A Case for Strongly Polynomial Time Sub-Polyhedral Scheduling Using Two-Variable-Per-Inequality Polyhedra

Ramakrishna Upadrasta
PARKAS group, INRIA, École Normale Supérieure,
and LRI, Paris-Sud 11 University
Ramakrishna.Upadrasta@inria.fr

Albert Cohen
PARKAS group, INRIA, École Normale Supérieure
Albert.Cohen@inria.fr

Abstract

We make a case for sub-polyhedral scheduling using (Unit-)Two-Variable-Per-Inequality or (U)TVPI Polyhedra. We empirically show that using general convex polyhedra leads to a scalability problem in a widely used polyhedral scheduler. We propose methods in which polyhedral schedulers can beat the scalability challenge by using sub-polyhedral under-aproximations of the polyhedra resulting from the application of the affine form of the Farkas lemma. We propose simple algorithms that under-approximate a general polyhedra into (U)TVPI polyhedra. These algorithms take worstcase polynomial time. We implement the above approximation algorithms in a modified PLuTo, and show that for a majority of the Polybench 2.0 kernels, the above under-approximation yield polyhedra that are non-empty. We also provide preliminary evidence 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, Algorithms, Optimization, Performance

Keywords Approximation Algorithms, Complexity Theory, Compilers, Optimization, Geometric Algorithms

1. Motivation

In the previous paper [19], 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 worstcase polynomial time guarantee. In this paper, we make solid progress towards polyhedral schedulers that are intrinsicly scalable in the program size, relying on strongly polynomial algorithms only, i.e., whose time complexity is polynomial in the number integers in the input and not only on the bit-size of the input. Our method applies to latency and depth minimization appraoches like Feautrier's algorithm [7], as well as on tiling-centric methods driven by the forward-communication-only property like Bondhugula et al.'s PLuTo [4, 9].

IMPACT 2012
Second International Workshop on Polyhedral Compilation Techniques
Jan 23, 2012, Paris, France
In conjunction with HiPEAC 2012.

http://impact.gforge.inria.fr/impact2012

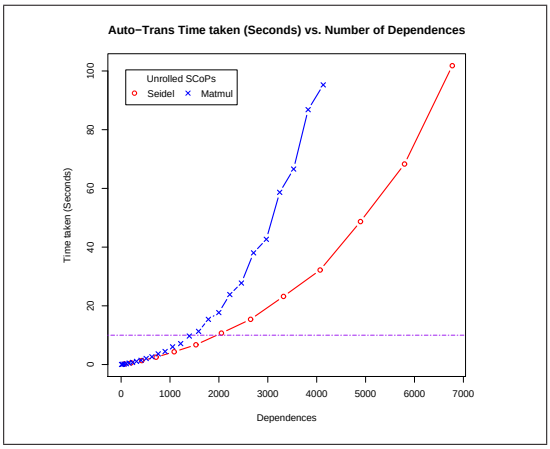

Figure 1. Unscalability for Large Versions of SCoPs

1.1 Introduction

Affine scheduling [6] now is a part and parcel of every compiler which aspires to compile for parallel architectures. Today and in the foreseeable future, it remains a difficult challenge to construct a "good" multidimensional affine transformation. The seminal work of Feautrier [7] 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. [4] extending the Forward Communication Only (FCO) principle of Griebl [9] for coarse-grain parallelization, and the complete, convex characterization and decoupled exploration heuristic of Pouchet et al. [14]. 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 appear to be strongly polynomial in the size of the program. The lowest complexity heuristics such as PLuTo appear to be reducible to linear programming, which is only weakly polynomial in theory, and only at the cost of significant (yet practically effective) restrictions of the optimization space

1.2 Unscalability of a current scheduler

In this section, we show an example of unscalability of current methods in PLuTo. We have artificially unrolled two typical ker-

nels, `matmul` and `seidel`, by a variable number of times so as to increase the number of dependences in the Static Control Parts (SCoPs), and we observe the compilation time. We have also enclosed the above unrolled loops in two or three "time loops", as it is done in compute intensive numerical processing kernels. The above methods increase the range of dependences in the input SCoPs to upto thousands of dependences. The plotted graphs 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) increase in a roughly n^5 complexity in the number of unknowns in the system. The time taken by the rest of the modules of PLuTo – in particular dependence analysis and code generation (CLooG) – were significantly smaller than the above times.

In the current benchmarks, like the ones currently in PLuTo or in testsuites like in PolyBench[1] , the range of dependences of the SCoPs is in tens, and it can arguably be said that there presently exists no scalability problem like in the above artificial examples. Even if it does not exist now, there will be a scalability problem when there is potential to increase the sizes and ranges of programs in compilers like GRAPHITE [18] by powerful dependence analysis. Or, there could be a restriction in the time limit in just-in-time compilers that would exacerbate the scalability problem further, such as online compilation applications of Polly [10].

The reader may wonder if part of the complexity is because PLuTo uses PIP to solve Integer Linear Programming (ILP) problems, rather than Linear Programming. But, even when the calls to PIP are changed to operate over rationals rather than integers, we found no significant decrease in time.

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 practical, just-in-time compilation applications.

