# HIGH-PERFORMANCE IMPLEMENTATIONS OF THE DESCARTES METHOD

Jeremy R. Johnson
Werner Krandick
Kevin M. Lynch
David G. Richardson and
Anatole D. Ruslanov

Technical Report DU-CS-06-04
Department of Computer Science
Drexel University
Philadelphia, PA 19104
May, 2006

# HIGH-PERFORMANCE IMPLEMENTATIONS OF THE DESCARTES METHOD

JEREMY R. JOHNSON, WERNER KRANDICK, KEVIN M. LYNCH,
DAVID G. RICHARDSON, ANATOLE D. RUSLANOV
DEPARTMENT OF COMPUTER SCIENCE
DREXEL UNIVERSITY
PHILADELPHIA, PA 19104
U.S.A.

Abstract. The Descartes method for polynomial real root isolation can be performed with respect to monomial bases and with respect to Bernstein bases. The first variant uses Taylor shift by 1 as its main subalgorithm, the second uses de Casteljau's algorithm. When applied to integer polynomials, the two variants have co-dominant, almost tight computing time bounds. Implementations of either variant can obtain speed-ups over previous state-of-the-art implementations by more than an order of magnitude if they use features of the processor architecture. We present an implementation of the Bernstein-bases variant of the Descartes method that automatically generates architecture-aware high-level code and leaves further optimizations to the compiler. We compare the performance of our implementation, algorithmically tuned implementations of the monomial and Bernstein variants, and architecture-unaware implementations of both variants on four different processor architectures and for three classes of input polynomials.

## 1. Introduction

Some years after Collins and Akritas [5] proposed an algorithm for polynomial real root isolation, Lane and Riesenfeld [24] presented a variant of the algorithm that uses Bernstein bases instead of monomial bases. Both methods proceed recursively and use the Descartes rule of signs as a termination criterion. For any input polynomial, the two methods compute the same isolating intervals since they generate the same recursion tree. An analysis of the recursion tree by Krandick and Mehlhorn [23] was improved by Eigenwillig et al. [10] who also gave a basis-free description of the Descartes method and provided explicit transformations between the two variants. When applied to integer

polynomials, both variants have co-dominant, almost tight computing time bounds [10]; the empirical computing times have never been compared [26].

The performance of the monomial variant critically depends on the efficiency of its main subalgorithm, Taylor shift by 1. It has been pointed out that asymptotically fast algorithms are available for this operation [27, 10, 11]. Indeed, von zur Gathen and Gerhard implemented and compared three asymptotically fast methods [33, 13], and Bostan et al. [4] proposed a fourth. However, these methods have not been empirically compared with implementations of classical Taylor shift by 1.

The main subalgorithm of the Bernstein-bases variant is de Casteljau's algorithm, a fundamental method in computer-aided design [12].

Since classical Taylor shift by 1 and de Casteljau's algorithm both consist of a sequence of integer additions, a high-performance version of these algorithms can be obtained by using architecture specific assembly language routines for integer addition. The implementation by Hanrot et al. [16] of the monomial Descartes method and the SYNAPS [25] implementation of the Bernstein-bases variant both build on the GMP library [14, 15] which provides such addition routines. Pursuing a different approach Johnson et al. [21] obtained high-performance Taylor shift by 1 on the UltraSPARC III processor architecture using high-level architecture-aware code and register tiling.

We show that the Taylor shift implementation of Johnson et al. achieves high performance on the Pentium EE and Opteron architectures, but not on the older Pentium 4. Von zur Gathen's and Gerhard's asymptotically fast Taylor shift by 1 turns out to be slower, for a wide range of inputs, than the classical method of Johnson et al. [21]. The register tiling techniques carry over to de Casteljau's algorithm. We implement the Descartes method using de Casteljau's algorithm with register tiling. We compare the performance of our method, the method by Hanrot et al., the SYNAPS method, and two architecture-unaware implementations from SACLIB [6]. A comparison of the five methods on four different processor architectures, and for three classes of input polynomials shows that the best absolute computing times are obtained on an Opteron processor using the Bernstein-bases variant of the Descartes method with register tiling.

## 2. The Descartes method

2.1. **Monomial vs. Bernstein bases.** The Descartes method, independent of the basis used to represent polynomials, uses binary search

to find isolating intervals and relies on the Descartes rule of signs to determine when an isolating interval has been found or when the search can stop since there are no roots in the given interval. Let $A(x) = a_m x^m + \cdots + a_1 x + a_0$. The Descartes rule states that the number of coefficient sign variations, $\mathrm{var}(A)$, is greater than or equal to the number of positive roots of $A$, and that the difference is even. This provides an exact test when $\mathrm{var}(A) \in \{0, 1\}$.

The following polynomial transformations are needed for the method and the mapping between the monomial basis and the Bernstein basis: 1) Translation $T_c(A(x)) = A(x - c)$, 2) Reciprocal transformation: $R(A(x)) = x^m A(1/x)$, and 3) Homothetic transformation: $H_a(A(x)) = A(x/a)$. The method proceeds by using a root bound and a homothetic transformation to transform the input polynomial to a polynomial, $A$, whose roots in the interval $(0, 1)$ correspond to the positive roots of the input polynomial. It can be advantageous to compute the negative roots separately using a separate root bound for the negative roots. When using the monomial basis, the Descartes rule is applied to the transformed polynomial $A^* = T_{-1}R(A)$ to determine whether $A$ has zero or one real roots in the interval $(0, 1)$. Bisection is performed by computing the transformed polynomials $A_1 = H_2(A)$ and $A_2 = T_{-1}H_2(A)$ whose roots in the interval $(0, 1)$ correspond to the roots of $A$ in the intervals $(0, 1/2)$ and $(1/2, 1)$ respectively. The Descartes rule is then applied to $A_1^* = T_{-1}R(A_1)$ and $A_2^* = T_{-1}R(A_2)$, and if more than one coefficient sign variation is obtained the algorithm proceeds recursively with the bisected polynomials.

