$\mathcal{NC}^2$ Computation of Gcd-free Basis and Application to Parallel Algebraic Numbers Computation.

T. Gautier and J.L. Roch

Institute for Scientific Computation
ETH-Zentrum IFW
CH-8092 Zürich, Switzerland
Email: gautier@inf.ethz.ch

LMC-IMAG
Institut Fourier, BP 53X
100, rue des Mathématiques
38041 Grenoble Cedex 9, France
Email: Jean-Louis.Roch@imag.fr

Abstract

We establish that the problem of computing a gcd-free basis for a set of polynomials is in $\mathcal{NC}_F^2$ for any arbitrary field F. This leads to a proof that arithmetic for a simple algebraic extension is in $\mathcal{NC}_F^2$. This result is applied to improve the complexity of the parallel deterministic algorithm to compute the Jordan normal form of a n dimensional matrix in time $\mathrm{O}(\log^2 n)$.

1 Introduction

The computation of the Jordan normal form of a matrix has many applications such as computing matrices functions, solving matrix equations or differential equations and systems. The Jordan form has been widely studied from a theoretical point of view [10], and sequential polynomial time algorithms are known [20, 14, 13]. From a parallel point of view, the first fast parallel algorithms [17, 13] are randomized. Those results have been improved in[21, 22] where algorithms are given to compute the Jordan normal form in parallel arithmetic time $\mathrm{O}(\log^3 n)$ using $n^{\mathrm{O}(1)}$ processors for any field F.

The later algorithm [22] is based on algebraic number computation in a parallel D5 [7, 8, 9] arithmetic manner and computes the symbolic Jordan normal form as defined in [17]. The main tool involved in parallel D5 arithmetic is the computation of gcd-free bases [16] of polynomials. Using algorithms proposed in [17, 1], this computation requires $\mathrm{O}(\log^3 n)$ arithmetic steps which dominates the cost of the computation of the Jordan form in [21].

This paper is devoted to a new algorithm that computes a gcd-free basis in parallel arithmetic time $\mathrm{O}(\log^2 n)$, for any field F. The method we propose, based on a weak-representation of polynomials, improves the algorithm [16] by moving costly gcd operations outside the recursive step. Our algorithm proves that the problem is in $\mathcal{NC}_F^2$. This result is applied to algebraic number computation in a parallel D5 arithmetic manner. A parallel model of computation is presented and we give bounds on the complexity of simulating it with PRAM arithmetic over a field F. We conclude this paper by reviewing result on computing Jordan normal form and we demonstrate that this problem is in $\mathcal{NC}_F^2$.

2 Fast Gcd-Free Basis Computations

Let F be an arbitrary commutative field. The computational model used in this section is the arithmetic PRAM model. We say that a problem lies in $\mathcal{NC}_F^k$ [4, 11] if there exists a parallel algorithm which solves it in time is bounded by $\mathrm{O}(\log^k n)$ using $n^{\mathrm{O}(1)}$ processors for all inputs of size n.

2.1 Introduction

This section aims at proving that computation of a *gcd-free basis* for a set of polynomials $\mathcal{P} = \{P_1, \cdots, P_n\}$ in $F[X]$ with $\deg(\prod_i P_i) = d$ is in $\mathcal{NC}_F^2$ when d is considered to be the size of the input.

We recall basic definitions from [16] which define a *gcd-free basis* of a set of polynomials.

Definition 1 . *Let $\mathcal{P} = \{P_1, \cdots, P_n\}$, each $P_i \in F[X]$. A set $\mathcal{Q} = \{Q_1, \cdots, Q_m\}$ is called a* gcd-free basis *for $\mathcal{P}$ if*

i) For all $1 \leq i < j \leq m$: $\gcd(Q_i, Q_j) = 1$

ii) For all $1 \leq i \leq n$, $1 \leq j \leq m$: there exists $e_{i,j} \in \mathbb{N}$: $P_i = \prod_{j=1}^{m} Q_j^{e_{i,j}}$.

In section 2.5 we present our algorithm to compute a gcd-free basis in two main steps :

Coarse factorization : from the set $\mathcal{P}$ we compute a set $\mathcal{R}$ of pairwise relatively prime factors which we call a *pseudo gcd-free basis* for $\mathcal{P}$. Such a basis can be quickly computed using a *weak-representation* of polynomials.

Highest power common divisor decomposition : this step computes a gcd-free basis $\mathcal{Q}$ for $\mathcal{P}$ from $\mathcal{R}$ by determining factors from which we compute a decomposition satisfying criterion *ii)* of definition 1.

This steps are in $\mathcal{NC}_F^2$ compared to the degree d of the product of the P_i.

Let us define some notations used in the following complexity analysis. We use $\bar{F}$ to denote the algebraic closure of the field F. A parallel algorithm is of complexity $\mathrm{O}(t(n), p(n))$ if it requires parallel time $\mathrm{O}(t(n))$ using $\mathrm{O}(p(n))$ processors for all inputs of size n.

