Massively Parallel Computing and Polynomial GCD's

by

Martin Santavy

School of Computer Science

McGill University

Montreal, Canada

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

January 1987

ABSTRACT

This thesis studies massively parallel synchronous processing models and algorithms. We survey the basic models, discuss their interrelationships and analyze properties of some feasible network models. A new definition of Gentleman's σ-function is given. We present routines that simulate the CUBE ASCEND/DESCEND class algorithms on the CCC (cube-connected cycles) and the PERFECT SHUFFLE machines of any sufficient size. We give the exact (non-asymptotic) computation times and prove the correctness of the algorithms. This extends the previous results of Stone (1971) and Preparata-Vuillemin (1979). Borodin-von zur Gathen-Hopcroft (1984) laid out a program to obtain a "theory package for parallel algebraic manipulation". We continue work in this program and focus on the GCD of two polynomials, which is one of the basic problems of algebraic manipulation algorithms. B-G-H gave a GCD algorithm that works over arbitrary fields in $O(\log^2 n)$ time and requires polynomial number of processors on a general type of parallel machine, such as P-RAM or algebraic circuits. The algorithm uses a system of $(n+m-2i)\times(n+m-2i)$ asymmetric matrices. If the result of Berkowitz (1984) is applied, the required number of processors is $O(n^{\alpha+2+\epsilon})$. We modify the algorithm and present the following results: A new matrix formula for polynomial GCD is given that uses a system of upper left principal minors of a symmetric $n\times n$ matrix. The Berkowitz (1984) parallelization of the Samuelson (1942) method is modified and combined with the previous result. This allows us to decrease the required number of processors by the factor of $O(n)$. Morover, the full strength of the general parallel models is not needed for the algorithm. We present a CUBE-feasible algorithm (composed of ASCEND/DESCEND subroutines) that computes the GCD of two polynomials over arbitrary field in $O(\log^2 n)$ time and requires $n^{\alpha+1+\epsilon}$ processors, $\alpha=3$. A Hoare-style axiomatic verification system for CUBE-feasible algorithms is developed and used to prove correctness of the presented routines.

RESUME

Cette thèse discute d'algorithmes et de modèles de traitement synchrones massifs. Nous présentons les modèles de base et nous discutons de leur réciprocité; de plus, nous analysons les propriétés de quelques modèles plausibles de réseaux. Une nouvelle définition de la fonction-σ de Gentleman est présentée. Nous discutons égallement de procédures qui simulent les algorithmes CUBE ASCEND/DESCEND auprès du CCC (cycles cube-connecté) et des machines PERFECT SHUFFLE de n'importe quelle taille. Nous donnons les temps de calculs exacts (non-asymptotiques) et nous prouvons l'exactitude des algorithmes. Ceci ajoute au résultats précédents de Stone (1971) at de Preparate-Vuillemin (1979). Borodin-von zur Gathen-Hopcroft (1984) précisa un programme afin d'obtenir un "groupement de théories pour la manipulation algébraïque parallèle". Nous ajoutons à ce programme tout en mettant un emphase sur le plus grand diviseur en commun (PGDC) de deux polynômes, qui se présente comme un des problèmes de base de la manipulation algébraïque d'algorithmes. B-G-H donne un algorithme pour déterminer le PGDC à l'intérieur de champs arbitraires; cet algorithme est d'un temps $O(\log^2 n)$ et requiert un nombre polynôme de processeurs sur une machine parallèle de genre commun, tel que P-RAM ou des circuits algébraïques. L'algorithme utilise un système de $(n+m-2i)\times(n+m-2i)$ matrices asymétriques. Si le résultat de Berkowitz (1984) est utilisé, le nombre requis de processeurs est $O(n^{\alpha+2+\epsilon})$. Nous modifions cet algorithme et nous présentons les résultats suivants: une nouvelle formule matricielle pour le PGDC d'un polynôme qui utilise un système de mineurs principals du coin supérieur de gauche de matrices symétriques $n\times n$ Le parallèlisme de Berkowitz (1984) avec la méthode de Samuelson (1942) est modifié et combiné avec le résultat précédent. Ceci nous permet de réduire le nombre de processeurs par un facteur de $O(n)$. De plus, la puissance totale des modèles générales parallèles n'est pas requise pour l'algorithme. Nous présentons un algorithme CUBE-plausible (composé de procédures ASCEND/DESCEND) qui calcul le PGDC de deux polynômes à l'intérieur d'un champs arbitraire en utilisant un temps $O(\log^2 n)$ et qui requirert $n^{\alpha+1+\epsilon}$ processeurs, $\alpha=3$. Une vérification axiomatique du style Hoare pour les algorithmes CUBE-plausible est développée et utilisée pour prouver l'exactitude des procédures présentées.

CONTENTS

ACKNOWLEDGEMENTS

I thank Prof. Nathan Friedman, my thesis supervisor, for his advice and support. I am indebted to the N.S.E.R.C. and McGill University for their financial support. I also thank the family van Wijlen and Prof. Michael Mackey for their help and encouragement.

Chapter 1
Introduction

In the last decade, there has been an enormous growth in the attention given to the field of parallel computing. Massively parallel computers based on various geometrical architectures offer an alternative to traditional supercomputers at far lower cost.

