A Parallel Cylindrical Algebraic Decomposition Algorithm for Quantifier Elimination on Real Closed Fields

Hari Krishna Malladi and Ambedkar Dukkipati
Indian Institute of Science
Bangalore
India
{harikrishnamalladi,ambedkar}@csa.iisc.ernet.in

ABSTRACT

In this paper we propose a parallel approach to quantifier elimination on real closed fields through a modification in the cylindrical algebraic decomposition (CAD) algorithm to accommodate parallelism. In this approach the speed-up due to parallelism obtained on an input formula is a property of that formula itself. Hence the time complexity of algorithm depends on not just the size of the input but on the input self. We classify prenex formulae depending on the amount of input-based parallelism that can be obtained from it based on which we obtain some insights into the complexity of the algorithm. We demonstrate performance of the proposed algorithm with experimental results.

Keywords

Cylindrical Algebraic Decomposition, parallelism, quantifier elimination

1. INTRODUCTION

The study of real algebraic geometry deals with the study of the real roots of an equation as, more often than not, the real roots are the most desired solutions to a system of equations. It is important to note that for some problems there are no counterparts in complex algebraic geometry. Three such examples are: (i) Hilbert's 16^{th} problem, (ii) Hilbert's 17^{th} problem, and (iii) Tarski-Seidenberg principle.

Given a first-order formula with both quantified and quantifier - free variables, the process of finding an equivalent first-order formula in these quantifier-free variables is called quantifier elimination. When the first-order formula is a boolean combination of polynomial equations and inequalities, we can consider quantifier elimination as a problem in real algebraic geometry.

The algorithm specified by Tarski [12] for quantifier elimination is highly resource intensive. So newer and more efficient algorithms have come up and replaced it. The fact that a semialgebraic set can be divided into a finite number of semialgebraic sets (Tarski-Seidenberg principle) has led to Cylindrical Algebraic Decomposition (CAD) [4], which eventually became the standard algorithm for quantifier elimination over real closed fields. The time complexity of CAD algorithm is doubly exponential in the number of variables (both quantified and quantifier-free).

It is a recursive algorithm, which has a sequence of projections, followed by a sequence of constructions, leading to the solution. CAD has undergone an extensive array of developments, such as Hong's Partial CAD [5], Hong's projection operator [9], Scott McCallum's projection operator [1], application of Gröbner Bases to CAD [3] and a first step towards parallelism in CAD [11].

This makes quantifier elimination through CAD impossible for a wide range of real-world applications. The goal of the paper is to address this issue and expand the class of problems where CAD could be used to perform quantifier elimination using a reasonable amount of resources.

The present work builds on [11] and extends the idea of parallelism to the entire algorithm. We show that parallelism can be introduced in various aspects of CAD. They are, (i) input to CAD can be made into independent units, (ii) projections can be made parallel, and (iii) constructions can be made parallel to the maximum possible extent.

The input to CAD cannot always be split into independent units, because of the constraints imposed by first-order logic itself. This motivate us to introduce Skolemization to eliminate the existential quantifiers. An ad-hoc improvement to these schemes is the introduction of chains of substitutions, which have the potential to reduce the number of variables in some polynomials, keeping the final truth value constant. These schemes benefit from the fact that speed-up happens at a doubly exponential rate. Reduction of just one variable reduces the time complexity greatly. These improvements are illustrated in the experimental results.

The paper is organized as follows. The preliminary notations and the original CAD algorithm are briefly introduced in Section 2. The idea of separability and the equi-satisfiable properties of substitutions are described in Section 3. The parallel CAD algorithm itself is described in Section 4. Empirical results for separability and substitutability are described in Section 5. A few directions sought for future research are mentioned in Section 6.

ISSAC '12 Grenoble, France

2. PRELIMINARIES

2.1 Basic Notation

Let $\mathbb{k}[x_1...x_n]$ denote the set of all polynomials in variables, $x_1...x_n$ and coefficients from the field $\mathbb{k}$. The base

field, $\Bbbk$ is assumed to be the set of real numbers, $\mathbb{R}$ throughout the paper. A field, $\Bbbk$ is called *algebraically closed* if every polynomial, $p \in \Bbbk[x_1, x_2, ...x_n]$ with coefficients from $\Bbbk$ has a root in $\Bbbk$ itself. The best known example is the set of complex numbers, $\mathbb{C}$.

DEFINITION 2.1. *A formula of the form*

$$(Q_1 x_1)...(Q_n x_n)[\psi(y_1...y_m, x_1...x_n)]$$

is called a prenex formula, where $Q_i \in \{\exists, \forall\}$ and ψ is a quantifier-free formula in $x_1, ...x_n, y_1, ...y_m$.

DEFINITION 2.2. *$V \subseteq \mathbb{R}$ is said to be a semialgebraic set if $\exists f_1...f_s \in \mathbb{R}[x_1...x_n]$ and $g_1...g_t \in \mathbb{R}[x_1...x_n]$ such that $V = \{(a_1...a_n) \in \mathbb{R} : f_i(a_1...a_n) = 0, i = 1...s, g_j(a_1...a_n) \geq 0, i = 1...t\}$.*

A connected subset of $\mathbb{R}^n$ is called a *region* in $\mathbb{R}^n$. Consider a region R and functions $f_i : \mathbb{R}^n \longrightarrow \mathbb{R}$ satisfying, $f_1 < f_2 < ... < f_l$. This ordering ensures that these functions do not intersect each other. A *f_i-section* is the graph of the function f_i. In other words, it's the set containing all points of the form (a, b), where a $\in \mathbb{R}^{n-1}$ and $b = f(a)$. An *(f_i,f_{i+1})-sector* is the set of all points (a, b), where a $\in \mathbb{R}^{n-1}$ and $f_i(a) < b < f_{i+1}(a)$. A *cylinder* over a region R is the set of all points (a, b), where $a \in R$ and $b \in \mathbb{R}$. A *stack* over a region R is the collection of sections and sectors that occur in the cylinder over R. A partition of $\mathbb{R}^n$ into semialgebraic partitions is called an *algebraic decomposition* of $\mathbb{R}^n$.