Using [6] $\mathrm{O}(\log d, d \log\log d)$ is the complexity to multiply two polynomials. The complexity of division is [2] $\mathrm{O}(\log d, d \log d)$. Using algorithm [3] the complexity of computing gcd of two polynomials is $\mathrm{O}\big(\log^2 d, d^3 \log\log d / \log d\big)$. We denote by $M(d)$, $D(d)$ and $G(d)$ the numbers of processors in this last three complexities. Over an arbitrary field, the complexity of computing the gcd of k polynomials [11, 3] is $\mathrm{O}\big(\log^2 d, kd^3 \log\log d / \log d\big)$. We assume that this last complexity can be written as $\mathrm{O}\big(\log^2 d, kG(d)\big)$.

Let us now to present the three main tools used to prove the complexity and the correctness of this algorithm.

2.2 Multiple multiplicity free decomposition

The results in this section are a short presentation of the results in [21, 22].

Given two polynomials P and Q in $F[x]$, we can write $P = p \times \gcd(P,Q) \times \mathrm{pp}(P,Q)$, where $\mathrm{pp}(P,Q)$ is the greatest divisor of P relatively prime to Q and p is a polynomial which is relatively prime to $\mathrm{pp}(P,Q)$ such that all irreducible factors (in a splitting field) of p are divisors of $\gcd(P,Q)$.

Proposition 1 *[21] Given two polynomials P and Q of degree at most d in $F[x]$, the parallel complexity of computing $\mathrm{pp}(P,Q)$ is $O\big(\log^2 d, G(d)\big)$.*

Proof. The proof is based on the fact that $\mathrm{pp}(P,Q) = P/\gcd(P,Q^d)$. □

Proposition 2 *[21] Given a polynomial P of degree d in $F[x]$, the multiple multiplicity free decomposition (up to a constant) of P consists of d polynomials $P_1, P_2, \cdots, P_d$ such that for all i the roots of multiplicity i in P are roots of multiplicity i in P_i (in a splitting field), and $P = c\prod_{i=1}^{d} P_i$, $c \in F$. The complexity of the computation of all the P_i is $O\big(\log^2 d, d^2 G(d)\big)$.*

Proof. Let us recall the proof in [21]. For any root λ of P, the multiplicity of λ in P is the order of x in $P(x+\lambda)$. If we rewrite P as

$$P(x+\lambda) = \sum_{i=0}^{d} a_i(\lambda) x^i$$

where the a_i are polynomials in $F[x]$, then for any fixed i, the roots of P which are roots of $a_0, \cdots, a_{i-1}$ but not of a_i are the roots of multiplicity i in P. Defining

$$q_i = \gcd(P, a_0, \cdots, a_i),\ 0 \le i \le d,$$

the roots of q_i are roots of P with multiplicity in P strictly greater than i. Thus, $P_i = P/\mathrm{pp}\,(P, \mathrm{pp}(q_{i-1}, q_i))$ for all $1 \le i \le d$ is the multiple multiplicity free decomposition of P.

Using the algorithm for the gcd of many polynomials in [11, 3] for q_i and applying proposition 1 for prime part computations, the computation of multiple multiplicity free has complexity $\mathrm{O}\big(\log^2 d, d^2 G(d)\big)$, which concludes the proof. □

2.3 Highest power common divisor

In this section we consider the computation of the highest power of a common divisor for a set of polynomials. This will be used during the second step of the final algorithm.

In the following of part 2, let $\mathcal{P} = \{P_1, \cdots, P_n\}$ be a set of n polynomials in $F[x]$ with $\deg P_i = d_i$. We assume that, for each $1 \le i \le n$, all roots of P_i have the same multiplicity m_i. Let C be a common divisor of $P_1, \cdots, P_n$ of degree c such that all roots of C have the same multiplicity m. In a splitting field, we can write $C = \bar{C}^m$, where all roots of $\bar{C}$ are of the same multiplicity equal to one. Our goal is to compute the polynomial $\tilde{C} = \bar{C}^g \in F[x]$ with the greatest $g \in \mathbb{N}$ such that each P_i can be written as $C_i \times (\bar{C}^g)^{t_i}$, $t_i \ge 1$ where $\gcd(C_i, \bar{C}^g) = 1$. We call $\bar{C}^g$ the *highest power common divisor* of C for the set $\mathcal{P}$.

Let $P = \prod_{i=1}^{n} P_i$ and $d = \deg P$. Then we have the following proposition.

Proposition 3 *Given C, the computation of $\bar{C}^g \in F[x]$, $g \in \mathbb{N}$ and all $t_i \in \mathbb{N}$ can be done using only gcd operations in $F[x]$ with complexity $O\big(\log^2(nd), nd^2 G(d)\big)$.*

Proof. Since for each $1 \le i \le n$, all roots of P_i have the same multiplicity m_i, there exists $u_i \in \mathbb{N} - \{0\}$ such that $\gcd(P_i, C^{d_i}) = \bar{C}^{u_i}$. Moreover, for all $1 \le i \le n$, $\bar{C}^{u_i}$ is in $F[x]$ and is computed with parallel complexity $\mathrm{O}\big(\log^2 d, G(d)\big)$.

Using proposition 2, for each $1 \le i \le n$, we can compute u_i by a multiple multiplicity free decomposition of $\bar{C}^{u_i}$. This step has complexity $\mathrm{O}\big(\log d(\log d + \log n), nd^2 G(d)\big)$.