An example of how rapid the development is, can be found in the history of the binary hypercube architecture. It has been known to researchers for a quarter of century [Squire, Palais (1962,1963)]. Only recently, however, has the technology to produce real machines been available. The first working hypercube architecture was the 64-node Cosmic Cube at Caltech in 1983 [Seitz (1985)]. The first commercial production started in the middle 1985 with the Intel Personal Supercomputer [Intel (1986)], which had 128 node processors. The Amdek System/14 which followed has 256 nodes, while NCUBE/ten [NCUBE (1986)] can accommodate 1024 processors with throughput potential 500 MFLOPS. The maximum sized 12-cube of Floating Point Systems, Inc. [Gustavson, Hawkinson and Scott (1986)] has 4096 processors with 65 GFLOPS peak performance. Other machines are under development [Hillis (1985)].

Such activity should have a good theoretical support. Unfortunately, a unified general theory of parallel computing is still missing. Various models exist. Their differences, however, are much deeper and much more fundamental than the differences among sequential models. Algorithms for the general models of parallel computing, widely used in the literature for their power and convenience, may prove hard or even impossible to implement on restricted but practical architectures of existing machines. Another important and unresolved issue is the verifiability of massively parallel algorithms. This thesis addresses both issues and, using the example of the problem of computing polynomial GCD's, demonstrates some problems associated with an efficient implementation of a theoretically fast algorithm on the restricted architecture of a practical model.

The GCD of two polynomials is an important practical problem which occurs frequently in fields such as symbolic and algebraic manipulation [Knuth (1973)] or error detecting codes [MacWilliams (1977)]. We use the main ideas of the fast parallel GCD algorithm of Borodin-von zur Gathen-Hopcroft (1984), which was designed for a general parallel machine, such as P-RAM. We decrease the required number of processors in a general model and also implement the algorithm on the CUBE. We also show how "CUBE-feasible" algorithms can be simulated on other practical models: the PERFECT SHUFFLE and the CCC. Finally, we design an axiomatic verification system for CUBE-feasible programs and use it to prove the correctness of our algorithm.

Chapters 2 and 3 survey main models of parallel machines. Chapter 4 characterizes some frequently used network machines. A new definition of Gentleman's σ-function is presented. The function is evaluated for given models. Chapter 5 extends results of Stone (1971) and Preparata and Vuillemin (1979,1981) and presents simulations of an ASCEND/DESCEND CUBE algorithms on the PERFECT SHUFFLE or the CCC of any sufficient size. The computation time is given in terms of the size of the problem. In chapter 6 a new matrix formula for the GCD of two polynomials is given. The Berkowitz parallelization of Samuelson's method is modified and combined with the previous formula. The resulting algorithm computes the GCD of two polynomials over arbitrary field in $O(\log^2 n)$ time using $O(n^{\alpha+1+\epsilon})$ processors. This is an $O(n)$ improvement of the processor bound of the algorithm of B-G-H (1984). Chapter 7 develops an axiomatic verification system and presents the implementation of the GCD algorithm on the CUBE and its verification. The implementation uses a simple matrix multiplication technique with $\alpha = 3$. We believe this is the first presentation of an efficient algorithm for this problem on this model.

Chapter 2
Taxonomy of Parallel Algorithms and Architectures

The following characteristics and taxonomies were given by Kung (1980). They reflect hardware considerations of algorithms for practical parallel architectures.

2.1. Characteristics of Parallel Algorithms

From a practical point of view, a parallel algorithm can be seen as a collection of independent task modules that can be executed in parallel and that communicate with each other during the execution of the algorithm. Because more than one module can be executed at a time, *concurrency control* is needed to enforce desired interactions among modules and to ensure the correctness of the concurrent execution. Kung (1980) recognizes three main categories of concurrency control:

1. centralized control (execution is synchronous),
2. distributed control (execution synchronous or asynchronous),
3. control via shared data (execution asynchronous).

Other practical characteristics are module granularity and communication geometry. *Module granularity* refers to the maximal amount of computation a typical task module can do before having to communicate with other modules. *Communication geometry* is the geometric layout of the network representing intermodule communication.

2.2. Matching Parallel Algorithms with Parallel Architectures

In order to assess the correspondence between parallel algorithms and parallel machines, the communication geometry properties or the concurrency control and module granularity properties of the algorithms can be used. The communication geometry classification is demonstrated in fig.1.

Flynn (1972) categorized various classes of computers based on the way they operate and handle data. These categories are: SISD (Single Instruction Stream, Single Data Stream), SIMD (Single Instruction Stream, Multiple Data Stream), MISD (Multiple Instruction Stream, Single Data Stream), MIMD (Multiple Instruction Stream, Multiple Data Stream). Kung (1980) used SIMD and MIMD as two of three categories of the matching of parallel algorithms and parallel machines according to their concurrency control and module granularity.

1. SIMD machines correspond to synchronous, lock-step algorithms that require central controls.
2. MIMD machines correspond to asynchronous algorithms with large module granularities.
3. Systolic machines.

The third, technologically practical category of *systolic* machines reflects the trend to have special purpose machines-on-a-chip with a large number of identical processors arranged in regular structures motivated by VLSI design technology. Each processor periodically moves data in and out, each time performing some short computation, so that a regular flow of data is kept up in the network. The geometry of the communication paths in a systolic machine must be simple and regular. Systolic machines correspond to synchronous algorithms that use distributed control achieved by simple local control mechanisms and have (small) constant module granularities.

