Sub-polyhedral scheduling using (unit-)two-variable-per-inequality polyhedra

Ramakrishna Upadrasta, Albert Cohen

Sub-Polyhedral Scheduling Using (Unit-)Two-Variable-Per-Inequality Polyhedra

Ramakrishna Upadrasta
INRIA and LRI, Université Paris-Sud (11)
Ramakrishna.Upadrasta@inria.fr

Albert Cohen
INRIA and École Normale Supérieure
Albert.Cohen@inria.fr

Abstract

Polyhedral compilation has been successful in the design and implementation of complex loop nest optimizers and parallelizing compilers. The algorithmic complexity and scalability limitations remain one important weakness. We address it using sub-polyhedral under-aproximations of the systems of constraints resulting from affine scheduling problems. We propose a sub-polyhedral scheduling technique using (Unit-)Two-Variable-Per-Inequality or (U)TVPI Polyhedra. This technique relies on simple polynomial time algorithms to under-approximate a general polyhedron into (U)TVPI polyhedra. We modify the state-of-the-art PLuTo compiler using our scheduling technique, and show that for a majority of the Polybench (2.0) kernels, the above under-approximations yield polyhedra that are non-empty. Solving the under-approximated system leads to asymptotic gains in complexity, and shows practically significant improvements when compared to a traditional LP solver. We also verify that code generated by our sub-polyhedral parallelization prototype matches the performance of PLuTo-optimized code when the under-approximation preserves feasibility.

Categories and Subject Descriptors D.3.4 [*Programming Languages*]: Processors—Compilers, Optimization

General Terms Approximations, Complexity, Scheduling, Optimization, Performance

Keywords Approximation Algorithms, Complexity Theory, Compiler Optimizations, Parallelism, Loop Transformations, Affine Scheduling, Optimization, Geometric Algorithms

1. Motivation

Polyhedral compilation is well established as the most effective framework to reason with loop programs. At its heart lie the representation of loop nests as polyhedra, and the search for their semantics-preserving loop transformations. In general, semantics preservation amounts to the *feasibility of rational polyhedra*, while performance and profitability are related to *optimization on rational polyhedra*. This simplicity is both the strength as well as the weakness of polyhedral compilation. Strength because polyhedral abstractions are extremely powerful to analyze loops, encoding multitudes of useful transformations, with the ability to automatically parallelize and tile them on newer architectures being one of the main practical applications. And weakness because the richness of polyhedra itself becomes an obstacle due to the complexity of its core algorithms. This latter issue creates a practical challenge when programs of non-trivial size are being compiled. Our view is that using approximations of polyhedra alleviates and provides a solution for this scalability challenge.

In a previous paper [53], we proposed different directions for sub-polyhedral compilation, where approximations of general convex polyhedra could be used so that the problems that are being solved by polyhedral compilers can be made scalable with worst-case polynomial time guarantee. This was followed by the introduction of sub-polyhedral scheduling in [54], where we proposed using (U)TVPI sub-polyhedral under-approximations to reduce the complexity of affine scheduling.

In the current paper, we build from the above works and make solid progress towards polyhedral schedulers that are intrinsically scalable in the program size. Our techniques rely on strongly polynomial algorithms, i.e., whose time complexity is polynomial only in the number of integers in the input and not on the bit-size of the input encoding. For example, we are able to substitute linear programming with min-cost flow and shortest paths problems, which are solvable using graph theoretic methods like the well known Bellman-Ford algorithm.

Our method applies to latency and depth minimization approaches like Feautrier's algorithm [25], as well as methods centered on loop tiling like Bondhugula et al.'s PLuTo [13, 29].

1.1 Polyhedral compilation

Affine scheduling [21] now is a part and parcel of many compilers which aspire to compile for parallel architectures (GCC, LLVM, IBM XL, Reservoir Labs R-Stream). The seminal work of Feautrier [25] opened the avenue of constraint-based affine transformation methods, building on the affine form of the Farkas lemma. This approach has been refined, extended and applied in many directions. To cite only two recent achievements at the two extremes of the complexity spectrum: the tiling-centric PLuTo algorithm of Bondhugula et al. [13] extending the Forward Communication Only (FCO) principle of Griebl et al. [29] for coarse-grain parallelization, and the complete, convex characterization of Vasilache [55] and decoupled exploration heuristic of Pouchet et al. [39]. Much progress has been made in the understanding of the theoretical and practical complexity of polyhedral compilation problems. Nevertheless, when considering multidimensional affine transformations, none of these are strongly polynomial in the size of the program. The lowest complexity heuristics such as PLuTo are reducible to linear programming, which is only weakly polynomial, its traditional Simplex implementation being associated with

POPL'13, January 23–25, 2013, Rome, Italy.

large memory requirements and having a worst-case exponential complexity.

1.2 Unscalability of a current scheduler

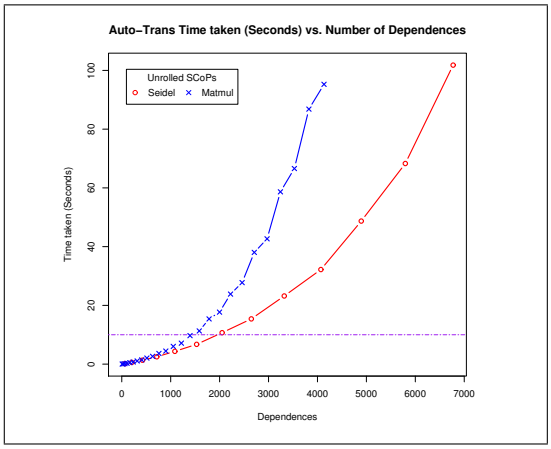

Figure 1. Unscalability for Large Loop Programs

In this section, we show an example of unscalability of current methods in PLuTo. We have artificially unrolled two typical kernels from PolyBench [40], `matmul` and `seidel`, by a variable number of times so as to increase the number of dependences in the loop nests. We have also enclosed the unrolled loops in two "time loops", mimicking the behavior of a scientific computing kernel. The above transformations induce thousands of dependences in the input program. The compilation times are shown in Figure 1. with `matmul` in blue (crosses) and `seidel` in red (circles).

We checked that the compilation time (`auto_trans` time of PLuTo) increases in a roughly n^5 complexity in the number of statements in the system. The rest of the modules of PLuTo—in particular, dependence analysis and code generation (CLooG)—took significantly less than the above.

It may seem that the above unrolling based method is an artificial way to induce unscalability, with inlining being a better candidate for the same in real world benchmarks. While unrolling is much simpler to simulate, limitations of current infrastructures do not provide a platform to study the asymptotic time complexity associated with code size increases associated with inlining. Hence, the above examples could only be taken as representatives of the unscalability problem. But it should also be remembered that when discussing the solution time with respect to the input code size increase, the number of constraints in the overall LP problem is linear in the number of dependences in the input code. So, a method like the above which gives a representative sample to increase the size of the LP program is not a limitation.

Further, in current benchmarks for loop nest optimization—like the PolyBench—the range of dependences is in tens, and it can arguably be said that presently there exists no scalability problem like in the above artificial examples. But, polyhedral compilers will soon face such large problems, arising from aggressive interprocedural optimization, domain-specific program generation, or simply as the applicability restrictions continue to be lifted [10, 51]. In addition, there could be a restriction of the time limit in just-in-time compilers that would further exacerbate the scalability problem, such as just-in-time applications of LLVM/Polly [30]. Note that IBM XL, LLVM, and R-Stream have schedulers similar to PLuTo.

In the following, we aim for lower complexity feasibility and optimization algorithms, with worst-case strongly polynomial bounds, and closer to n^2 time complexity for large-scale and/or just-in-time compilation applications.

1.3 Contributions

In this paper, we make the following contributions:

- We show that state-of-the-art parallelization and affine scheduling heuristics such as PLuTo can be adapted to (U)TVPI subpolyhedra, thereby reducing their algorithmic complexity.
- Using elementary polyhedral concepts, we present a simple and powerful framework (an approximation scheme) which can be used for designing Under-Approximation (UA) algorithms of general convex polyhedra, linearizing the UA problem.
- We evaluate these methods by integrating them into PLuTo. We show that for a significant percentage of Farkas-polyhedra arising from a wide range of test cases from affine scheduling, the (U)TVPI-UAs proposed above are precise enough to preserve feasibility. We show that our approximations when solved with a Bellman-Ford algorithm show considerable improvement in running time over a well established Simplex implementation. Further, we show that preliminary integration of the above UA polyhedra into PLuTo yields code in most cases that does not suffer significant increase in execution time.
- We show how our framework is general enough to extend to various other problems in compiler scheduling, either within the affine transformations or beyond.

The paper is structured as follows. Section 2 introduces TVPI and UTVPI sub-polyhedra. Section 3 introduces polyhedral scheduling, our method to tackle with the scalability problem, and the mathematical framework for the linearization of the under-approximation problem and for establishing its correctness. Section 4 proposes simple algorithms that under-approximate a general convex polyhedron into a (U)TVPI polyhedron. Section 5 discusses the theoretical and practical implications of the above algorithms. In Section 6, we discuss the various methods for having better control of feasibility in the presence of multiple polyhedra. In Section 7 we discuss the results of implementing these algorithms in PLuTo. In Section 8 we discuss extensions and related work, and we conclude in Section 9.

2. Sub-Polyhedra: TVPI and UTVPI

In this section, we briefly cover some basics of TVPI and UTVPI approximations of polyhedra. A more extensive discussion on these, including other flavors of sub-polyhedra as used by the static analysis community can be found in our earlier work [53]. For a polyhedron described in constraint form, let m be the number of inequalities, n be the number of variables and B the upper bound on the absolute value of the coefficients describing the system.

2.1 TVPI sub-polyhedra

In TVPI polyhedra, each constraint is of the form: $ax_i + bx_j \leq c$ $a, b, c \in \mathbb{Q}$. TVPI are obviously closed under projection, and hence many algorithms on geometric operations that are developed for planar polyhedra (polygons) are directly applicable to general TVPI, giving rise to simple algorithms with low complexity. Furthermore, the dual of a TVPI program is a generalized min-cost flow problem, which can be solved using graph algorithms. So, the linear programming community has been interested in TVPI polyhedra because it can be dealt with strongly polynomial time algorithms.

Application of graph theory to linear programming using TVPI systems was pioneered by Shostak [46]. Aspvall and Shiloach [4] showed the polynomiality of the feasibility problem of TVPI-LP

formulations by introducing a unique strongly polynomial time procedure that can be used to decide the range of a particular variable with respect to a given constant. This latter procedure is a Bellman-Ford style propagation of values assigned to variables through inequalities in the system, and is the heart of all subsequent algorithms in the TVPI literature. The following result by Wayne [58] is the best to date for the TVPI optimization problem:

Lemma 2.1.1 [LP optimization on TVPI] Linear programming optimization on TVPI systems can be solved in $\mathcal{O}(m^3n^2 \log m \log B)$ worst case time.

It is well known that for general polyhedra, the optimization and the feasibility problems have the same weakly-polynomial time hardness. But it is interesting to note that till date, they have different complexities on TVPI systems. The feasibility problem on TVPI systems has lower complexity than the above weakly polynomial time result by Wayne on the optimization problem. Network flow based ("combinatorial") strongly polynomial time algorithms for the feasibility problem were given by Cohen and Megiddo [16]. Hochbaum and Naor [32] showed that feasibility of TVPI polyhedra can be determined in strongly polynomial time:

Lemma 2.1.2 [Feasibility on TVPI] Feasibility of TVPI systems can be solved in $\mathcal{O}(mn^2 \log m)$ worst case time.