1.3 Polyhedral scheduling and Under-Approximation

In Feautrier's algorithm [7], the dependence constraints of a particular dependence edge are pseudo-linear (or quasi-affine) constraints involving (μ, I, N), where μ are the schedule Farkas multipliers, and the I and N vectors are iterator vectors and parameter vectors respectively. These are converted into a per-dependence edge polyhedron $P_e(\mu, \lambda)$ by application of the affine form of the Farkas lemma, where the newly created λ-variables (along with μ-variables) are called the Farkas multipliers. By putting together all the per-edge Farkas polyhedra, one obtains a overall Farkas polyhedron $P = \cap_{e \in E} P_e$, which is amenable to Linear Programming or Fourier-Motzkin elimination. Any rational point that satisfies P is considered a valid schedule.

The above application of the Farkas lemma results in almost all the constraints in the Feautrier's scheduler. In PLuTo, a different but conceptually similar method also results in the majority of constraints. The very few remaining ones are dependence satisfaction variables, non-negativity, orthogonality constraints.

In this paper, we propose that P be Under-Approximated (UA) for scalability purposes. 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. The overall process has to ensure that the approximation algorithm, as well as the solution finding time be scalable algorithms. We restrict these requirements further and say that both of these algorithms should have *worstcase strongly polynomial time* running times.

It can be noticed that the above approximation can also be done on a per-dependence basis. In this method, the per-dependence edge Farkas polyhedra are under-approximated, and the solution is found from the overall polyhedron obtained by putting together all the per-dependence UAs. Namely, by doing $P_a = \mathrm{UA}(P) = \cap_{e \in E} \mathrm{UA}(P_e)$.

[1] http://www.cse.ohio-state.edu/ pouchet/software/polybench

1.4 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 use sub-polyhedra, thereby reducing their complexity of finding schedules to worstcase strongly polynomial time.
- We propose two alternative methods – either with a direct Under-Approximation of all the feasible schedules, or as an intersection of Under-Approximations of Farkas polyhedra per each dependence edge – which can be used to solve feasibility problems of large polyhedra arising in affine scheduling.
- Of the many sub-polyhedra used in static analysis ([19]), we show that TVPI and UTVPI sub-polyhedra could be good alternatives to be used in polyhedral scheduling, limiting its worstcase complexity.
- Using elementary polyhedral concepts of homogenization and polarity, we present a simple and powerful framework (an approximation scheme) which can be used for designing UA algorithms of general convex polyhedra. This framework gives a sound background to linearize the UA finding problem.
- Using the above framework, we present simple algorithms that under-approximate a general polyhedra in constraint representation ($\mathcal{H}$-form) into (U)TVPI sub-polyhedra. For a single non-TVPI constraint, these algorithms are linear in the sparsity of the particular constraint; for the multiple constraint case, we propose many variations in which we can under-approximate the overall polyhedron as one-shot or iterative methods.
- 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 also 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.

The paper is structured as follows. In Section 2, we briefly introduce TVPI and UTVPI sub-polyhedra. In Section 3 we give the mathematical framework for the linearization of the UA problem and for establishing its correctness. In Section 4 we propose simple algorithms that under-approximate a general convex polyhedron into a (U)TVPI polyhedron. In Section 5 we discuss the theoretical and practical implications of the above UA algorithms. In Section 6 we discuss the results of implementing these algorithms in PLuTo. In Section 7 we discuss related work, and in Section 8 we conclude and present some future work.

2. Sub-Polyhedra: TVPI and UTVPI

In this section, we briefly cover some basics of TVPI and UTVPI approximations of polyhedra (or sub-polyhedra) that are needed for our purposes. A more extensive discussion on these, as well as other flavors of sub-polyhedra as used by the static analysis community can be found in detail in our earlier work [19]. 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: $ax_i + bx_j \leq c$. TVPI are obviously closed under projection, and hence many algo-

rithms on geometric operations that are developed for planar polyhedra (polygons) are directly applicable to general TVPI giving rise to simple algorithms with small complexity. Further, the dual of TVPI programs is a generalized min-cost flow problem, which could be solved using graph theoretic techniques. So, the linear programming community has been interested in TVPI polyhedra because of the strongly polynomial time algorithms for the linear programming feasibility problem on TVPI polyhedra.

The early work on using graph theory techniques for linear programming on TVPI systems was by Shostak [16]. Aspvall and Shiloach [1] 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, it is the heart of all subsequent algorithms in the TVPI literature. The following result by Wayne in [21] is the best to date for the TVPI optimization problem:

Lemma 2.1 [LP optimization on TVPI] Linear programming optimization on TVPI systems can be solved in $\mathcal{O}(m^3n^2 \log B)$ worst case time.

It 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 that 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 [5]. Hochbaum and Naor in [11] showed that feasibility of TVPI polyhedra can be determined in strongly polynomial time:

Lemma 2.2 [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 and uses the previously mentioned decision procedure of Aspvall-Shiloach embedded in a binary search along with a selected application of Chernikova on a planar polyhedra to detect the feasibility of the given TVPI system. 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 the abstract interpretation and verification like in [17].