By hypothesis, each integer $0 \le u_i \le d$ has $\mathrm{O}(\log d)$ bits. So the computation of the n Bezout's coefficients $\delta_i v_i$, $\delta_i \in \{-1, 1\}$, such that $\sum_{i=1}^{n} \delta_i u_i v_i = \gcd(u_1, \cdots, u_k) = g$, where $0 \le v_i \le d$, can be done using the extended gcd algorithm in [18] (see also [19, 15]). The running time is $\mathrm{O}(\log d \log n)$ using $\mathrm{O}(n \log\log d\, M(\log d))$ processors, Note that g is the maximal power in the sense previously defined.

The computation of each $(\bar{C}^{u_i})^{v_i}$, $1 \le i \le n$, can be done in $\mathrm{O}\big(\log^2 d, n\, M(d^2)\big)$ by repeating squaring algorithm. By a binary tree of multiplication if $\delta_i = 1$ or division if $\delta_i = -1$, the computation of

$$\bar{C}^g = \prod_{i=1}^{n} (\bar{C}^{u_i})^{\delta_i v_i} = \bar{C}^{\sum_{i=1}^{n} \delta_i u_i v_i}$$

has complexity $\mathrm{O}\big(\log d \log n, nM(d^2)\big)$.

Thus, using the fact that $\log d + \log n = log(nd)$, the algorithm has the complexity $\mathrm{O}\big(\log^2(nd), nd^2 G(d)\big)$. □

2.4 *Pseudo* gcd-free basis computation

Before presenting our final algorithm, this section introduces an algorithm to compute a pseudo gcd-free basis from which a gcd-free basis is easily recovered using previous sections.

Definition 2 *Let $\mathcal{P} = \{P_1, \cdots, P_n\}$ a set of polynomials in $F[x]$, and $d_i = \deg P_i$, $1 \le i \le k$. Let $P = \prod_{i=1}^{n} P_i$ and $d = \deg P$. A* pseudo gcd-free basis *$\mathcal{Q}$ for $\mathcal{P}$ is a set $\{Q_1, \cdots, Q_m\}$ of polynomials in $F[x]$ which satisfies the criteria:*

i) For all $1 \le i < j \le m$, $\gcd(Q_i, Q_j) = 1$.

ii) In a splitting field, for each root r of P there exists $j \in \mathbb{N}$ such that r is a root of Q_j.

iii) For all $1 \le i \le n$, $1 \le j \le m$, either $Q_j | P_i$ or $gcd(P_i, Q_j) = 1$.

Let us remark that criteria *ii)* and *iii)* of the previous definition correspond to a weak form of the criterion *ii)* of definition 1, in which no assumptions are made about the decomposition o P_i into products of polynomials Q_j.

Studying the algorithm in [16] to compute a gcd-free basis, at most two points can be improved. To avoid costly gcd operations in the recursive merging step, we will work with a weak-representation of polynomials. This weak representation allows us to move all gcd operations to the beginning and to the end of the algorithm [16]. Moreover, discarding unit elements at each step is done by divisibility tests instead of primality tests: elimination of unit element is important to ensure that the number of polynomials handled during the computation remains polynomial.

Definition 3 *Let P and B be two polynomials in $F[X]$ such that any root of B is root of P. A weak representation of B relative to P is a pair $[\mathcal{B};\tilde{B}]$, where $\mathcal{B}=\{B_1,\cdots,B_k\}$, $B_i\in F[x]$, $1\le i\le k$, $\tilde{B}\in F[x]$ and satisfying:*

- *i)* $\gcd(\mathcal{B})=\gcd(B_1,\cdots,B_k)=B$,
- *ii)* $\gcd(B,\tilde{B})=1$,
- *iii)* $\forall a\in\bar{F}$, $(P(a)=0)\Rightarrow\big((B\times\tilde{B})(a)=0\big)$

In such a representation the set of roots of P is split into two subsets: the one corresponding to the roots of B and the other one to the roots of polynomials in $\tilde{B}$.

Note that the such a representation is not unique. Given a set of polynomials $\{P_1,\cdots,P_n\}$ and $P=\prod_i P_i$, for all $1\le i\le n$ a weak representation of P_i relative to P could be $[\{P_i\};\mathrm{pp}(P,P_i)^k]$ for any $k\ge1$. Likewise a weak representation of $\mathrm{pp}(P,P_i)$ relative to P could be

$$[\{\mathrm{pp}(P,P_i)\};P_i] \text{ or } [\{\mathrm{pp}(P,P_i)\};P/\mathrm{pp}(P,P_i)].$$

Under some assumptions about the multiplicity of roots of P into two polynomials A and B, the following proposition leads to the replacement of primality tests by divisibility tests. For a polynomial P in $F[x]$, $V_a(P)$ is the multiplicity of $a\in\bar{F}$ in P.