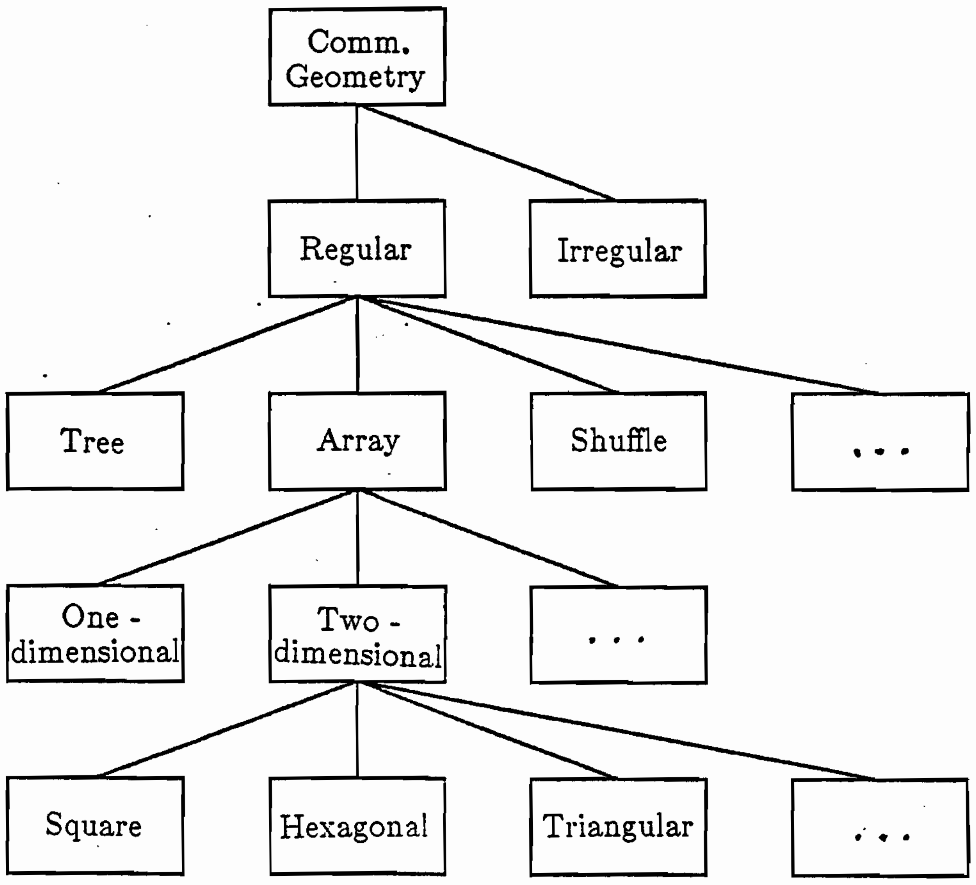


Fig.1: Classification of the communication geometry.

Chapter 3
Synchronous parallel machines

3.1. Fixed versus modifiable structure models

Cook (1981) has classified the synchronous parallel models according to whether the interconnection among processors during a computation is fixed or modifiable. This classification has its analog in the sequential computing theory. The sequential fixed structure models are represented by various types of Turing machines. The examples of sequential modifiable structure models are storage modification machines [Schönhage (1979)] and random access machines (SSM's are equivalent to RAM's that can only add and subtract one).

The parallel fixed structure models include uniform Boolean circuits [Borodin (1977), Ruzzo (1981)], aggregates [Dymond and Cook (1980)], conglomerates [Goldschlager (1978)], and alternating Turing machines [Chandra, Kozen and Stockmeyer (1981)].

The parallel modifiable structure models include P-RAM's [Fortune and Wyllie (1978)], SIMDAG's [Goldschlager (1978)], and hardware modification machines [Dymond and Cook (1980)].

Time bounds of all these models are roughly equivalent to each other, and they are equivalent to the space bound of a deterministic Turing machine. This is stated in the "parallel computation thesis" [Goldschlager (1978)]: The sets of functions computed by a parallel computer in time $S^{O(1)}$ (i.e time polynomial in

S) are the same as those computed by a deterministic Turing machine in space $S^{O(1)}$.

For the models mentioned above the thesis can be formulated in a more specific way [Cook (1981)]:

$$\textit{Fixed-Time}\,(T) \subseteq \textit{DSPACE}\,(T) \subseteq \textit{Modifiable-Time}\,(T) \subseteq \textit{Fixed-Time}\,(T^2) \quad ,$$

where *Fixed-Time* (T) can represent the class of languages accepted in time T by any one of the fixed structure models mentioned above. Similarly, *Modifiable-Time* (T) can represent the class of languages accepted in time T by any one of the modifiable structure models. *DSPACE* (T) refers to the languages accepted by a T space bounded deterministic Turing machine [Hopcroft and Ullman (1979)].

3.2. Speedups of sequential machines by synchronous parallel machines

The processing of the input and output of a sequential algorithm alone requires time that is linear in their size. There is, however, no lower time limit on parallel machines.

The most favorable extreme is a completely parallelizable problem which can be totally decomposed into a reasonable number of independent parallel operations. A proper parallel machine can process such an algorithm in constant time.

The opposite extreme is a completely unparallelizable problem. An example [Kung (1979)] is the task of raising a number x to a large power x^{2^k}. One processor can compute the output by successive squarings. No speedup, however, can be achieved by using more than one processor (of the same type).