Associated with this bisection process is a binary tree, where each node in the tree has an associated subinterval and polynomial. Each internal node requires the computation of three polynomial translations $T_{-1}$, called Taylor shift by 1, to compute the bisection polynomial and the two applications of the Descartes rule, while leaf nodes only require the polynomial translations for the application of the Descartes rule. The bulk of the computing time for the method is devoted to Taylor shift by 1. Figure 1(a) shows the classical computation of $A(x+1) = \sum_{h=0}^{m} a_{m-h,h} x^h$. Note that it is possible to avoid the complete computation of the Taylor shift in the application of the Descartes rule by stopping as soon as more than one sign variation is detected.

Let $B_{m,i}(x) = \binom{m}{i} x^i (1-x)^{m-i}$, $i = 0, \ldots, m$ be the Bernstein basis, and write $A(x) = \sum_{i=0}^{m} b_i B_{m,i}(x)$. Since $T_{-1}R(B_{m,i}(x)) = \binom{m}{i} x^{m-i}$, $T_{-1}R(A(x)) = \sum_{i=0}^{m} \binom{m}{i} b_i x^{m-i}$ and $\mathrm{var}(A^*(x)) = \mathrm{var}(b_0, ..., b_m)$. The Bernstein representation of the bisection polynomials, $A_1(x)$ and $A_2(x)$, can be obtained from the coefficients of the Bernstein representation

of $A(x)$ using de Casteljau's algorithm. In order to preserve integer coefficients a fraction-free variant is used. For $0 \le i \le m$ set $b_{0,i} = b_i$, and for $1 \le j \le m$ and $0 \le i \le m-j$ set $b_{j,i} = b_{j-1} + b_{j-1,i+1}$. As Figure 1(b) shows, this computation is similar to the computation of the Taylor shift by 1, except that the computation proceeds in the revserse direction. Eigenwillig et al. [10] remark that if $b'_i = 2^{m-i} b_{i,0}$ and $b''_i = 2^i b_{m-i,i}$, $A_1(x) = \sum_{i=0}^{m} b'_i B_{m,i}(x)$ and $A_2(x) = \sum_{i=0}^{m} b''_i B_{m,i}(x)$. This establishes a one-to-one mapping between the nodes and the associated polynomials in the search trees for the monomial and Bernstein variants of the algorithm. Moreover, the cost of the computation at each node, assuming classical algorithms for de Casteljau and Taylor shift, for both variants is codominant and hence the total computing time is codominant. In contrast to the monomial basis, each internal node requires one application of de Casteljau's algorithm instead of three Taylor shifts by one and no transformations are required for leaf nodes. A similar approach, called the dual algorithm, which also reduces the number of Taylor shifts by computing $A_1^*(x)$ and $A_2^*(x)$ directly from $A^*(x)$ using monomial bases was suggested by Johnson [19].

$$
\begin{array}{ccccccccc}
 & & 0 & 0 & 0 & 0 & & \\
 & & \downarrow & \downarrow & \downarrow & \downarrow & & \\
a_n & \rightarrow & a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} & & \\
a_{n-1} & \rightarrow & a_{1,0} & a_{1,1} & a_{1,2} & & \searrow & \text{(a)} \\
a_{n-2} & \rightarrow & a_{2,0} & a_{2,1} & & \searrow & & \\
a_{n-3} & \rightarrow & a_{3,0} & & \searrow & & & \\
 & & & \searrow & & & & \\
 & & b''_3 & b''_2 & b''_1 & b''_0 & & \\
 & & \uparrow & \uparrow & \uparrow & \uparrow & & \\
b'_3 & \leftarrow & b_{3,0} & b_{2,1} & b_{12} & b_{0,3} & & \\
b'_2 & \leftarrow & b_{2,0} & b_{1,1} & b_{0,2} & & \nwarrow & \text{(b)} \\
b'_1 & \leftarrow & b_{1,0} & b_{0,1} & & \nwarrow & & \\
b'_0 & \leftarrow & b_{0,0} & & \nwarrow & & & \\
 & & & \nwarrow & & & &
\end{array}
$$

FIGURE 1. (a) The pattern of integer additions in Pascal's triangle, $a_{i,j} = a_{i,j-1} + a_{i-1,j}$, can be used to perform Taylor shift by 1. (b) In de Casteljau's algorithm all dependencies are reversed, the intermediate results are computed according to the recursion $b_{j,i} = b_{j-1,i} + b_{j-1,i+1}$.

2.2. **The monomial SACLIB method IPRRID.** The program IPRRID in the SACLIB library [6] processes the bisection tree in breadth-first order [22, 27]. IPRRID tries to avoid the complete computation of the Taylor shift in the application of the Descartes rule by stopping as soon as more than one sign variation is detected. Also, IPRRID checks whether $\mathrm{var}(A) \neq 0$ before computing $T_{-1}R(A)$. The program IUPTR1 that implements Taylor shift by 1 operates on a single array containing the coefficients of the input polynomial. IUPTR1 avoids the overhead of calling integer addition routines and of normalizing after each integer addition [21].

2.3. **The Bernstein SACLIB method IPRRIDB.** The program IPRRIDB in the SACLIB library [6] converts the input polynomial from its monomial representation into a fraction-free Bernstein-basis representation. IPRRIDB processes the bisection tree in the same way as the program IPRRID of Section 2.2. IPRRIDB uses a fraction-free version of de Casteljau's algorithm that avoids the overhead of calling integer addition routines and of normalizing after each integer addition—in the same way as the program IUPTR1 of Section 2.2.

2.4. **The method by Hanrot et al.** Hanrot et al. [16] provide an efficient implementation of the monomial version of the Descartes method that incorporates the memory-saving technique of Rouillier and Zimmermann [27]. Their implementation uses GMP [15] for the integer additions required by Taylor shift operations. Additional algorithmic optimizations are included to reduce the time spent on Taylor shift. The complete execution of the Taylor shift used to compute $T_{-1}R$ prior to the application of the Descartes rule is not needed in many situations. If all of the input coefficients are of the same sign, then the transformed polynomial will have zero coefficient sign variations and the Taylor shift can be avoided. If all of the intermediate coefficients in a column of the Taylor shift computation have the same sign, then the remaining result coefficients will have the same sign, and the computation can be aborted. If exactly two sign variations are reported then there are either zero or two roots in the interval. If the signs of the polynomial evaluated at 0 and 1 are equal but different from the sign at 1/2, then two roots have been found and the algorithm can terminate avoiding the additional Taylor shifts needed for the Descartes test to report the termination. This test is efficient to apply since the polynomial evaluated at 1 is equal to the sum of the coefficients and the sum is known after computing the first column of intermediate coefficients in the Taylor shift computation. In practice, computation of a partial

rather than a complete Taylor shift along with the early termination tests can save a substantial amount of time.