Proposition 4 *Using the notation of definition 3, let A and B be two polynomials in $F[x]$ with weak representations $[\mathcal{A};\tilde{A}]$ and $[\mathcal{B};\tilde{B}]$ relative to a polynomial P in $F[x]$. If for all roots $a\in\bar{F}$ of $\tilde{A}$ and for all roots $b\in\bar{F}$ of $\tilde{B}$, $V_a(\tilde{A})\ge V_a(P)$ and $V_b(\tilde{B})\ge V_b(P)$, then*

$$P|\tilde{A}\tilde{B}\Leftrightarrow\gcd(A,B)=\gcd(\mathcal{A},\mathcal{B})=1.$$

Proof. Let $a\in\bar{F}$ be any root of P. Due to the definition of weak representations of A and B, $\gcd(A,\tilde{A})=\gcd(B,\tilde{B})=1$. Thus, we have

$$X-a|\tilde{A}\tilde{B}\Leftrightarrow\text{not}\,(X-a|\gcd(A,B)).\qquad(1)$$

Since

$$P|\tilde{A}\tilde{B}\Leftrightarrow\big(\forall b\in\bar{F},X-b|P\Rightarrow X-b|\tilde{A}\tilde{B}\big)$$

using (1), we obtain:

$$\begin{aligned}P|\tilde{A}\tilde{B}&\Rightarrow\big(\forall b\in\bar{F},X-b|P\Rightarrow\text{not}\,(X-b|\gcd(A,B))\big)\\&\Rightarrow\gcd(A,B)=1\end{aligned}$$

because any root of A or B is a root of P by definition of weak representation.

Conversely, it is sufficient to remark that if $\gcd(A,B)=1$, then any irreducible factor of P divides either A and thus $\tilde{B}$ (since B is relatively prime to A) or B and thus $\tilde{A}$.

Besides, for any root $a\in\bar{F}$ of P, we have $V_a(P)\le\max\{V_a(\tilde{A}),V_a(\tilde{B})\}\le V_a(\tilde{A}\tilde{B})$. So $P|\tilde{A}\tilde{B}$. □

The next proposition proves that two pseudo gcd-free bases of P can be merged by an arithmetic circuit of depth $\mathrm{O}(\log^2 d)$ in which primality tests (gcd operations) are replaced by divisibility tests.

Proposition 5 *Let $\mathcal{A}=\{A_i,i=1..r\}$ and $\mathcal{B}=\{B_i,i=1..s\}$ be two gcd-free bases of $\mathcal{P}$. For all $1\le i\le r$, let $[\mathcal{A}_i;\tilde{A}_i]$ be a weak-representation of A_i relative to P, and for $1\le j\le s$ $[\mathcal{B}_j;\tilde{B}_j]$ be a weak-representation of B_j relative to P.*

Under the hypothesis of proposition 4 for each weak representation polynomial, the set $\mathcal{C}$ of weak representation polynomials defined by

$$\left\{[\mathcal{A}_i\cup\mathcal{B}_i;\tilde{A}_i\tilde{B}_j],\forall i,j,1\le i\le r,q\le j\le s:\ not(P|\tilde{A}_i\tilde{B}_j)\right\}$$

is a pseudo gcd-free basis of P. Moreover, each weak representation polynomial in $\mathcal{C}$ satisfies the hypothesis of proposition 4.

Proof. Let C_{l1} and C_{l2} be two polynomials of $\mathcal{C}$ with weak-representations $[\mathcal{A}_{i1}\cup\mathcal{B}_{i1};\tilde{A}_{i1}\tilde{B}_{j1}]$ and $[\mathcal{A}_{i2}\cup\mathcal{B}_{i2};\tilde{A}_{i2}\tilde{B}_{j2}]$. Then $\gcd(C_{l1},C_{l2})=\gcd(\mathcal{A}_{i1},\mathcal{B}_{i1},\mathcal{A}_{i2},\mathcal{B}_{i2})=1$ because $A_{i1}=\gcd(\mathcal{A}_{i1})$ and $A_{i2}=\gcd(\mathcal{A}_{i2})$ are relatively prime by definition of the pseudo gcd-free basis $\mathcal{A}$.

Let a any root of P; then there exist two unique integers i and j such that a is a root of both $A_i=\gcd(\mathcal{A}_i)$ and $B_j=\gcd(\mathcal{B}_j)$. So $\gcd(A_i,B_j)$ is not equal to one and following proposition 4, $[\mathcal{A}_i\cup\mathcal{B}_j;\tilde{A}_i\tilde{B}_j]$ is in the set $\mathcal{C}$. Moreover, due to the uniqueness of i and j, there exists a unique integer l such that a is a root of C_l.

It is evident that elements of the set $\mathcal{C}$ are weak representations of pairwise relatively prime polynomials. Moreover, each element in the resulting basis satisfies the multiplicity hypothesis of proposition 4.

Thus, $\mathcal{C}$ is a pseudo gcd-free basis for the set $\{\mathcal{A}\cup\mathcal{B}\}$. □

Proposition 5 allows us to prove the next lemma:

Lemma 1 *The computation of a pseudo-gcd free basis for a set $\mathcal{P}$ is in $\mathcal{NC}^2_F$ and its complexity is $O\big(\log^2(nd),nd^2G(d)\big)$.*

Proof. The algorithm consists in recursively merging two bases using weak-representation polynomials.

At the beginning we consider the set of the following pseudo gcd-free bases:

$$\{[\{P_i\};pp(P,P_i)],[\{pp(P,P_i)\};P/pp(P,P_i)],\ 1\le i\le n\}.$$

At the end of the algorithm, we must recover polynomials from their weak-representation. This is done by computing the gcd of many polynomials using algorithm [11, 3].

