Approaches to Parallel Quantifier Elimination

Andreas Dolzmann
Department of Mathematics and Computer Science
University of Passau, Germany
dolzmann@uni-passau.de
http://www.fmi.uni-passau.de/~dolzmann/

Oliver Gloor
Institute for Computer Science
University of Tübingen, Germany
gloor@informatik.uni-tuebingen.de
http://www-sr.informatik.uni-tuebingen.de/~gloor/

Thomas Sturm
Department of Mathematics and Computer Science
University of Passau, Germany
sturm@uni-passau.de
http://www.fmi.uni-passau.de/~sturm/

MIP-9803
February 25, 1998

Abstract

Special-purpose quantifier elimination procedures for problems of low degree using virtual substitution of test terms have recently turned out to be applicable to a variety of non-trivial non-academic problems. We study parallel algorithms based on these methods for several parallelization environments: a Cray YMP4/T3D, a workstation cluster, and a multi-processor Sparc. Our implementations show remarkable though sublinear speed-ups in all these environments.

1 Introduction

The basic motivation of real quantifier elimination is to "eliminate" unwanted variables representing real quantities from an algebraic description of some situation. As an example, consider conditions for the solvability of the equation $ax^2 + b = 0$ where a and b are real parameters: This equation has a real solution if and only if $ab < 0$ or $b = 0$. Formally, we have eliminated the quantifier "$\exists x$" from $\exists x(ax^2 + b = 0)$ yielding the equivalent $ab < 0 \vee b = 0$. The latter provides a better idea of the range of a and b described by the two formulas.

For a long time the use of quantifier elimination in application problems outside pure mathematics has been fairly limited due to the poor efficiency of the implemented methods. Recently, implementations of quantifier elimination have been able to solve problems of interesting size in science, engineering, and also in economics, namely in operations research, cf. [DSW97] and the references there. Quantifier elimination by *virtual substitution* of test terms [Wei88, LW93, Wei94, Wei97a] plays a crucial role for this success, in particular for problems involving numerous parameters.

The purpose of our work is to study parallelization and distribution of the virtual substitution method. Here, *distribution* refers to parallelization on distributed-memory hosts or network clusters of several hosts. We have worked with the following environments, of which the first two are distributed ones:

- PVM and RR on a Cray YMP4/T3D supercomputer using a REDUCE implementation of a special case of the method for non-parametric linear optimization.

- PVM based DTS on a workstation cluster.

- Multithreading on a multi-processor Sun Sparc-10 running Solaris 2.

For the latter two cases, we have realized a SACLIB/PARSAC-2 based C implementation "QEDTS."

In all these environments we obtain remarkable speed-ups. Since our parallelization proceeds in such a way that we perform essentially the same computations as in the sequential case, the speed-ups are limited by the number of processors. For decision problems, there are straightforward extensions of our approaches which will obviously give rise to superlinear speed-ups.

2 History and Related Work

The first real quantifier elimination procedure was found by Tarski in the 1930's. It remained unpublished until 1948 [Tar48]. Tarski's procedure was very inefficient, more precisely it was not elementary recursive.

In 1975 Collins introduced a new method called *cylindrical algebraic decomposition* (CAD) [Col75], which is worst-case doubly exponential in the number of variables. A full implementation of CAD within the computer algebra system SAC-2 [Col85] has been finished by Arnon in 1981. In the past 20 years of research, CAD has gone through numerous improvements [McC88, Hon90, Hon92], resulting in *partial* CAD [CH91] implemented in Hong's QEPCAD program based on SACLIB [BCE+93], a C version of SAC-2.

Around 1988 it has been shown that real quantifier elimination is inherently hard for some problem classes [DH88, Wei88]. Thus the attention turned to special procedures for restricted problem classes, where the elimination procedures can be tuned to the structure of the problem. The focus was on considering formulas in which the occurrence of quantified variables is restricted to low degrees [Hon93b, Hon93c, GV93]. This was initiated by a theoretical paper of Weispfenning in 1988 [Wei88], who introduced the method of *virtual disjunctive term substitution*, which has been continuously extended and improved [LW93, Wei94, Wei97a, DSW96, DS97b].

The worst-case complexity of this method is doubly exponential only in the number of the *quantifier blocks* of the input formula, but singly exponential in the number of *quantified* variables. This makes the method attractive for problems containing many parameters. It is implemented in the REDUCE [HF95] package REDLOG [DS96, DS97a]. In addition, there is an implementation in the computer algebra system Risa/Asir [Stu96]. For the purpose of this paper, the authors have reimplemented the method once more in C based on SACLIB/PARSAC-2 [BCE+93, Küc90a, Küc90b, Küc94, GM98]. All these implementations are currently restricted to input obeying certain degree restrictions with respect to the quantified variables. In principle, the method can be extended to arbitrary degrees [Wei97a].

In 1993 Weispfenning introduced a new complete elimination procedure based on comprehensive Gröbner bases in combination with multivariate real root counting, cf. [Wei93] and [Wei92, BW94, PRS93]. It has been implemented within the computer algebra system MAS [Dol94].