DEFINITION 2.3. *A partition of $\mathbb{R}^n$ satisfying the following two properties is called a cylindrical algebraic decomposition:*

1. *If $n = 1$, then the CAD is a set of points and open intervals.*
2. *If $n > 1$, every region of the CAD of $\mathbb{R}^{n-1}$ has a stack over it, which is a disjoint subset of some of the regions of the CAD of $\mathbb{R}^n$.*

A set of polynomials, F is said to be *sign invariant* on the region R iff no polynomial in F changes its sign anywhere on R.

DEFINITION 2.4. *A CAD of $\mathbb{R}^n$ in which each region is sign-invariant with respect to an input set of polynomials is called a sign-invariant CAD.*

The CAD algorithm has two phases in its description, namely projection and construction. Projection is specified by a projection operator.

2.2 Cylindrical Algebraic Decomposition Algorithm

This algorithm takes as input, a set of polynomials in n variables and generates a cylindrical algebraic decomposition of the n-dimensional real space. This algorithm works recursively to produce a series of projections on lesser dimensions and builds the CAD of each dimension while returning the recurring functions. An outline of CAD algorithm [7] is presented below.

1. Project each polynomial onto lesser dimensions. Polynomials in n variables are taken as input and a set of polynomials in $n-1$ variables is obtained as output, upon applying a projection operator. This process is continued till a set of univariate polynomials is obtained.
2. These univariate polynomials can easily be solved (using Sturm's theorem iteratively or otherwise). The roots of these univariate polynomials and the intervals between them are taken as the regions in the CAD of one dimensional real space. Designate a randomly selected point in each region as a sample point.
3. For the two dimensional CAD, substitute the sample point found in each one-dimensional region, R in each of the polynomials in the 2-dimensional projection to again get univariate polynomials. Find the roots of these polynomials and create two dimensional regions that form the stack over the region R. Designate sample points for these regions as well. Suppose we are given a CAD for k-dimensional real space, we use the projected set for $k+1$-dimensions and substitute the sample points of the k-dimensional CAD in them to get univariate polynomials. These can now be solved to get $k+1$-dimensional CAD.

Now, this information can be used by another recursive algorithm to perform quantifier elimination [6]. Assume that the variables $x_1, ..., x_k$ are quantifier-free and $x_{k+1}, ..., x_n$ have quantifiers, and let Q_i be the corresponding quantifier for $i \in \{k+1, ..., n\}$. Now we list the QE algorithm.

$$\mathbf{QE}((D_1, ..., D_n), (F_1, ..., F_l))$$

Input: The CADs of $\mathbb{R}^1$ to $\mathbb{R}^n$ $(D_1, ..., D_n)$ and the formulae describing the regions of the CAD of $\mathbb{R}^k$ $(F_1, ..., F_l)$.
Output: The quantifier-free formula equivalent to the input prenex formula.

1. For $k \leq i < n$, we have
   (a) If Q_{i+1} is $\exists$, then $C \in D_i$ is valid if at least one region in the stack over C is valid.
   (b) If Q_{i+1} is $\forall$, then $C \in D_i$ is valid if all regions in the stack over C are valid.
2. A region, C in D_n is valid if $\psi(t_C)$ is TRUE, where t_C is the sample point of that region.
3. Obtain the cells of D_k which are TRUE. The disjunction of the formulae of these cells is the required quantifier-free formula and is returned.

The projection operator takes as input a set of polynomials in n variables and returns another set of polynomials in $n-1$ variables. Assume $PSRC_j(P, Q)$ denotes the j^{th} principal subresultant coefficient of P and Q.

Collins' projection operator takes as input the polynomials $A = \{A_1, ..., A_m\}$. Then it returns (i) The coefficient of each and every term of each A_i, (ii) $PSRC_i(U_j, U_j')$ for every i and j and, (iii) $PSRC_i(U_j, U_k)$ for every i and $j < k$. U_i represents every element in the reducta set of A_i.

3. SOME OBSERVATIONS THAT LEAD UP TO PARALLELISM

3.1 Separability

The concept of separability is used when we need to convert one instance of quantifier elimination into multiple instances of quantifier elimination.