The complexity of the algorithm is dominated by the complexity of this last step.

The number of elements in the resulting basis is at most d the degree of P. The depth of the binary tree of merging is $\mathrm{O}(\log n)$, at each step the degree of polynomial $\tilde{A}_i\tilde{B}_j$ squares. Thus the final degree of this polynomial is at most $\mathrm{O}(nd)$ the complexity in the recursive merging at each step is bounded by $\mathrm{O}(\log(nd),dD(nd))$.

The number of polynomials handled in weak-representation growths as the degree of polynomials $\tilde{A}_i\tilde{B}_j$. Thus, the complexity of computing at most d gcd of nd polynomials of degree d is bounded by $\mathrm{O}\big(\log^2(nd),nd^2G(d)\big)$, which concludes the proof. □

2.5 Fast gcd-free basis algorithms

We can now prove the first main theorem of this section :

Theorem 1 . *Let $\mathcal{P} = \{P_1, \cdots, P_n\}$ be a set of polynomials in $F[x]$, $P = \prod P_i$ and $d = \deg P$, such that for all i, all roots of P_i have the same multiplicity. The complexity of computing a gcd-free basis $\mathcal{Q} = \{Q_1, \cdots, Q_m\}$ for $\mathcal{P}$ is $O\big(\log^2(nd), nd^2G(d)\big)$.*

Proof. First we compute $\mathcal{R} = \{R_1, ..., R_k\}$ a pseudo gcd-free basis for $\mathcal{P}$ using proposition 5. Let $I_i = \{j \text{ such that } R_i | P_j\}$ and denote by $\mathcal{P}_{I_i}$ the subset of $\mathcal{P}$ such that R_i divides P_j, for all $j \in I_i$. For any fixed i, $1 \leq i \leq k$, using proposition 2, we can compute $\{r_i^{(1)}, \cdots, r_i^{(k_i)}\}$ the multiple multiplicity free decomposition of R_i. By definition, for any $1 \leq j \leq k_i$, all roots of $r_i^{(j)}$ have same multiplicity j and $r_i^{(j)}$ is a common divisor for the set $\mathcal{P}_{I_i}$. So using proposition 3, we can compute for all j, $1 \leq j \leq k_i$ the highest power common divisor $\tilde{r}_i^{(j)}$ of $r_i^{(j)}$ for the set $\mathcal{P}_{I_i}$, the multiplicity $g_i^{(j)}$ of roots of $\tilde{r}_i^{(j)}$ and the power $t_i^{(j)} \in \mathbb{N}$. Finally, for all $P_j \in \mathcal{P}_{I_i}$, we can write $P_j = c_{i,j} \times (\tilde{r}_i^{(j)})^{t_i^{(j)}}$ with $c_{i,j}$ and $\tilde{r}_i^{(j)}$ relatively prime.

So, we have built a gcd-free basis for $\mathcal{P}$. In fact, let

$$\mathcal{Q} = \{\tilde{r}_i^{(j)},\ 1 \leq i \leq k,\ 1 \leq j \leq k_i\}$$

and $m = \text{card}\ \mathcal{Q}$. By construction, for $1 \leq i \leq n$, $1 \leq j \leq m$, we have $P_i = C_{i,j} \times Q_j^{e_{i,j}}$ where $C_{i,j}$ and Q_j are relatively prime polynomials. Setting $e_{i,j} = 0$ if Q_j does not divide P_i, we can write

$$P_i = C_i \prod_{j=1}^{m} Q_j^{e_{i,j}}$$

where $C_i \in F$.

Due to the operations involved in the computation of the pseudo gcd-free basis, the complexity of this algorithm is $\mathrm{O}\big(\log^2(nd), nd^2G(d)\big)$. □

The next theorem treats the general case:

Theorem 2 . *Let $\mathcal{P} = \{P_1, \cdots, P_n\}$ be a set of polynomials in $F[x]$, $P = \prod P_i$ and $d = \deg P$. The complexity of computing a gcd-free basis $\mathcal{Q} = \{Q_1, \cdots, Q_m\}$ for $\mathcal{P}$ is $O\big(\log^2(nd), nd^2G(d)\big)$.*

Proof. First of all, for all $1 \leq i \leq n$, we compute $\{p_i^{(1)}, ..., p_i^{(k_i)}\}$, the multiple multiplicity free decomposition of P_i. Then, applying theorem 1, we compute a gcd-free basis for the set $\{p_i^{(j)}, 1 \leq i \leq n, 1 \leq j \leq k_i\}$. The complexity for this two steps is $\mathrm{O}\big(\log^2(nd), nd^2G(d)\big)$. □

3 Parallel Algebraic Number Computations

In this section we define a model of computation, called the *A-PRAM*, for parallel algebraic number computations. Our approach is to represent algebraic numbers over arbitrary commutative fields F like the $D5$ method [7, 8, 9, 5]. The main aim of this section is to give bounds on the simulation of the *A-PRAM* model over an arithmetic PRAM over F in the following sense: given a parallel program $\mathcal{A}$ on arithmetic PRAM over F with parallel complexity $\mathrm{O}(t,p)$, our goal is to give bounds on the complexity of the evaluation of $\mathcal{A}$ for any entries of size n on an arithmetic PRAM over $F(\lambda)$, for all roots λ of a polynomial P in $F[X]$ of degree d.

