Yuki. 2019. The Limit of Polynomials. In *Ninth International Workshop on Polyhedral Compilation Techniques (IMPACT'19), in conjunction with HiPEAC'19*. Valencia, Spain. https://acohen.gitlabpages.inria.fr/impact/impact2019/
[71] Yutao Zhong, Xipeng Shen, and Chen Ding. 2009. Program Locality Analysis Using Reuse Distance. *ACM Trans. Program. Lang. Syst.* 31, 6, Article 20 (aug 2009), 39 pages. https://doi.org/10.1145/1552309.1552310
[72] G.M. Ziegler. 2006. *Lectures on polytopes.* Springer Science.