In a pre-processing step the method determines the greatest $k$ such that the input polynomial $A(x)$ is a polynomial in $x^k$, and replaces $A(x)$ by $A(\sqrt[k]{x})$. If $k$ is even, the method isolates only the positive roots.

2.5. **The SYNAPS method.** The SYNAPS [25] implementation IslBzInteger<QQ> [11] of the Descartes method uses GMP [15] for the integer additions required by the de Casteljau operations. Otherwise, the method is a straightforward implementation of the Bernstein-bases variant. A hardcoded limitation of the recursion depth to 96 prevents the method from isolating the roots of Mignotte polynomials of degrees greater than 80.

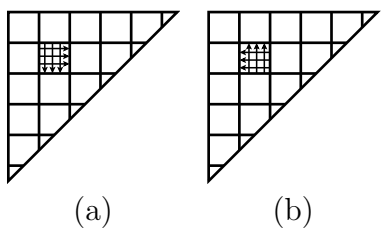


FIGURE 2. Register tiling can be applied to (a) Taylor Shift and (b) de Casteljau's algorithm.

2.6. **The register tile method.** Figure 2 shows that both, Taylor shift and de Casteljau's method, are amenable to register tiling, a method that reduces the number of memory accesses and exploits instruction-level parallelism [21].

A tile can be described by three parameters: tile shape, tile size, and addition schedule. The shape of the tiles is determined by the computational structure of the Taylor shift and de Casteljau algorithms. Possible addition schedules are also constrained by the computation structure of the Taylor shift and de Casteljau algorithms. In general, more than one addition schedule is possible, and the number of possible schedules increases with the number of addition units in the target CPU. The optimal tile size depends on the number of registers in the target CPU and the quality with which the CPU schedules memory operations. We found that the best performing parameter values varied widely depending on the target CPU.

In order to allow our tiled implementations to be used on multiple architectures, we have implemented an automatic code generator in

Perl [34]. It produces portable ANSI C++ [18, 30] code for tiles of different sizes, compiles and executes the code, and searches through successively larger tile sizes until a tile size with peak performance is discovered. A single addition schedule is produced by the code generator for a CPU with two addition units. The purpose of this part of the code generator is not to produce an optimal addition schedule, but rather to produce a schedule that exposes the dependencies of the computation in such a way that the C++ compiler can then schedule the computation appropriately for the target architecture. All compilers we have used were unable to infer these dependencies when the computation is programmed in "for"-loops. The code generator for the Taylor shift and the code generator for de Casteljau's method both follow the same architecture and share a good deal of common code. This is the case because changes in index computations suffice to "reverse the arrows" when going from Taylor shift to de Casteljau's algorithm.

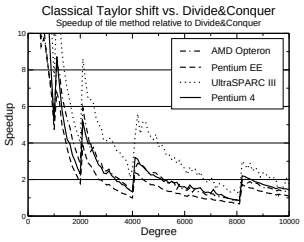


FIGURE 3. Classical Taylor shift with register tiling is faster than the fastest known asymptotically fast Taylor shift for a wide range of inputs.

2.7. **Asymptotically fast Taylor shift.** Von zur Gathen and Gerhard experimentally compared six methods for computing Taylor shifts. They implemented all methods on top of the NTL library [29, 28] and identified a divide-and-conquer method as the fastest method.

We performed experiments using their implementation and confirmed that, on the Pentium EE, the Opteron, and the UltraSPARC III, the divide-and-conquer method—for degrees $\geq 255$ and "large" coefficients–is indeed faster than the other implementations of asymptotically fast Taylor shift.

Using the same NTL-based implementation we compared its speed to classical Taylor shift with register tiling. We note that NTL represents integers using just 32-bits of each word due to the way it implements integer multiplication while our Taylor shift uses the full 64-bit word. Figure 3 shows that our method is faster on a wide range of inputs.

None of the Descartes methods we compare in this paper uses asymptotically fast Taylor shift.

## 3. Experimental procedures

3.1. **Processor architecture.** The tiled Bernstein method primarily achieves its speedup from delaying carries and using register tiling to improve locality of reference. The computation schedule for the register tiles allows multiple integer execution units to be used simultaneously in the processing of a register tile. When implementing the tiled Bernstein method for a given processor, the precision of the native integer arithmetic, the number of general purpose integer registers, and the number of integer execution units determines the maximum speedup that can be obtained by the method; the speedup will be larger with high native integer precision and a larger number of general purpose integer registers and integer execution units.

3.1.1. *64-bit processors.* Current processors such as the Pentium EE [9], Opteron [2, 1], and UltraSPARC III [17, 32] support native 64-bit integer operations, have at least 16 64-bit general purpose integer registers, and at least 2 integer execution units. These are the kind of processors for which the tiled Bernstein method was developed. Below, we briefly summarize the salient features of these three processors.

**Pentium EE.** The Intel Pentium Extreme Edition dual-core processor supports both the 32-bit x86 and 64-bit EM64T instruction sets. Each core of the Pentium EE provides 16 64-bit general purpose integer registers and has an 8-way set-associative 16 kilobyte L1 data cache and an 8-way set-associative 1 megabyte L2 cache. [9]

**Opteron.** The AMD Opteron processor supports the 32-bit x86 and 64-bit AMD64 instruction sets. The Opteron provides 16 64-bit general purpose integer registers and has a 2-way set-associative 64 kilobyte L1 data cache and a 4-way set-associative 1 megabyte L2 cache. [2, 1]

**UltraSPARC III.** The Sun UltraSPARC III processor [17, 32] supports the SPARC V9 instruction set. The UltraSPARC III provides 32 64-bit general purpose integer registers and has a 64 kilobyte 4-way set-associative L1 data cache and an 8 megabyte 2-way set-associative L2 cache.

3.1.2. *32-bit processors.* The Pentium 4 [8] is included for comparison only and is not expected to perform well with the tiled Bernstein method due to the unavailability of native 64-bit integer arithmetic and the small number of general purpose integer registers.