The above nearly cubic time algorithm by Hochbaum-Naor is surprisingly simple. It embeds the mentioned decision procedure of Aspvall-Shiloach into a binary search, along with a selected application of Chernikova on a planar polyhedron. It can be seen that the above result can as well be used to derive strongly polynomial time cubic bounds for Fourier-Motzkin elimination, and for projection of variables from TVPI systems.

TVPI systems have been used for various problems in abstract interpretation and verification [47].

2.2 UTVPI sub-polyhedra (octagons)

Octagons have constraints of the form $ax_i + bx_j \leq c; a, b \in \{0, \pm 1\}, c \in \mathbb{Q}$, and are called so because in 2-dimensions, their geometric shape is octagonal. They are also referred to as Unit Two Variables Per Inequality (UTVPI) because of the nature of their constraints. Since UTVPI $\subset$ TVPI, the complexity bounds of TVPI polyhedra apply to UTVPI polyhedra as well. But, as the dual of the LP formulation of shortest-paths problem has just difference constraints [1] with the form $x_i - x_j \leq c$, general UTVPI systems can be solved with same quadratic complexity as Bellman-Ford, giving the following lemma:

Lemma 2.2 [Feasibility on UTVPI] Feasibility of UTVPI stystems can be solved in $\mathcal{O}(mn)$ worst case time and in $\mathcal{O}(m + n)$ space.

The above decision algorithm can also return a feasibility certificate of the UTVPI polyhedron.

UTVPI polyhedra have successfully been used for various problems in abstract interpretation and program verification [38]. Also, they have well supported implementations, in Apron [34], the Astrée analyzer [12, 17], and in the Parma Polyhedra Library (PPL) [5].

2.3 Sub-polyhedra vs. general (convex) polyhedra

The use of sub-polyhedra by the static analysis community has different requirements than our proposed application to affine scheduling. In the former, they serve as abstract domains and provide operations like union, intersection, projection etc., which are efficiently solved by (U)TVPI polyhedra providing better worst-case complexity. The objective functions in polyhedral scheduling are very simple functionals with unit coefficients, and lexicographic minima. Our (U)TVPI sub-polyhedral approximate solutions are based on the assumption that the objective function itself could be approximated and any valid feasibility certificate is usually enough. In this aspect, sub-polyhedra fare better than general convex polyhedra for which efficient algorithms are usually based on the widely used Simplex [18] algorithm.

The worst-case complexity of LP is known to be (weakly) polynomial in time [44], and the Simplex algorithm, while taking exponential time on worst-case scenario squashed cube inputs (as shown by Klee-Minty), is known to run well in practice [11].

In practice, we are more interested in the average-case complexity of LP, to evaluate the merit of our approximation methods. Let $Z(m, n)$ be the complexity of LP using the Simplex algorithm with m constraints and n variables. Determining typical value of Z is not an easy task, as it is well known to depend on many details of the algorithm, like relative ratio of m and n, pivoting rule, method of exploiting sparsity, and even many implementation details. In this paper however, we will be using the following folklore result which does not count the bit-size complexity: on a *typical* input, $Z = \mathcal{O}((m + n)mn)$ on *average*.

In the above empirical estimate, we assume that the Gaussian elimination steps of the Simplex are very fast, exploiting sparsity, and assuming a linear number $\mathcal{O}(m + n)$ of pivoting steps. Also, the above estimate is practically very accurate for the LP programs we observe, despite the recent invalidation of Hirsch conjecture [43, 61]. Note that advanced algorithmic analyses using novel perturbation-based probabilistic techniques have even been proposed by Spielman and Teng [48] to establish the average-case polynomial running time; but these results go way beyond the empirical estimate we need to evaluate our algorithms.

3. Polyhedral Scheduling and Approximations

In this section, we show how the above classes of sub-polyhedra could be used to help in overcoming the scalability challenge in affine scheduling.

3.1 Polyhedral scheduling and Farkas lemma

Let us first recall some essential notations and results about polyhedral compilation.

The input to any polyhedral scheduling algorithm is a polyhedral dependence graph G, which is a result of a dependence analysis, and is defined to be a multi-graph $G = (V, E)$, where V is the set of statements, and each particular dependence edge $e \in E$ is annotated with a parametrized polyhedron $\mathcal{D}_e$. Each of the constraints of $\mathcal{D}_e$ is affine and involves (I, N) where vectors I and N are the iteration and parameter vectors, respectively.

In Feautrier's algorithm [25], these are converted into a per-dependence edge polyhedron $P_e(\mu, \lambda)$, with μ-variables being the Farkas multipliers that come from domain constraints and λ-variables being the Farkas multipliers that come from dependence constraints. This conversion is done by application of the affine form of the Farkas lemma [44, Corollary 7.1h] given as:

Lemma 3.1 [Affine Form of Farkas's Lemma] Let $\mathcal{D}$ be a nonempty polyhedron defined by p inequalities $\mathbf{a}_k\mathbf{x} + b_k \geq 0$, for any $k \in \{1, \ldots, p\}$. An affine form Φ is non-negative over $\mathcal{D}$ if and only if it is a non-negative affine combination of the affine forms used to define $\mathcal{D}$, meaning:

$$\Phi(\mathbf{x}) \equiv \lambda_0 + \sum_{k=1}^{p} \lambda_k(\mathbf{a}_k\mathbf{x} + b_k);\ \forall k \in [0, p]\ \lambda_k \geq 0$$

The nonnegative values λ_k are called Farkas' multipliers.

In the per-edge Farkas polyhedron $P_e(\mu, \lambda)$, both of the newly created λ and μ variables are called the Farkas multipliers. By putting together all the per-edge Farkas polyhedra, one obtains an

overall Farkas polyhedron $P = \cap_{e \in E} P_e$, which is amenable to Linear Programming. Any rational point that satisfies P is considered a valid schedule.

The above application of the Farkas lemma results in all the constraints in the Feautrier's scheduler, with some additional variables to model the strong/strict satisfaction of dependences at a given dimension of the affine schedule. In PLuTo, a different but conceptually similar method results in a majority of dependence constraints of the same form as Feautrier's.

It has been shown elsewhere [27, 53] that these methods result in an LP problem of size $m \times n \approx (d \cdot |E|) \times (d \cdot |V|)$, where d is the mean depth of the loop nests. Assuming a usual simplex method—whose complexity has been alluded in the previous section—for solving systems with bounded d leads to a close to $\mathcal{O}(|E|^2|V|) = \mathcal{O}(|V|^5)$ asymptotic complexity (not counting the bit-size complexity, and assuming $|E| = \mathcal{O}(|V|^2)$), closely matching the curves in Figure 1 and leading to unscalability problems.

3.2 Schedule space under-approximation

In this paper, we propose that P be Under-Approximated (UA) to improve the scalability of the scheduling algorithm. This means that instead of searching for an optimal feasible point in P, we search in $P_a = \mathrm{UA}(P)$. The above approximation is legal and only leads to a conservative approximation of losing schedules, though it is well proven [39] that P is highly redundant with respect to schedule points. The overall process has to ensure that the approximation algorithm, as well as the solution finding time be scalable algorithms. We can restrict these requirements further and say that both of these algorithms should have *worst-case strongly polynomial time* running times matching the complexities of (U)TVPI polyhedra introduced in Section 2.

The above approximation can also be done on a per-dependence-edge basis. In this method, the per-dependence-edge Farkas polyhedra P_e are under-approximated, and the solution is found from the overall polyhedron obtained by putting together all the per-dependence UAs. Namely, by doing $P_a = \mathrm{UA}(P) = \cap_{e \in E} \mathrm{UA}(P_e)$. If the above approximation leads to a non-empty polyhedron, then we can find schedule using $\mathrm{UA}(P)$ instead of P.

In the above, we are exploiting the property that each of the individual P_e's of polyhedral compilation are guaranteed to be non-empty [25] directly as a result of dependence analysis. It also helps that each of the per-dependence-edge Farkas polyhedra P_e are comparably much smaller than the overall polyhedron.

In the rest of this section, we define a simple and sound mathematical background to build (U)TVPI approximations of polyhedra, to linearize the problem of finding approximations, and to prove that the algorithms we construct in later sections return valid approximations. For giving these sufficiency conditions, we take two approaches: an intuitive and geometric explanation using duality/polarity in next section, followed by a more direct way using the equivalent Farkas solution.

3.3 Convexity and approximations

Starting with a polyhedron given in constraint form as $P = \{\mathbf{x} | A\mathbf{x} + \mathbf{b} \geq \mathbf{0}\}$, we show that simple approximations of P can be obtained by reasoning about over-approximations of the dual (polar), associated with the transpose matrix $[A \mid \mathbf{b}]^T$. To accomplish the above, we introduce some basic lemmas about convexity and then show how these geometric concepts help reasoning about under-approximations and over-approximations in a unified manner. The reader may refer to standard books for a complete coverage [44, 61]. A more detailed presentation of this material can be found in [52, 54].

3.3.1 Homogenization and conical polarity

In projective geometry, homogenization can be done on polyhedra in $\mathcal{H}$-form—constraint form—or in $\mathcal{V}$-form—vertex form or generator form. The following definition is when P is in $\mathcal{H}$-form.

Definition 3.3.1a [Homogenization] Let $P = P(A, \mathbf{b})$ ($P = \{\mathbf{x} | A\mathbf{x} + \mathbf{b} \geq \mathbf{0}\}$) be a $\mathcal{H}$-polyhedron, then its homogenization is a cone and is also a $\mathcal{H}$-polyhedron:

$$\mathrm{homog}(P) = \left(\begin{pmatrix} A & \mathbf{b} \\ \mathbf{0}^T & 1 \end{pmatrix}, \begin{pmatrix} \mathbf{0} \\ 0 \end{pmatrix} \right) = C(P) \qquad (1)$$

It can be noted that if A is a $m \times n$-matrix (m constraints and n variables), and $\mathbf{b}$ is a $m \times 1$-vector, then the homogenized constraint system C (or $\mathrm{homog}(P)$) is of size $(m + 1) \times (n + 1)$. Note that the constants dimension has become an additional dimension in the $(n + 1)$-dimensional space. We would be referring to this dimension as *homogenizing* dimension and the other dimensions as *non-homogenizing* dimensions. Though the homogenization is a rather trivial process, involving special marking of the dimensions, it however needs to be mentioned because this paper deals with (U)TVPI constraints having at most two non-zero coefficients in the non-homogenizing dimensions.

The following is the definition of polar of a polyhedron:

Definition 3.3.1b [Conical Polarity] For $C \subseteq \mathbb{R}^n$, the polar set is defined by

$$C^* = \{\mathbf{c} \in (\mathbb{R}^n)^* : \mathbf{c}\mathbf{x} \leq 0 \text{ for all } \mathbf{x} \in C\} \subseteq (\mathbb{R}^n)^*$$

From the above definition, K, the polar cone corresponding to P can directly be constructed from $\mathrm{homog}(P)$ in (1), and whose generator vectors are columns of the following matrix:

$$K = \mathrm{cone} \begin{pmatrix} A^T & \mathbf{0} \\ \mathbf{b}^T & 1 \end{pmatrix} \qquad (2)$$

In matricial form, K is of size $(n+1) \times (m+1)$, the transpose of constraint matrix of (1), and can also be written as the following:

$$K = \mathrm{cone} \left\{ \begin{pmatrix} \mathbf{a}_1^T \\ b_1 \end{pmatrix}, \ldots, \begin{pmatrix} \mathbf{a}_j^T \\ b_j \end{pmatrix}, \ldots, \begin{pmatrix} \mathbf{a}_m^T \\ b_m \end{pmatrix}, \begin{pmatrix} \mathbf{0} \\ 1 \end{pmatrix} \right\} \qquad (3)$$

3.3.2 Polarity, (U)TVPI and (U)TCPV

Though a polyhedron can be represented in either of $\mathcal{H}/\mathcal{V}$-forms there is certain naturality in describing the primal in $\mathcal{H}$-form for problems that occur in polyhedral compilation. The polar can be built in linear time in $\mathcal{V}$-form as in (3). But converting the primal to $\mathcal{V}$-form or the polar to $\mathcal{H}$-form is very costly; it involves the Chernikova algorithm which takes exponential time. In the following lemma, we will implicitly be using the above *dual* interpretation, *without* making an actual call to Chernikova.