DEFINITION 3.1. *For prenex formulae, f and g, if we have,*

$$d(x_1, ...x_n) = (Q_r x_r)...(Q_n x_n)\ f(x_{f_1}, ..., x_{f_k})\ C\ g(x_{g_1}, ..., x_{g_l}),$$

where $Q_i \in \{\exists, \forall\}, \forall i \in \{r, ..., n\}$ and $C \in \{\vee, \wedge\}$, we say that g is separable from d if, supposing $F = \{x_{f_1}, x_{f_2}, ..., x_{f_k}\}$ and $G = \{x_{g_1}, x_{g_2}, ..., x_{g_l}\}$ and $H = F \cap G$ and $H = \{x_{h_1}, x_{h_2}, ..., x_{h_j}\}$,

1. *C is $\vee$, then every $x \in H$ should not be the universal quantifier, $\forall$.*
2. *C is $\wedge$, then every $x \in H$ should not be the existential quantifier, $\exists$.*
3. *g does not evaluate to TRUE or FALSE upon quantifier elimination, and g has quantifier-free variables.*

Essentially, we would be converting

$$(Q_{k+1}x_{k+1})...(Q_n x_n)[\psi(x_1...x_n)],\ Q_i \in \{\exists, \forall\}$$

into two (or more, if recursively applied) quantifier elimination problems,

$$X\ C\ Y,$$
$$X = (Q_{k+1}x_{k+1})...(Q_n x_n)[\psi'(x_1...x_n)] \text{ and}$$
$$Y = (Q_{k+1}x_{k+1})...(Q_n x_n)[\psi''(x_1...x_n)]$$
$$Q_i \in \{\exists, \forall\}, C \in \{\wedge, \vee\} \text{ and}$$
$$\psi'(x_1...x_n) C \psi''(x_1...x_n) = \psi(x_1...x_n).$$

The theory of separability seeks to address the conditions on ψ' and ψ'', to allow such a splitting.

The projection and construction phases of CAD algorithm are inherently sequential. A projection (construction) of a set of polynomials in k variables happens only after the corresponding set of polynomials in $k+1$ dimensions ($k-1$ dimensions for constructions) has been projected (constructed). Hence, none of the major phases of the original CAD algorithm can be executed in parallel. We can separate the input, leading to multiple instances of the CAD algorithm with lesser number of variables.

Note that the concept of separability is derived from the following two well known properties of quantifiers:

1. $(\forall x)[f(x) \wedge g(x)] \leftrightarrow (\forall x)f(x) \wedge (\forall x)g(x)$.
2. $(\exists x)[f(x) \vee g(x)] \leftrightarrow (\exists x)f(x) \vee (\exists x)g(x)$.

In the above two cases, we effectively separated $f(x)$ (or $g(x)$) from the formula on the left side. We now need to generalize this notion of separation to an n variable case.

THEOREM 3.2. *Assume that*

$$d(x_1...x_n) = (Q_r x_r)...(Q_n x_n) f(x_{f_1}...x_{f_k}) C g(x_{g_1}...x_{g_l})$$

where $C \in \{\vee, \wedge\}$ and $Q_i \in \{\forall, \exists\}$, is a prenex formula. If f is separable from g and

$$X = (Q_r x_r)(Q_{r+1}x_{r+1})...(Q_n x_n)[f(x_{f_1}, x_{f_2}, ..., x_{f_k})]$$

and

$$Y = (Q_r x_r)(Q_{r+1}x_{r+1})...(Q_n x_n)[g(x_{g_1}, x_{g_2}, ..., x_{g_l})]$$

then,

$$d(x_1, ...x_n) = X\ C\ Y.$$

PROOF. Suppose $F = \{x_{f_1}, x_{f_2}, ..., x_{f_k}\}$ and $G = \{x_{g_1}, x_{g_2}, ..., x_{g_l}\}$ and $H = F \cap G$ and $H = \{x_{h_1}, x_{h_2}, ..., x_{h_j}\}$. Then we have two cases, based on C:

1. C **is** $\vee$**:** The definition of separability gives us that g can be separated if every $x \in H$ does not have the universal quantifier. We know that d is satisfied if we choose values for $G = \{x_{g_1}, x_{g_2}, ..., x_{g_l}\}$ in such a way that g is satisfied. Some variables, namely $H = \{x_{h_1}, x_{h_2}, ..., x_{h_j}\}$, are common between f and g, and do not have the universal quantifier. If we are able to choose values for the variables in G or F to satisfy d, we implicitly chose values for the variables in H to satisfy d. This choice is feasible due to the absence of universal quantifiers for the variables in H.
2. C **is** $\wedge$**:** This is the dual of the previous case.

□

3.2 Parallelism in Projections and Constructions

Projections and constructions form the recursive steps of the CAD algorithm. So, it's natural to ask whether these can be parallelized. A step in this direction has been taken by Saunders et al. [11]. This paper proposed a way to find parallelism in the construction phase. We borrow this idea, suitably modified, and call it 'Pipelined Constructions'. A recurring feature in most of the projection operators is the fact that they deal with polynomials pair-wise. This can be utilized to work in a parallel fashion.

The projection operators utilize polynomials pair-wise to compute resultants and discriminants. This can be done in a parallel fashion, using the divide and conquer paradigm. Assuming there are n polynomials in r variables, we could assign $\frac{n}{2}$ of them to one processor and $\frac{n}{2}$ to another processor. This goes on recursively till we get each processor working on just two polynomials. These now generate the discriminants and resultants simultaneously. Assuming we have found the projections of $X = \{a_1, a_2, ..., a_{\frac{n}{2}}\}$ and $Y = \{a_{\frac{n}{2}+1}, ..., a_n\}$ separately, we now combine them by pairing up each polynomial from X with every polynomial from Y. This procedure ensures that projections are done at effectively at a logarithmic function of the time taken by the original projection operator. This way of projecting shall be referred to as *Parallel Projections*.