2.2 UTVPI Sub-Polyhedra (Octagons)

Octagons have constraints of the form $\pm x_i \pm x_j \leq c$. They are mainly from [2, 13], 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. It is obvious to see that UTVPI $\subset$ TVPI and hence the complexity bounds of TVPI polyhedra apply to UTVPI polyhedra as well. But, as the dual of Bellman-Ford LP formulation is a UTVPI problem, general UTVPI systems can be solved with same quadratic complexity. The following result by [12] also confirms the same.

Lemma 2.3 [Feasibility on UTVPI] Feasibility of UTVPI stystems can be solved in $\mathcal{O}(mn)$ worst case time and in $\mathcal{O}(m+n)$ space.

UTVPI polyhedra also have successfully been used for various problems in abstract interpretation and verification like in [13]. Also, they have well supported tools like in Apron of the Astree project, and with support from PPL as well.

3. Convexity, Approximation and (U)TVPI

In this section, we defe a simple framework which gives us a 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.

Informally, the overall goal of this section is to begin with a polyhedron given in constraint form as $P = \{\mathbf{x}|A\mathbf{x} + \mathbf{b} \geq \mathbf{0}\}$ and show that simple approximations of P can be obtained by reasoning about over-approximations of the dual/polar "transpose" matrix $[A \mid \mathbf{b}]^T$.

To accomplish the above, in Section 3.1, we introduce some basic lemmas about convexity like homogenization and polarity. Next, in Section 3.2, we construct a simple framework for reasoning about under-approximations and over-approximations in a unified manner. Finally, in Section 3.3, we show how the above construction could be used to define a per-constraint approximation scheme which helps to obtain TVPI approximations of the non-TVPI constraints in the input polyhedron.

3.1 Some background in convexity

The overall goal of this section is to introduce some convexity concepts and relate them to TVPI under-approximations. Much of the math here can be verified from standard books like [15, 22].

3.1.1 Homogenization and Polarity

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 3.1 [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 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 $m \times n$-system (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. Also note that homogenizing is a process that can be reversed by special marking of the dimensions. It should also be noted that homogenization is a rather trivial process, which can be constructed in linear time, and routinely done so in libraries like PIP/PolyLib. It however needs to be mentioned because this paper deals with (U)TVPI constraints, which are defined to be having at most two non-zero coefficients in the non-homogenizing dimensions.

The following is the definition of polar of a polyhedron.

Definition 3.2a [Polarity] For $P \in \mathbb{R}^d$, the polar set is defined by

$$P^* = \left\{ \mathbf{c} \in (\mathbb{R}^d)^* : \mathbf{c}\mathbf{x} \leq 1 \text{ for all } \mathbf{x} \in P \right\} \subseteq (\mathbb{R}^d)^*$$

In standard books like [15], some of the theorems of polarity assume that origin is an internal point of a polyhedron, namely that $\mathbf{0} \in \mathrm{int}(P)$. For ease of notation, we use subscript o to denote that kind of pre-supposition on the polyhedra: P_o is a polyhedron such that $\mathbf{0} \in \mathrm{int}(P)$. The polyhedron which satisfies this restriction can be expressed in $\mathcal{H}$-form as $P_o = \{\mathbf{x}|A\mathbf{x} + \mathbf{b} \geq \mathbf{0}; \mathbf{b} \geq \mathbf{0}\}$. (Note the additional non-negativity restriction on $\mathbf{b}$.)

For a polyhedral cone C however, there is no necessity of origin being an internal point, as a polyhedral cone is always homogeneous ($\mathbf{0} \in C$).

The following are some properties of the polar.

Definition 3.2b [Polarity Properties: Polyhedra] For $P_o^* \in \mathbb{R}^d$, whose vertices are columns of the matrix V and whose rays are the columns of the matrix Y, the following are equivalent:

$$P_o^* = \text{conv}(\mathbf{0}, V) + \text{cone}(Y) \iff$$
$$P_o = \left\{\mathbf{x} \in \mathbb{R}^n | V^T\mathbf{x} \leq \mathbf{1}; Y^T\mathbf{x} \leq \mathbf{0}\right\}$$

The above correspondences give rise to some well known polarity properties namely that: the vertices (respectively, edges, and facets) of the primal correspond to the facets (respectively, ridges, and vertices) of the polar. (A facet is a $(n-1)$-dimensional face.) Further, the constraints of a polyhedron correspond to vertices/rays of the polar. We would be using these in the proofs of this paper.

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

A polyhedron can be represented in either of $\mathcal{H}$-form or $\mathcal{V}$-form. But, there is certain naturality in describing the primal in $\mathcal{H}$-form as polyhedra that occur in polyhedral compilation are almost always described that way. This means that the polar can be described, using a straightforward linear time construction, in $\mathcal{V}$-form. Converting the primal to $\mathcal{V}$-form or the polar to $\mathcal{H}$-form is however costly as it involves a call to Chernikova algorithm. In the following lemma, we will implicitly be using the above *dual* interpretation, *without* making an actual call to Chernikova. This dual interpretation is different from the dual representation in libraries like PolyLib, where a polyhedron (in either primal or dual space) is described using both $\mathcal{H}$-form and $\mathcal{V}$-form. We have the following lemma:

Lemma 3.1.2 [(U)TVPI and (U)TCPV] For a TVPI-polyhedron, the polar has vertices and rays which have not more than two non-zero components in the non-homogenizing dimensions. For a UTVPI-polyhedron, the polar has vertices and rays 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.*

Note that in the above lemma, as we are using $\mathcal{H}$-form (U)TVPI in primal, it implies that by a straightforward construction, the polar is in $\mathcal{V}$-form.