Lemma 3.3.2 [(U)TVPI and (U)TCPV] For a TVPI-polyhedron, the polar has vertices and rays (generators) which have not more than two non-zero components in the non-homogenizing dimensions. For a UTVPI-polyhedron, the polar has generators which have not more than two non-zero components each in the non-homogenizing dimensions, with them being from $\{-1, +1\}$. We define the polar of a (U)TVPI constraint as *(Unit-)Two-Components-Per-Vector* or *(U)TCPV* vector. Further, we define the polar of a (U)TVPI polyhedron as a *(U)TCPV polyhedron*.

3.3.3 Under-approximation and over-approximation

The following lemma and its corollary are standard results:

Lemma 3.3.3a [Polarity and Inclusivity] For any two cones K_1 and K_2, $K_1 \subseteq K_2 \Leftrightarrow K_1^* \supseteq K_2^*$.

Corollary 3.3.3b [Toggling between OA/UA] $(\mathrm{OA}(K^*))^* \subseteq K$

The above corollary means that by taking the polar of a cone and taking the resultant's Over-Approximation (OA) one obtains a cone whose polar is an UA of the original cone. By using it, the problem of finding a UA of a polyhedron in $\mathcal{H}$-form is reduced to the problem of finding a OA of its polar cone in $\mathcal{V}$-form. More specifically, if we let P_a (respectively K_a) be the approximation of P (respectively K), we have the following:

$$K_a \supseteq K \Leftrightarrow P_a \subseteq P \tag{4}$$

So, the objective in the polar space of finding the TCPV-OA of cone K given in $\mathcal{V}$-form as in (3), is equivalent to finding a cone K_a such that:

$$K \in \mathrm{cone}\,(K_a) \tag{5}$$

By this equation, if each column of K as in (3) can be written as a conical sum of the vectors in K_a, then the approximation remains a valid approximation. If each vector in K_a is a (U)TCPV vector, then the corresponding P_a would be a (U)TVPI approximation of P.

3.4 Approximation scheme for TVPI-UA

In this section, we formulate the sufficient conditions to prove that the UAs we will construct in Section 4 are valid approximations. In fact, with the Farkas lemma in Section 3.1 and the equivalent homogenization-polarity intuition in Section 3.3, we do have all the necessary ammunition to construct an approximation scheme which allows us to built TVPI-UAs of general polyhedra. (UTVPI-UAs are simple extensions of the material in this section.)

In the rest of this section, for ease of exposition, we assume that the polyhedron P is 3-dimensional; these are also called 3VPI polyhedra. Generalization to $n > 3$ is straightforward.[1]

Let the j-th column of K, with $1 \leq j \leq m$, be $K^j = (\mathbf{a}_j, b_j)^T = \begin{pmatrix} a_{j,1} & a_{j,2} & a_{j,3} & b_j \end{pmatrix}^T$. Then we have

$$K^j = \begin{pmatrix} a_{j,1} \\ a_{j,2} \\ a_{j,3} \\ b_j \end{pmatrix} \in \mathrm{cone} \begin{pmatrix} t_1^{j,1} & t_2^{j,1} & 0 \\ t_1^{j,2} & 0 & t_3^{j,2} \\ 0 & t_2^{j,3} & t_3^{j,3} \\ p_1^j & p_2^j & p_3^j \end{pmatrix} = \mathrm{cone}(T^j) \tag{6}$$

where the (t, p)-variables constitute the elements of the unknown T-matrix and are to be found out.

It can be noticed that each column of the matrix T^j is a TCPV vector and hence is a TVPI constraint in the primal space. The above is what we call an *approximation scheme*. In this scheme:

Definition 3.4 [Approximation scheme] A non-TVPI constraint $a_{j,1}x_1 + a_{j,2}x_2 + a_{j,3}x_3 + b_j \geq 0$ in the original system is replaced by the set of constraints $\mathrm{UA}(x_1, x_2, x_3) = \{t_1^{j,1}x_1 + t_1^{j,2}x_2 + p_1^j \geq 0; t_2^{j,1}x_1 + t_2^{j,3}x_3 + p_2^j \geq 0; t_3^{j,2}x_2 + t_3^{j,3}x_3 + p_3^j \geq 0\}$.

In the above scheme, every column vector of K which has more than two non-zero components is replaced by a set of TCPV vectors such that the original vector remains in the conical sum of the replacements. The conical combination of the replacement vectors would hence be an OA of the original vector K^j satisfying (5). The above scheme remains valid as long as the (t, p) variables and the (a, b) constants satisfy the convexity requirement. One way for ensuring the same is by making the (t, p)-variables satisfy the following additional constraints, which we would be referring to as *context constraints* for reasons that will be exposed later:

$$\{t_1^{j,1} + t_2^{j,1} = a_{j,1}; t_1^{j,2} + t_3^{j,2} = a_{j,2}; t_2^{j,3} + t_3^{j,3} = a_{j,3}; \\ p_1^j + p_2^j + p_3^j = b_j\} \tag{7}$$

If such a set of T matrices can be found for each non-TCPV vector of K, then we have $K_a = \{T^1, T^2, \ldots, T^m\}$ and the resultant TCPV approximation would be $K \in \mathrm{cone}\,\{T^1, T^2, \ldots, T^m\}$. We have the following theorem:

Theorem 3.4 [TVPI and UA] P_a is a valid TVPI-UA of the original Polyhedron P.

Proof We employ the affine form of Farkas lemma, with the premise of non-emptiness of P guaranteed because of the per-constraint method. For the UA to be proper, the affine form corresponding to the original constraint $\mathbf{a}_j\mathbf{x} + b_j$ should be expressible as a positive sum of the replacement TVPI vectors: $a_{j,1}x_1 + a_{j,2}x_2 + a_{j,3}x_3 + b_j \equiv \lambda_0^j + \lambda_1^j(t_1^{j,1}x_1 + t_1^{j,2}x_2 + p_1^j) + \lambda_2^j(t_2^{j,1}x_1 + t_2^{j,3}x_3 + p_2^j) + \lambda_3^j(t_3^{j,2}x_2 + t_3^{j,3}x_3 + p_3^j)$, where the λ-multipliers satisfy $\lambda_0^j, \lambda_1^j, \lambda_2^j, \lambda_3^j \geq 0$. Pairwise matching for each of the x-variables yields:

$$\{a_{j,1} = \lambda_1^j t_1^{j,1} + \lambda_2^j t_2^{j,1}; a_{j,2} = \lambda_1^j t_1^{j,2} + \lambda_3^j t_3^{j,2}; \\ a_{j,3} = \lambda_2^j t_2^{j,3} + \lambda_3^j t_3^{j,3}; b_j = \lambda_0^j + \lambda_1 p_1^j + \lambda_2^j p_2^j + \lambda_3^j p_3^j\} \tag{8}$$

Setting $\lambda_1^j = \lambda_2^j = \lambda_3^j = 1$ yields the context constraints defined in (7) proving the validity of the approximation, except for the additional λ_0^j in the homogenizing-dimension-matching. Setting $\lambda_0^j = 0$ is safe, and we will see later that it actually contributes to the quality of the approximation. ■

The problem remains to find such a set of (t, p) variables satisfying the above context constraints (7), so that the approximation remains valid. It can be seen that searching for the (t, p) variables directly could lead to a non-linear (quadratic) formulation. Even searching for t-variables that satisfy the above constraints turns out to be non-linear. But, as they are existentially quantified, the equations can be solved using advanced quantifier elimination techniques like [49], which are well known to be unscalable beyond small inputs and certainly not polynomial.

The next section proposes some linearizations of the above mentioned approximation scheme, so that the approximation algorithms remain scalable. This is done first as a heuristic where both (t, p)-variables are arbitrarily fixed, then as a more careful method where only the t-variables are fixed, while the p-variables are found by an LP formulation.

4. TVPI and UTVPI UA Algorithms

In this section, we use the framework developed in earlier section and develop worst-case polynomial time algorithms for obtaining (U)TVPI under-approximations of Polyhedra.

4.1 The median method for TVPI-UA

In this section, we introduce a simple, per-constraint, strongly polynomial heuristic, using the framework developed in Section 3.4. The main idea of this approximation is (5), saying that the original vector can be approximated by any set of the replacement TCPV vectors, as long as the former remains in the cone of the latter.

Definition 4.1 [Median method: 3-d case] The inequality $ax + by + cz + 1 \geq 0$ can be approximated by the set of inequalities $\{ax + by + \frac{2}{3} \geq 0; ax + cz + \frac{2}{3} \geq 0; by + cz + \frac{2}{3} \geq 0;\}$.

[1] It is also possible to reduce a general polyhedron into an equivalent 3VPI one. According to Shostak [46], the transformation has been suggested by R. Tarjan. It is similar to the reduction of an arbitrary boolean satisfiability ("SAT") problem to a 3SAT problem. This transformation is not an approximation, which is the subject of this paper.

Geometric intuition for the above derives from the observation of the polar space, where the above approximation is the following:

$$\begin{bmatrix} a \\ b \\ c \\ 1 \end{bmatrix} \in \text{cone} \begin{pmatrix} \frac{1}{2}a & \frac{1}{2}a & 0 \\ \frac{1}{2}b & 0 & \frac{1}{2}b \\ 0 & \frac{1}{2}c & \frac{1}{2}c \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix}$$

The intuition for the above approximation comes from (6). The values of the t-variables and p-variables need to satisfy (7). As explained earlier, the t-variables have to be instantiated a priori to avoid solving a non-linear problem. The method in this section fixes the p variables also in a heuristic manner by dividing the available "budget" in the homogenizing dimensions equally between the values in the homogenizing dimensions of the replacement TCPV vectors. We call it the median method because the original vector is the median of the replacement vectors in the polar space.

General n-d case The above can be easily generalized to n-d polyhedra. Let s be the sparsity of a polyhedron, i.e., the number of non-zero variables (outside the homogenizing dimension) for a specific constraint, with $1 \leq s \leq n$. Let $q = \binom{s}{2} = s(s-1)/2$ and let $r = s - 1$. The resultant set of TCPV vectors corresponding to the particular constraint are $n + 1$ dimensional, with cardinality q. The coefficients of the non-homogenizing dimensions are divided by r, while the homogenizing dimension is uniformly divided by q.

In each of the cases, it can be seen that the approximation being proposed is a TCPV approximation, making its polar a TVPI approximation. It can also be verified using the construction given in the earlier sections and equations (5) and (6) that the UA proposed in each case is a valid approximation.