**Pentium 4.** The Intel Pentium 4 processor supports the 32-bit x86 instruction set. The Pentium 4 provides 8 32-bit general purpose integer registers and has a 16 kilobyte 8-way set-associative L1 data cache and a 1 megabyte 8-way set-associative L2 cache. [8]

3.2. **Hardware configuration and timing.** The getrusage system call is used on all platforms to obtain timings. The hardware platforms used in this study are configured as follows.

**Pentium EE.** We use a Pentium Extreme Edition 840 Dual-Core CPU with a clock speed of 3.2 GHz and 1 GB of main memory. The Gentoo Linux distribution with the 2.6.14-gentoo-r2 kernel is installed. Hyper-Threading is disabled in the BIOS.

**Opteron.** We use an Opteron 244 with a clock speed of 1.8 GHz and 2 GB of main memory. The Gentoo Linux distribution with the 2.6.14-gentoo-r2 kernel is installed.

**UltraSPARC III.** We use a Sun Blade 2000 with two 900 MHz UltraSPARC III processors and 2 GB of main memory. The Solaris 9 operating system is installed.

**Pentium 4.** We use a Pentium 4 with a clock speed of 3.0 GHz and 1 GB of main memory. The Fedora Core 2 Linux distribution is installed.

3.3. **Compilation protocol. SACLIB monomial and SACLIB Bernstein.** The SACLIB [6] programs IPRRID and IPRRIDB are compiled using gcc 3.4.4 with the flags "-O3 -march=nocona -m64" on the Pentium EE, gcc 3.4.4 with the flags "-O3 -march=opteron -m64" on the Opteron, Sun Studio 9 compilers [31] with the flags "-xO3" on the UltraSPARC III, and gcc 3.3.3 with the flags "-O3 -march=pentium4" on the Pentium 4.

**Tiled Bernstein.** The tiled Bernstein method is compiled with the same compilers and flags as the SACLIB methods. An additional step for the tiled Bernstein method is searching for the correct register tile size. We search over tiles of size $n \times n$ for $n = 4$, 6, 8, 10, 12, 14, and 16. Smaller tiles are too small to offer a speedup and larger tiles will result in register spilling that will negate the locality of reference advantages of the tiled Bernstein method. The outcome of the tile search is shown in Figure 4. Based on the outcome of the search we use register tile sizes of $12 \times 12$ for the Pentium EE and Opteron, $8 \times 8$ on the UltraSPARC III, and $6 \times 6$ on the Pentium 4. The dip that appears in the graphs is due to increased execution time of the tiled Taylor shift at degree 4100 caused by an increased cache miss rate that is likely caused by the low set associativity organization of the

UltraSPARC and Opteron data L1 caches. The dip also appears when GMP adds integers that are 4100 words long.

**NTL.** On the Pentium EE, Opteron, and Pentium 4, NTL 5.4 [28] is compiled with the same compilers as the SACLIB method. Compiler flags are the defaults set by NTL. On the UltraSPARC III, NTL 5.4 is compiled with the Sun Studio 9 compiler [31] with the flags "-xO3 -xarch=v9b". Because of the way it performs multiplication, NTL is limited to 32-bit integer arithmetic; however, for compatibility, NTL is compiled to use the 64-bit ABI.

**GMP.** On the Pentium EE, Opteron, UltraSPARC III, and Pentium 4, GMP 4.2 [15] is compiled using the same compilers as the SACLIB method. Compiler flags are the defaults set by GMP.

**SYNAPS.** On the Pentium EE, Opteron, and Pentium 4, SYNAPS 2.4 is compiled with the same compilers and flags as the SACLIB method. On the UltraSPARC III, SYNAPS 2.4 is compiled with the Sun Studio 9 C++ compiler with the flags "-x03 -xarch=v9b". SYNAPS required minor porting before it could be compiled with the Sun Studio 9 C++ compiler.

**Hanrot et al.** The code of Hanrot et al. is compiled with the same compilers and flags as SYNAPS.

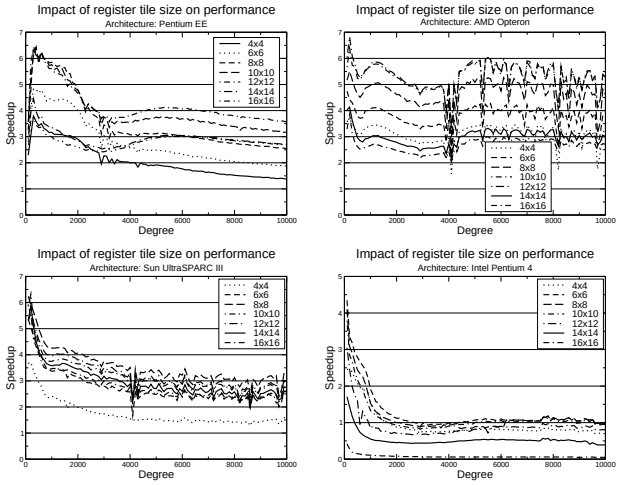


FIGURE 4. Speedup obtained by the tiled version of the Taylor shift by 1 algorithm on the Pentium EE, Opteron, UltraSPARC III, and Pentium 4 for different register tile sizes. Speedup is calculated with respect to a straightforward GMP-based implementation of Taylor shift by 1. The dips in the Opteron and UltraSPARC curves are discussed in Section 3.3.

3.4. **Input polynomials.** In our experiments we isolate the real roots of three kinds of polynomials.

**Random polynomials** are polynomials with integer coefficients of absolute value less than $2^{20}$; the coefficients are pseudo-randomly generated from a uniform distribution. For each degree we generate 50 random polynomials and report average computing times for degrees 100, 200, ..., 1000. For random polynomials, the Descartes method produces recursion trees that typically have few nodes.

**Chebyshev polynomials** are the polynomials defined by the recurrence relation $T_0(x) = 1$, $T_1(x) = x$, $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$. The roots of Chebyshev polynomials are well-known values of the cosine function. We apply the Descartes method to Chebyshev polynomials in order to obtain wide recursion trees with many nodes. We report computing times for degrees 100, 200, ..., 1000. Since all these degrees are even, the corresponding Chebyshev polynomials are polynomials in $x^2$. Since, for even $n$, the method by Hanrot et al. reduces $T_n(x)$ to $T_n(\sqrt{x})$ we apply the same pre-processing step also to the other methods. We call the polynomials $T_n(\sqrt{x})$ somewhat ambiguously "reduced Chebyshev polynomials of degree $n$"; of course, $\deg(T_n(\sqrt{x})) = n/2$.