In recent years there have been impressive theoretical results on asymptotically fast real elimination algorithms [Ren92, BPR94]. Since there is no implementation available, the question whether these methods are of practical relevance remains unanswered.

The first work in parallelizing quantifier elimination has been done by Saunders et al. in 1989 [SLA89]. They have parallelized the original Collins CAD algorithm based on SAC-2 on an 8-processor Sequent Symmetry shared-memory machine. They report a speed-up of about factor 3 using 7 processors.

In 1993 Hong has parallelized his SACLIB based QEPCAD on a workstation network based on an apparently unpublished communication protocol [Hon93a]. Using a master-slave model on 6 hosts, he obtains speed-ups ranging from factor 4 up to superlinear.

Hong has also developed a parallel extension of SACLIB named PACLIB [HNS95], which runs on shared-memory machines. There are numerous symbolic algorithms implemented based on PACLIB [SH93], not including quantifier elimination, however.

Another independent parallel extension of SACLIB is PARSAC-2 [Küc90a, Küc90b, GM98], originally developed by Küchlin. PARSAC-2 is a parallel Computer Algebra library targeted for execution on multi-processor workstations. Up to now, the applications of PARSAC-2 have been limited to the area of integer and polynomial arithmetic and Gröbner basis computation [Küc94, AGK96, AGK97].

3 Components of our Parallelization Environments

3.1 PVM

PVM [GBD$^+$94] stands for Parallel Virtual Machine. It is a software package that allows a network of parallel and serial computers to appear as single concurrent distributed-memory machine. Under PVM, C or Fortran-77 applications can be parallelized using common message-passing constructs. It is available for a variety of architectures including the Cray YMP4/T3D and Sun Workstations used for our studies.

3.2 RR

RR [MN95] provides an extension of the Portable Standard Lisp (PSL) system underlying REDUCE. It supports both the Cray YMP4/T3D and workstation clusters. Communication and control are implemented on top of the PVM manager. RR supports the fork/join paradigm by means of remote procedure calls and remote-wait/remote-receive primitives.

3.3 The S-thread System

PARSAC-2 contains the S-threads parallel symbolic programming environment [Küc90a]. S-threads is *middle-ware* between the operating system below and the application software above. Most importantly, an S-thread extends an operating system thread by a control block that supports memory management. Furthermore, the memory allocation mechanism as well as the mark-sweep garbage collection of SACLIB are modified in such a way that concurrent allocation is enabled. Practically all sequential procedures of the SACLIB library can now be executed concurrently, by forking them on a separate S-thread.

S-threads support *virtual parallel programming:* we parallelize an algorithm according to its inherent logical parallel structure. The resulting logical threads of control are mapped at run-time to real concurrency provided by the hardware and the operating system. Thus the same executable can run on 1, 2, or more processors.

The implementation of S-threads is based on operating system threads. For the current work, we used the S-thread system on Sparc workstations under Solaris 2, which in turn relies on Solaris/POSIX threads [GM98].

The following parallelization paradigms of the S-thread system are of importance for our quantifier elimination algorithm:

Work Parallelism in its pure form ($\forall$-Parallelism) is the concept supported most directly by the fork/join paradigm of multithreading. A given amount of work is divided into multiple threads of control and forked onto a number of threads. Each thread is finally collected (joined) again by the parent, and its result is retrieved.

In our quantifier elimination algorithm, we employ work parallelism by applying quantifier elimination to subformulas. The final result is then obtained as the disjunction over the quantifier-free equivalents obtained for the subformulas.

Search Parallelism in its pure form ($\exists$-Parallelism) reflects a parallel search for some item. A number of search threads are forked, and the first child to find the item synchronizes with the parent to deliver the result. The parent subsequently terminates the other children, not being interested in their results.

In our quantifier elimination algorithm, in particular with decision problems, i.e. parameter-free elimination problems, we could use a form of search parallelism: As soon as one of the forked subproblems yields "true," all other subproblems become irrelevant and these computations can be aborted.

The synchronization needed by search parallelism is now supported in S-threads by the concept of *thread groups.* A thread group is a collection of threads, similar to a process group in the traditional Unix world. We designed our thread groups to support both work and search parallelism in a uniform way.

After a thread group has been opened, threads can be forked into the group at any time. The parent can wait for the next thread of the group, or it can send a signal to the entire group, for example to terminate or to suspend all threads (and their children, of course).

On shared-memory machines with fast context switching, the main use of a thread group is in supporting search parallelism. Then, all threads of the group execute as concurrent operating system threads. On a uniprocessor machine, execution of these threads will be interleaved to maintain logical concurrency.

Of course, all these features are recursively inherited to forked threads. This is of crucial importance for the parallelization of our quantifier method as we use a recursive approach in our parallelization.