Example 1 Let the input system be the triangular pyramid (3d-simplex) $\{x, y, z \geq 0; x + y + z \leq 1\}$. Only the inequality $x + y + z \leq 1$ is not TVPI and is approximated by the set of inequalities $\{x+y \leq \frac{2}{3}; x+z \leq \frac{2}{3}; y+z \leq \frac{2}{3}\}$. It can be seen that the approximation is a non-empty TVPI system (it is a UTVPI system), with vertices $\{(0,0,0), (\frac{2}{3},0,0), (0,\frac{2}{3},0), (0,0,\frac{2}{3}), (\frac{1}{3},\frac{1}{3},\frac{1}{3})\}$, each of which are inside the vertices of the original system $\{(0,0,0), (1,0,0), (0,1,0), (0,0,1)\}$. (The above example is similar [21, 25] to the Farkas system induced in the compilation of the matrix-vector-product: $\mathtt{s[i]} += \mathtt{A} * \mathtt{x[i]}$.) □

With reference to the choice of Farkas multiplier λ_0 in Theorem 3.4, it can be seen that setting $\lambda_0 = 0$ in the above example has the advantage that the three replacement TVPI hyperplanes intersect exactly on the original non-TVPI hyperplane. Any other strictly positive choice for λ_0 would mean that the point of intersection would lie strictly inside the positive half-space of the original and thus giving rise to an UA which is in a way less effective.

Example 2 Let the input system be the skewed triangular pyramid $\{x, y, z \geq 0; 1000x + 100y + 10z \leq 1\}$. Only the inequality $1000x + 100y + 10z \leq 1$ is not TVPI and is approximated by the set of inequalities $\{1000x + 100y \leq \frac{2}{3}; 1000x + 10z \leq \frac{2}{3}; 100y + 10z \leq \frac{2}{3}\}$. It can be seen that the resultant approximation is a non-empty TVPI system with vertices: $\{(0,0,0), (\frac{1}{1500},0,0), (0,\frac{1}{150},0), (0,0,\frac{1}{15}), (\frac{1}{3000},\frac{10}{3000},\frac{100}{3000})\}$, each of which are inside the vertices of the original system, which are $\{(0,0,0), (\frac{1}{1000},0,0), (0,0,\frac{1}{10}), (0,\frac{1}{100},0)\}$. □

It can be seen that the median method is simple and easy to implement, but does not have any guarantee of ensuring that the resultant approximation is non-empty. In the next section, we will generalize this method to formulate a parametrized approximation and formulate an LP problem to find the approximation.

4.2 LP-based parametrized TVPI approximation

To ease the exposition, we will primarily deal with 3-dimensional polyhedra again; higher dimensional extensions are straightforward.

The median method can easily be extended by defining the approximation as a *parametrization* on the values in the homogenizing dimension entries: as $\text{UA}_2(\mathbf{a}_j\mathbf{x} \geq b_j) = \{(\mathbf{x}, \mathbf{p}^j) | a_{j,1}x_1 + a_{j,2}x_2 \geq 2p_1^j; a_{j,1}x_1 + a_{j,3}x_3 \geq 2p_2^j; a_{j,2}x_2 + a_{j,3}x_3 \geq 2p_3^j; \sum \mathbf{p}^j = b_j; \}$ where the values of the 3-dimensional $\mathbf{p}^j$-vector are unknown and have to be found out.

In the above approximation, it can be observed that the coefficients in the non-homogenizing dimensions (t-variable values) have been fixed, much similar to their choice in the median method. But, the coefficients in the homogenized dimension (p-variable values) are unknown and have to be found out. The context constraint $\sum \mathbf{p}^j = p_1^j + p_2^j + p_3^j = b_j$ is not arbitrary. It is determined by the choice of the multipliers for the t-variables so that the K^j vector is in the convex-sum of T^j, as given in (6).

The resultant system is $\mathcal{S}_{HD}(\mathbf{x}, \mathbf{p}^j) = \{(\mathbf{x}, \mathbf{p}^j) | \mathbf{a}_1\mathbf{x} \geq b_1; \ldots; \mathbf{a}_{j-1}\mathbf{x} \geq b_{j-1}; \text{UA}_2(\mathbf{a}_j\mathbf{x} \geq b_j); \mathbf{a}_{j+1}\mathbf{x} \geq b_{j+1}; \ldots; \mathbf{a}_m\mathbf{x} \geq b_m; \}$. In the following discussion, we show that the higher dimensional system $\mathcal{S}_{HD}(\mathbf{x}, \mathbf{p}^j)$ can be interpreted in two ways, a geometric and an algorithmic ways, each having its own merits.

4.2.1 A parametrized approximation

$\mathcal{S}_{HD}$ can geometrically be considered as a parametrized approximation, with $\mathbf{p}^j$ being the parametric vector and the context constraint $\sum \mathbf{p}^j = b_j$ considered as the parametric context. When the values of the vector $\mathbf{p}^j$ are known, then the system:

$$\mathcal{S}_2(\mathbf{x}) = \mathcal{S}_{HD}(\mathbf{x}, \mathbf{p}^j) | \mathbf{p}^j \tag{9}$$

is a non-parametrized LP problem, which can be tested for feasibility in the usual $\mathbf{x}$-variables. Since the context constraint $\sum \mathbf{p}^j = b_j$ is respected, it follows from the proofs of earlier sections that $\mathcal{S}_2(\mathbf{x})$ is a proper approximation of $\mathcal{S}(\mathbf{x})$. If the value of the vector $\mathbf{p}^j$ is not known, $\mathcal{S}_2(\mathbf{x})$ can be considered as a parametrized approximation of $\mathcal{S}(\mathbf{x})$. Note that the context constraint is only on the p-variables, while the t-variables have been assigned a fixed value.

4.2.2 An LP formulation

Algorithmically we can consider $\mathcal{S}_{HD}(\mathbf{x}, \mathbf{p}^j)$ to be a non-parametric LP system with unknown variables $(\mathbf{x}, \mathbf{p}^j)$. Such an interpretation is possible because there exist no non-linear terms in the definition of $\mathcal{S}_2(\mathbf{x}, \mathbf{p}^j)$ and it is only the feasibility of the approximation that is interesting. Supposing the system $\mathcal{S}_{HD}(\mathbf{x}, \mathbf{p}^j)$ is solved for feasibility, with the unknown variables vector as $(\mathbf{x}, \mathbf{p}^j)$, and a valid assignment for the values of both $\mathbf{x}$-variables as well as the $\mathbf{p}^j$-variables is found, then the system $\mathcal{S}_2(\mathbf{x})$ as given by (9) can be considered an approximation of the system $\mathcal{S}(\mathbf{x})$, as long as the $\mathbf{p}^j$-variables satisfy the constraint $\sum \mathbf{p}^j = b_j$. On the other hand, $\mathcal{S}_{HD}(\mathbf{x}, \mathbf{p}^j)$ is a higher dimensional system than $\mathcal{S}(\mathbf{x})$ and hence cannot be considered as an approximation of the latter. It is an intermediate form useful for algorithmic purposes.

Example 3 Here is a reduced example from Banerjee's book [8]. $\mathcal{S}(x, y, z) = \{(x, y, z) \mid -z + 3 \geq 0; x - z \geq 0; -y + z - 1 \geq 0; -x + y + z + 1 \geq 0\}$. It is clearly not a TVPI system as the fourth constraint is non-TVPI. The median method applied to this constraint yields the system $\mathcal{S}_1(x, y, z) = \{(x, y, z) | -z + 3 \geq 0; x - z \geq 0; -y + z - 1 \geq 0; -x + y + \frac{2}{3} \geq 0; -x + z + \frac{2}{3} \geq 0; y + z + \frac{2}{3} \geq 0\}$, which turns out to be an empty system. On the other hand, the method in this section would lead to the following higher dimensional system: $\mathcal{S}_{HD}(x, y, z, p_1, p_2, p_3) = \{(x, y, z, p_1, p_2, p_3) | -z+3 \geq 0; x-z \geq 0; -y+z-1 \geq 0; -x+$

$y+2p_1 \geq 0; -x+z+2p_2 \geq 0; y+z+2p_3 \geq 0; p_1+p_2+p_3 = 1\}$, where the variables p_1, p_2, p_3 are additional context variables and the last constraint $p_1 + p_2 + p_3 = 1$ is a context constraint. When system $\mathcal{S}_{HD}$ is solved for a feasible point, and the set of p-variables that are obtained as solution $(\frac{1}{2}, 0, \frac{1}{2})$ are substituted, we obtain $\mathcal{S}_2(x, y, z) = \{(x, y, z)| - z + 3 \geq 0; x - z \geq 0; -y + z - 1 \geq 0; -x + y + 1 \geq 0; -x + z \geq 0; y + z + 1 \geq 0\}$. The reader can verify the satisfiability of the two approximations $\mathcal{S}_1$ and $\mathcal{S}_2$. □

The above method involves only one call to a standard LP solver. The disadvantage is that the system $\mathcal{S}_{HD}(\mathbf{x}, \mathbf{p})$ has $n + \binom{\|\mathbf{a}_j\|}{2} = n + \|\mathbf{a}_j\|(\|\mathbf{a}_j\| - 1)/2 \approx \mathcal{O}(n + \|\mathbf{a}_j\|^2)$ dimensions, where $\|\mathbf{a}_j\|$ is the number of non-zero elements in the vector $\mathbf{a}_j$. As the method involves a call to an LP solver, its theoretical cost is not strongly polynomial time.

4.3 Multiple constraint LP formulations

When there exist multiple non-TVPI constraints in $\mathcal{S}(\mathbf{x})$, each one of them has to approximated to find a TVPI approximation of the polyhedron. Let m_k be the number of non-TVPI constraints in $\mathcal{S}(\mathbf{x})$, with $m_k \leq m$. Without loss of generality, we can assume that the constraints have been ordered such that the non-TVPI constraints come first, followed by the TVPI constraints. This means that the constraints of $\mathcal{S}(\mathbf{x})$ are $\{1, \ldots, m_k, \ldots, m\}$, with the constraints $\{1, \ldots, m_k\}$ being non-TVPI constraints and the constraints $\{m_k + 1, \ldots, m\}$ being TVPI constraints.

4.3.1 One-shot method

A straightforward way in which one can find a TVPI approximation of the above non-TVPI system $\mathcal{S}(\mathbf{x})$ is to construct a system $\mathcal{S}_{HD}(\mathbf{x}, \mathbf{p}^1, \ldots, \mathbf{p}^{m_k})$ in which all the non-TVPI constraints in $\mathcal{S}(\mathbf{x})$ are approximated using the scheme described in Section 4.2.2. This system can be solved using an LP formulation in variables as $\{\mathbf{x}, \mathbf{p}^1, \ldots, , \mathbf{p}^{m_k}\}$. An approximation of $\mathcal{S}(\mathbf{x})$ could be found as $\mathcal{S}_{HD}(\mathbf{x}, \mathbf{p}^1, \ldots, \mathbf{p}^{m_k})|\mathbf{p}^1, \ldots, \mathbf{p}^{m_k}$. It can easily be shown that the latter system is a proper approximation as long as the context constraints $\sum \mathbf{p}^1 = b_1, \ldots, \sum \mathbf{p}^{m_k} = b_{m_k}$ are respected.

But, finding the approximations of all the non-TVPI constraints simultaneously in the above fashion would lead to a large LP system. $\mathcal{S}_{HD}(\mathbf{x}, \mathbf{p}^1, \ldots, \mathbf{p}^{m_k})$ could have up to $n + \sum_{l=1}^{m_k} \binom{\|\mathbf{a}_l\|}{2} = n + \sum_{l=1}^{m_k} \|\mathbf{a}_l\|(\|\mathbf{a}_l\| - 1)/2$-dimensions which could be as large as $\mathcal{O}(n + \widehat{\|\mathcal{S}_{m_k}\|}^3)$, with $\widehat{\|\mathcal{S}_{m_k}\|}$ being the average number of non-zero coefficients in the non-TVPI constraints in $\mathcal{S}(\mathbf{x})$.