**Mignotte polynomials** are defined by $x^n - 2\,(5x-1)^2$. We are not aware of any applications that involve Mignotte polynomials; however, the Descartes method generates extremely deep recursion trees for Mignotte polynomials, and it requires computing times that are approximately proportional to its worst-case computing time function. We report computing times for degrees 100, 200, ..., 600.

An analysis of the computing time needed to isolate the roots of polynomials from these classes was given by Johnson [19, 20].

## 4. Results and discussion

We measured the executions times of the five implementations of the Descartes method on the four processor architectures for input polynomials of various degrees from the three classes of polynomials. The raw data are given in Table 1. Figures 5, 6, 7 and 8 show, for input polynomials of various degrees, the speed-up that the various methods obtain with respect to the SACLIB routine IPRRID. Speed-ups by an order of magnitude are typical. The largest speed-up is by a factor of 24, and it is obtained by the Bernstein-based variant of the Descartes method with register tiling on an Opteron processor for the Chebyshev polynomial of degree 1000.

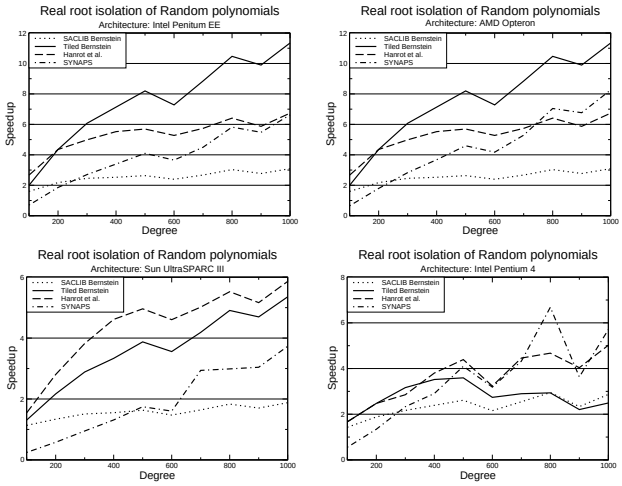


FIGURE 5. Speedup with respect to the monomial SACLIB implementation for random polynomials on four architectures.

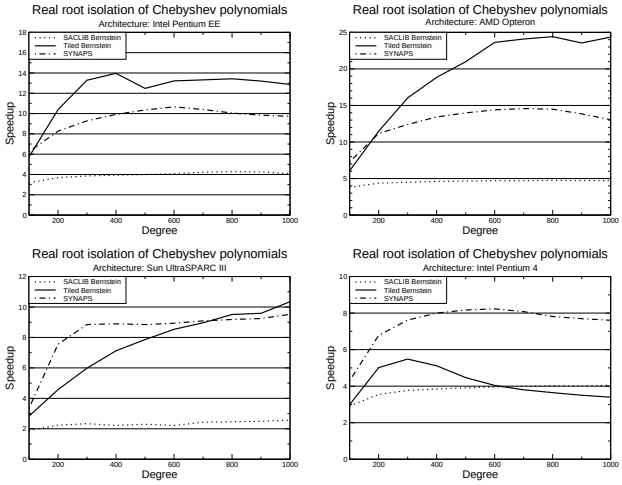


FIGURE 6. Speedup with respect to the monomial SACLIB implementation for Chebyshev polynomials on four architectures.

4.1. **Processor architectures.** The data show that register tiling works best on architectures where the optimal tile size is large. According to Section 3.3 and Figure 4 the optimal register tile size on the Pentium EE and Opteron is $12 \times 12$, on the UltraSPARC III it is $8 \times 8$, and on the Pentium 4 it is $6 \times 6$. In order to have a large optimal tile

size an architecture must provide a large word length and a large number of registers. By Section 3.2, the Pentium EE and Opteron provide both, the Pentium 4 provides neither. Hence, register tiling achieves large speed-ups on the Pentium EE and Opteron and small speed-ups on the Pentium 4.

Register tiling can also profit from the instruction level parallelism (ILP) that is afforded by the number of integer execution units since the rows of each tile can be processed concurrently. The UltraSPARC III can execute up to 2 integer instructions per cycle. The Opteron processor is based on Athlon's QuantiSpeed architecture [3, pp. 250–252][1] which is capable of executing 3 simple integer instructions per cycle. Both Pentium 4 and Pentium EE are based on the NetBurst microarchitecture [7] which is capable of executing up to 4 simple integer instructions per cycle on 2 ALUs per core. The difference in the number of integer execution units might explain the observation that the Descartes method involving register tiling is the one that profits the most when ported from the UltraSPARC III to the Pentium EE and Opteron. Since the adaptation to the new architecture is obtained by optimizing the register tile size, and this change is performed automatically, no human effort is required to take advantage of the new architectures.

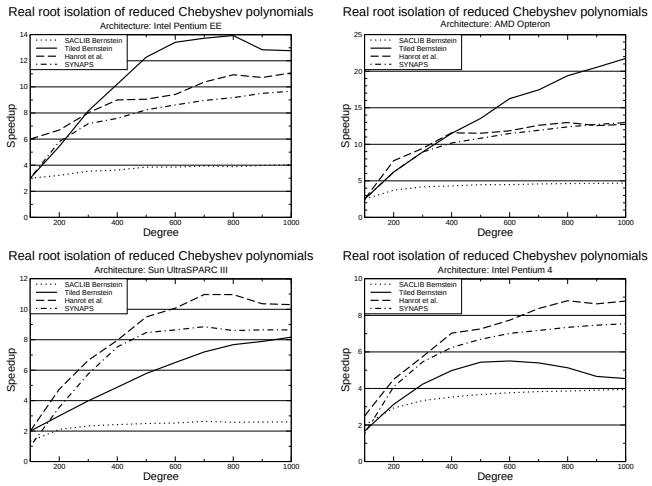


FIGURE 7. Speedup with respect to the monomial SACLIB implementation for reduced Chebyshev polynomials on four architectures.

4.2. **Algorithms.** The data show that high performance can be achieved using a number of algorithmic devices.