3.1 Introduction to D5

Using $D5$, computing in $\bar{F}$ reduces to the construction of a tower of subfields

$$F \subset F_1 \subset \cdots \subset F_n \subset \bar{F}$$

such that each F_i is a simple algebraic extension of F_{i-1} by a root λ_i of a univariate polynomial P_i with coefficients in F_{i-1} [8]. F_i is denoted $F_{i-1}(\lambda_i)$. The common way that algebraic extensions appear in computer algebra is to represent elements of F_i as polynomials in $F_{i-1}[\lambda]/P(\lambda)$. The problem is that generally P_i is not irreducible and characterizes not only one root λ_i but a set of roots. Therefore, the result of a computation may not be the same for all the roots of P_i, typically for the boolean outputs of $\overset{?}{=} 0$ gates (*i.e.* test to zero) of an arithmetic circuit over $\bar{F}$. As an example, let λ denote a root of $P = x^5 - 2x^3 - x^2 + 2$ in $\bar{\mathbb{Q}}$, the algebraic closure of the field of the rationals. The output of the gate $\lambda^2 - 2 \overset{?}{=} 0$ may be true or false, depending on the chosen root λ of P. This leads to an automatic discussion, called *splitting* in D5, which reduces to a partial factorization of P. Notice that this partial factorization does not require polynomial factorization, but only polynomial gcd computations, and thus is in $\mathcal{NC}^2$.

In the sequential implementation of D5, the test $\lambda^2 - 2 \overset{?}{=} 0$ of the previous example returns: "*true* if λ is a root of $x^2 - 2$" else "*false* if λ is a root of $x^3 - 1$".

The evaluation of a sequential program using algebraic number arithmetic of D5 leads to a *splitting tree* [5, 9] such that each node corresponds to a split, the root node corresponds to the beginning of the evaluation and each edge is attributed by definition of handling roots.

Splitting that occurs permits a parallel evaluation of the program by separate threads of control since the computations involved in each subtree are independent. A basis for the parallel implementation can be founded in [5], where implementation of *continuations* are studied. As a result, several independent threads of control are created and may be mapped onto different processors as on a shared memory parallel computer. Let us remark that parallelism of the evaluation only occurs due to the splitting into independent subcases of algebraic numbers.

The evaluation of parallel program using D5 arithmetic has been studied in [12], both from theoritical and pratical points of view. In addition to the need of continuation, the parallel case requires synchronisation mecanisms. In the following *A-PRAM* model of computation, this feature is incorporated in the model: it can be implemented in a lazy fashion using inherent synchronisation of parallel programs [12] or by using a weak coherency management protocol for global shared memory.

3.2 Parallel D5

In fact, in the framework of parallel complexity, parallel evaluation must consider the problem of synchronizing simultaneous splitting due to the explicit parallelism of a program. Consider the previous example and assume that two independent tests $\lambda^2 - 2 \overset{?}{=} 0$ and $\lambda - 1 \overset{?}{=} 0$ are evaluated in parallel. In such a case, we must ensure global definition of splitting that occurs. We will show that it relies on the computation of a gcd-free basis.

The next section describes a model of computation that treats simultaneous splitting and gives bounds on parallel complexity evaluation of program using a parallel D5 arithmetic. Because of the possible exponential growth of representation when considering a tower of k extensions due to arithmetic over polynomials with k variables, we focus on the problem of simultaneous splitting during evaluation of a parallel program over a simple algebraic extension.

3.3 Parallel model of computation

The A-PRAM model is based on a set of d synchronous arithmetic PRAMs over $F[X]$ and is designed to handle computation over $F(\lambda)$, for all roots of a polynomial P in $F[X]$ of degree d. Four main features are defined:

- an *execution context* which contains the definition of the root λ,
- a function, called *split*, which has as a side effect the modification of the execution context,
- a *synchronous mechanism* that manages simultaneous splits,
- a *dump mechanism* that copies the state of a machine to another machine.

Initially, one PRAM runs the parallel program. Other PRAMs are reserved for the continuation of the evaluation after splittings.

Like the D5 representation, the representation of an element of $F(\lambda)$ is an element of $F[X]/P$. We assume that arithmetic operations $(+, -, \times)$ on $F[X]/P$ are unit cost, as are the inversions $(1/)$ and "equal to zero" tests $(\overset{?}{=} 0)$.

More precisely, the function *split* can be used to implement the $\overset{?}{=} 0$ operator. Let $Q(\lambda)$ $(Q \in F[X])$ be the result of an arithmetic expression which is to be compared to zero. By the way that D5 operates, the result of $Q(\lambda) \overset{?}{=} 0$ is *true* if λ is root of $P_1 = \gcd(Q, P)$ and *false* if λ is root of $P_2 = \mathrm{pp}(P, Q)$, the prime part of P relative to Q. Given such a test $Q(\lambda) \overset{?}{=} 0$, the A-PRAM calls *split*(P_1, P_2) which returns either 1 or 2 depending which new definition (P_1 or P_2) has been chosen. The side effect is the modification of the execution context by P_1 or P_2.