4.3.2 Iterative methods

We can also iterate the above process described in Section 4.2.2 for each of the m_k non-TVPI constraints in $\mathcal{S}(\mathbf{x})$ on an iterative basis. Clearly, there could be choice in the methods in whether the original system is being updated with the approximation constraints of each non-TVPI constraint or not. We refer to the case when the original system is immediately updated as the *incremental* method. We refer to the case when the approximations of all non-TVPI constraints are found by constructing LP formulations on the same LP system as the *independent* method.

It could be noticed that each of the above methods involves multiple LP calls: one for each non-TVPI constraint $\mathcal{S}(\mathbf{x})$. This means making upto $\mathcal{O}(m)$ LP calls in total for building the approximation system. But, the dimension of each of the LP systems is in the order of $\mathcal{O}(n + \widehat{\|\mathcal{S}_{m_k}\|}^2)$, which is much more reasonable than the previous one-shot formulation.

4.4 Per constraint UTVPI-UA of TVPI

Let us sketch a simple per-constraint algorithm that takes a TVPI constraint and returns its UTVPI under-approximation.

From Lemma 3.3.2, a vector in the polar space needs to be TCPV, and should have equal magnitude components in the non-homogenizing dimensions for the original to be UTVPI. So, the intuition for this algorithm is similar to the TCPV-OA, namely that reasoning about the original TCPV vectors in the polar space and computing a set of UTCPV vectors which OA the original TCPV vector resulting in an OA.

Suppose the TCPV vector has components as (a, b, p) in the (x_i, x_j, x_0) dimension (with x_0 being the homogenizing dimension), then we can replace it with two UTCPV vectors. The replacement has to just take care of the fact that the new UTCPV vectors are such that the original vector is in the conical sum of the replacements. As the case when $a = b$ means that the vector is already a UTCPV vector, the other cases can be handled in the following way: $\begin{cases} a > b: & \text{cone}\{(b, b, \frac{p}{2})^T, (a - b, 0, \frac{p}{2})^T\} \\ a < b: & \text{cone}\{(a, a, \frac{p}{2})^T, (0, b - a, \frac{p}{2})^T\} \end{cases}$. In either of the above cases, it can be noticed that the first vector is a UTCPV vector and the second is an interval vector. The first case is illustrated in Figure 2.

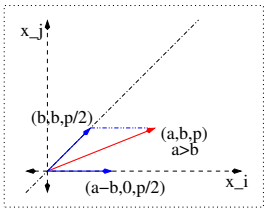

Figure 2. TCPV to UTCPV approximation

Lemma 4.4 [Validity of the above UTVPI approximation] Given a TVPI constraint, the above method returns a valid UTVPI-UA of the constraint.

Proof The geometric proof derives from the observation that the sum of the corresponding UTCPV replacement vectors is the original vector. Hence every vector in the polar space that is reachable by the original vector is reachable by these replacement vectors, meaning that they give an OA in the polar space. In the primal space, the replacements necessarily give an UA (Cor. 3.3.3b). ■

Example 4 Let $P^{\mathcal{V}} = \text{conv}\{(1, 1), (4, 1), (1, 2)\}$, with the $\mathcal{H}$-form of P being $P^{\mathcal{H}} = \{1 \leq x \leq 4; 1 \leq y \leq 2; x + 3y \leq 7\}$. It can be seen that P is a TVPI system, but not a UTVPI system, as the third constraint is a non-UTVPI constraint. The UTVPI approximation by the above method is $\{1 \leq x \leq 4; 1 \leq y \leq 2; x + y \leq \frac{7}{2}; 2y \leq \frac{7}{2}\}$ which can be seen to be non-empty. □

The cost of finding the UA is linear and the method is simple to implement, just like the median method mentioned in Section 4.1. Just like that method, this method does not have any guarantee that the approximated system is non-empty.

4.5 LP based parametrized UTVPI approximation

This approximation is similar to the parametrized TVPI approximation of Section 4.2, in the sense of searching for p variables instead of t's. We are not covering it for lack of space.

5. Metrics and Discussion

Let us now study the size of the new system, the complexity of the conversion, the overall complexity of finding a solution, and discuss some fundamental and methodological limitations of our approach.

5.1 Sizes

Let the original matrix be of size $m \times n$: m constraints and n variables with m_k and m_t being the number of non-TVPI and

TVPI constraints respectively. Also let the overall sparsity factor of the system be $\hat{s}$, which means that for a constraint in the input system, on average $\hat{s}$ variable elements are non-zero. It will be seen in Sections 7.1 that for practical purposes, $\hat{s}$ is a small constant (little more than 4) and relatively independent of n.

TVPI-UA For a system described as above, each of the TVPI-UA methods (given in Sections 4.1 and 4.2) replace a non-TVPI constraint $\mathbf{a}_j$ with the same number of $\binom{\|\mathbf{a}_j\|}{2} = \|\mathbf{a}_j\|(\|\mathbf{a}_j\|-1)/2$ TVPI constraints. Doing the above process for each of the m_k non-TVPI constraints creates an approximated TVPI system of size $m_a \times n$, where $m_a = m_t + \widehat{\|\mathcal{S}_{m_k}\|}(\widehat{\|\mathcal{S}_{m_k}\|} - 1)m_k/2$. It can be assumed that the new system is approximately of the size $\hat{s}^2 m \times n$. In the rare case that $\hat{s} = n$ (which means that the constraint matrix does not have any zero entries) then the size of the resultant TVPI constraint system is $\frac{n(n-1)m}{2} \times n$ which is of the order of $n^2 m \times n$.

UTVPI-UA For UTVPI approximation given in Section 4.4, we have the same order of complexity of additional constraints as in the TVPI-UA case, though double in number.

Constraint graph The Bellman-Ford algorithm constructs a constraint graph, or incidence matrix, of nearly twice the size of the input UTVPI system before solving it: a $m \times n$ input system results in $(2n + 1)$ nodes and $(2m + 2n)$ edges, due to simple transformations of addition of positive and negative forms of each variable [38, Section 2.2], and addition of a pseudo-source-vertex for the shortest-paths problem [1]. The former is a relaxation of UTVPI into difference constraints and is exact for rational points [38], though some (odd) integer points in the original are lost.

5.2 Complexity of conversion

Both the median method of TVPI approximation covered in Section 4.1 and the UTVPI approximation covered in Section 4.4 are strongly polynomial time algorithms as they do not use any LP call when constructing the approximation. The parametric approximation covered in Section 4.2.2 is a weakly polynomial time algorithm for it needs to solve an LP problem for finding the homogenizing dimension values.

For the case of multiple constraints, complexity of conversion is one LP call for the one-shot algorithm of Section 4.3.1, and one LP call per non-TVPI constraint for both incremental and independent algorithms. Each of the above numbers are weakly polynomial time, but are worst-case times nonetheless.

5.3 Complexity of finding a solution

For the TVPI approximation, as the worst-case complexity of Hochbaum-Naor is $\mathcal{O}(mn^2 \log m)$ [32], applying it to the resultant approximated system would lead to $\mathcal{O}(\hat{s}^2 mn^2 \log \hat{s}m)$ time with constraint sparsity $\hat{s}$, and $\mathcal{O}(n^4 m \log nm)$ in the unlikely case when $\hat{s} = O(n)$.

For the UTVPI approximation, as the theoretical worst-case complexity of solving UTVPI systems is $\mathcal{O}(mn)$ (if we use the traditional Bellman-Ford algorithm on the difference constraints), the corresponding complexities would be $\mathcal{O}(\hat{s}^2 mn)$, and $\mathcal{O}(n^3 m)$ respectively. We will see in Section 7.1 empirical evidence suggesting that TVPI constraints are *always* UTVPI in practice, making this method very attractive.

5.4 Pre/post-processing

It can be noticed that our algorithms do not ask for removal of redundant inequalities, which is as hard as determining if the input system is feasible.

Polyhedra in compilation have lot of duplicate constraints, which gets reflected in the approximations as well. Compilers like PLuTo remove these by methods like textual matching which leads to a large decrease in the number of constraints. A more advanced scheme, finely-tuned to UTVPI constraints, would use a hashing or radix-sort technique (like in [3]) packing the UTVPI constraint in a few integers: exploiting the fact that the non-homogenizing dimension values are from $\{0, \pm 1\}$, while the homogenizing dimension values are *always* from a very small set $[-5, 5]$. It would bring the simplification complexity to linear time. We thus believe that the problem of duplicate elimination is asymptotically insignificant.

The following example shows an inherent limitation of the process of TVPI-UA itself.

Example 5 The polyhedron described using the constraints $\{x + y + z \geq -1; x + y + z \leq 1\}$ is not TVPI. Though it is unbounded in both directions, the only TVPI approximations that are possible of the above polyhedron are bounded TVPI polyhedra, which can be considered a failure of the UA approach. □

The above kinds of polyhedra can be considered as degenerate cases and identified by a pre-processing step that removes the lineality space from the input polyhedron. Further, the Farkas polyhedra that arise in polyhedral scheduling are always pointed polyhedra and can never have non-trivial lineality space.

5.5 Integer scaling and TU polyhedra

The parametric approximations of Sections 4.2 and 4.5 can ensure that the resulting difference constraints are integer constraints of the type $x_i - x_j \leq c$, where $c \in \mathbb{Z}$. This would involve scaling up the pseudo-parametric variables, changing the context constraints accordingly, and *solving an ILP problem*—instead of the usual LP problem—though on a per-P_e basis. Such a local transformation will not only induce integer schedules because incidence matrices are a sub-class of network matrices, but also has the additional property that all vertices of the resultant overall polyhedron are integral (because such matrices are Totally Unimodular [44, Chapter 19]). We will show in Section 8 that this method would help in polynomial time approximations of NP-Complete problems and code-generation.

6. Feasibility Control With Many Polyhedra

The previous sections discuss algorithms for the under approximation of one single polyhedron, so that the UA polyhedron can be solved as a means of trading expressiveness (e.g., the ability to find "good" affine transformations) for scalability. In affine scheduling however, the overall system P is an intersection of many polyhedra, each of them arising from a dependence-edge as in the case of Feautrier's scheduler [25, 26], or the above along with some additional constraints as in the case of Bondhugula's scheduler [13, 29]. More specifically:

- Feautrier's scheduler to minimize latency: $P_L = \cap_{e \in E} P_e$, where P_e is the per-dependence-edge Farkas polyhedron. A given dependence edge induces a (weak satisfaction) constraint in the system until it is strongly satisfied by outer dimensions of the multidimensional affine schedule (i.e., the sink of a dependence is scheduled at a strictly higher time step than the source).
- Bondhugula's FCO scheduler to expose loop tiling opportunities: $P_{FCO} = \cap_{e \in E} P_e \cap_{v \in V} H_v \cap_{v \in V} N_v$, where P_e is similar to Feautrier's scheduler's per-dependence polyhedron, enforcing weak dependence satisfaction only, but *at all dimensions* of the multidimensional affine schedule; while H_v collects the per-statement linear independence constraints, and N_v collects the per-statement non-negativity or trivial solution avoidance constraints.

These two cases are different because, thanks to the conditions in [26], the polyhedron built by Feautrier's multi-dimensional scheduler is known to be *always* feasible. This condition however is

not necessarily true for FCO-based systems: though the number of problematic, additional constraints $|H_V| + |N_V|$ is small (around $2|V|$ or less than 10%) when compared to Farkas multiplier constraints $|P_E|$, the overall polyhedron could be empty, complicating the UA procedure.

The feasibility preservation problem is to find a non-empty $\mathrm{UA}(P)$ in each of the above cases, *without* making any queries—like an LP call—on the overall system P.[2]

In this section we discuss practical methods to increase the control on feasibility for the overall approximated system.