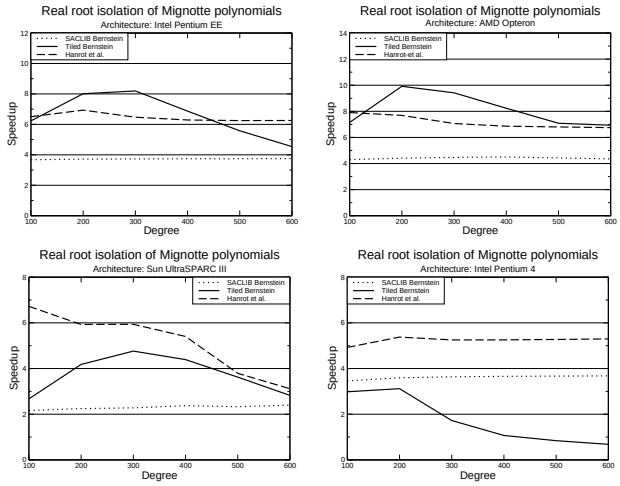


FIGURE 8. Speedup with respect to the monomial SACLIB implementation for Mignotte polynomials on four architectures.

(1) The use of Bernstein bases can be viewed as a way to reduce the number of $n^3$-operations per internal node of the recursion tree in the monomial variant from 3 Taylor shifts to 1 de Casteljau transformation. A comparison between the SACLIB methods IPRRID and IPRRIDB shows that this approach is successful—despite the fact that the initial transformation of the input polynomial into the Bernstein-base representation can increase the coefficient length.

(2) Hanrot et al. achieve a similar reduction in the number of $n^3$-operations by partial execution of certain Taylor shifts. In addition, their early termination test avoids all $n^3$-operations at certain leaf nodes. For reduced Chebyshev polynomials this device reduces the number of complete Taylor shifts by 40%.

(3) The use of the assembly-language integer addition routines of GMP makes the SYNAPS method faster than the SACLIB method IPRRIDB for polynomials with long coefficients, and it contributes to making the method by Hanrot et al. faster than the SACLIB method IPRRID.

(4) The use of register tiling is orthogonal to devices (1) and (2). In fact, in an additional experiment we replaced the complete Taylor shift in the method by Hanrot et al. with our tiled Taylor shift and obtained an additional speed-up by a factor of about 1.33.

The three implementations of the Bernstein-bases variant might be further sped-up by incorporating the early termination test by Hanrot et al.

The data show that—with minor exceptions—for all classes of polynomials, the best absolute computing times are achieved on the Opteron processor using the Bernstein-bases variant of the Descartes method with register tiling.

## 5. Acknowledgements

The authors would like to thank Jürgen Gerhard, Guillaume Hanrot et al., Bernard Mourrain et al., and George Collins, who developed IPRRIDB, for making their code available.