3.4 DTS

The fork/join concept of shared-memory parallel programming can be carried over to the distributed-memory network, provided all global access to data-structures can be confined to parameters that can be packed with fork/join. An extension of the S-threads called DTS (Distributed Thread System [BHKR95]) is based on PVM which transports fork/join calls across the net. Thus, we can use the same parallelization paradigm (and essentially the same code), regardless whether we compute on a shared-memory workstation or on a network of single-processor or even multi-processor workstations. In addition, as DTS relies on S-threads, we do not take care about the number of forked tasks. That is, we usually overload the processors and thus overlap the network communication by actual computation tasks.

4 Quantifier Elimination by Virtual Substitution

The applicability of the virtual substitution method in the form described here is restricted to formulas in which the quantified variables occur at most quadratically. Moreover, quantifiers are eliminated one by one, and the elimination of one quantifier can increase the degree of other quantified variables.

For eliminating the quantifiers from an input formula

$$\varphi(u_1, \ldots, u_m) \equiv \mathsf{Q}_1 x_1 \ldots \mathsf{Q}_n x_n \psi(u_1, \ldots, u_m, x_1, \ldots, x_n),$$

where $\mathsf{Q}_i \in \{\exists, \forall\}$, the elimination starts with the innermost quantifier regarding the other quantified variables within ψ as extra parameters. Universal quantifiers are handled by means of the equivalence $\forall x \psi \longleftrightarrow \neg \exists x \neg \psi$. We may thus restrict our attention to a formula of the form

$$\varphi^*(u_1, \dots, u_k) \equiv \exists x \psi^*(u_1, \dots, u_k, x),$$

where the $u_{m+1}, \dots, u_k$ are actually x_i quantified from further outside.

We fix real values $a_1, \dots, a_k$ for the parameters $u_1, \dots, u_k$. Then all polynomials occurring in ψ^* become linear or quadratic univariate polynomials in x with real coefficients. So the set

$$M = \{\, b \in \mathbb{R} \mid \psi^*(a_1, \dots, a_k, b) \,\}$$

of all real values b of x satisfying ψ^* is a finite union of closed, open, and half-open intervals on the real line. The endpoints of these intervals are among $\pm\infty$ together with the real zeros of the linear and quadratic polynomials occurring in ψ^*. Candidate terms $\alpha_1, \dots, \alpha_r$ for the zeros can be computed uniformly in $u_1, \dots, u_k$ by the solution formulas for linear and quadratic equations.

If all inequalities in ψ^* are weak, then all the intervals constituting M will, into each direction, be either unbounded or closed. In the latter case, such an interval will contain its real endpoint. Thus M is non-empty if and only if the substitution of $\pm\infty$ or of one of the candidate solutions α_j for x satisfies ψ^*. The substitution of $\pm\infty$ into a polynomial equation or inequality is evaluated in the obvious sense. The substitution of expressions in $u_1, \dots, u_k$ of the form $(a + b\sqrt{c})/d$ among the α_j can be rewritten in such a way that all denominators involving the u_i and all square-root expressions are removed from the result [Wei97a]. By disjunctively substituting all candidates into ψ^* we obtain a quantifier-free formula equivalent to $\exists x \psi^*$.

If ψ^* contains also strict inequalities, we need to add to our candidates for points in M expressions of the form $\alpha \pm \varepsilon$, where α is candidate solution for some left-hand side polynomial occurring in a strict inequality. The symbol ε stands for a positive infinitesimal number. Again the substitution of these expressions into a polynomial equation or inequality can be rewritten in such a form that there occur neither denominators involving any of the u_i, nor any square root expressions, nor the symbol ε in the result [Wei97a]. Again this yields a quantifier-free formula equivalent to $\exists x \psi^*$. For practical applications this method, of course, has to be refined by a careful selection of a smaller number of candidate solutions and by a combination with powerful simplification techniques for quantifier-free formulas, cf. [DS97b] for details.

Recall that the well-known solution formula for quadratic equations

$$ax^2 + bx + c = 0$$

requires $a \neq 0$. In our situation, a is a term in $u_1, \ldots, u_k$. So $a \neq 0$ can in general not be decided uniformly, but depends on the interpretation of the u_i. Thus a quadratic polynomial $ax^2 + bx + c$ does not only deliver two square-root expressions α_1 and α_2 as candidate solutions, but also $\alpha_3 = -c/b$, which in turn requires $b \neq 0$. Let t_1, t_2, and t_3 be the candidate points for M obtained from α_1, α_2, and α_3, respectively, by possibly adding or subtracting ε. With the substitution of all the t_i into ψ^*, it is necessary to add the conditions on the non-vanishing of a and b. Formally, we obtain

$$(a \neq 0 \wedge \Delta \geq 0 \wedge (\psi^*[x/t_1] \vee \psi^*[x/t_2])) \vee (a = 0 \wedge b \neq 0 \wedge \psi^*[x/t_3]),$$

where Δ denotes the discriminant of the equation $ax^2 + bx + c = 0$. If, however, a is a rational constant, then the case distinction is superfluous. In particular, if a is non-zero, we drop the second case.