6.1 Feautrier's scheduler

Let us first discuss a technique for Feautrier's scheduler. Let us start by approximating each of the P_e by calling the one-shot algorithm discussed in Section 4.3.1, and construct the overall approximation by putting all the individual approximations together: $\mathrm{UA}(P) = \cap_{e \in E} \mathrm{UA}(P_e)$. The overall approximation is the result of $|E|$ calls to the one-shot procedure. This will be affordable because each of the P_e-s are quite small.

Feautrier's algorithm always finds a (rational) multi-dimensional affine schedule, which guarantees that P is a feasible polyhedron. The main question is whether feasibility will be preserved when intersecting the per-dependence approximations. It may sound as a surprise that the answer is yes. Notice that he original schedule of the source program is always feasible. The key idea is to *seed* the approximation of the per-dependence P_e polyhedra, forcing this original schedule into the UA. It simply amounts to substituting the values of the coefficients of the original schedule in the Farkas constraints, and adding the resulting (in)equalities to the systems before applying the one-shot UA method.

This result is important because of the large number of affine scheduling heuristics deriving from Feautrier's algorithm. Feautrier's fine-grain parallelization method is already useful for loop vectorization [26]. Our seeding approach is directly applicable to coarse-grain parallelization heuristics as well [36].

We could stop there and declare success. But more advanced methods combining tiling for locality enhancement and parallelism extraction require additional effort to preserve feasibility. The next section studies the most successful of these heuristics, pushing us to enhance the under-approximation method to tackle feasibility preservation in presence of more complex affine constraints.

6.2 FCO scheduling in PLuTo

In this section, we discuss the approximation of P in the context of Bondhugula's FCO scheduler, as implemented in PLuTo. Here, seeding with the original schedule is not possible since the loops of the source program may not be directly permutable/tilable. We study a variety of Per-Dependence constraint clustering techniques.

PD1: Fully-separate Each P_e is approximated alone, and all the additional constraints are approximated together as in: $\mathrm{UA}(P) = \cap_{e \in E} \mathrm{UA}(P_e) \cap \mathrm{UA}(\cap_{v \in V} H_v \cap_{v \in V} N_v)$. The overall approximation is the result of $|E| + 1$ applications of the one-shot method.

PD2: Total augmentation Each P_e is augmented with *all* the additional constraints, and this augmented polyhedron is approximated, as in: $\mathrm{UA}(P) = \cap_{e \in E} \mathrm{UA}(P_e \cap_{v \in V} H_v \cap_{v \in V} N_v)$. The overall approximation is the intersection of $|E|$ one-shot approxiations, and each element in H_V and N_V could be approximated multiple-times.

PD3: Selected augmentation Each P_e is augmented with *its section* of the additional constraints H_u and N_u (for statements u on either side of dependence edge) and the augmented per-dependence polyhedron is approximated. If $e : v_1 \to v_2$, $\mathrm{UA}(P) = \cap_e \mathrm{UA}(P_e \cap H_{v_1} \cap H_{v_2} \cap N_{v_1} \cap N_{v_2})$. The overall approximation is the intersection of $|E|$ one-shot approximations.

PD4: Just Farkas Each P_e is approximated independently, just like in Feautrier's scheduler, while the additional constraints are thrown away. The overall approximation is the intersection of $|E|$ one-shot approximations.

This method clearly does not give an UA. It is meant to better characterize the source of the infeasibility in the previous three methods. Indeed, considering the intersection of UAs of per-dependence, weak-satisfaction constraints we know that the P_e's are homogeneous cones. This tells us that the origin can be used as a seed to build a non-empty approximation.

7. Experimental Results

We conduct systematic experiments, tiling and parallelizing the PolyBench (2.0) [40] with the source-to-source polyhedral optimizer PLuTo (PLuTo-0.6). Subsequent to our experiments, later versions of PolyBench(3.x) have been released, and which have similar numbers. Farkas systems are extracted from PLuTo. We compare the default linear programming calls to PIP [24] with the TVPI-UA and then the UTVPI-UA algorithms. The UTVPI-UA is further relaxed into a constraint graph (or incidence matrix) which is fed to a custom implementation of the Bellman-Ford algorithm.

Note that we still use PIP for the much smaller linear programming problems arising from the feasibility enhancing heuristics. Also, Pluto makes use of PIP in parts unrelated with the scalability of affine scheduling problems: the PIP calls from PLuTo originate from the functions
`dep_satisfaction_test` (DS), `get_dep_direction` (DIR) and `find_permutable_hyperplanes` (FPH).

We focus on FPH calls which correspond to affine scheduling problems. There are a lot more calls of the DS and DIR variety when compared to the FPH calls, optimization of the former pair of LP calls is entirely a different problem from the FPH variety in many ways. Primarily, the former need to be *over-approximated* [21, 53] as otherwise, it results in incorrect transformations. The latter of course are the main topic of this paper and need to be *under-approximated* leading to a conservative approximation and perhaps loss of useful schedules.

7.1 Features of the polyhedra

Here we are referring to Table 1.a. The initial three columns refer to the size of the loop nest: L is the number of loops, S the number of statements, and D the number of dependences. The next sets of columns indicate the polyhedral characteristics of the different varieties of calls from PLuTo. As the DS and DIR variety are similar types of calls, they have been summarized together. The remaining 8 columns (Table 1.b) will be discussed in the next section.

The number of polyhedra of different types are indicated in the P columns ($\mathrm{P_{DS}}$, $\mathrm{P_{DIR}}$ and $\mathrm{P_{FPH}}$). As written earlier, there are lot more polyhedral calls of DS/DIR than when compared to FPH. But the former are smaller calls and are linearly dependent on the size of the loop nest. For a particular benchmark and variety of polyhedra (DS/DIR or FPH), n and $\hat{m}$ columns indicate the number of variables (unknowns), and average number of constraints in the particular LP formulation respectively. $\hat{m}_t$ column indicates the average number of TVPI constraints for that benchmark and variety of polyhedra.

DS/DIR As it can be expected, most of the constraints in the DS/DIR polyhedra are TVPI constraints. In these polyhedra, there

[2] If such a query could be made, then it could as well be used to solve for the objective function resulting in the schedule...

	Loop nest			$P_{DS}(1)/P_{DIR}(2)$					P_{FPH}						
Bench	#L	#S	#D	$\#P_1$	$\#P_2$	n	$\hat{m}$	$\hat{m}_t$	#P	n	$\hat{m}$	$\hat{m}_t$	$\widehat{\|\mathcal{S}_{m_k}\|}$	$\hat{m}_a$	$\hat{m}_u$
adi	12	4	54	90	564	9	20	19	3	20	200	65	5	1844	614
corcol	12	6	14	38	194	9	17	16	3	22	22	13	5	130	77
covcol	13	7	26	41	228	15	25	24	3	24	18	14	4	29	55
dsyr2k	3	1	3	9	18	8	18	17	3	7	8	6	3	11	18
dsyrk	3	1	3	9	18	8	18	17	3	7	8	7	3	11	18
fdtd-2d	11	4	48	39	168	10	22	21	3	20	96	35	6	1010	367
gemver	7	4	13	29	161	6	13	12	2	14	21	15	4	48	47
jacobi-1d	4	2	10	16	88	7	14	13	2	10	32	14	5	232	104
jacobi-2d	6	2	14	21	84	9	19	18	3	12	65	15	7	1135	212
lu	5	2	10	12	60	11	23	22	3	10	35	17	5	232	106
matmul	3	1	3	9	18	10	21	21	3	9	9	7	4	20	24
mvt	4	2	11	31	68	6	13	12	2	9	20	12	3	46	52
seidel	3	1	27	37	162	12	24	23	3	8	33	15	5	168	179
ssymm	8	3	15	33	126	8	19	19	3	14	15	10	3	22	36
strmm	3	1	4	8	24	8	17	16	3	7	12	8	3	20	30
tmm	3	1	3	9	18	10	21	21	3	9	11	8	4	23	30

Median		LP (indep)		PD2		PD4	
YY	YN	YY	YN	YY	YN	YY	YN
0	3	0	3	0	3	3	0
1	2	0	3	3	0	3	0
3	0	3	0	3	0	3	0
3	0	3	0	3	0	3	0
3	0	3	0	3	0	3	0
0	3	1	2	0	3	3	0
0	2	2	0	1	1	2	0
0	2	0	2	0	2	2	0
0	3	0	3	0	3	3	0
0	3	0	3	0	3	3	0
3	0	3	0	3	0	3	0
0	2	2	0	2	0	3	0
0	3	3	0	0	3	3	0
3	0	3	0	3	0	3	0
0	3	3	0	0	3	3	0
3	0	3	0	3	0	3	0
19	26	29	16	24	21	45	0

Table 1. (a) Problem size, Polyhedral and TVPI-ness characteristics, (b) UA effectiveness

is *never* more than 1 constraint per polyhedron which is non-TVPI. In all the cases, whenever a constraint is TVPI, it is *always* a UTVPI constraint, having the same absolute magnitude for the two coefficients, when it has two entries.

FPH In the FPH variety, it can be noticed that the sizes of some of the FPH polyhedra are small and comparable to the DS/DIR polyhedra. This is either the result of a small problem size or because PLuTo uses Fourier-Motzkin elimination, along with a syntactic heuristic to reduce the duplicate constraints. As it can be expected, $\hat{m}_t$, number of TVPI constraints in the original system, is highly benchmark dependent. But, just like in DS/DIR polyhedra, a TVPI constraint is always a UTVPI constraint.

$\widehat{\|\mathcal{S}_{m_k}\|}$ is the average number of non-zero coefficients in the non-TVPI constraints for that benchmark. It can be seen that this number, though again being benchmark dependent, is a small constant when compared to the dimension size n. $\hat{m}_a$ is the number of constraints in the new (approximated TVPI) system. As seen earlier, $\hat{m}_a = \widehat{\|\mathcal{S}_{m_k}\|}(\widehat{\|\mathcal{S}_{m_k}\|} - 1)(m - m_t)/2 + m_t$.

The relative growth of the approximated system with respect to the original one is defined as the ratio between the sum of entries in the $\hat{m}_a$ and $\hat{m}$ columns. We found the average value of this to be 8, meaning that the overall sparsity factor $\hat{s}$ is a little more than 4. Sometimes the growth of the approximated system is significant, but it has to be remembered that $\hat{m}_a$ is the number of constraints without any simplification, while m is obtained after systematic simplification and elimination of duplicates in PLuTo.

For comparison purposes with $\hat{m}_a$, we have added the $\hat{m}_u$ column, which is the average unsimplified system size when the simplification techniques used in PLuTo are turned off. It can be observed that $\hat{m}_u$ and $\hat{m}_a$ are of comparable sizes. Our experience is that when the approximated system undergoes simplification and duplication removal techniques, it leads to a much smaller system, comparable to the one in m columns. Also, asymptotically, the size increase that only depends on sparsity does not matter much.

7.2 UA feasibility

Here we are referring to Table 1.b. These numbers are for TVPI-UA. (The results for UTVPI-UA are similar and are not being given because of space constraint.) These columns refer to the median method discussed in Section 4.1, to the LP based independent method discussed in Section 4.3.2 and the per-dependence methods discussed in Section 6 respectively. All the columns except for the LP-indep method refer to the per-dependence methods. Only the LP-indep method was implemented on the overall Farkas system, one that is obtained after putting together all the individual systems and after simplification by PLuTo.