3.1.3 Polar of Polar

The proofs of the following can be found from Schrijver [15] and hence are omitted.

Lemma 3.1.3a [Polar of Polar and Equality] (i) $(P_o^*)^* = P_o$, (ii) $(P^*)^* = P \cup \{\mathbf{0}\}$, and (iii) $(C^*)^* = C$.

Lemma 3.1.3b [Polarity and Inclusivity] For any two Polytopes P_o and Q_o, $P_o \subseteq Q_o \Leftrightarrow P_o^* \supseteq Q_o^*$.

The above lemmas can obviously be extended for any general polyhedra and for cones as well.

3.1.4 Under-Approximation and Over-Approximation

The following corollary can be derived from the above lemmas.

Corollary 3.1.4 [Toggling between OA and UA] (i) $(\text{OA}(P_o^*))^* \subseteq P_o$, and (ii) $(\text{OA}(C^*))^* \subseteq C$

The above corollary means that by taking the polar of a polyhedron and taking the resultant's Over-Approximation (OA) one obtains a polyhedron whose polar is an UA of the original polyhedron. Hence, the first half above corollary can be used to find the UA of a polyhedron by converting into polar space and taking the OA of its polar polyhedron.

3.2 A construction for Under-Approximation

Using the theorems of the previous section, one can reason about UA by talking about OA in the dual/polar space. But, using OA to find valid UA using duality-properties, means that one has to deal with the many cases of the polyhedron being bounded or not, whether it is homogenized or not (like in Lemma 3.1.3), and whether origin is an internal point or not (like in Corollary 3.1.4.i). Further, the Farkas constraint polyhedra (even the per-dependence ones) that are obtained in polyhedral analysis do not necessarily have origin as an internal point. Also, they are never bounded. So, what is needed is a single framework that deals with all these cases.

In this section, we create a simple construction, using which 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. The construction uses homogenization and conical polarity and leads to all the above cases being handled in a unified and simple manner. The construction is the mathematical framework that gives us the basis to prove that the UAs we later construct in Section 4 are valid approximations.

The super-scripts $\mathcal{H}$ and $\mathcal{V}$ show whether the particular polyhedron is in constraint or generator form respectively.

$$P^{\mathcal{H}} \xrightarrow{\text{homog}} C^{\mathcal{H}} \xrightarrow{\text{polar}} K^{\mathcal{V}} \xrightarrow{\text{OA}} K_a^{\mathcal{V}} \xrightarrow{\text{depolar}} C_a^{\mathcal{H}} \xrightarrow{\text{dehomog}} P_a^{\mathcal{H}}$$

The following is some notes on the above construction:

- [Step 0] $P^{\mathcal{H}}$: Input P, the Polyhedron to be under-approximated in $\mathcal{H}$-form.
- [Step 1] $P^{\mathcal{H}} \xrightarrow{\text{homog}} C^{\mathcal{H}}$: Compute Cone C by homogenizing P. Namely, $C = \text{Homog}(P)$. C is a cone in $\mathcal{H}$-form.
- [Step 2] $C^{\mathcal{H}} \xrightarrow{\text{polar}} K^{\mathcal{V}}$: Compute $K = C^*$, the Polar cone of the cone C. K is a cone in $\mathcal{V}$-form.
- [Step 3] $K^{\mathcal{V}} \xrightarrow{\text{OA}} K_a^{\mathcal{V}}$: Compute K_a, the Over-Approximation of K such that $K \subseteq K_a$. K_a is a cone in $\mathcal{V}$-form.
- [Step 4] $K_a^{\mathcal{V}} \xrightarrow{\text{depolar}} C_a^{\mathcal{H}}$: Take the polar cone (DePolar) of K_a to result in C_a, which will be a cone in $\mathcal{H}$-form.
- [Step 5] $C_a^{\mathcal{H}} \xrightarrow{\text{dehomog}} P_a^{\mathcal{H}}$: Dehomogenize C_a to result in P_a, which will be a Polyhedron in $\mathcal{H}$-form.

In Step 0, P could be a general polyhedron in $\mathcal{H}$-form, possibly in non-minimal form (having redundant constraints). Step 1 follows from Definition 3.1. Here, the dimensions have to be marked as *homogenizing* and *non-homogenizing* dimensions so that it can help in approximation (Step 3), as well as in the de-homogenization (Step 5) later. No new homogenization is needed for performing this step other than what is performed in standard libraries. As mentioned earlier, this is needed in particular because the subject of this paper is (U)TVPI systems, which by definition cannot directly operate on cones without this special marking. Step 2 follows from Definition 3.2.