Therefore, for a general unspecified problem, the replacement of sequential machines by parallel ones can be expected to save no significant amount of computation time. The general speedups of deterministic machines by parallel machines reflect mostly the structural differences among models. They correspond to similar speedups attained by a structural change of a sequential model. For a better evaluation of the speedups, only a subclass of "reasonably" parallelizable algorithms should be considered. Dymond and Tompa (1983) gave the general speedup of deterministic Turing machines by (fixed structure) alternating Turing machines as

$$DTIME(T) \subseteq ATM\text{-}TIME(T / \log T) \ ,$$

which corresponds to the sequential speedup of time-bounded deterministic Turing machines by space-bounded machines [Hopcroft, Paul and Valiant (1977)]

$$DTIME(T) \subseteq DSPACE(T / \log T) \ ,$$

The second general speedup reflects a quadratic advantage of modifiable structure machines (namely P-RAM's) over fixed structure machines:

$$DTIME(T) \subseteq PRAM\text{-}TIME(\sqrt{T}) \ .$$

(The SIMDAG's are at least as fast as P-RAM's.) Similar speedup can be achieved for sequential RAM's [Hopcroft, Paul and Valiant (1975)].

Classes of problems with a "good" parallel solution on some models of parallel machines and their relationships are given in [Cook (1985)]. In the following sections we will discuss the main models of synchronous parallel machines and relationships among them.

3.3. Uniform circuits

A *combinational (Boolean) circuit* [Borodin (1977), Pippinger (1979), Ruzzo (1981), Cook (1981), Cook (1985)] is a labeled acyclic directed graph (a network). Each node of the graph can be labeled as an input node, an AND-gate, an OR-gate, or a NOT-gate (or possibly another boolean function-gate). Input nodes must have fan-in zero, and NOT-gates must have fan-in one. Fan-in of AND- and OR-gates is bounded by two in some models, or unbounded in others. In addition, certain nodes are designated as output nodes. (There is no fan-out bound on any node.)

The *size* of a circuit is the number of gates. The *depth* of a circuit is the length of the longest path from some input to some output. Let the nodes of a network be assigned to *levels* in the following way. The inputs are assigned to level zero; gates and outputs to the level one greater than the maximum level of the inputs and gates upon which they depend. The *thickness* of a network at level l is the number of gates at levels not exceeding l upon which one or more gates at levels exceeding l depend. The *width* of a network is the maximum of its thicknesses at all levels.

An *interconnection function* determines the gate whose output is connected to a given input of a given gate. A *gate function* determines the boolean function performed by a given gate. For k input nodes and l output nodes, the circuit computes a function $f : \{0,1\}^k \rightarrow \{0,1\}^l$ in the obvious way.

A *uniform circuit* is an infinite family $C = (C_0, C_1, \cdots)$ of combinational circuits, one for each input size, such that the interconnection and gate functions can be computed by a deterministic Turing machine in space $O(\log c(n))$, where $c(n)$ is the size of C_n.

The uniform circuit (uniform Boolean circuit family) is considered to be a fundamental model, since it reflects the basic hardware structure of real computers without many additional restrictions. The circuit complexity of Boolean functions is an appealing mathematical subject, studied since Shannon (1949), and the uniform circuit model is reasonably attractive for an enduring mathematical theory.

A drawback of the model is that the circuit depends on the problem and its input size. Uniform circuits are incompatible with a concept of a practical universal machine with simple geometry and (preferably) identical processors, which could be easily reconfigured for many different computational problems.

Note: Some algorithms use a modification of boolean circuits, called *arithmetic circuits* and *arithmetic networks*. An arithmetic circuit corresponds to a boolean circuit with boolean gates replaced by arithmetic gates (performing arithmetic operations). An arithmetic network combines both boolean and arithmetic circuits [Berkowitz (1984), Eberly (1984)].

The equivalences between the size and the depth of uniform circuits and time and space of deterministic Turing machines can be written as

$$USIZE(n^{O(1)}) = DTIME(n^{O(1)}) \ ,$$
$$UDEPTH(n^{O(1)}) = DSPACE(n^{O(1)}) \ ,$$

where *USIZE* and *UDEPTH* are the classes of languages accepted by size bounded and depth bounded uniform circuits, respectively [Pippinger (1979), Cook (1981)].

The simultaneous bound on the size and depth of uniform circuits relates to the simultaneous bound on the time and reversal of deterministic Turing machines, where the resource *reversal* is the number of so called reversal steps in a computation, when one or more heads change direction.

$$USIZE\text{-}DEPTH(n^{O(1)}, \log^{O(1)} n) = DTIME\text{-}REVERSAL(n^{O(1)}, \log^{O(1)} n)$$

The simultaneous bound on the time and size of deterministic Turing machines relates to the simultaneous bound on the size and width of uniform circuits.

$$USIZE\text{-}WIDTH(n^{O(1)}, \log^{O(1)} n) = DTIME\text{-}SPACE(n^{O(1)}, \log^{O(1)} n) \ .$$

Note: The relations above can be reformulated with only slight changes for different definitions of circuit uniformity. Instead of comparing Turing machines to uniform families of circuits, one can also compare "nonuniform" Turing machines to (nonuniform) families of circuits [Pippinger (1979)].

3.4. Alternating Turing machines

An *alternating Turing machine* [Kozen (1976), Chandra and Stockmeyer (1976), Chandra, Stockmeyer and Kozen (1979,1981)] is a generalization of a nondeterministic multitape Turing machine. A nondeterministic machine has *existential* states, for which there are several possible next states. At least one of the alternatives must lead eventually to an accepting state. In addition to the existential states, an alternating Turing machine (ATM) has *universal* states, for which all possible next states must lead to an accepting state. The accepting state can be, for example, a universal state with no successors. (No rejecting states are then defined.) An alternative definition uses special *accepting* and *rejecting* states, which are halting states.

ATM M *accepts* input x iff there is a finite tree whose nodes are labeled with configurations of M, such that the root of the tree is the initial configuration, all leaves are accepting configurations, every universal node (i.e. node whose configuration has a universal state) has all possible next configurations as children, and every existential node has at least one possible next configuration as a child. Such a tree is called an *accepting computational tree* of M on input x.