If different operations *split* are performed simultaneously by different processors, the synchronization mechanism ensures consistent modification of the execution context and starts new evaluations of other machines (see figure 1).

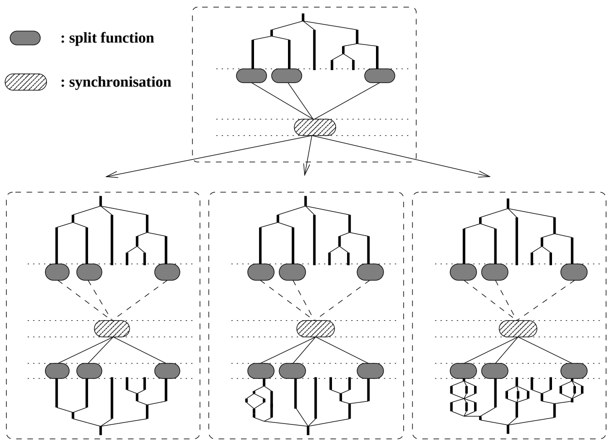

Figure 1: Illustration of continuation processes of multiple simultaneous splits.

3.4 Managing simultaneous splittings using gcd-free basis

In case of simultaneous splitting, let $(P_1^{(i)}, P_2^{(i)})_{i=1..k}$ be the set of the $2k$ polynomials that correspond to the k splittings of P. To recover a consistent definition for the execution context from these polynomials, we compute a gcd-free basis.

Let $(Q_j)_{j=1..m}$ be a gcd-free basis of the set $(P_1^{(i)}, P_2^{(i)})_{i=1}^{k}$. Since, for each i, $P_1^{(i)}$ and $P_2^{(i)}$ are relatively prime and they define the same set of roots as P, then for all $1 < j < m$, either $Q_j | P_1^{(i)}$ or $Q_j | P_2^{(i)}$. Always one of these cases occurs.

Each polynomial that occurs in a splitting allows the definition of a common execution context such that all local split function calls can return a correct value for this context.

Continuing the example given in the introduction of this section, the splitting polynomials of the test $t_1 = \lambda^2 - 2 \overset{?}{=} 0$ are $P_1^{(1)}(x) = x^2 - 2$ (and the test is true) and $P_2^{(1)}(x) = x^3 - 1$ (and the test is false). Likewise, for the test $t_2 = \lambda - 1 \overset{?}{=} 0$, the polynomials are $P_1^{(2)}(x) = x - 1$ and $P_2^{(2)}(x) = x^4 - x^2 - 2x - 2$. A gcd-free basis of these polynomials is $Q_1(x) = x^2 - 2$, $Q_2(x) = x - 1$, $Q_3(x) = x^2 + x + 1$ such that $P_1^{(1)} = Q_1$, $P_2^{(1)} = Q_2 Q_3$, $P_1^{(2)} = Q_2$ and $P_2^{(2)} = Q_1 Q_3$.

For each polynomial Q_i, values of the tests t_1 and t_2 are: $\begin{cases} t_1 = true \\ t_2 = false \end{cases}$ if $Q_1(\lambda) = 0$, $\begin{cases} t_1 = false \\ t_2 = true \end{cases}$ if $Q_2(\lambda) = 0$, and $\begin{cases} t_1 = false \\ t_2 = false \end{cases}$ if $Q_3(\lambda) = 0$.

3.5 Simulation complexity

During simultaneous splittings, several cases are generated. Each subcase runs with an independent *execution context* on a new parallel machine.

The costs to simulate the A-PRAM on a PRAM over F are decomposed into the cost of gcd-free basis computations, costs to recopy the computation state, and the cost to support polynomial arithmetic.

The problem that we consider is the following. Let $\mathcal{A}$ be a parallel program on PRAM over a field H and let $O(t, p)$ be its complexity for any input of size n. Let t_a be the greatest number of additions or subtractions performed by one processor and let p_a be the greatest number of processors that simultaneously execute such an operation. Define t_m and p_m (respectively t_t and p_t) for multiplication operation (respectively for tests or divisions). Then t is bounded by $t_a + t_m + t_t$ and p by $p_a + p_m + p_t$.

Theorem 3 *The time of the execution of the program $\mathcal{A}$ on the A-PRAM model, with λ a root of P of degree d in input, is bounded by*

$$O\big(t_a + t_m \log d + t_t \log^2(dp_t)\big)$$

using a number of processor bounded by

$$O\big(p_a d + p_m M(d) + p_t n d^2 G(d) + t d^2 p\big).$$

Proof. The main point concerns the case of test for zero operations. Other costs comes from complexity operations for realizing arithmetic operations on $F[x]/P$ on an arithmetic PRAM on the field F.

Using the algorithm given by theorem 1, the cost to compute a gcd-free basis of $2p_t$ polynomials from p_t simultaneous splittings is $O\big(\log^2(dp_t), p_t d^2 G(d)\big)$.

The number of registers to be recopied for the continuation mechanism is bounded by $O(tdp)$: the work of the parallel program times the number of coefficients in each polynomial, *i.e.* $d+1$ memory registers for each polynomial. The complexity to recopy its registers on at most d machines is $\mathrm{O}\big(\log(tdp), td^2p\big)$. □

3.6 Application to Jordan normal form computation

As direct consequence of the previous theorem, we can deduce the next corollary.