We can obtain dramatic improvements of the general procedure sketched above by reducing the number of test candidates for M depending on the structure of the formula ψ^* [LW93, Wei97a]. One simple instance for such an improvement is the following natural extension of *Gauss elimination*: Suppose ψ^* is of the form

$$bx + c = 0 \wedge \psi^{**},$$

where at least one of the coefficient terms b, c is a rational non-zero constant. Then we know that, under any interpretation of the u_i, the equation is *non-trivial*, i.e. different from $0 = 0$. Hence the only test candidate required is $-c/b$ substituted, of course, with the condition $b \neq 0$. No additional test candidates that arise from equations or inequalities in the remainder ψ^{**} of ψ^* need be considered. This idea can easily be extended to a quadratic equation instead of a linear one, taking into account again the discriminant.

The notion of *virtual* substitution refers to both adding conditions and resolving non-standard subterms such as surds or infinitesimals. This method is doubly exponential in the number of quantifier changes. For pure "$\exists$" or "$\forall$" blocks, it is singly exponential in the number of quantified variables. In theory, parameters play a minor role for the complexity. They turn in fact out to be very cheap in practice, too.

5 The Elimination Tree

In the sequel, we restrict our attention to the elimination of pure prenex "$\exists$" blocks of quantifiers.

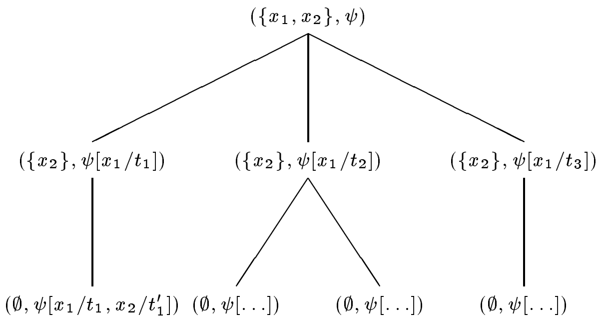

Figure 1: An elimination tree

Suppose we have eliminated an existential quantifier. Then we have in general obtained a disjunction $\psi_1' \vee \cdots \vee \psi_r'$. If the next quantifier to be eliminated is also an existential one, then we make use of the equivalence

$$\exists x_{n-1}(\psi_1' \vee \cdots \vee \psi_r') \longleftrightarrow \exists x_{n-1}(\psi_1') \vee \cdots \vee \exists x_{n-1}(\psi_r')$$

and eliminate all $\exists x_{n-1}(\psi_j')$ independently. As a consequence, no candidate solutions obtained from, say, ψ_1' are substituted into the other ψ_j. This decreases the complexity class of our procedure for single quantifier blocks from doubly exponential to singly exponential in the number of quantifiers [Wei88].

Proceeding this way, the elimination procedure for

$$\exists x_1 \ldots \exists x_n \psi_0(u_1, \ldots, u_m, x_1, \ldots, x_n)$$

can be considered as the computation of an *elimination tree* of the following form:

- Each node consists of a quantifier-free formula ψ plus a list V of variables. The corresponding partial elimination problem is $\exists V \psi$.
- The root is $(\psi_0, \{x_1, \ldots, x_n\})$.
- For each node $(\psi, \{x_i, \ldots, x_n\})$, the children are determined as

  $$(\psi[x_i/t_1], \{x_{i+1}, \ldots, x_n\}), \quad \ldots, \quad (\psi[x_i/t_k], \{x_{i+1}, \ldots, x_n\}),$$

  where $\{t_1, \ldots, t_k\}$ is a set of candidate points with respect to x_i for ψ.

Figure 1 shows an example.

Elimination trees have the following properties: On each level all variable lists are identical. The number of variables in these lists is decreased by one

$W := \{(\psi_0, \{x_1, \dots, x_n\})\}$
$R := \emptyset$
while $W \neq \emptyset$ **do**
  select (ψ, V) from W
  $W := W \setminus \{(\psi, V)\}$
  if $V = \emptyset$ **then**
    $R := R \cup \{(\psi, V)\}$
  else
    determine the children C of (ψ, V)
    $W := W \cup C$
  fi
od

Figure 2: The sequential elimination algorithm

with each new level. All leaves of the tree contain the empty list. For each level with nodes (ψ'_1, V'), ..., (ψ'_k, V') we have

$$\left(\bigvee_{i=1}^{k} \exists V' \psi'_i\right) \longleftrightarrow \exists x_1 \dots \exists x_n \psi_0.$$

This holds in particular for the last level for which $V' = \emptyset$. Hence the left hand side of the above equivalence is then quantifier-free.

6 The Sequential Algorithm

Figure 2 shows a sequential algorithm for constructing the elimination tree. It maintains a set W of working nodes, and another set R of result nodes, i.e., leaves. The set W is implemented either as a queue or as a stack, with the elimination tree being constructed breadth-first or depth-first, respectively. Both approaches have certain advantages: Using a queue allows us to detect identical sibling nodes which are known to come into existence systematically [Wei97b]. Using a stack saves space during the computation, and whenever a leaf happens to become "true," we can immediately abort the computation and prune the remaining tree. As a rule of thumb, it has turned out reasonable to use the depth-first approach only for decision problems and parameter-free optimization problems.