An ATM M is $S(n)$ *space* bounded if any configuration reachable from the initial configuration of M on input x uses at most $S(|x|)$ cells on the worktape, where $|x|$ is the size of input x. An ATM M is $T(n)$ alternation bounded if the accepting tree of M on input x has any path from root to leaf of length at most $T(|x|)$.

ATM's represent a sightly restricted form of parallel computation, since they limit the "processors" to be Turing machines organized as an and-or-tree. However, there is a close correspondence between resources of an ATM and resources of a deterministic Turing machine. Ruzzo (1979,1981) has shown the equivalence of the simultaneous bound on the depth and size of uniform circuits and that on the alternating time and space of ATM's:

$$ATM\text{-}TIME\text{-}SPACE\,(\log^{O(1)} n\,,\ O(\log n)) = USIZE\text{-}DEPTH\,(n^{O(1)}, \log^{O(1)} n)\ .$$

The space bound definition of uniformity for the families of circuits does not need to be so strong in this case. The equivalence above still holds for a weaker definition of uniformity with only time bound of $O(\log c(n))$ for the deterministic Turing machine that computes interconnection and gate functions of a circuit.

Several definitions of uniformity of circuits can be used for specific purposes. One advantage of ATM's over uniform circuits is that there is no uniformity problem. Each ATM is automatically uniform.

3.5. Conglomerates

Conglomerates, introduced in [Goldschlager (1978)], are a generalization of parallel machines which "could be feasibly built using fixed connections". A *conglomerate* is an infinite set of identical finite controls connected together in some manner. Each finite control has $r \geq 1$ inputs and one output. A connection function f specifies the finite control whose output is connected to a given input of a given finite control. Cycles are allowed in the connection graph.

Conglomerate time corresponds to the space bound of deterministic Turing machines. In order to relate both resources, conglomerates must satisfy a *uniformity* condition given by a linear space bound computability of the connection function f by a deterministic Turing machine. Goldschlager (1978), however, did not discuss the size of conglomerates and their possible relationship to resources of other models.

3.6. Aggregates

Dymond (1980) developed a generalization of circuits called an *aggregate*. Unlike a circuit, the directed graph of an aggregate is not necessarily acyclic. This offers a better relation of model resources to the hardware size. A *computation* of the aggregate is a sequence of configurations. A configuration is an assignment of 0 or 1 to each node. In the initial configuration, values of all nodes except the input nodes are 0 . Subsequent configurations assign a value of the gate function to each node. Arguments of the gate function are values assigned by the previous configuration to those nodes that are given by the interconnection function. The input nodes of an aggregate are not fed directly by the input values, but rather provided with a $\lceil \log n \rceil$ register and a $\lceil \log n \rceil$ initial time delay. This construction allows the input to be read by fewer input nodes than its size is, i.e. sublinear hardware bounds can be considered for aggregates. There are two output nodes. The output of the aggregate is the the value of the first output node in the first configuration that assigns 1 to the second node. That configuration also ends the computation.

A *hardware size* of an aggregate is the number of its nodes. A *running time* is the maximum length of a successful computation over all inputs of a given length. The *uniformity* condition of a family $\{\beta_n\}$ of aggregates is given by computability of the interconnection and gate functions of β_n by

$O(\log(h(\beta_n) + \log n))$ space bounded deterministic Turing machine, where $h(\beta_n)$ is the hardware size of aggregate β_n (with an input of length n).

The hardware size and running time of uniform families of aggregates are equivalent to the space of deterministic Turing machines and the depth of uniform circuits, respectively.

3.7. Universal parallel machines

An efficient general-purpose parallel machine should simulate any special-purpose machine with only a small loss of efficiency. Galil and Paul (1983) proposed a parametrized class of computers. By fixing the parameter, namely the type of the individual processors, different models are obtained.

A *parallel computer* consists of an infinite recursive graph G. On designated nodes in this graph the input is read, and on other nodes the output is produced. Identical processors are attached to the nodes of G by a recursive function. Formally, the parallel computer is specified by a pair $C = (S, \Delta)$. The geometrical structure of the computer is described by a *skeleton* $S = (\Sigma, V, D, g, I, O)$, where Σ is a finite alphabet, and $V \subseteq \Sigma^*$ a set of names of processors. Connections between processors are assumed to be fixed; each processor can communicate with a bounded number, d, of other processors. D is a set of "directions" with cardinality $|D| = d$, and $g : V \times D \rightarrow V \cup \Omega$, $\Omega \notin V$, is a recursive interconnection function. The value Ω means that a given processor in a given direction has no neighbor. Two one-to-one recursive functions $I, O : \{0,1\}^* \rightarrow V$ specify the input and output nodes. Processors are described by a recursive mapping $\Delta : V \longrightarrow \{0,1\}$, which specifies a binary encoding for a given processor.

Galil and Paul (1983) explicitly mentioned five processor models:

1. finite automata;
2. RAM;
3. k-RAM, i.e. a RAM with only k registers;
4. RAC, i.e. a RAM which in computation of t steps and with p active processors generates register contents of length $O(\log(t+p))$;
5. k-RAC.

In one computation step of the parallel computer every processor makes one step. In the beginning only input nodes are active. Inactive nodes are activated during the computation by an activity of its neighbors. The obvious complexity measures are the number of steps and the number of processors active during a computation.

There are further (uniformity) restrictions imposed on the model:

1. functions Δ, g, I, and O must be computable by an $O(n)$ space bounded and $O(n^2)$ time bounded deterministic Turing machine, and
2. the address of any active node is bounded by $O(\log p)$, where p is the number of all active nodes.