Corollary 1 *The complexity to compute the symbolic Jordan normal of a matrix of order n over a field F is in* $\mathcal{NC}^2_F$.

Proof. For the complete proof, please see [21]. The main idea of the algorithm [21, 22] is to split the caracteristic polynomial of the input matrix using its geometric structure (generalized null-space computation). The last step refines the splitted polynomials using a gcd-free basis computation and gives the parallel time complexity of the algorithm.

Thus, using our gcd-free basis algorithm we improve the complexity of this algorithm to $\mathcal{NC}^2_F$ instead of $\mathcal{NC}^3_F$. □

4 Conclusions

Dealing with algebraic numbers is a central problem in computer algebra. We have given complexity bounds concerning parallel D5 arithmetic in the case of conjugated roots of a same polynomial. These bounds are based on gcd-free basis computations and we improve known results. Futur work will be to improve parallel work complexity of the algorithm with the same time complexity.

Acknowledgments

The authors thank Gilles Villard for useful discussions.

The content of this paper has been studied during the PhD thesis of the first author, which was bound to LMC-IMAG, Grenoble.

References

[1] Ben-Or, M., Kozen, D., and Reif, J. The complexity of elementary algebra and geometry. *JCSS 32*, 2 (1986), 251–264.

[2] Bini, D., and Pan, V. Improved Parallel Polynomial Division. *SIAM J. Comput. 22*, 3 (1993), 617–626.

[3] Bini, D., and Pan, V. *Polynomial and Matrix Computations. Fundamental Algorithms*, vol. 1. Birkhäuser, 1994.

[4] Borodin, A. On relating time and space to size and depth. *SIAM J. Comput. 6*, 4 (1977), 733–743.

[5] Broadbery, P., Gómez-Díaz, T., and Watt, S. On the Implementation of Dynamic Evaluation. In *Proc. of ISSAC'95* (Montréal, Canada, 1995), ACM Press.

[6] Cantor, D., and Kaltofen, E. On fast multiplication of polynomials over arbitrary rings. *Acta Inf. 28*, 7 (1991), 697–701.

[7] Della Dora, J., Dicrescenzo, C., and Duval, D. About a new method for computing in algebraic number fields. In *Proc. EUROCAL'85* (1985), LNCS 204, Springer Verlag, pp. 289–290.

[8] Duval, D. *Diverses questions relatives au calcul formel avec des nombres algébriques.* Thèse de Doctorat d'Etat, Université Joseph Fourier, Grenoble, France, 1987.

[9] Duval, D. Algebraic Numbers: An Example of Dynamic Evaluation. *Journal of Symbolic Computations* (1994), 429–445.

[10] Gantmacher, F. *Théorie des Matrices.* Jacques Gabay, 1957.

[11] Gathen, J. v. z. Parallel Algorithms for Algebraic Problems. *SIAM J. Comput. 13*, 4 (1984), 802–824.

[12] Gautier, T. *Calcul Formel et Parallélisme : conception du système* Givaro *et application au calcul dans les extensions algébriques.* PhD thesis, Institut National Polytechnique de Grenoble, France, 1996.

[13] Giesbrecht, M. Nearly optimal algorithms for canonical matrix forms. *SIAM J. Comp. 24* (1995), 948–969.

[14] Gil, I. *Contribution à l'algèbre linéaire formelle. Formes normales de matrices et applications.* PhD thesis, Institut National Polytechnique de Grenoble, 1993.

[15] Iliopoulos, C. Worst-case complexity bounds on algorithms for computing the canonical structure of inite abelian groups and the Hermite and Smith normal forms of an integer matrix. *SIAM J. Comput. 18*, 4 (1989), 658–669.

[16] Kaltofen, E. Sparse Hensel lifting. In *Proc. EUROCAL'85, vol. 2* (1985), vol. 204 of *LNCS*, Springer-Verlag, pp. 4–17.

[17] Kaltofen, E., Krishnamoorthy, M., and Saunders, B. D. Fast Parallel Computation of Hermite and Smith Forms of Polynomial Matrices. *SIAM J. Alg. Disc. Meth. 8*, 4 (Oct. 1987), 683–690.

[18] Majewski, B., and Havas, G. The complexity of greatest common divisor computations. In *Algorithmic Number Theory* (1994), vol. 877 of *LNCS*, pp. 184–193.

[19] Majewski, B., and Havas, G. A solution of the extended gcd problem. In *Proc. of ISSAC'95, Montreal, Canada* (1995), ACM Press, pp. 248–253.

[20] Ozello, P. *Calcul exact des formes de Jordan et de Frobenius d'une matrice.* PhD thesis, Université Scientifique Technologique et Médicale de Grenoble, 1987.

[21] Roch, J., and Villard, G. Parallel computations with algebraic numbers, a case study: Jordan normal form of matrices. In *PARLE'94, Athens Greece* (1994), vol. 817 of *LNCS*.

[22] Roch, J., and Villard, G. Fast parallel computation of the jordan normal form of matrice. *Parallel Processing Letters 6*, 2 (1996), 203–212.