$W := \{(\psi_0, \{x_1, \dots, x_n\})\}$
$R := \emptyset$
run breadth-first elimination until $|W| = 3|P|$
while $W \neq \emptyset$ **or** $H \neq \emptyset$ **do**
  while $W \neq \emptyset$ **and** $H \neq P$ **do**
    select h from $P \setminus H$
    $H := H \cup \{h\}$
    select (ψ, V) from W
    $W := W \setminus \{(\psi, V)\}$
    start depth-first elimination of (ψ, V) on h
  od
  if $H \neq \emptyset$ **then**
    wait until $H' \subseteq H$ have finished
    $H := H \setminus H'$
    for each h **in** H' **do**
      receive the result C^*
      $R := R \cup C^*$
    od
  fi
od
return R

Figure 3: A parallel master-slave elimination algorithm

7 A Master-Slave Approach on the Cray

The first approach to distributing the virtual substitution method has been done by the first and the third author in cooperation with W. Neun at the Konrad Zuse Center in Berlin. This implementation is restricted to conjunctions of parameter-free constraints involving only "$\geq$," "$\leq$," and "$=$." It is part of the REDUCE package REDLOG. It runs on a Cray YMP4/T3D using PVM plus a special version of REDUCE based on PSL extended by RR. The idea is to have a master node generate by breadth-first elimination a *problem base*, which is then distributed to several slaves in order to be processed completely by depth-first elimination. A corresponding algorithm is given in Figure 3. We denote by P the list of available processor nodes, and by $H \subseteq P$ the list of busy processors. The initial problem base size $3|P|$ has been found by practical elimination experiments.

This implementation has been applied to three benchmark problems from linear programming: Using 8 processors, i.e. 1 master plus 7 slaves, with 64 MB of memory each, a speed-up factor of about 3 compared to sequential

Problem	# Var.	# Constr.	Seq.	Par.	Factor
afiro	32	59	333 s	95 s	3.5
sc50a	48	97	>900 s	272 s	>3.3
sc50b	48	96	70 s	25 s	2.8

Table 1: Timings (in seconds) obtained on the Cray

procedure first_process(W)
  $L := \emptyset$
  for each (ψ, V) **in** W **do**
    determine the children C of (ψ, V)
    $L := L \cup C$
  od
  if the V in L are empty **then**
    return L
  else
    split L into L_1 and L_2 of equal lengths
    $h :=$ fork(first_process, L_1)
    $L_2' :=$ first_process(L_2)
    $L_1' :=$ join(h)
    return $L_1' \cup L_2'$
  fi
end

Figure 4: The "first_process" divide-and-conquer algorithm

depth-first elimination was obtained, cf. Table 1. The input problems there are taken from the Konrad Zuse Center's NETLIB online library of benchmark examples. Our speed-ups are very similar to those obtained by Saunders et. al with their shared-memory parallelization of CAD [SLA89].

8 Divide-and-Conquer on a Workstation Cluster using DTS

The S-thread system and DTS allow us to fork essentially arbitrary many threads over the net. Therefore, we can apply the concept of divide-and-conquer, where there is no a priori bound for the number of concurrent tasks. Divide-and-conquer algorithms distribute the problem over the computational resources in the fastest possible manner. When the elimination is distributed over several networked hosts, however, any fork of a function

Problem	Seq.	1 Host	2 Hosts	3 Hosts
rp-2	198.05 s	320.0 s	211.7 s	175.6 s

Table 2: Timings (in seconds) with "first_process" on a workstation cluster

Problem	Seq.	1 Proc.	2 Proc.	3 Proc.	4 Proc.
period-9	50.4 s	53.4 s	29.0 s	21.4 s	17.4 s
schedule	93.2 s	109.0 s	60.9 s	44.6 s	36.1 s
rp-2	266.1 s	289.9 s	168.2 s	156.4 s	154.2 s

Table 3: Timings (in seconds) with "first_process" on a 4-processor Sparc

causes its arguments to be sent over the net. Therefore we have to take care about the communication costs. This means that whenever a host processes some forked function, all the data passed to this function should actually be processed there at least partially. The algorithm "first_process" of Figure 4 has been carefully designed to meet this issue.

Compare, in contrast, the algorithm "first_divide" of Figure 5, where half of the nodes contained in the argument W are simply passed to yet another host.

Our first set of distributed timings is collected in Table 2. It has been obtained with three 140 MHz Sun Ultra Sparc-1 connected by a conventional 10 Mbit Ethernet. All hosts are running Solaris 2. We observe an initial overhead of about 60% caused by the parallelization environment, i.e. PVM and DTS, and the fact that the distributed version on one host does not construct the tree in a breadth-first manner, and hence we cannot identify as many identical nodes. On more than one host, we reach moderate speed-ups, with three hosts even compared to the sequential version.