In the approximation process in Step 3, the Over-Approximation K_a has to satisfy (U)TCPV properties, so as to respectively obtain the (U)TVPI approximation of P. More particularly, for K_a to be a TCPV over-approximation of K, it should have not more than two non-zero components in the non-homogenizing dimensions. Further, for K_a to be a UTCPV over-approximation of K, K_a should be a TCPV vector and the non-zero components should be from $\{-1, +1\}$ (Lemma 3.1.2). In the approximation, one can neither throw away the offending dimensions, not can throw away the offending vectors themselves (ones that are not (U)TCPV), as the remaining vectors may result in an under-approximation of P. If K is being approximated on a per-constraint basis, the OA process

could be such that K_a has multiple vectors for each vector of K. The result of Step 3 is $K_a = \mathrm{OA}(K)$. Steps 4 and 5 are reverse of Steps 2 and 1 respectively.

In the above, P_a (respectively C_a and K_a) are approximations of P (respectively C and K). More particularly, we have the following theorem:

Theorem 3.2 [Approximations and the UA-OA Construction] (i) $K_a \supseteq K$, (ii) $C_a \subseteq C$ and (iii) $P_a \subseteq P$.

Proof: In (i) $K_a = \mathrm{OA}(K)$ by construction. In (ii), $C_a = \mathrm{UA}(C)$, because of Corollary 3.1.4.ii. Finally, in (iii) $P_a = \mathrm{UA}(P)$ because of Corollary 3.1.4.i.

The above construction can be done in linear time and can be considered to be a reinterpretation of the input polyhedron in $\mathcal{H}-$form so that the validity of the UA can be proved, and simple scalable algorithms could be obtained for (U)TVPI approximation. In the above construction, using homogenization leads to taking care of origin being an internal point. And, using polarity reduces the UA problem to the OA problem. But, one cannot swap the two steps of homogenization and polarity (swapping Steps 1 and 2 and following it by swapping Steps 4 and 5), for example, by taking the polar of P and then taking the resultant's homogenization. Doing so means that one has to "chop away" the origin from the polar of the resultant's OA as in Lemma 3.1.3a.ii. Such a process involves the generator representation and is costly.

Observe that K in the above construction is in $\mathcal{V}$-form, whose generator vectors are columns of the following matrix:

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

In matricial form, K is of size $(n+1)\times(m+1)$. K could thus be seen to be a transpose of Equation 1. K can also be written as the following:

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

The objective of Step 3 remains to find the TCPV-OA of Polyhedron K given in $\mathcal{V}$-form. This problem can be seen as trying to find a cone K_a such that:

$$K \in \mathrm{cone}\,(K_a) \tag{2}$$

By this equation, if each column of K 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 TCPV vector, then the corresponding P_a would be a TVPI approximation of P.

3.3 Approximation Scheme for TVPI-UA using TCPV-OA

In this section, we will formulate the sufficient conditions for a TVPI approximation such that it is a valid UA of an arbitrary polyhedron. In fact, with the construction in the previous section, we do have all the necessary ammunition to construct an approximation scheme which allows us to built TVPI-UAs of general polyhedra using TCPV-OA as a mathematical intermediary.

In the rest of this section, for ease of exposition, we assume that the polyhedron P is in 3-dimensions ($n = 3$). The latter polyhedra (also called as 3VPI polyhedra) are general enough to represent any arbitrary polyhedron. For polyhedra that are not 3VPI, a well known reduction[2] exists that transforms any arbitrary polyhedron into its equivalent 3VPI polyhedron.

[2] According to [16], the transformation from general polyhedra to 3VPI polyhedra is said to have been suggested by R.Tarjan and is much similar to the reduction of an arbitrary n-variable boolean satisfiability ("SAT") problem to a 3SAT problem. This transformation is not an approximation, which is the subject of this paper.

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

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

It can be noticed that each column of the matrix T^j is a TCPV vector and hence when it is subjected to the reverse transformation of Steps 4 and 5 as discussed in the earlier section would yield a set of TVPI constraints. The above is what we call as an *approximation scheme*. In this scheme, a non-TVPI constraint $a_{j,1}x_1 + a_{j,2}x_2 + a_{j,3}x_3 + b_j \ge 0$ in the original system is replaced by the set of constraints $\mathrm{UA}(x_1, x_2, x_3) = \{t_1^{j,1}x_1 + t_1^{j,2}x_2 + p_1^j \ge 0; t_2^{j,1}x_1 + t_2^{j,3}x_3 + p_2^j \ge 0; t_3^{j,2}x_2 + t_3^{j,3}x_3 + p_3^j \ge 0\}$.

In the above approximation 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. 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:

$$\{t_1^{j,1} + t_2^{j,1} = a_{j,1}; t_1^{j,2} + t_3^{j,2} = a_{j,2}; t_2^{j,3} + t_3^{j,3} = a_{j,3}; \\ p_1^j + p_2^j + p_3^j = b_j\} \tag{4}$$

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 $K \in \mathrm{cone}\{T^1, T^2, \ldots, T^m\}$. With the above, we give the following theorem, whose proof builds on Theorem 3.2 and the arguments given in this section.

Theorem 3.3 [TVPI/TCPV and UA/OA] (i) If P is empty, then P_a is also empty. (ii) If P is non-empty, and P_a is non-empty, then it is always a valid TVPI UA of the original Polyhedron

The problem remains to find such a set of (t, p) variables. If they can be found satisfing the above context constraints, then the above scheme remains a valid approximation. 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 context constraints turns out to be non-linear.