Construction phase parallelism depends on many issues. Clustering prevents any parallelism from entering construction phase as some, if not all the regions would be interdependent. Hence, we consider a case without clustering. This idea has been explored in [11]. Suppose the n-dimensional CAD has regions, $R = \{r_1, r_2, ..., r_k\}$. In the construction stage, the stack over each region is constructed iteratively. Suppose the stack over the region r_1 has regions, $R' = \{r_{1_1}, r_{1_2}, ...r_{1_l}\}$. The construction of the stack over r_1 is followed by the construction of the stack over the region r_2. The observation to be made is that the construction of the stack over r_2 is independent of the construction of stacks

over the regions of R'. These can be done in parallel. This is analogous to the stages in a CPU pipeline. Hence, this work shall refer to this process as *Pipelined Constructions.*

3.3 Substitutability

Most real-world problems have more than one polynomial in their prenex formulae, connected by boolean connectives. The observation to be made here is that, in most cases, none of these polynomials have all the variables that are used in the whole formula. The time complexity of the CAD algorithm has an upper bound of $(mn)^{k^r}d^k$, where m is the number of polynomials occurring in the input formula, r is the total number of variables, n is the maximum degree of any polynomial in any variable, k is some constant and d is the length of any integer coefficient of any polynomial. We observe here, that in general, if we could decrease r by even one variable, but add more number of polynomials to m, we'd get a faster running algorithm. There are instances where the elimination of one variable makes a problem computable in reasonable amounts of space and in reasonable time. But, there's no closed form for the roots of quintic and higher degree equations, as proved by Niels Henrik Abel. Hence, we can use only the variables, which have a degree of at most 4 to perform substitutions.

In our definition of substitutability, we classify polynomials into two classes, namely (i) substitutor and (ii) substituend.

Substitutors are further sub-divided into two types, namely (i) substitutor-type I and (ii) substitutor-type II.

DEFINITION 3.3. *A substitutor-type I polynomial should form an equation and has to contain atleast one variable whose exponent is at-most 4.*

DEFINITION 3.4. *A substitutor-type II polynomial forms an equation and has a variable of exponent at-most 4 in one of its factors, and is not a substitutor-type I.*

DEFINITION 3.5. *A substituend polynomial does not come under the substitutor-types I and II.*

DEFINITION 3.6. *Assume that P is any polynomial and T is a substitutor-type I. We say that T is substitutable in P if, the process of replacing a variable in P by substituting it with other variables from T does not result in an increase in the number of variables in the polynomial P.*

Using the above defined terms and operation, we can now state the following lemma, which essentially states that substitutability produces an equi-satisfiable formula for a given prenex formula. We denote equi-satisfiability by the symbol '$\Longleftrightarrow$' in the following lemma.

LEMMA 3.7. *Assume a prenex formula,*

$$h(x_1, ...x_n) = f(x_1, ..., x_r, ..., x_n) \wedge g(x_1, ..., x_n),$$

where f and g are both prenex formulae. Then,

$$h(x_1, ...x_n) \Longleftrightarrow f'(x_1, ..., x_{r-1}, x_{r+1}, ..., x_n) \wedge g(x_1, ..., x_n),$$

where the polynomial f' is obtained by substituting x_r in f, using g.

PROOF. For any given n-tuple, $(x_1, ..., x_n)$, $g(x_1, ..., x_n)$ is either true or false. If it were true,

$$f(x_1, ..., x_n) \Longleftrightarrow f'(x_1, ..., x_{r-1}, x_{r+1}, ..., x_n)$$

. If it were false, $g(x_1, ..., x_n) \wedge f'(x_1, ..., x_{r-1}, x_{r+1}, ..., x_n)$ and $h(x_1, ...x_n)$ would both be false. Hence, $h(x_1, ...x_n)$ and $f'(x_1, ..., x_{r-1}, x_{r+1}, ..., x_n) \wedge g(x_1, ..., x_n)$ have the same truth table, and hence are equi-satisfiable. □

4. PARALLEL CYLINDRICAL ALGEBRAIC DECOMPOSITION

In this section we present a parallel CAD algorithm that is motivated by the observation we made above.

4.1 Preliminary Considerations

Skolemization is a process discovered by Thoralf Skolem [8], which replaces all existentially quantified variables by new functions, called Skolem functions, which depend on the presently quantified variables. This leaves only the universal quantifiers intact. When Skolemization is performed on a CNF formula, each term of the CNF becomes separable from the rest of the formula. This allows each term of the resulting CNF to be a separate parallel process. But, the newly introduced Skolem functions should also be accounted for, in the input. This may add to the number of variables, thus increasing the running time.

If separability was done before substitutions, we would not be considering the terms that would've become separable if some variables have been substituted for others. Thus, in the algorithms that follow, substitution is done prior to separations. This also derives from the fact that separability depends exclusively on the variables, and substitutability never adds new variables to an existing formula (it can only remove variables and replace them with expressions containing other variables that occur in that formula).

The proposed algorithms depend on the following definitions of Conjunctive and Disjunctive Normal Forms,

DEFINITION 4.1. *For polynomial equations and inequalities f_{ij} for some i, j, a prenex formula of the type,*

$$[(f_{11} \wedge f_{12} \wedge ... f_{1k_1}) \vee (f_{21} \wedge f_{22} \wedge ... f_{2k_2}) \vee ... \vee (f_{n1} \wedge f_{n2} \wedge ... f_{nk_n})]$$

is called a disjunctive normal form, and each $(f_{i1} \wedge f_{i2} \wedge ...f_{ik_i})$, for some i is called a 'term'.