We shall turn back to the distributed case once more in the next section.

9 Divide-and-Conquer on a Multi-processor Sparc

Our shared-memory parallel computations based on S-threads have been performed on a multi-processor Sun Sparc-10 with four 90 MHz Ross Hyper-Sparc processors running Solaris 2. Such a Hyper-Sparc processor is roughly equivalent to a 50 MHz Super-Sparc.

Table 3 collects parallelization timings for several input problems using the algorithm "first_process" of Figure 4 already discussed in the previous section.

```
procedure first_divide(W)
  if |W| = 1 then
    determine the children C of (ψ, V) ∈ W
    if the V in C are empty then
      return C
    else
      return first_divide(C)
    fi
  else
    split W into L1 and L2 of equal lengths
    h := fork(first_divide, L1)
    L2' := first_divide(L2)
    L1' := join(h)
    return L1' ∪ L2'
  fi
```

Figure 5: The "first_divide" divide-and-conquer algorithm

Problem	Seq.	1 Proc.	2 Proc.	3 Proc.	4 Proc.
period-9	50.4 s	53.8 s	29.4 s	21.9 s	17.6 s
schedule	93.2 s	110.6 s	59.2 s	43.1 s	35.8 s
rp-2	266.1 s	297.1 s	170.2 s	130.3 s	111.2 s

Table 4: Timings (in seconds) with "first_divide" on a 4-processor Sparc

Here, the system overhead of the concurrent computation using 1 processor compared to the sequential one is much smaller than in the distributed case. In fact, the overhead of the S-thread system can be neglected. The speed-ups obtained with more processors are similar to those in the distributed case. The absolute running times are, of course, much better. Note that the hosts used for the distribution are significantly faster than 1 processor on the parallel Sparc.

In the example "rp2," we observe almost no additional speed-up by the 4th processor. Traced output indicates that there are not enough threads forked to exploit this additional resource. Therefore, we have devised the algorithm "first_divide" shown in Figure 5. It forks half of the work at the earliest possible time before doing any computation itself.

Table 4 shows the timings obtained with "first_divide." It actually yields additional speed-up with 3 and 4 processors. On the other hand we loose performance with 1 and 2 processors caused by the fact that the construction of the elimination tree by "first_process" is much closer to breadth-first than

	1 Host	2 Hosts
1 Processor	459.5 s	290.7 s
4 Processors	206.6 s	149.8 s

Table 5: Timing (in seconds) obtained for "rp2" by connecting multi-processor Sparcs with "first_process"

that by "first_divide."

Finally, we have combined distribution with parallelization by connecting two 4-processor Sparcs. Since the S-thread system and DTS cooperate perfectly, there is no additional code necessary for doing so, i.e., the same executable as in the previous section is running. The timings collected in Table 5, which we obtained by the algorithm "first_process," suggest that also with a distributed approach, one can make good use of multi-processor shared-memory machines.

10 Conclusions and Future Work

We have demonstrated that remarkable speed-ups are obtained with parallel quantifier elimination. This holds for shared-memory multi-processor workstations as well as for distributed-memory hosts and networks of workstations.

Our parallel algorithms retain the structures of their corresponding sequential variants. Therefore, we can distinguish between the system effects and the algorithmic effects on the speed-ups.

In the case of decision problems, one wishes to determine as fast as possible whether there is a leaf containing "true." Search parallelism allows us to abort the computation immediately when such a leaf is found. We expect superlinear speed-ups from a corresponding implementation.

Acknowledgments

We would like to thank Winfried Neun and the Konrad Zuse Center in Berlin for the cooperation which led to the parallelization on the Cray. The second author is supported by grant Ku 966/2 from Deutsche Forschungsgemeinschaft (DFG). We are indebted to Benno Fuchssteiner for his generous financial support of this project.

References

[AGK96] Beatrice Amrhein, Oliver Gloor, and Wolfgang Küchlin. A case study of multi-threaded Gröbner basis completion. In Y. N. Lakshmann, editor, *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation (ISSAC 96)*, pages 95–102, Zurich, Switzerland, July 1996. ACM, ACM Press.

[AGK97] Beatrice Amrhein, Oliver Gloor, and Wolfgang Küchlin. On the walk. *Theoretical Computer Science*, 187:179–202, 1997.

[BCE+93] Bruno Buchberger, George E. Collins, Mark J. Encarnacion, Hoon Hong, Jeremy R. Johnson, Werner Krandick, Rüdiger Loos, Ana M. Mandache, Andreas Neubacher, and Herbert Vielhaber. Saclib 1.1 user's guide. RISC-Linz Series Technical Report 93-19, Research Institute for Symbolic Computation, Johannes Kepler University, A-4040 Linz, Austria, 1993.