In the next section however, we will propose 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, and then as a more formal 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 will use the framework developed in earlier section and develop worstcase polynomial time algorithms for obtaining (U)TVPI under-approximations of Polyhedra. Firstly, in Section 4.1, we will give a strongly polynomial time per-constraint algorithm for TVPI-UA. Then, in Section 4.2, we will give an LP based per-constraint weakly polynomial time algorithm for TVPI-UA. In Section 4.3 we will propose various ways in which the above LP based algorithm can be applied in the case when multiple non-TVPI constraints are present. In Section 4.4, we will give a strongly polynomial algorithm that gives a UTVPI-UA of

a TVPI polyhedron. Finally, in Section 4.5, we briefly sketch a parametrized UTVPI-UA approximation method.

4.1 The median method for TVPI-UA

In this section, we introduce a simple per-constraint heuristic using the framework developed in Section 3.2 and 3.3. The main idea of this approximation is (2), 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.

3-d case For 3d case, the 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 above means that for 3d case, the inequality $ax+by+cz+1 \geq 0$ is approximated by the set of inequalities $\{ax + by + \frac{2}{3} \geq 0; ax + cz + \frac{2}{3} \geq 0; by + cz + \frac{2}{3} \geq 0;\}$.

The intuition for the above approximation comes from Equation 3. In that equation, both the values of t-variables and p-variables are to be found out such that they satisfy Equation 4. As explained earlier, the t-variables have to be fixed by a choice so that linearity is satisfied. 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 this as 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 be generalized to the n-d case. Let n be the dimension of the original constraints and let s be the sparsity – number of non-zero variables (or number of non-homogenizing dimensions) – for a particular 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 obviously $n+1$ dimensional, and are q in number. 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 2 and 3 that the UA proposed in each case is a valid approximation.

Example 1 Let the input system be the triangular pyramid $\{x, y, z \geq 0; x+y+z \leq 1\}$. 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 $\{(0,0,0), (1,0,0), (0,1,0), (0,0,1)\}$.

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

In this section, for ease of exposition, we will primarily deal with 3d case. The higher dimensional extensions are straightforward.

In Section 4.1, we saw a per-constraint approximation scheme in which the system $\mathcal{S}(\mathbf{x}) = \{\mathbf{x}|\mathbf{a}_1\mathbf{x} \geq b_1; \ldots; \mathbf{a}_j\mathbf{x} \geq b_j; \ldots; \mathbf{a}_m\mathbf{x} \geq b_m;\}$, with $\mathbf{a}_j\mathbf{x} \geq b_j$ being a non-TVPI constraint is being approximated by the system $\mathcal{S}_1(\mathbf{x}) = \{\mathbf{x}|\mathbf{a}_1\mathbf{x} \geq b_1; \ldots; \text{UA}_1(\mathbf{a}_j\mathbf{x} \geq b_j); \ldots; \mathbf{a}_m\mathbf{x} \geq b_m;\}$, with $\text{UA}_1(\mathbf{a}_j\mathbf{x} \geq b_j)$ defined as the following: $\text{UA}_1(\mathbf{a}_j\mathbf{x} \geq b_j) = \{\mathbf{x}|a_{j,1}x_1 + a_{j,2}x_2 \geq \frac{2}{3}; a_{j,1}x_1 + a_{j,3}x_3 \geq \frac{2}{3}; a_{j,2}x_2 + a_{j,3}x_3 \geq \frac{2}{3};\}$. As seen earlier, the above median method is simple, but does not have any guarantee of ensuring that $\mathcal{S}_1$ is non-empty.

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 Equation 3.

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:

$$\mathcal{S}_2(\mathbf{x}) = \mathcal{S}_{HD}(\mathbf{x}, \mathbf{p}^j)|\mathbf{p}^j \tag{5}$$

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 (5) 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 an reduced example from Banerjee book ("Loop Transformations for Restructuring Compilers"), which has been used as a test case in FMLib[3]. The original system be $\mathcal{S}(x,y,z) = \{(x,y,z) | -z+3 \geq 0; x-z \geq 0; -y+z-1 \geq 0; -x+y+z+1 \geq 0\}$. It is clearly not a TVPI system as the fourth constraint is non-TVPI. The median method of approximating the fourth constraint gives 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 are substituted, we obtain $\mathcal{S}_2(x,y,z) = \{(x,y,z) | -z+3 \geq 0; x-z \geq 0; -y+z-1 \geq 0; -x+y+\frac{1}{2} \geq 0; -x+z \geq 0; y+z+\frac{1}{2} \geq 0\}$. The reader can verify the satisfiability of the two approximations $\mathcal{S}_1$ and $\mathcal{S}_2$.

The above method involces 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 an LP system, with too high increase in the number of dimensions. $\mathcal{S}_{HD}(\mathbf{x},\mathbf{p}^1,\ldots,\mathbf{p}^{m_k})$ could have upto $n + \sum_{l=1}^{m_k}\binom{\|\mathbf{a}_l\|}{2} = n + \sum_{l=1}^{m_k}\|\mathbf{a}_l\|(\|\mathbf{a}_l\|-1)/2$-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/elements in the non-TVPI constraints in $\mathcal{S}(\mathbf{x})$.