DEFINITION 4.2. *For polynomial equations and inequalities f_{ij} for some i, j, a prenex formula of the type,*

$$[(f_{11} \vee f_{12} \vee ... f_{1k_1}) \wedge (f_{21} \vee f_{22} \vee ... f_{2k_2}) \wedge ... \wedge (f_{n1} \vee f_{n2} \vee ... f_{nk_n})]$$

is called a conjunctive normal form, and each $(f_{i1} \vee f_{i2} \vee ...f_{ik_i})$, for some i is called a 'term'.

4.2 Proposed Algorithms

We now list the proposed algorithms.

PARALLEL-CAD(A)
Input: A set of polynomials, A
Output: The CAD for $\mathbb{R}^1$ to $\mathbb{R}^n$, with respect to A

1. Perform parallel projections to A to obtain A_1 in $n-1$ variables.

2. Project the polynomials in the set A_i to get the set A_{i+1}. This continues till we obtain the set A_{n-1}, which consists of univariate polynomials.

3. Perform pipelined constructions on the set of projected polynomials, $\{A_1, ..., A_{n-1}\}$. We obtain the decompositions, $D_1, ..., D_n$ from these constructions, corresponding to the CAD for $\mathbb{R}^1$ to $\mathbb{R}^n$, with respect to A.

4. Return $D_1, ..., D_n$.

This algorithm is called simultaneously for each set of polynomials, derived from separations, after performing substitutions. These are described in the following algorithm.

PREPROCESSOR(f)

Input: A prenex formula containing a boolean combination of polynomial equations and inequalities, f

Output: Sets of polynomials, $S_1, ..., S_k$, where each S_i is a set of polynomials.

1. Consider the prenex formula, f. Convert f to a CNF, c, or to a DNF, d, whichever has the most number of terms. If it was converted to a DNF, complement it twice to get a complement of a CNF. Denote this CNF by c.

2. Identify the substitutor-type-I polynomials in c. Use them to substitute in all other polynomials and also among themselves. Retain the substitutor polynomials, which weren't substituted in, as part of the CNF.

3. Consider all substitutor-type-I polynomials as substitutor-type-II polynomials. Factorize them and substitute among the factors, along with other substitutor-type-II polynomials. The resulting formula shall be denoted by r.

4. Perform Skolemization on r to get r_s. Perform separations on r to get the prenex formulae, $r_1, ..., r_l$. If every term of r_s has at most as many variables as the number of variables in $\max(r_1, ..., r_l)$, use r_s for further computations, and denote it by r. Otherwise use r.

5. Assume r gives rise to k prenex formulae when separated. Collect these polynomials in sets, $S_1, ..., S_k$.

6. Return the sets $S_1, ..., S_k$.

The algorithm, PARALLEL-CAD(A) is executed for each $A \in \{S_1, ..., S_k\}$. These k instances of Parallel-CAD are run simultaneously. Quantifier elimination can be done by considering the connectives and the equalities and inequalities specified among the member polynomials of each set, S_i, $i \in \{1, ..., k\}$. The resulting k quantifier-free formulae are united using conjunctions.

4.3 Analysis of the Algorithms

In the following sections and subsections, 'structure' in a prenex formula refers to the dependence of the constituent polynomials on each other, resulting in conditions/constraints being imposed on the formula because of these polynomials.

For the analysis of the proposed algorithms, we use the classical notion of parallel complexity [10]. The parallel complexity analysis of PARALLEL-CAD can be studied using three parameters,

1. *Number of Processors:* The number of processors is assumed to be constant, denoted by an upper bound of K. This assumption is very rigid as it does not allow the increase in the number of processing cores. This way, we can study the effects of parallelism in a case where we obtain more number of partitions than the number of processors. This leads to an optimization problem.

2. *Parallel Time Complexity:* Assuming that we have a collection of independent parallel processes, the time taken by the longest process is called the parallel time complexity of the collection. A formula is separated only when none of the resulting partitions have all the variables present in the original formula. This ensures that, when separability happens, the parallel time complexity is exponentially lower than the original CAD algorithm. This reduction in time complexity happens even though the total work done by PARALLEL-CAD is greater than the original algorithm.

3. *Total Work Done:* Given a collection of parallel independent processes, the amount of work done by all the independent processes combined is called the total work done by the collection. Equational constraints in a prenex formula eliminate the need for many constructions, thus speeding up CAD. But, if these constraints were to be separated, then the resulting prenex formula without the constraints would perform unnecessary constructions, thus increasing the resource consumption. This would also occur with inequalities.

Separability is a property of the prenex formula. Separability might result in the loss of structure in a given prenex formula. By this we mean that the interdependencies among the formula's constituent polynomials are negated, resulting in wasteful and redundant computations, which would not have been required if the formula was not separated. We need a formal, quantitative definition of 'structure' in a given formula to study it's effects on the running time of the proposed algorithm.

DEFINITION 4.3. *The 'sharing factor' existing among two given prenex formulae, f and g, is the number of variables that are shared between f and g. It's denoted by $T_{f,g}$.*

The sharing factor essentially captures the distribution of variables among the constituent polynomials of a prenex formula. If formulae f and g have k variables in common, it implies that f imposes conditions on g in k dimensions. The sharing factor gives a measure of structure present in a formula, as the sharing of variables between polynomials causes interdependencies among them. Having defined the sharing factor, we now define the operation, separation, where the sharing of variables comes into consideration.