[BHKR95] Tilmann Bubeck, Martin Hiller, Wolfgang Küchlin, and Wolfgang Rosenstiel. Distributed symbolic computation with DTS. In Afonso Ferreira and José Rolim, editors, *Parallel Algorithms for Irregularly Structured Problems, 2nd International Workshop, IRREGULAR'95*, volume 980 of *Lecture Notes in Computer Science*, pages 231–248, Lyon, France, September 1995. Springer-Verlag, Berlin, Heidelberg, New York.

[BPR94] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. On the combinatorial and algebraic complexity of quantifier elimination. In Shafi Goldwasser, editor, *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 632–641, Los Alamitos, CA, USA, November 1994. IEEE Computer Society Press.

[BW94] Eberhard Becker and Thorsten Wörmann. On the trace formula for quadratic forms. In William B. Jacob, Tsit-Yuen Lam, and Robert O. Robson, editors, *Recent Advances in Real Algebraic Geometry and Quadratic Forms*, volume 155 of *Contemporary Mathematics*, pages 271–291. American Mathematical Society, Providence, Rhode Island, 1994. Proceedings of the RAGSQUAD Year, Berkeley, 1990–1991.

[CH91] George E. Collins and Hoon Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12(3):299–328, September 1991.

[Col75] George E. Collins. Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. In H. Brakhage, editor, *Automata Theory and Formal Languages. 2nd GI Conference*, volume 33 of *Lecture Notes in Computer Science*, pages 134–183. Gesellschaft für Informatik, Springer-Verlag, Berlin, Heidelberg, New York, May 1975.

[Col85] George E. Collins. The SAC-2 computer algebra system. In Bob F. Caviness, editor, *EUROCAL '85; Proceedings of the European Conference on Computer Algebra*, number 104 in Lecture Notes in Computer Science, pages 34–35, Berlin, Heidelberg, New York, Tokyo, April 1985. Springer.

[DH88] James H. Davenport and Joos Heintz. Real quantifier elimination is doubly exponential. *Journal of Symbolic Computation*, 5(1–2):29–35, February–April 1988.

[Dol94] Andreas Dolzmann. Reelle Quantorenelimination durch parametrisches Zählen von Nullstellen. Diploma thesis, Universität Passau, D-94030 Passau, Germany, November 1994.

[DS96] Andreas Dolzmann and Thomas Sturm. Redlog user manual. Technical Report MIP-9616, FMI, Universität Passau, D-94030 Passau, Germany, October 1996. Edition 1.0 for Version 1.0.

[DS97a] Andreas Dolzmann and Thomas Sturm. Redlog: Computer algebra meets computer logic. *ACM SIGSAM Bulletin*, 31(2):2–9, June 1997.

[DS97b] Andreas Dolzmann and Thomas Sturm. Simplification of quantifier-free formulae over ordered fields. *Journal of Symbolic Computation*, 24(2):209–231, August 1997.

[DSW96] Andreas Dolzmann, Thomas Sturm, and Volker Weispfenning. A new approach for automatic theorem proving in real geometry. Technical Report MIP-9611, FMI, Universität Passau, D-94030 Passau, Germany, May 1996. To appear in the Journal of Automated Reasoning.

[DSW97] Andreas Dolzmann, Thomas Sturm, and Volker Weispfenning. Real quantifier elimination in practice. Technical Report MIP-9720, FMI, Universität Passau, D-94030 Passau, Germany, December 1997.

[GBD+94] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM 3 User's Guide and Reference Manual*. Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, September 1994.

[GM98] Oliver Gloor and Stefan Müller. PARCAN—a parallel computer algebra nucleus. In preparation, 1998.

[GV93] Laureano González-Vega. A combinatorial algorithm solving some quantifier elimination problems. Technical Report 11-1993, University of Cantabria, Departemento de Mathematicas, Estadistica y Computación, Facultad de Ciencias, Avenida de Los Castros, 39071 Santander, Spain, November 1993.

[HF95] Anthony C. Hearn and John P. Fitch. *Reduce User's Manual for Version 3.6*. RAND, Santa Monica, CA 90407-2138, July 1995. RAND Publication CP78.

[HNS95] Hoon Hong, Andreas Neubacher, and Wolfgang Schreiner. The design of the SACLIB/PACLIB kernels. *Journal of Symbolic Computation*, 19(1–3):111–132, January–March 1995.

[Hon90] Hoon Hong. An improvement of the projection operator in cylindrical algebraic decomposition. In Shunro Watanabe and Morio Nagata, editors, *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC 90)*, pages 261–264, Tokyo, Japan, August 1990. ACM, ACM Press.

[Hon92] Hoon Hong. Simple solution formula construction in cylindrical algebraic decomposition based quantifier elimination. In Paul S. Wang, editor, *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC 92)*, pages 177–188, Berkeley, California, July 1992. ACM, ACM Press.

[Hon93a] Hoon Hong. Parallelization of quantifier elimination on a workstation network. In Gérard Cohen, Teo Mora, and Oscar Moreno, editors, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes. Proceedings of the 10th International*

Symposium, AAECC-10, volume 673 of *Lecture Notes in Computer Science*, pages 170–179, San Juan de Puerto Rico, Puerto Rico, May 1993. Springer-Verlag, Berlin, Heidelberg, New York.