The PLuTo calls of the FPH variety are for LexMin. In the current table, we discuss the feasibility results only. The columns YY (Yes-Yes), YN (Yes-No), denote the feasibility (Y) or infeasibility (N) of the original and approximated systems respectively. They have been highlighted accordingly. Since the FPH system is used to find an optimization point in the overall system, a YN entry would mean loss of parallelization.

It can be seen that the LP-indep method (29 YY and 16 YN cases or 10 out of 16 PolyBench problems) performs much better than the median method (19 YY and 26 YN cases or 6 out of 16 PolyBench problems). The latter performs not as poorly as the simplicity of the approach would hint to. We expect the incremental method to have much better performance than the current independent method. The results of PD1 (not shown) fare marginally better than the median method. The results of PD2 are close to LP-indep (24 YY and 21 YN cases or 8 out of 16 PolyBench loop nests). Advanced clustering strategy PD3 has not been implemented. PD4 (Just Farkas) gives 100% feasibility preservation results (45 YY and 0 YN), making it quite attractive. The next sections study the impact on compilation time and on the performance of the generated code.

7.3 Scalability comparison: Simplex vs. Bellman-Ford

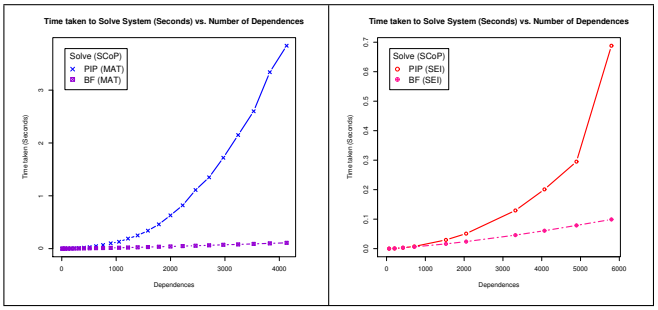

Figure 3. Simplex (PIP) vs. Bellman-Ford (BF)

Figure 3 shows the running times of the Simplex on the original system vs. Bellman-Ford (BF) on a UTVPI approximated system (PD4). The former uses the quite well used dual-simplex implementation of PIP [24] (similar [50] to many LP solvers like CPLEX [11]) called on a *rational polyhedron for feasibility*, while

the latter is our own implementation of a standard Bellman-Ford algorithm called for testing for presence of *negative weight cycles*.

The input programs `matmul` and `seidel` are the same as in Figure 1, and each call of `auto_trans` of PLuTo to find schedule hyperplanes has tens of calls of the above variety, which explains the difference in scales. Prior to solving either of these, the duplicates were eliminated using a syntactic matching like currently in PLuTo. Even on the systems with duplicates, the improvements were similar, though being more prominent.

It can be seen from the graphs that for their respective inputs, BF (worst-case $\mathcal{O}(\hat{s}^2mn) \approx \mathcal{O}(|E||V|) \approx \mathcal{O}(|V|^3)$) is asymptotically as well as empirically faster than PIP's Simplex (observed $\mathcal{O}((m+n)mn) \approx \mathcal{O}(|V|^5)$), answering positively the scalability challenge. The regression coefficients (linear R^2) for the above formulae are around 99.9%. The curves for BF appear linear because the x-axis counts the number of dependences, which is practically $\mathcal{O}(|V|^2)$. Though the memory improvements are not shown in the above graphs, the linear memory of BF is considerably lighter than the quadratic memory of Simplex.

The relatively lesser magnitudes and hence lesser improvements of `seidel` when compared to `matmul` could be attributed to the fact that while the former has just uniform dependences which are easily amenable to the Gaussian eliminations of PIP, the latter has more complex affine dependences and also induces many more Farkas multiplier variables.

7.4 UA generated code performance

		Perf. Comparison (Seconds)			
Benchmark	Orig	Par cur	Par new	Til cur	Til new
gemver	0.31	0.15 (2x)	0.15 (2x)	0.15 (2x)	0.15 (2x)
mvt	1.40	0.27 (5x)	0.28 (5x)	0.42 (5x)	0.43 (5x)
seidel	11.8	3.6 (3x)	3.6 (3x)	11.5 (1x)	11.5 (1x)

Table 2. UA Code Performance

We are referring here to Table 2, limiting ourselves to a subset of the YY cases in the previous table that our current implementation could handle; we will later consider all YY and YN cases with a more robust implementation. In each case, we replaced the original system(s) by the approximated TVPI ones obtained by the independent LP method. The cost function was unchanged and the solution was found using PIP. It can be seen that performance gains closely match the default polyhedral method in PLuTo, despite the approximation taking place. The impact of YN approximations on PLuTo's effectiveness is yet to be studied, but we express some hope on PLuTo's loop distribution heuristic to break infeasible systems.

8. Extensions and Related Work

In this section we will first see how our framework can be applied to other (scalability) problems, either in affine scheduling framework or beyond, and then cover some related work.

8.1 Applications to other loop transformation problems

One important strength of our under-approximation framework is its ability to trade precision for scalability using (U)TVPI approximations of polyhedra. Scalability arises from the excellent worst-case complexities of sub-polyhedral operations, and the ability to largely implement the approximation as an independent optimization problem for each dependence-edge. The latter property is another strength of our approach, which happens to be general enough to be applied to other compilation problems, related and unrelated to scheduling.

Darte-Vivien's scheduling As the framework of Feautrier which we build from is the most powerful among the class of algorithms that find affine schedules [56], the LP formulation in Darte-Vivien's scheduling [22] can directly be approximated by our method.

Shifting 1-d loops for pipelining and compaction To solve the cyclic scheduling (decomposed software pipelining) problem, Calland et al. [14] give an LP formulation whose constraint matrix is built from incidence matrices of the original and a retimed dependence graph, and prove it to be a Totally Unimodular (TU) matrix. As given, the formulation is not a (U)TVPI polyhedron having at most 3 non-zero elements per row. But, it can benefit from our framework as the constraints are constructed on a per-dependence-edge basis, and the constraint matrix elements belong to the set $\{0, \pm 1\}$. Darte-Huard's loop shifting for compaction [19] uses a very similar framework to the above paper, where an LP formulation which is a variation of a min-cost flow problem is used, and hence could as well benefit from a (U)TVPI approximation resulting in a strongly polynomial time algorithm.

n-d loop alignment for fusion Darte-Huard show that external shifting in the multi-dimensional loop alignment problem is NP-Complete [20]. They provide an ILP formulation constructed on a per-dependence-edge basis, with 4 variables per constraint and general coefficients (not just $\{0, \pm 1\}$). A possible heuristic for their formulation that builds on our techniques would be to under-approximate each per-edge constraint and to solve the overall system using the Bellman-Ford algorithm. This is correct because rational relaxation and subsequent UA may result in less integer points, but no new ones. The overall UA, if non-empty, can be solved in polynomial time. Further, it can be made to directly result in an integer multi-dimensional shift using the integer scaling discussed earlier in Section 5.5. In this method, an *ILP problem* is solved on a per-P_e-basis, while solving a *normal Bellman-Ford problem* on the overall UA polyhedron resulting in an integer multi-dimensional shift. In this aspect, UTVPI polyhedra are very special, and such a polynomial time shifting algorithm will not be possible even with a TVPI-UA, as solving integer-TVPI polyhedra is known to be NP-Complete [35].

Simpler code generated As seen earlier, by the TU property, a restriction of the homogenizing dimensions to integers can ensure that *all the vertices* of the resultant UTVPI-UA are integers. Using these UAs in scheduling has the additional advantage of simpler code being generated, even with affine schedules generated through (rational) linear programming (PLuTo uses integer linear programming by default). Rational schedule coefficients may indeed induce modulos in loop bounds and conditional expressions of the generated code, a risk completely eliminated with our approach.

8.2 Related work

Feautrier's scalable modular scheduling Feautrier's approach [27] starts with Gaussian elimination, and suggests a Minkowski decomposition (generator representation) of P_e for projection. We consider this approach as complementary to ours in the sense of per-dependence methods, but the use of generator representation makes it less likely to be scalable than ours.

Approximations in loop optimization Previous approaches to approximations in loop optimization concerned themselves with restricting the kinds of input programs or the kinds of transformations being searched for [21, 60]. The most notable examples of the above are the Dependence Levels of Allen and Kennedy [2] and Direction Vectors of Wolf and Lam [59]. Each of the above are less powerful than the affine scheduling model of Feautrier [25]. Our approach tackles the scalability problem from within the affine scheduling model and is more powerful.

(U)TVPI in loop optimization UTVPI polyhedra have previously been used by Balasundaram and Kennedy [7] in task level loop parallelization and pipelining applications, by data dependence analysis, domain simplification and (limited form of) code generation. Their use is similar to the framework of classical dependence over-approximations [21, 53, 60]. Our use of (U)TVPI is more powerful than theirs because of our affine scheduling framework and UA algorithms. Further, Pugh [42] and Maydan et al. [37] also hint that (U)TVPI polyhedra could be solved with better complexity and hence used in dependence analysis citing some statistics from parallelization benchmarks. Our statistics in Table 1.a could be seen as extensions of the above.

(U)TVPI in static analysis General polyhedra are known [31, 33, 38] to be considerably more expensive than (U)TVPI polyhedra for abstract domain operations. Hence, sub-polyhedra have been widely employed in abstract interpretation problems by the static analysis community as a means of trading precision for scalability. A comparison of the use of sub-polyhedra, along with their applicability to polyhedral compilation can be found in [53]. Of the many classes of sub-polyhedra, (U)TVPI polyhedra have extensively been used [6, 38, 47], with impressive results such as in Astrée [12, 17].

Approximations to sub-polyhedra We are not aware of any previous algorithm for finding approximations of general polyhedra into sub-polyhedra other than finding the interval ("box") polyhedral OAs. The literature is scarce about polyhedral approximations and more so for UAs. (The naïve method of projecting out the extra variables on a per-constraint basis using Fourier-Motzkin method leads to an over-approximation with no bound on the complexity.) Vivien-Wicker [57] propose an algorithm to find the parallelepiped-OA of a 3d-polyhedron in vertex representation. Miné [38, Section 4.3] adapts the vertex method for finding the UTVPI-OA of a general polyhedron. Simon et al. [47, Section 3.2.6] propose an iterative algorithm for the TVPI-OA of a general polyhedron using LP. Our algorithm for TVPI-UA in Section 4.3 can be seen as complementary to the latter, though our algorithm formulates the LP problem by searching only for the replacements in the homogenizing dimension.

Feasibility and optimization algorithms The state of the art in feasibility testing for TVPI systems is by Hochbaum-Naor [32], and for optimization of TVPI systems by Wayne [58]. We are not aware of existing implementations of the above algorithms. The state of the art for feasibility and optimization of UTVPI systems is by the well understood Bellman-Ford algorithm [41], which involves constructing a nearly $2m \times 2n$ size constraint graph encoding the difference constraints [23] and testing it for presence of negative weight cycles using the shortest-path problem. The latter has been extensively studied, beginning from [9, 28] to most recently in [15]. Efficient, though closure based methods for the same [6, 38] are available in Apron [34] and PPL [5]. More general forms than ours, like boolean formulae involving UTVPI constraints have been considered in constraint programming, like by Seshia et al. [45].

9. Conclusions and Future Work

We have presented sub-polyhedral scheduling using (U)TVPI polyhedra. We have proposed worst-case polynomial time algorithms to compute (U)TVPI under-approximations from a general convex polyhedron. We have shown initial results of the above approximations as well as their integration in PLuTo. Our experiments clearly indicate an asymptotic improvement in scalability, answering positively the scalability challenge.