DEFINITION 4.4. *Consider a prenex formula, f, which has n variables, out of which, there are k quantified variables and $n-k$ quantifier-free variables. We say that the separation of f is the ordered pair (F, C), if we obtain a set of prenex formulae, $F = \{f_1, ...f_l\}$ and a set of boolean operators, $C = \{c_1, ..., c_{l-1}\}$, such that the following conditions hold.*

1. $f \iff f_1 c_1 f_2 ... c_{l-1} f_l$
2. *Assume that* $f_j = q_j f'_j, j \in \{1, ..., l\}$, *where* q_j *is the string of quantifiers and quantified variables,* $Q_1 x_1 ... Q_{j_k} x_{j_k}$, $Q_i \in \{\exists, \forall\}$, $i \in \{1, ..., k\}$, *and* f'_j *is the formula without the string of quantifiers. The condition for* (F, C) *to be a separation of* f *is,*

$$f \iff Q_1 x_1 ... Q_k x_k [f'_1 c_1 f'_2 c_2 ... c_{l-1} f'_l].$$

Loss of structure in a formula leads to redundant computations.

DEFINITION 4.5. *Consider a prenex formula,* f *and it's separation,* (F, C). *While executing PARALLEL-CAD on the set* F, *a computation is said to be redundant if it need not be performed while executing the original CAD algorithm on* f. *The redundancy ratio is given by* $\frac{redundant\ computations}{total\ computations}$.

The definition of separation defines separability as a property of the formula. In the remainder of this section, we assume that all prenex formulae are in variables, $x_1, ...x_n$. It is natural to classify prenex formulae into classes based on the number of partitions it can be separated into. Thus, we have the following definition.

DEFINITION 4.6. *A prenex formula is defined to be* k *- separable if and only if it can be separated into* k *prenex formulae, which are connected by boolean connectives. The set of all k-separable formulae shall be denoted by* S_k.

According to this definition, we can observe that $S_k \subseteq S_{k-1}$. The set of 1-separable formulae is the set of all prenex formulae, which shall be denoted by P. We assume that we have only a constant number of processors, and we denote this by K. This leads to a definition of the class of separable problems.

DEFINITION 4.7. *Consider a set* $S \subseteq P$. S *is called the 'separable class' if the following conditions hold.*

1. $S \subseteq S_2$
2. *If* f *is a prenex formula and* (F, C) *is it's separation,* $\forall f \in F$, $\exists x_i \in \{x_1, ..., x_n\}$ *such that* f *is independent of* x_i.

The speed-up obtained by a k-separable formula depends on the distribution of variables among the k separated formulae and their interdependencies. Thus, we need to define a subset of S_k, which has the most ideal distribution of variables.

DEFINITION 4.8. *The centre of* S_k *is the set of all prenex formulae in* n *variables, which can be separated into* k *formulae,* $f_1, ...f_k$, *each with* $\frac{n}{k}$ *variables and every variable is present in exactly one formula in* $f_1, ..., f_k$. *The centre of* S_k *is denoted by* C_k.

4.3.1 PARALLEL-CAD and Separable Class

The above defined terms help us in analysing the extent of improvement in running time and space a problem can possibly achieve when PARALLEL-CAD is used. The notion of the centre of separable class indicates the areas where the maximum amount of speed-up is possible. We now define the conditions which determine whether a given problem is closer to the centre than another problem or not. Denote the number of variables in a polynomial, p, by $\#(p)$.

DEFINITION 4.9. *Given two* k*-separable prenex formulae* f *and* g *and their separations,* (F, C_1) *and* (G, C_2), *we say that* f *is closer to the centre than* g *(denoted by* $f \preceq g$*) if*

1. $\max_i \#(f_i) < \max_j \#(g_j)$, $f_i \in F$ *and* $g_j \in G$, $1 \leq i, j \leq k$.
2. *If* $\max_i \#(f_i) = \max_j \#(g_j)$, *then* $\sum_{i,j} T_{f_i, f_j} \leq \sum_{m,l} T_{g_m, g_l}$, $f_i, f_j \in F$ *and* $g_m, g_l \in G$, $1 \leq i, j, m, l \leq k$.

In the above definition, g is said to be farther away from the centre, compared to f.

The time complexity of CAD algorithm is bounded above by $(mn)^{k^r} d^k$, where m is the number of polynomials occurring in the input formula, r is the total number of variables, n is the maximum degree of any polynomial in any variable, k is some constant and d is the length of any integer coefficient of any polynomial. This motivates the definition of comparability.

DEFINITION 4.10. *Consider two prenex formulae,* f *and* g. *The maximum degrees of any polynomial in any variable in* f *and* g *are denoted by* n_1 *and* n_2. *The number of variables in* f *and* g *are denoted by* r_1 *and* r_2. *Without loss of generality, assume that* $r_1 > r_2$. *Consider a function,* $\alpha : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, *such that,* $\alpha(n, r) = (mn)^{k^r} d^k$. *The formulae,* f *and* g *are said to be comparable if* $\alpha(n_1, r_1) > \alpha(n_2, r_2)$.

We now characterize the relationship between PARALLEL-CAD and the separable class. If the first condition in the definition 4.9 is satisfied, then the formula closer to the centre has fewer variables than the farther one, which might it execute faster (if both the formulae are comparable). On the other hand, if the second condition is satisfied, then a lesser amount of structure is lost on separations, thus necessitating a lower number of redundant computations. This motivates the following observation.

OBSERVATION 4.11. *Given prenex formulae,* f *and* g *and their separations,* (F, C_1) *and* (G, C_2), *if* $f \preceq g$, *then PARALLEL - CAD takes at least as much time and has at least as much redundancy ratio to execute* g *as it would take to execute* f, *provided that* f *and* g *remain comparable.*

5. SIMULATION RESULTS

The simulation results have been divided into two categories which are mutually independent, namely (i) The cases where only separations are considered and (ii) The cases where only substitutions are considered.