An efficient general-purpose parallel machine U is then defined as a parallel computer with a sorting network (e.g. cube connected cycles) as its skeleton. Galil and Paul (1983) showed that U can simulate any parallel computer C that uses p processors and makes t steps in only $O(t \log^2 p)$ steps using

$O(p)$ processors. Moreover, the same result holds when a much stronger model of parallel computer C is to be simulated, in which there is no underlying graph but each processor can request information from any other processor.

3.8. Hardware modification machines

Hardware modification machines, developed by Dymond (1980), are similar to conglomerates where each finite control is given an additional power to modify its input connections. The machine is automatically uniform because it constructs itself.

Hardware modification machines have a "truly" modifiable structure in the sense that processors modify the explicit links among themselves. Processors of other modifiable structure machines, P-RAM and SIMDAG, have no direct links to each other and communicate only indirectly via shared (or global) memory. These communication patterns can be viewed as logical modifiable links among processors. The modifiable structure of the links, however, has nothing in common with the hardware structure of the computer.

3.9. Global memory machines

A *global memory machine* consists of an infinite number of processors attached to a globally accessible shared memory. Such a machine was introduced in the SIMDAG model [Goldschlager (1978)] or in the P-RAM model [Fortune and Wyllie (1978)]. "SIMDAG" stands for "single instruction stream, multiple data stream, global memory"; "P-RAM" stands for "parallel random access machine". (Note: PRAM's of Savitch and Stimson (1979) have no global memory, but a given processor can initiate offspring processors. They are not global machines but rather a special case of a parallel computer model of Galil and Paul (1983).)

Each processor in a global memory machine possesses an infinite number of general purpose registers and a unique read-only processor identity register which is preset to i in the i-th processor, $i \in N$. A *program* consists of a finite list of instructions in one of the following forms:

1. Read a value from a specified place in the global memory.
2. Write a value to a specified place in the global memory.
3. Perform an internal computation.
4. Conditional transfer, halt.

The allowable internal computations usually consist of direct and indirect register transfers, logical and arithmetic operations.

Each machine is specified by program P and a processor bound $P(n)$. The computation starts with the n words of an input of size n placed in the first n locations of common memory. All other memory locations and general purpose registers are set to zero. The first $P(n)$ processors are activated simultaneously; they synchronously execute program P. The computation halts when all $P(n)$ processors are halted. The output is then to be found in some specified place in the global memory.

The most important resources are the *processor bound* $P(n)$, the number of processors used as a function of input size, and the *time bound* $T(n)$, the number of steps executed as a function of input size.

In order to obtain closer relation with other models, Parberry (1985) used another two resource bounds. *Space* $S(n)$ is the maximum number of non-zero entries in the global memory and registers at any time during the computation. The machine is said to have *wordsize* $W(n)$ if every value placed into a register or global memory location during the computation has absolute value less than $2^{W(n)}$.

Memory access conflicts can be dealt with in several ways. SIMDAG's, or CRCW PRAM's (for concurrent-read concurrent-write), allow simultaneous reading and writing of several processors from and to the same global memory location. In the case of the writing conflict only the lowest numbered processor succeeds. P-RAM's, or CREW PRAM's (for concurrent-read exclusive-write), allow no simultaneous writing to the same location. EREW PRAM's allow neither writing nor reading conflicts.

Stockmeyer and Vishkin (1984) studied the correspondence between CRCW PRAM's and circuits. They obtained the following:

Theorem. A CRCW PRAM with $P(n)$ processor bound that operates in time $T(n)$ can be simulated by a family of circuits of size polynomial in $P(n)$, $T(n)$, and n, and depth linear in $T(n)$. The result holds also for CREW PRAM.

Theorem. A circuit of size S and depth T with n inputs and at most n outputs can be simulated by a nonuniform CRCW PRAM with processor bound linear in $(S+n)$, program size logarithmic in $(S+n)$, that runs in time linear in T. For CREW PRAM the time bound is to be changed to $O(T + \log n)$.

Note: A *nonuniform* CRCW PRAM allows programs to depend on the size of the input.

3.10. Practical and impractical models and their relations

Global memory models are popular for their theoretical power and universality. They are, however, highly impractical. Another universal but impractical model is a network machine, which is similar to a conglomerate [Goldschlager (1978)] or a parallel machine of Galil and Paul (1983).

A *network machine* consists of an infinite family of finite graphs, one for each input size. Each node represents a processor. Each edge represents a communication link between processors. The resources defined for global memory machines, $(P(n),\ S(n),\ T(n),\ W(n))$ remain the same for network machines. Since there is no global memory for a network machine, only the registers are considered for $S(n)$ and $W(n)$.

Amongst more practical models are uniform circuits and feasible network machines. A *feasible network machine* is a network machine with:

1. constant number of general purpose registers in each processor,
2. degree 3 of underlying graphs
3. interconnection function computable by a $O(\log P(n))$ time bounded deterministic Turing machine.

These constraints are designed to make the model more suitable for fabrication in a VLSI-like environment.

Parberry (1985) showed that all of the abovementioned machines can be unified by "reduction to sorting". An important consequence of the reduction is a possibility to simulate unpractical global memory machines and networks by practical models of feasible networks and uniform circuits. His results include:

Theorem. There is a feasible network machine which can simulate any global memory machine or network of $P(n)$ processors, space $S(n)$, time $T(n)$ and wordsize $W(n)$ using $S(n)$ processors, wordsize $W(n)$ and time

$$O\ (\ T(n)\frac{\log^2 P(n)}{\log S(n) - \log P(n) + 1} + T(n) \log S(n)\ )\ .$$

Theorem. Every global memory or network machine of $P(n)$ processors, space $S(n)$, time $T(n)$ and wordsize $W(n)$ can be simulated by a uniform circuit of depth $O(\ T(n) \log S(n) \log W(n)\ )$ and width $O(\ S(n)\ W(n)\ )$.