4.3.2 Iterative methods

We could however 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. The case when the original system is immediately updated, we call the method as *incremental* method. The case when the approximations of all non-TVPI constraints are found by constructing LP formulations on the same LP system each time, we call 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, each of the LP systems is of $\mathcal{O}(n+\widehat{\|\mathcal{S}_{m_k}\|}^2)$ -dimension and is not that high-dimensional when compared to the above one-shot formulation.

[3] http://www.cse.ohio-state.edu/ pouchet/software/fm

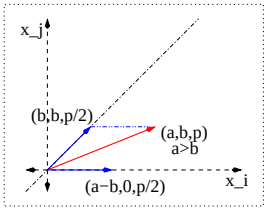

Figure 2. TCPV to UTCPV approximation

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 3.1.2, a vector in the polarity construction needs to be TCPV and to 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 TCPV original vectors in the polar space and computing a set of UTCPV vectors which OA the original TCPV vector would result 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.

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 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.

Example 4 Let $P = \text{cone}\{(1,1),(4,1),(1,2)\}$, with the $\mathcal{H}$-form of P being $P = \{1 \leq x \leq 4; 1 \leq y \leq 2; x+3y \leq 7\}$. 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 mostly similar to the parametrized TVPI approximation covered in Section 4.2, in the sense of searching for p variables instead of t's. We are not covering it in detail because of 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 later sections 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.

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 multiple constraints case, 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 worstcase times nonetheless.

5.3 Complexity of Finding a Solution

For the TVPI approximation, as the worstcase complexity of Hochbaum-Naor is $\mathcal{O}(mn^2 \log m)$ [11], 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, the theoretical worstcase complexity of solving UTVPI systems is $\mathcal{O}(mn)$ (if we use the Lahiri-Musuvathi algorithm [12]). the corresponding complexities would be $\mathcal{O}(\hat{s}^2 mn)$, and $\mathcal{O}(n^3 m)$ respectively. We will see later empirical evidence suggesting that a TVPI constraint is *always* a UTVPI constraint, making this method very attractive.

5.4 Preprocessing and Limitations

It can be noticed that our algorithms do not ask for any kind of preprocessing. In particular, we are not asking for removal of redundant inequalities, which is as hard as determining if the input system is feasible. Polyhedra in compilation are usually highly redundant. Also, there are lot of duplicate constraints. Compilers like PLuTo remove the duplicates by methods like textual matching which lead to a huge decrease in the number of constraints. We have not employed any of these techniques to remove redundancy.

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 the above polyhedron 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.

6. Experimental Results

In all our work, we used the widely used source-to-source polyhedral optimizer PLuTo (PLuTo-0.6). Our experimental framework used the well used PiPLib and PolyLib libraries, with an option to use FMLib as well.

The PIP calls from PLuTo originate from the functions: `dep_satisfaction_test` (DS), `get_dep_direction` (DIR) and `find_permutable_hyperplanes` (FPH).

Though there are 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* 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.

6.1 Polyhedral characteristics

Here we are referring to Table 1. The initial three columns refer to the SCoP size: L number of loops, S number of statements and D 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 4 columns 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 SCoP size. 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 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. An interval constraint, with only one non-zero coefficient, is of course UTVPI as well.

	SCoP			DS/DIR					FPH							Median		LP (indep)	
Bench	#L	#S	#D	P_{DS}	P_{DIR}	n	$\hat{m}$	$\hat{m}_t$	P_{FPH}	n	$\hat{m}$	$\hat{m}_t$	$\widehat{\|\mathcal{S}_{m_k}\|}$	$\hat{m}_a$	$\hat{m}_u$	YY	YN	YY	YN
adi	12	4	54	90	564	9	20	19	3	20	200	65	5	1844	614	0	3	0	3
corcol	12	6	14	38	194	9	17	16	3	22	22	13	5	130	77	1	2	0	3
covcol	13	7	26	41	228	15	25	24	3	24	18	14	4	29	55	3	0	3	0
dsyr2k	3	1	3	9	18	8	18	17	3	7	8	6	3	11	18	3	0	3	0
dsyrk	3	1	3	9	18	8	18	17	3	7	8	7	3	11	18	3	0	3	0
fdtd-2d	11	4	48	39	168	10	22	21	3	20	96	35	6	1010	367	0	3	1	2
gemver	7	4	13	29	161	6	13	12	2	14	21	15	4	48	47	0	2	2	0
jacobi-1d	4	2	10	16	88	7	14	13	2	10	32	14	5	232	104	0	2	0	2
jacobi-2d	6	2	14	21	84	9	19	18	3	12	65	15	7	1135	212	0	3	0	3
lu	5	2	10	12	60	11	23	22	3	10	35	17	5	232	106	0	3	0	3
matmul	3	1	3	9	18	10	21	21	3	9	9	7	4	20	24	3	0	3	0
mvt	4	2	11	31	68	6	13	12	2	9	20	12	3	46	52	0	2	2	0
seidel	3	1	27	37	162	12	24	23	3	8	33	15	5	168	179	0	3	3	0
ssymm	8	3	15	33	126	8	19	19	3	14	15	10	3	22	36	3	0	3	0
strmm	3	1	4	8	24	8	17	16	3	7	12	8	3	20	30	0	3	3	0
tmm	3	1	3	9	18	10	21	21	3	9	11	8	4	23	30	3	0	3	0
																19	26	29	16