These experiments were run on an Intel i7 quad-core CPU using 4GB of DDRAM. QEPCAD Version B 1.65, 10 May 2011 [2] is used to perform the computations. In the following plots, time is in milliseconds and space is in cells used by QEPCAD.

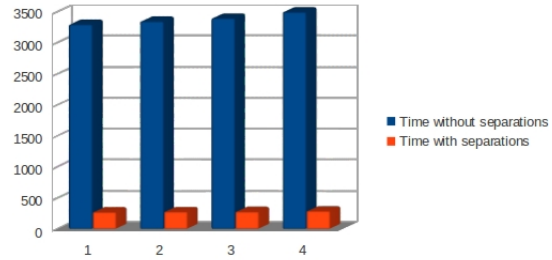
5.1 Simulation Results for Separability

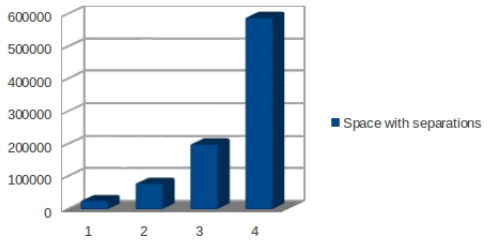
The simulation has been performed for two cases, depending on the distribution of variables among the constituent polynomials of the prenex formula. This is based on the fact that the running time and the space required for CAD depend almost entirely on the number of variables, the time being doubly exponential in the number of variables.

Case Study 1 - Homogeneous Distribution of Variables

This case study involves four prenex formulae, each with two polynomial equations and with varying number of quantifier - free variables being common between them. The space and time analyses of this situation yields quite interesting consequences, when separability is applied. The polynomials considered were:

1. $(\exists x)[ax^2 + bx + c = 0 \vee dx^2 + ex + f = 0]$
2. $(\exists x)[ax^2 + bx + c = 0 \vee dax^2 + ex + f = 0]$
3. $(\exists x)[ax^2 + bx + c = 0 \vee dax^2 + ebx + f = 0]$
4. $(\exists x)[ax^2 + bx + c = 0 \vee dax^2 + ebx + fc = 0]$



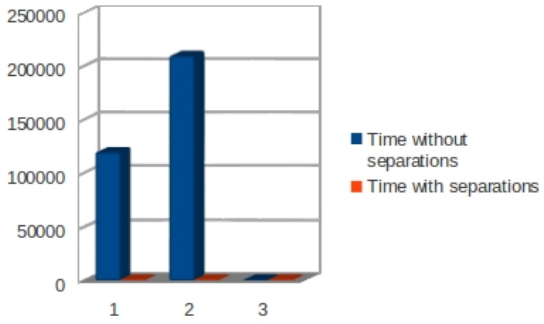


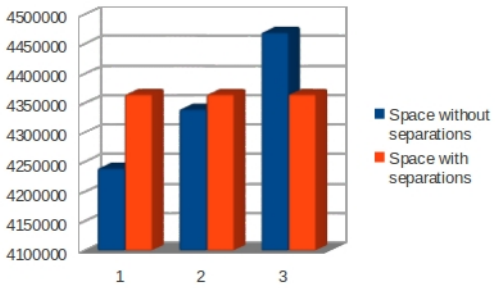
The time complexity reduced nearly five-fold in all the four cases. The space complexity remained same across all the four cases if CAD was applied without separations, as there would still be the same number of projection operations. But, when separations were applied, the space required increased exponentially as the overlap in the variables of the two polynomials increased. This was expected, considering the doubly exponential nature of CAD algorithm.

Case Study 2 - Heterogeneous Distribution of Variables

This case study considers a similar system as above, except for the fact that the variables are unevenly distributed among the two polynomials and that there is a disparity among the degrees of the two polynomials. The formulae considered are:

1. $(\exists x)[dafx^5 + ebx^3 + ec = 0 \vee fc + x = 0]$
2. $(\exists x)[dafx^5 + ebx^3 + ec = 0 \vee f + x = 0]$
3. $(\exists x)[dafx^5 + ebx^3 + ec = 0 \vee x = 0]$





Here, due to separations, the structure imposed by the second equation on the first one is lost, thus leading to possibly more number of constructions. But, as the dependency reduces, the 'amount of structure' lost is also reduced, and hence we observe that the separated case starts to perform better.

Case Study 3 - Validity of Observation 4.11 Two case studies have been performed demonstrating the validity of observation 4.11. The inputs for the two case studies were

1. - $\exists x[ax^5 + abx^3 + bc = 0] \vee [dx + ef = 0] \vee [gfx^{10} + hgx^6 + ix^4 + j = 0]$
   - $\exists x[ax^5 + abx^3 + bc = 0] \vee [dx + e = 0] \vee [gfx^6 + hix + j = 0]$.
2. - $\exists x[ax^3 + bx + c = 0] \vee [cdx^2 + de = 0]$
   - $\exists x[ax^3 + bx + c = 0] \vee [dx^2 + de = 0]$.

The time taken for execution and the space required were:

		First Formula	Second Formula
Case 1	Time	1584ms	2128ms
	Space	15126564 cells	23304694 cells
Case 2	Time	40ms	36ms
	Space	73998 cells	66287 cells

In the first case, the time taken by the first formula was considerably lesser, even though it had a larger exponent in it's constituent polynomials. This is because of a more even distribution of variables. According to observation 4.11, the first formula is closer to the centre of S_3 than the second formula.