The correspondence to deterministic Turing machines can be stated as follows:

Theorem. Every global memory or network machine of $P(n)$ processors, space $S(n)$, time $T(n)$ and wordsize $W(n)$ can be simulated by a deterministic Turing machine using $O(\ S(n)\ W(n)\ )$ space and $O(\ T(n)\ (\ \log^2 P(n) + \log S(n)\ )\ )$ reversals.

Theorem. An $S(n)$ space, $R(n)$ reversal bounded k-tape deterministic Turing machine can be simulated on a global memory machine with processors and space $O(\frac{S(n)^k}{\log S(n)})$, time $O(R(n) \log S(n))$ and wordsize $O(\log S(n))$.

Upfal (1984) gave an interesting result for probabilistic simulation of (CRCW) PRAM:

Theorem. Any PRAM of $P(n)$ processors, space $S(n)$ and time $T(n)$ can be simulated by a feasible network machine of $P(n)$ processors and space $S(n)$. The simulation terminates within $O(t \log^2 n)$ steps with probability $1 - O(\min[e^{-St}, e^{-S \log n}])$, for some $S > 0$ (independent of n and t).

Chapter 4
Practical network machines

4.1. Notation

Suppose we are given n processors, $n = 2^q$, and i, $i \in [0, 2^q - 1]$, an address with binary representation $i_{q-1} i_{q-2} \cdots i_0$. Each processor has local registers and there is some communication geometry between the processors. Then

$i_k = (i \text{ div } 2^k) \bmod 2$ is the k-th bit of i ;

$\bar{i}_k = 1 - i_k$ is the complement of i_k ;

$i^{(k)} = i + \bar{i}_k 2^k - i_k 2^k = i_{q-1} i_{q-2} \cdots i_{k+1} \bar{i}_k i_{k-1} \cdots i_0$;

$i_{<k} = i \bmod 2^k = i_{k-1} i_{k-2} \cdots i_0$;

$i_{\geq k} = i \text{ div } 2^k = i_{q-1} i_{q-2} \cdots i_k$;

$PE(i)$ is the processor with address i ;

$A(i)$, $B(i)$, etc. are the contents of registers of $PE(i)$.

A general type of operation a network machine can do in one computational step is to replace the contents of a processor register by a new value, which is given by some function applied to the previous register contents of the processor and its neighbors. This is done for all processors at once.

Assuming only one register, A, per processor, the computational step is represented by an assignment

$$A(p) \leftarrow g(A(x_1), A(x_2), \cdots ; y_1, y_2, \cdots), \ (P),$$

which means that if condition P is satisfied, function $g(\cdot\,;\,y_1, y_2, \cdots)$ is applied to arguments $A(x_1)$, $A(x_2)$, $\cdots$ and the value is then assigned to register A of $PE(p)$. The variables x_1, x_2, $\cdots$, y_1, y_2, $\cdots$ must not depend on the contents of the registers.

4.2. Function σ

The communication geometry of the more technologically practical models from the previous chapter, feasible networks and universal circuits, may depend on the computed problem and its size. In this chapter, a more restricted subclass of network machine models is discussed. The communication geometry of the following models is extremely simple and regular, and universal for a wide class of problems. The power of the models is, however, limited.

Two classes of communication geometries can be recognized. Structures of the first class, which we call "σ-polynomial" structures, have links between physically close processors. Structures of the second, "σ-exponential" class, have links between processors with similar addresses. The names of the classes come from the behavior of a function $\sigma(m)$, defined by Gentleman (1978) as "the maximum number of processors at which data originally available only at a single processor can be made available in m or fewer data movement steps". A more precise definition will be used here.

Definition 4.2.1. If a communication geometry model is represented by a family of graphs Γ, then its $\sigma(m)$ is the cardinality of the largest graph G from the family Γ of radius at most m :

$$\sigma(m) = \max_{G \in \Gamma,\ \mathrm{rad}(G) \le m} |V(G)| \,, \qquad \mathrm{rad}(G) = \min_{x \in V(G)} \max_{y \in V(G)} d_G(x,y) \,,$$

where d_G is the distance between vertices x and y in graph G .

Definition 4.2.2. A model is called *σ-polynomial* iff its $\sigma(m)$ is $m^{O(1)}$.

Definition 4.2.3. A model is called *σ-exponential* iff its $\sigma(m)$ is not $m^{O(1)}$.

4.3. σ-polynomial models

A RING computer (fig.0c) provides processors with only two links. $PE(i)$ is connected to $PE((i+1) \bmod n)$ and $PE((i-1) \bmod n)$.

A k-dimensional MESH computer (fig.0a) requires $2k$ connections per processor. In this model, processors may be thought of as logically arranged as in a k-dimensional array. Processors are connected to their neighbors along each dimension. Let $i_{(j)}$ be the part of address i that correspond to the j-th dimension, i.e. j-th "coordinate" of $PE(i)$. Let $n_{(j)}$ be the number of processors in the j-th dimension. $PE(i_{(k-1)}, \cdots, i_{(j)}, \cdots, i_{(0)})$ is connected to $PE(i_{(k-1)}, \cdots, i_{(j)}+1, \cdots, i_{(0)})$ and $PE(i_{(k-1)}, \cdots, i_{(j)}-1, \cdots, i_{(0)})$, for $0 \le j < k$ and $0 < i_{(j)} < n_{(j)}$. Processors on the boundaries ($i_{(j)}=0$ or $i_{(j)}= n_{(j)}$) have less than $2k$ connections.