| | | Pentium EE | | | | | Opteron | | | | | UltraSPARC III | | | | | Pentium 4 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Deg | SACLIB monomial | SACLIB Bernstein | Tiled Bernstein | Hanrot et al. | SYNAPS | SACLIB monomial | SACLIB Bernstein | Tiled Bernstein | Hanrot et al. | SYNAPS | SACLIB monomial | SACLIB Bernstein | Tiled Bernstein | Hanrot et al. | SYNAPS | SACLIB monomial | SACLIB Bernstein | Tiled Bernstein | Hanrot et al. | SYNAPS |
| Random | 100 | 8 | 5 | 4 | 4 | 11 | 8 | 5 | 4 | 3 | 12 | 17 | 15 | 13 | 11 | 69 | 10 | 7 | 6 | 6 | 18 |
| Random | 200 | 81 | 44 | 20 | 25 | 44 | 87 | 40 | 20 | 20 | 48 | 143 | 107 | 66 | 51 | 247 | 94 | 50 | 38 | 38 | 70 |
| Random | 300 | 311 | 148 | 57 | 103 | 115 | 334 | 136 | 55 | 67 | 118 | 531 | 352 | 184 | 139 | 560 | 405 | 187 | 128 | 142 | 174 |
| Random | 400 | 787 | 360 | 128 | 194 | 232 | 856 | 339 | 120 | 155 | 233 | 1347 | 869 | 404 | 292 | 1027 | 996 | 417 | 283 | 262 | 343 |
| Random | 500 | 1708 | 733 | 252 | 430 | 417 | 1885 | 716 | 230 | 331 | 410 | 2976 | 1824 | 768 | 600 | 1705 | 2468 | 945 | 687 | 562 | 602 |
| Random | 600 | 2257 | 1071 | 376 | 687 | 617 | 2526 | 1055 | 347 | 479 | 605 | 4020 | 2732 | 1130 | 873 | 2509 | 2875 | 1334 | 1050 | 889 | 902 |
| Random | 700 | 4416 | 1884 | 641 | 1058 | 982 | 4981 | 1864 | 563 | 868 | 942 | 7922 | 4829 | 1892 | 1579 | 2695 | 6108 | 2401 | 2108 | 1370 | 1414 |
| Random | 800 | 8706 | 3309 | 1143 | 2361 | 1496 | 9870 | 3258 | 943 | 1540 | 1402 | 15546 | 8482 | 3168 | 2815 | 5203 | 14193 | 4814 | 4836 | 3037 | 2124 |
| Random | 900 | 11679 | 4832 | 1610 | 2090 | 2132 | 13190 | 4754 | 1333 | 2247 | 1949 | 21250 | 12483 | 4526 | 4115 | 6986 | 10946 | 4690 | 4967 | 2712 | 3014 |
| Random | 1000 | 18155 | 6761 | 2274 | 3417 | 2741 | 20536 | 6645 | 1810 | 3048 | 2486 | 32647 | 17380 | 6096 | 5570 | 8750 | 22100 | 7712 | 8871 | 4402 | 3874 |
| Chebyshev | 100 | 344 | 108 | 60 | 4 | 55 | 364 | 96 | 60 | 8 | 50 | 510 | 270 | 180 | 20 | 152 | 344 | 118 | 115 | 8 | 80 |
| Chebyshev | 200 | 6316 | 1712 | 608 | 52 | 764 | 6700 | 1532 | 584 | 48 | 600 | 8820 | 3940 | 1930 | 120 | 1168 | 6488 | 1829 | 1294 | 80 | 957 |
| Chebyshev | 300 | 34606 | 8980 | 2604 | 240 | 3728 | 37134 | 8268 | 2316 | 216 | 2995 | 48470 | 20740 | 8100 | 450 | 5477 | 35613 | 9455 | 6495 | 343 | 4675 |
| Chebyshev | 400 | 115675 | 29409 | 8284 | 708 | 11664 | 128004 | 27777 | 6792 | 584 | 9535 | 168280 | 75840 | 23620 | 1220 | 18924 | 119135 | 30989 | 23292 | 942 | 14895 |
| Chebyshev | 500 | 296434 | 74340 | 23748 | 1860 | 28625 | 330172 | 70956 | 15737 | 1524 | 23610 | 439010 | 190800 | 55880 | 2630 | 49640 | 305045 | 78069 | 68264 | 2365 | 37335 |
| Chebyshev | 600 | 638131 | 156701 | 48267 | 3720 | 59842 | 704420 | 149721 | 29806 | 3172 | 48929 | 940020 | 423620 | 110160 | 5260 | 105222 | 650096 | 164264 | 160848 | 4691 | 78986 |
| Chebyshev | 700 | 1207523 | 285578 | 90662 | 6540 | 116087 | 1342976 | 285493 | 55752 | 5844 | 92164 | 1804440 | 741890 | 201320 | 9710 | 198604 | 1240013 | 310566 | 326062 | 8334 | 153477 |
| Chebyshev | 800 | 2106963 | 492447 | 156935 | 10688 | 209550 | 2341022 | 492447 | 95889 | 9973 | 161582 | 3149370 | 1278380 | 331230 | 16590 | 343092 | 2167662 | 538754 | 594845 | 13734 | 277534 |
| Chebyshev | 900 | 3455690 | 811711 | 261720 | 18033 | 351207 | 3839292 | 811711 | 163023 | 17093 | 277293 | 5172400 | 2076420 | 539700 | 29300 | 559625 | 3556937 | 885691 | 1015461 | 23189 | 462300 |
| Chebyshev | 1000 | 5388421 | 1320835 | 418654 | 27326 | 553752 | 5988138 | 1268569 | 245834 | 26258 | 458819 | 8282860 | 3215740 | 800600 | 45970 | 870306 | 5550107 | 1372243 | 1633486 | 35406 | 729782 |
| Reduced | 100 | 24 | 8 | 8 | 4 | 8 | 20 | 8 | 8 | 8 | 8 | 40 | 30 | 20 | 20 | 40 | 20 | 10 | 12 | 8 | 13 |
| Reduced | 200 | 348 | 108 | 64 | 52 | 60 | 372 | 100 | 60 | 48 | 60 | 570 | 270 | 190 | 120 | 160 | 360 | 123 | 115 | 80 | 88 |
| Reduced | 300 | 1924 | 544 | 236 | 240 | 268 | 2040 | 488 | 228 | 216 | 228 | 2990 | 1280 | 750 | 450 | 520 | 1969 | 590 | 465 | 343 | 361 |
| Reduced | 400 | 6372 | 1756 | 624 | 708 | 840 | 6760 | 1572 | 588 | 584 | 664 | 9710 | 4000 | 1990 | 1220 | 1290 | 6619 | 1874 | 1329 | 942 | 1060 |
| Reduced | 500 | 16845 | 4372 | 1372 | 1860 | 2044 | 17545 | 3920 | 1296 | 1524 | 1620 | 24980 | 10000 | 4320 | 2630 | 2950 | 17166 | 4679 | 3156 | 2365 | 2565 |
| Reduced | 600 | 35046 | 9092 | 2612 | 3720 | 4068 | 37574 | 8380 | 2312 | 3172 | 3276 | 52990 | 20970 | 8150 | 5260 | 6130 | 36299 | 9642 | 6594 | 4691 | 5175 |
| Reduced | 700 | 67820 | 17237 | 4940 | 6540 | 7568 | 73672 | 16105 | 4224 | 5844 | 6184 | 106540 | 40390 | 14800 | 9710 | 12030 | 69815 | 18231 | 12932 | 8334 | 9730 |
| Reduced | 800 | 116783 | 30049 | 8380 | 10688 | 12729 | 129424 | 27993 | 6676 | 9973 | 10453 | 181930 | 70670 | 23700 | 16590 | 21150 | 120908 | 31333 | 23556 | 13734 | 16459 |
| Reduced | 900 | 193304 | 48583 | 15049 | 18033 | 20333 | 214697 | 46211 | 10456 | 17093 | 16885 | 303660 | 117100 | 38490 | 29300 | 35110 | 200134 | 51231 | 42960 | 23189 | 26815 |
| Reduced | 1000 | 302510 | 75012 | 23685 | 27326 | 31286 | 334341 | 71400 | 15377 | 26258 | 25766 | 473670 | 181870 | 57990 | 45970 | 54780 | 311011 | 78886 | 68456 | 35406 | 41241 |
| Mignotte | 100 | 4736 | 1288 | 760 | 728 | N/A | 5092 | 1184 | 712 | 644 | N/A | 6790 | 3140 | 2540 | 1010 | N/A | 4851 | 1402 | 1627 | 985 | N/A |
| Mignotte | 200 | 143673 | 38642 | 17932 | 20721 | N/A | 159362 | 36122 | 16069 | 20729 | N/A | 214470 | 95480 | 51320 | 36170 | N/A | 147705 | 41103 | 47363 | 27469 | N/A |
| Mignotte | 300 | 1083263 | 290385 | 132196 | 167387 | N/A | 1209179 | 270337 | 128412 | 171159 | N/A | 1619100 | 710360 | 339900 | 272690 | N/A | 1118406 | 307373 | 648885 | 212914 | N/A |
| Mignotte | 400 | 4536036 | 1213095 | 660173 | 721453 | N/A | 5081274 | 1127943 | 617024 | 740666 | N/A | 7208290 | 3037420 | 1641610 | 1333880 | N/A | 4691741 | 1282456 | 4383761 | 892932 | N/A |
| Mignotte | 500 | 13777640 | 3682927 | 2470037 | 2204722 | N/A | 15445227 | 3488480 | 2179611 | 2270234 | N/A | 24018450 | 10311010 | 6627000 | 6335480 | N/A | 14239431 | 3881311 | 16920331 | 2700368 | N/A |
| Mignotte | 600 | 34211121 | 9110764 | 7533232 | 5468834 | N/A | 38326948 | 8799792 | 5526025 | 5678407 | N/A | 67752340 | 28252170 | 23996430 | 21738150 | N/A | 35312569 | 9598795 | 51724931 | 6666565 | N/A |

TABLE 1. Root isolation timings in milliseconds.