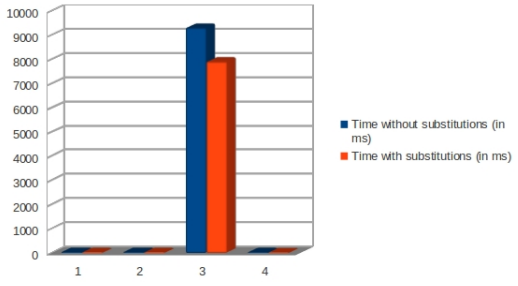
In the second case, the second condition of closeness to the centre is considered. Thus, the time consumed by both the formulae are almost identical. Due to sharing of variables in the first case, a greater amount of wasteful total work has been done, than in the second case. According to observation 4.11, the second formula is closer to the centre of S_2 than the first one.

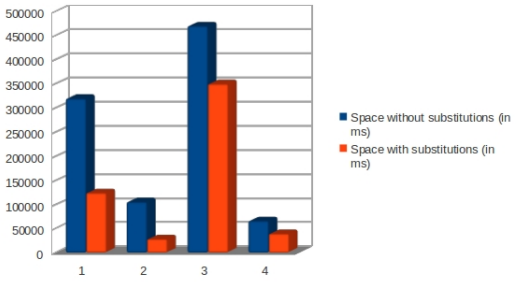
5.2 Simulation Results for Substitutability

The experiments have been done for 4 test cases. They do not dynamically schedule new instances to different cores. The 4 cases are,

1. $(\forall x)(\forall y)[[b^2x^2 + a^2y^2 = a^2b^2] \Longrightarrow [x^2 + y^2 < 1]]$
2. $(\exists y)(\exists z)[[x^3y - 29x^2 = 0] \wedge [x^4z^2 - 101x^3 - 19x = 0] \wedge [x^2 + y^2 + z^2 > 0] \wedge [x + y > 0] \wedge [y + z < 0]]$
3. $(\forall v)[[x^3y - 29w^2 = 0] \wedge [x^4z^2 - 101w^2 - 19v = 0] \wedge [x^2 + y^2 + z^2 > 0] \wedge [x + y > 0] \wedge [w + v + x + z > 0]]$
4. $(\exists v)[[x^5 + 4y^2 - 3z = 0] \wedge [x^6 + 2w^2 - v = 0] \wedge [x + y + w + z + v > 0]]$

This simulation is intended to demonstrate the extent till which substitutions can improve the time and space required.





In the above plots, we observe that the space required for computations decreased to a greater extent, as compared to the time. This is because of the fact that substitutions preserve the inter-dependency among the constituent polynomials in a prenex formula, unlike separations. Substitutions only change the way these polynomials are represented. This explains the difference between the space and time plots of the separations case and the substitutions case. The plots indicate that the space consumed and the time elapsed are disproportionate, suggesting that a large amount of memory has been used on projecting and constructing operations, which required small amounts of processing time.

6. CONCLUDING REMARKS

In this paper we proposed a parallel Cylindrical Algebraic Decomposition algorithm. The ongoing work is to implement this algorithm for execution on a GPU. Based on the concepts of separable class and it's centre, we propose to give the average case complexity analysis of PARALLEL-CAD. This algorithm still uses the projection operators defined for the sequential CAD algorithm. Our future work is to explore for, and define a new projection operator, which embodies the structure imposed by parallelism. This projection operator, when found, may speed-up CAD further.

7. REFERENCES

[1] Caviness B.F. and J.R. Johnson. *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Springer Verlag, 1998.

[2] C.W. Brown. QEPCAD B: A Program for Computing with Semi-Algebraic Sets using CADs. *ACM SIGSAM Bulletin*, 37(4):97–108, 2003.

[3] B. Buchberger and H. Hong. *Speeding-up Quantifier Elimination by Gröbner Bases*. Johannes Kepler University, Linz, 1991.

[4] G. Collins. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decompostion. In *Automata Theory and Formal Languages 2nd GI Conference Kaiserslautern, May 20–23, 1975*, pages 134–183. Springer, 1975.

[5] G.E. Collins and H. Hong. Partial Cylindrical Algebraic Decomposition for Quantifier Elimination. *Journal of Symbolic Computation*, 12(3):299–328, 1991.

[6] Andreas Dolzmann, Thomas Sturm, and Volker Weispfenning. Real Quantifier Elimination in Practice. In *Algorithmic Algebra and Number Theory*, pages 221–247. Springer, 1998.

[7] Collins G.E. Quantifier Elimination by Cylindrical Algebraic Decomposition–Twenty Years of Progress. *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 8–23, 1998.

[8] J Hintikka. Truth definitions, Skolem Functions and Axiomatic Set Theory. 1998.

[9] H. Hong. An Improvement of the Projection Operator in Cylindrical Algebraic Decomposition. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pages 261–264. ACM, 1990.

[10] Parberry, I. Parallel Complexity Theory. 1987.

[11] B.D. Saunders, H.R. Lee, and S.K. Abdali. A Parallel Implementation of the Cylindrical Algebraic Decomposition Algorithm. In *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation*, pages 298–307. ACM, 1989.

[12] A. Tarski. A Decision Method for Elementary Algebra and Geometry: Prepared for Publication with the Assistance of JCC McKinsey. 1951.