[Hon93b] Hoon Hong. Quantifier elimination for formulas constrained by quadratic equations. In Manuel Bronstein, editor, *Proceedings of the 1993 International Symposium on Symbolic and Algebraic Computation (ISSAC 93)*, pages 264–274, Kiev, Ukraine, July 1993. ACM, ACM Press.

[Hon93c] Hoon Hong. Quantifier elimination for formulas constrained by quadratic equations via slope resultants. *THE Computer Journal*, 36(5):440–449, 1993. Special issue on computational quantifier elimination.

[Küc90a] Wolfgang Küchlin. PARSAC-2: A parallel SAC-2 based on threads. In Shojiro Sakata, editor, *Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes: 8th International Conference, AAECC-8*, volume 508 of *Lecture Notes in Computer Science*, pages 341–353, Tokyo, Japan, August 1990. Springer-Verlag, Berlin, Heidelberg, New York, 1991.

[Küc90b] Wolfgang Küchlin. The S-threads environment for parallel symbolic computation. In Richard E. Zippel, editor, *Computer Algebra and Parallelism, Second International Workshop*, volume 584 of *Lecture Notes in Computer Science*, pages 1–18, Ithaca, USA, May 1990. Springer-Verlag, Berlin, Heidelberg, New York, 1992.

[Küc94] Wolfgang Küchlin. PARSAC-2: Parallel computer algebra on the desk-top. In J. Fleischer, J. Grabmeier, F. Hehl, and W. Küchlin, editors, *Computer Algebra in Science and Engineering*, pages 24–43, Bielefeld, Germany, August 1994. World Scientific, Singapore, 1995.

[LW93] Rüdiger Loos and Volker Weispfenning. Applying linear quantifier elimination. *The Computer Journal*, 36(5):450–462, 1993. Special issue on computational quantifier elimination.

[McC88] Scott McCallum. An improved projection operation for cylindrical algebraic decomposition of three-dimensional space. *Journal of Symbolic Computation*, 5(1–2):141–161, February–April 1988.

[MN95] Herbert Melenk and Winfried Neun. RR: Parallel symbolic algorithm support for PSL based REDUCE. Preliminary Draft, 1995.

[PRS93] P. Pedersen, M.-F. Roy, and A. Szpirglas. Counting real zeroes in the multivariate case. In F. Eysette and A. Galigo, editors, *Computational Algebraic Geometry*, volume 109 of *Progress in Mathematics*, pages 203–224. Birkhäuser, Boston, Basel; Berlin, 1993. Proceedings of the MEGA 92.

[Ren92] James Renegar. On the computational complexity and geometry of the first-order theory of the reals. *Journal of Symbolic Computation*, 13(3):255–352, March 1992. Part I–III.

[SH93] Wolfgang Schreiner and Hoon Hong. A new library for parallel algebraic computation. In Richard F. Sincovec et al., editors, *Sixth SIAM Conference on Parallel Processing for Scientific Computing*, volume 2, pages 776–783, Norfolk, Virginia, March 1993. SIAM.

[SLA89] B. David Saunders, Hong R. Lee, and S. Kamal Abdali. A parallel implementation of the cylindrical algebraic decomposition algorithm. In Gaston H. Gonnet, editor, *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation (ISSAC 89)*, pages 298–307, Portland, Oregon, July 1989. ACM Press, New York.

[Stu96] Thomas Sturm. Real quadratic quantifier elimination in Risa/Asir. Research Memorandum ISIS-RM-5E, ISIS, Fujitsu Laboratories Limited, 1-9-3, Nakase, Mihama-ku, Chiba-shi, Chiba 261, Japan, September 1996.

[Tar48] Alfred Tarski. A decision method for elementary algebra and geometry. Technical report, RAND, Santa Monica, CA, 1948.

[Wei88] Volker Weispfenning. The complexity of linear problems in fields. *Journal of Symbolic Computation*, 5(1–2):3–27, February–April 1988.

[Wei92] Volker Weispfenning. Comprehensive Gröbner bases. *Journal of Symbolic Computation*, 14:1–29, July 1992.

[Wei93] Volker Weispfenning. A new approach to quantifier elimination for real algebra. Technical Report MIP-9305, FMI, Universität Passau, D-94030 Passau, Germany, July 1993.

[Wei94] Volker Weispfenning. Quantifier elimination for real algebra—the cubic case. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC 94)*, pages 258–263, Oxford, England, July 1994. ACM Press.

[Wei97a] Volker Weispfenning. Quantifier elimination for real algebra—the quadratic case and beyond. *Applicable Algebra in Engineering Communication and Computing*, 8(2):85–101, February 1997.

[Wei97b] Volker Weispfenning. Simulation and optimization by quantifier elimination. *Journal of Symbolic Computation*, 24(2):189–208, August 1997. Special issue on applications of quantifier elimination.