## References


[1] AMD. AMD Eighth-Generation Processor Architecture. `http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/Hammer_architecture_WP_2.pdf`, October 2001.

[2] AMD. Processor Reference. `http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/23932.pdf`, June 2004.

[3] AMD. *Software Optimization Guide for AMD64 Processors*, September 2005.

[4] Alin Bostan, Philippe Flajolet, Bruno Salvy, and Éric Schost. Fast computation of special resultants. *Journal of Symbolic Computation*, 41(1):1–29, 2006.

[5] George E. Collins and Alkiviadis G. Akritas. Polynomial real root isolation using Descartes' rule of signs. In R. D. Jenks, editor, *Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation*, pages 272–275. ACM Press, 1976.

[6] George E. Collins et al. `SACLIB` User's Guide. Technical Report 93-19, Research Institute for Symbolic Computation, RISC-Linz, Johannes Kepler University, A-4040 Linz, Austria, 1993.

[7] Intel Corporation. *A Detailed Look Inside the Intel NetBurst Micro-Architecture of the Intel Pentium 4 Processor*, November 2000.

[8] Intel Corporation. *The IA-32 Intel Architecture Optimization: Reference Manual*, 2004.

[9] Intel Corporation. *Intel Pentium D Processor 800 Sequence: Datasheet*, 2006.

[10] Arno Eigenwillig, Vikram Sharma, and Chee K. Yap. Almost tight recursion tree bounds for the Descartes method. In J.-G. Dumas, editor, *International Symposium on Symbolic and Algebraic Computation*. ACM Press, 2006. To appear.

[11] I. Z. Emiris, B. Mourrain, and E. Tsigaridas. Real algebraic numbers: Complexity analysis and experimentations. Research Report 5897, INRIA, 2006.

[12] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, 1988.

[13] Jürgen Gerhard. *Modular Algorithms in Symbolic Summation and Symbolic Integration*, volume 3218 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.

[14] Torbjörn Granlund. *GNU MP: The GNU Multiple Precision Arithmetic Library*. Swox AB, September 2004. Edition 4.1.4.

[15] Torbjörn Granlund. *GNU MP: The GNU Multiple Precision Arithmetic Library*. Swox AB, March 2006. Edition 4.2.

[16] Guillaume Hanrot, Fabrice Rouillier, Paul Zimmermann, and Sylvain Petitjean. Uspensky's algorithm. `http://www.loria.fr/equipes/vegas/qi/usp/usp.c`, 2004.

[17] Tim Horel and Gary Lauterbach. UltraSPARC-III: Designing third-generation 64-bit performance. *IEEE MICRO*, 19(3):73–85, 1999.

[18] International Standards Organization, `http://www.iso.org`. *ISO/IEC 14882:2003: Programming languages—C++*, 2003.

[19] J. R. Johnson. *Algorithms for Polynomial Real Root Isolation*. Technical research report OSU-CISRC-8/91-TR21, The Ohio State University, Department of Computer and Information Science, 1991.

[20] J. R. Johnson. Algorithms for polynomial real root isolation. In B. F. Caviness and J. R. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 269–299. Springer-Verlag, 1998.

[21] Jeremy R. Johnson, Werner Krandick, and Anatole D. Ruslanov. Architecture-aware classical Taylor shift by 1. In M. Kauers, editor, *International Symposium on Symbolic and Algebraic Computation*, pages 200–207. ACM Press, 2005.

[22] Werner Krandick. Isolierung reeller Nullstellen von Polynomen. In J. Herzberger, editor, *Wissenschaftliches Rechnen*, pages 105–154. Akademie Verlag, Berlin, 1995.

[23] Werner Krandick and Kurt Mehlhorn. New bounds for the Descartes method. *Journal of Symbolic Computation*, 41(1):49–66, 2006.

[24] Jeffrey M. Lane and R. F. Riesenfeld. Bounds on a polynomial. *BIT*, 21(1):112–117, 1981.

[25] B. Mourrain, J. P. Pavone, P. Trébuchet, and E. Tsigaridas. SYNAPS: A library for symbolic-numeric computation. Software presentation. MEGA 2005, Sardinia, Italy, May 2005. `http://www-sop.inria.fr/galaad/logiciels/synaps/`.

[26] Bernard Mourrain, Fabrice Rouillier, and Marie-Françoise Roy. The Bernstein basis and real root isolation. In J. E. Goodman, J. Pach, and E. Welzl, editors, *Combinatorial and Computational Geometry*, volume 52 of *Mathematical Sciences Research Institute Publications*, pages 459–478. Cambridge University Press, 2005.

[27] Fabrice Rouillier and Paul Zimmermann. Efficient isolation of a polynomial's real roots. *Journal of Computational and Applied Mathematics*, 162:33–50, 2004.

[28] Victor Shoup. *NTL: A Library for Doing Number Theory.* `http://www.shoup.net/ntl`.

[29] Victor Shoup. A new polynomial factorization algorithm and its implementation. *Journal of Symbolic Computation*, 20(4):363–397, 1995.

[30] Bjarne Stroustrup. *The C++ Standard: Incorporating Technical Corrigendum No. 1.* John Wiley and Sons, 2003.

[31] Sun Microsystems. Sun Studio Collection. `http://www.sun.com/software/products/studio/`.

[32] Sun Microsystems. *UltraSPARC III Cu: User's Manual*, 2004.

[33] Joachim von zur Gathen and Jürgen Gerhard. Fast algorithms for Taylor shifts and certain difference equations. In W. W. Küchlin, editor, *International Symposium on Symbolic and Algebraic Computation*, pages 40–47. ACM Press, 1997.

[34] Larry Wall, Tom Christiansen, and Jon Orwant. *Programming Perl.* O'Reilly, 3rd edition, 2000.