Table 1. Problem size, Polyhedral and TVPI-ness characteristics, UA effectiveness

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 Gaussian and 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 the 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, which means that when the approximated system undergoes simplification and duplication removal techniques, it could lead to a much smaller system comparable to the one in m columns. Though we have not implemented this, we expect a good improvement by use of this technique: namely that the resultant approximated system would be not more than twice-thrice bigger than the original system.

6.2 TVPI-UA performance

Here we are referring to the rightmost columns of Table 1. These columns refer to the median method discussed in Section 4.1 and to the LP based independent method discussed in Section 4.3.2 respectively. The latter, we have implemented only the overall Farkas system, one that is obtained after putting together all the per-dependence systems and after simplification of PLuTo, and not on a per-dependence basis.

The PLuTo calls of the FPH variety are for LexMin. In the current table, we discuss the results of feasibility 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 based independent method performs much better than the median method. The latter performs poorly (19 YY and 26 YN cases or 6 out of 16 PolyBench problems), but surprisingly well considering the simplicity of the approach. We expect the incremental method to have much better performance than the current independent method (29 YY and 16 YN cases or 10 out of 16 PolyBench problems).

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

Table 2. UA Code Performance

6.3 UA code performance

We are referring here to Table 2, limitting ourselves to a subset of the YY cases in the previous table that our current implementation could hanle; 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 remain unknown, but we express some hope on PLuTo's loop distribution heuristic to break infeasible systems.

7. Related Work

Feautrier's scalable modular scheduling Feautrier's approach [8] starts with Gaussian elimination, and combines a Minkowski decomposition of P_e with parametric linear programming. We consider this approach as complementary to ours.

TVPI and UTVPI Sub-polyhedra have been widely employed [3] 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 [19]. Of the many classes of sub-

polyhedra TVPI and UTVPI have extensively been used [2, 13, 17], such as in the Astree project [3].

Approximations 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. Vivien and Wicker (in [20]) propose an algorithm to find the parallelepiped-OA of a 3d-polyhedron in vertex representation. Mine [13] (in Section 4.3) adapts the vertex method for finding the UTVPI-OA of a general polyhedron. Simon, King and Howe [17] (in 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 [11], for optimization of TVPI systems by Wayne [21], and for feasibility of UTVPI systems by Lahiri and Musuvathi [12]. We are not aware of any existing implementations of the above algorithms, though UTVPI feasibility can be solved using existing implementations of the well understood Bellman-Ford algorithm.

8. Conclusions and Future Work

We have presented a case for 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.

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 median method takes a strongly polynomial time, while the linear-programming methods take a weakly polynomial time but allows for better control of the feasibility of the under-appoximation. Indeed, the most difficult unsolved 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. It is unlikely that a strongly polynomial method exists to offer this guarantee, as it would then lead to a strongly polynomial feasibility test for general polyhedra. But we are looking for heuristics with quadratic time complexity.

Our experimental results show that the TVPI constraints are extremely likely to be UTVPI as well, and we suggested a direct under-approximation method into UTVPI rather than a two-step approximation through TVPI. This enables the Bellman-Ford algorithm for feasibility testing as well as for objective function optimization, reducing the complexity to quadratic time.

Acknowledgments We are grateful to Paul Feautrier for the tremendously helpful suggestions and feedback. We are also thankful to the reviewers of our previous IMPACT 2011 work [19], and to Cédric Bastoul and Armin Größlinger for their feedback at different stages. Further, the detailed comments from the IMPACT 2012 reviewers really improved the presentation.

References

[1] 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.

[2] 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.

[3] 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.

[4] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan. A practical automatic polyhedral parallelizer and locality optimizer. In R. Gupta and S. P. Amarasinghe, editors, *PLDI*, pages 101–113. ACM, 2008.

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

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

[7] P. Feautrier. Some efficient solutions to the affine scheduling problem: I. one-dimensional time. *Int. J. Parallel Program.*, 21:313–348, October 1992.

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

[9] 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.

[10] T. Grosser, H. Zheng, R. A, A. Simbürger, A. Größlinger, and L.-N. Pouchet. Polly - Polyhedral Optimization in LLVM. In *First International Workshop on Polyhedral Compilation Techniques (IMPACT 2011), in conjunction with CGO 2011*, Chamonix, France, Apr 2011.

[11] 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.

[12] S. K. Lahiri and M. Musuvathi. An efficient decision procedure for UTVPI constraints. In B. Gramlich, editor, *FroCos*, volume 3717 of *Lecture Notes in Computer Science*, pages 168–183. Springer, 2005.

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

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

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

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

[17] A. Simon, A. King, and J. M. Howe. The Two Variable Per Inequality Abstract Domain. *Higher Order and Symbolic Computation*, 2010.

[18] 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.

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

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

[21] 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.

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