To stay within polynomial time, care was taken to linearize the under-approximation model, avoiding the exponential vertex and ray construction associated with the Chernikova algorithm. The most difficult problem is the lack of a scalable way to reliably preserve the non-emptiness of the underapproximation when the original polyhedron is non-empty. The one-shot linear programming method offers this guarantee, but not in strongly polynomial time. We explore different heuristics relying on bounded-size linear programs to trade complexity for feasibility. We know this problem is difficult because a strongly polynomial method with this guarantee could directly lead to a solution of finding strongly polynomial feasibility test for general polyhedra.

The scalability experiments indicate an asymptotic improvement of Bellman-Ford over linear programming. Future experiments should include a fast duplicate elimination, a complete evaluation of the performance impact of the approximation on all PolyBench using an approximated objective function compatible with the Bellman-Ford algorithm, and a complementary study of the memory complexity (known to be overwhelmingly in our favor).

An illustration of the power of our approximation scheme has been shown on important affine scheduling problems. Future work would involve applying the scheme to a wider range of compilation problems, including ILP or mixed-ILP formulations, so that either heuristics with better complexity measures are obtained, or the resulting approximations provide practical improvements on complex systems.

Acknowledgments We are grateful to Paul Feautrier for the tremendously helpful suggestions and insights. We are also thankful to Cédric Bastoul and Armin Größlinger for their feedback at different stages. Our work also benefited from Uday Bondhugula's PLuTo framework, and from the detailed comments from anonymous reviewers. This work was partly supported by the French Nano2012 project Mediacom, in collaboration with STMicroelectronics, and by the European FP7 project CARP id. 287767.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice-Hall, Inc., NJ, USA, 1993.

[2] R. Allen and K. Kennedy. Automatic translation of fortran programs to vector form. *ACM Trans. Program. Lang. Syst.*, 9(4):491–542, Oct. 1987.

[3] A. Andersson and S. Nilsson. Implementing radixsort. *J. Exp. Algorithmics*, 3, Sept. 1998.

[4] B. Aspvall and Y. Shiloach. A polynomial time algorithm for solving systems of linear inequalities with two variables per inequality. *SIAM J. Comput.*, 9(4):827–845, 1980.

[5] R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.

[6] R. Bagnara, P. M. Hill, and E. Zaffanella. Weakly-relational shapes for numeric abstractions: improved algorithms and proofs of correctness. *Formal Methods in System Design*, 35(3):279–323, 2009.

[7] V. Balasundaram and K. Kennedy. A technique for summarizing data access and its use in parallelism enhancing transformations. In *PLDI*, pages 41–53, 1989.

[8] U. Banerjee. *Loop Transformations for Restructuring Compilers: The Foundations*. Kluwer Academic Publishers, Boston, 1992.

[9] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[10] M.-W. Benabderrahmane, L.-N. Pouchet, A. Cohen, and C. Bastoul. The polyhedral model is more widely applicable than you think. In *Proceedings of the International Conference on Compiler Construction (ETAPS CC'10)*, number 6011 in LNCS, Paphos, Cyprus, Mar. 2010. Springer-Verlag.

[11] R. E. Bixby. Solving real-world linear programs: A decade and more of progress. *Oper. Res.*, 50(1):3–15, Jan. 2002.

[12] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *PLDI*, pages 196–207. ACM, 2003.

[13] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan. A practical automatic polyhedral parallelizer and locality optimizer. In *PLDI*, pages 101–113, 2008.

[14] P.-Y. Calland, A. Darte, and Y. Robert. Circuit retiming applied to decomposed software pipelining. *IEEE Trans. Parallel Distrib. Syst.*, 9(1):24–35, Jan. 1998.

[15] B. V. Cherkassky, L. Georgiadis, A. V. Goldberg, R. E. Tarjan, and R. F. Werneck. Shortest-path feasibility algorithms: An experimental evaluation. *J. Exp. Algorithmics*, 14:7:2.7–7:2.37, Jan. 2010.

[16] E. Cohen and N. Megiddo. Improved algorithms for linear inequalities with two variables per inequality. *SIAM J. Comput.*, 23:1313–1350, December 1994.

[17] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, and X. Rival. Why does Astrée scale up? *Formal Methods in System Design*, 35(3):229–264, Dec 2009.

[18] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.

[19] A. Darte and G. Huard. Loop shifting for loop compaction. *Int. J. Parallel Program.*, 28(5):499–534, Oct. 2000.

[20] A. Darte and G. Huard. Complexity of multi-dimensional loop alignment. In *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '02, pages 179–191, London, UK, UK, 2002. Springer-Verlag.

[21] A. Darte, Y. Robert, and F. Vivien. *Scheduling and Automatic Parallelization*. Birkhaüser, 2000.

[22] A. Darte and F. Vivien. Optimal fine and medium grain parallelism detection in polyhedral reduced dependence graphs. *Int. J. Parallel Program.*, 25:447–496, December 1997.

[23] H. Edelsbrunner, G. Rote, and E. Welzl. Testing the necklace condition for shortest tours and optimal factors in the plane. *Theor. Comput. Sci.*, 66(2):157–180, 1989.

[24] P. Feautrier. Parametric integer programming. *RAIRO Recherche Opérationnelle*, 22(3):243–268, 1988. `http://www.piplib.org/`.

[25] P. Feautrier. Some efficient solutions to the affine scheduling problem: I. one-dimensional time. *IJPP*, 21:313–348, October 1992.

[26] P. Feautrier. Some efficient solutions to the affine scheduling problem: Part ii: Multidimensional time. *Int. J. Parallel Program.*, 21:389–420, December 1992.

[27] P. Feautrier. Scalable and structured scheduling. *International Journal of Parallel Programming*, 34(5):459–487, 2006.

[28] L. R. Ford, Jr. and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[29] M. Griebl, P. Feautrier, and A. Größlinger. Forward communication only placements and their use for parallel program construction. In W. Pugh and C.-W. Tseng, editors, *LCPC*, volume 2481 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2002.

[30] T. Grosser, H. Zheng, A. Raghesh, A. Simbürger, A. Größlinger, and L.-N. Pouchet. Polly - Polyhedral Optimization in LLVM. In *IMPACT 2011, in conjunction with CGO 2011*, Chamonix, France, Apr 2011.

[31] N. Halbwachs, D. Merchat, and L. Gonnord. Some ways to reduce the space dimension in polyhedra computations. *Form. Methods Syst. Des.*, 29:79–95, July 2006.

[32] D. S. Hochbaum and J. Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM J. Comput.*, 23(6):1179–1192, 1994.

[33] J. Jaffar, M. J. Maher, P. J. Stuckey, and R. H. C. Yap. Beyond finite domains. In A. Borning, editor, *PPCP*, volume 874 of *Lecture Notes in Computer Science*, pages 86–94. Springer, 1994.

[34] B. Jeannet and A. Miné. Apron: A library of numerical abstract domains for static analysis. In *CAV*, pages 661–667, 2009.

[35] J. C. Lagarias. The computational complexity of simultaneous diophantine approximation problems. *SIAM J. Comput.*, 14(1):196–209, 1985.

[36] A. W. Lim and M. S. Lam. Communication-free parallelization via affine transformations. In *POPL*, pages 201–214, Paris, France, jan 1997.

[37] D. E. Maydan, J. L. Hennessy, and M. S. Lam. Efficient and exact data dependence analysis. In *Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation*, PLDI '91, pages 1–14, New York, NY, USA, 1991. ACM.

[38] A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.

[39] L.-N. Pouchet, U. Bondhugula, C. Bastoul, A. Cohen, J. Ramanujam, P. Sadayappan, and N. Vasilache. Loop transformations: convexity, pruning and optimization. In *POPL*, pages 549–562, 2011.

[40] L.-N. Pouchet, U. Bondhugula, et al. The polybench benchmarks. `http://www.cse.ohio-state.edu/~pouchet/software/polybench`.

[41] V. R. Pratt. Two easy theories whose combination is hard. Technical report, Massachusetts Institute of Technology, Cambridge, Mass, 1977. `http://boole.stanford.edu/pub/sefnp.pdf`.

[42] W. Pugh. A practical algorithm for exact array dependence analysis. *Commun. ACM*, 35(8):102–114, Aug. 1992.

[43] F. Santos. A counterexample to the hirsch conjecture. *Annals of Mathematics*, 176:383–412, 2012.

[44] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, Inc., New York, NY, USA, 1986.

[45] S. A. Seshia, K. Subramani, and R. E. Bryant. On solving boolean combinations of UTVPI constraints. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 3(1-2):67–90, 2007.

[46] R. Shostak. Deciding linear inequalities by computing loop residues. *J. ACM*, 28:769–779, October 1981.

[47] A. Simon and A. King. The two variable per inequality abstract domain. *Higher Order Symbol. Comput.*, 23(1):87–143, Mar. 2010.

[48] D. A. Spielman and S.-H. Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM*, 51:385–463, May 2004.

[49] A. Tarski. *A decision method for elementary algebra and geometry*. Univ. of California Press, Berkeley, 2nd edition, 1951.

[50] M. J. Todd. The many facets of linear programming. *Mathematical Programming*, 91:417–436, 2002.

[51] K. Trifunovic, A. Cohen, D. Edelsohn, F. Li, T. Grosser, H. Jagasia, R. Ladelsky, S. Pop, J. Sjödin, and R. Upadrasta. Graphite two years after: First lessons learned from real-world polyhedral compilation. In *GCC Research Opportunities Workshop (GROW'10)*, Pisa, Italy, Jan. 2010.

[52] R. Upadrasta. *Scalability Challenges in the Polyhedral Model: An Algorithmic Approach using (Unit-)Two-variable Per Inequality Sub-Polyhedra*. PhD thesis, Université Paris-Sud (11), Orsay, France, January 2013.

[53] R. Upadrasta and A. Cohen. Potential and Challenges of Two-Variable-Per-Inequality Sub-Polyhedral Compilation. In *First International Workshop on Polyhedral Compilation Techniques (IMPACT'11), in conjunction with CGO'11*, Chamonix, France, Apr. 2011.

[54] R. Upadrasta and A. Cohen. A Case for Strongly Polynomial Time Sub-Polyhedral Scheduling Using Two-Variable-Per-Inequality Polyhedra. In *Second International Workshop on Polyhedral Compilation Techniques (IMPACT'12), in conjunction with HiPEAC'12*, Paris, France, Jan. 2012.

[55] N. Vasilache. *Scalable Program Optimization Techniques In The Polyhedral Model*. PhD thesis, Paris-Sud 11 University, Sept. 2007.

[56] F. Vivien. On the optimality of feautrier's scheduling algorithm. *Concurrency and Computation: Practice and Experience*, 15(11-12):1047–1068, 2003.

[57] F. Vivien and N. Wicker. Minimal enclosing parallelepiped in 3d. *Comput. Geom. Theory Appl.*, 29:177–190, November 2004.

[58] K. D. Wayne. A polynomial combinatorial algorithm for generalized minimum cost flow. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, STOC '99, pages 11–18, New York, NY, USA, 1999. ACM.

[59] M. E. Wolf and M. S. Lam. A loop transformation theory and an algorithm to maximize parallelism. *IEEE Trans. Parallel Distrib. Syst.*, 2(4):452–471, Oct. 1991.

[60] Y.-Q. Yang, C. Ancourt, and F. Irigoin. Minimal data dependence abstractions for loop transformations. In K. Pingali, U. Banerjee, D. Gelernter, A. Nicolau, and D. A. Padua, editors, *LCPC*, volume 892 of *Lecture Notes in Computer Science*, pages 201–216. Springer, 1994.

[61] G. Ziegler. *Lectures on polytopes*. Graduate texts in mathematics. Springer Science, 2006.