A k-dimensional TOROID computer (fig.0b) is a combination of a RING and a k-dimensional MESH. Like in a k-dimensional MESH, processors have a logical form of a k-dimensional array. Connections in each dimension, however, form a ring. $PE(i_{(k-1)}, \cdots, i_{(j)}, \cdots, i_{(0)})$ is connected to $PE(i_{(k-1)}, \cdots, (i_{(j)}+1) \bmod n_{(j)}, \cdots, i_{(0)})$ and $PE(i_{(k-1)}, \cdots, (i_{(j)}-1) \bmod n_{(j)}, \cdots, i_{(0)})$, for $0 \le j < k$ and $0 \le i_{(j)} \le n_{(j)}$.

By a simple geometrical argument, $\sigma(m)$ is $O(m^k)$ for both k-dimensional MESH and TOROID models. Since the radius of a k-dimensional MESH or TOROID with $x_1 \times x_2 \times \cdots \times x_k$ processors is

$$\mathrm{rad}(G) = \lfloor x_1/2 \rfloor + \lfloor x_2/2 \rfloor + \cdots + \lfloor x_k/2 \rfloor = \sum_{i=1}^{k} \lfloor x_i/2 \rfloor ,$$

function $\sigma(m)$ is given by

$$\sigma(m) = \max_{\sum_{i=1}^{k} \lfloor x_i/2 \rfloor \leq m} \prod_{i=1}^{k} x_i = \max_{\sum_{i=1}^{k} y_i = m} \prod_{i=1}^{k} (2y_i+1) = (a+1)^b \, a^{k-b} ,$$

where $a = m \operatorname{div} k$ and $b = m \bmod k$.

Note: a RING is a special case of a one-dimensional TOROID.

Since the MESH and the TOROID have more than three connections per processor (for $k > 1$), only the RING is a feasible network. However, the most common types of a MESH and a TOROID are the 2-dimensional versions with 4 connections per processor. If the definition of feasible networks is extended to allow degree 4 graphs, the 2-dimensional MESH and TOROID are feasible.

The three abovementioned structures and other similar models have been used in many graph and matrix algorithms [e.g. Levitt and Kautz (1972), Nassimi and Sahni (1979), Kung and Leiserson (1980), Brent and Kung (1983)]. These architectures are very attractive for VLSI design because of the simple interconnection pattern and high utilization of processors. Unlike the more powerful models of parallel computation, σ-polynomial structures do not achieve logarithmic times for standard numeric problems, such as FFT. Gentleman (1978) studied lower bounds on the time requirements of computers with various $\sigma(m)$.

He showed that if $\sigma(m)$ is $O(m^2)$, matrix multiplication requires at least linear time. Similar results can be obtained for other algorithms with large data movements and for other σ-polynomial models.

Note: The number of processors n for the RING, the MESH and the TOROID is not required to be a power of 2.

4.4. σ-exponential models

A CUBE computer (fig.2d) with $n = 2^q$ processors has q links per processor [Pease (1977)]. Processors are connected along edges of a q-dimensional cube, i.e. $PE(i)$ is connected to $PE(i^{(k)})$, for $0 \leq k < q$. Set of all links along k-th dimension, i.e. $\{ (PE(i), PE(i^{(k)})) \mid 0 \leq i < 2^q \}$, is called a *sheaf* k.

Trivially, $\sigma(m) = 2^m$. Since q is not bounded by 3, a CUBE is not a feasible network.

Note: A CUBE and a MESH can be viewed as opposite extremes of one type of architecture. While the MESH has a bounded number of dimensions and an unbounded number of processors along each dimension, the CUBE has an unbounded number of dimensions and a bounded (by 2) number of processors along each dimension.

A PERFECT SHUFFLE computer (fig.3a) has three links per processor [Stone (1971)]. $PE(i)$ is connected to $PE(i^{(0)})$, $PE(\text{shuffle}(i))$ and $PE(\text{unshuffle}(i))$, where shuffle(i) and unshuffle(i) are defined to be, respectively, the integers with binary representations $i_{q-2} \cdots i_0 i_{q-1}$ and

$i_0 i_q{-}1 \cdots i_1$. Another, equivalent definition of shuffle and unshuffle maps (fig.3b) has the form

$$\text{shuffle}(i) = \textbf{if } i < h/2 \textbf{ then } 2i \textbf{ else } 2i - n + 1 \ ,$$

$$\text{unshuffle}(i) = \textbf{if } i \text{ is even } \textbf{then } i/2 \textbf{ else } (i-1)/2 + n/2 \ .$$

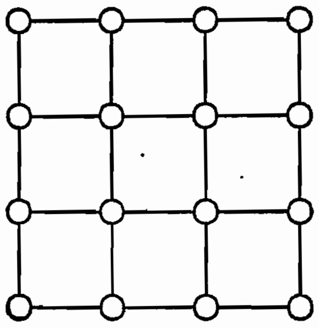
Since the distance (the number of links) between $PE(0)$ and any other $PE(p)$ is at most $2n-1$, the radius of a PERFECT SHUFFLE graph is at most $2n-1$, i.e. $\sigma(m) \geq 2^{m/2}$. A PERFECT SHUFFLE is a feasible network.

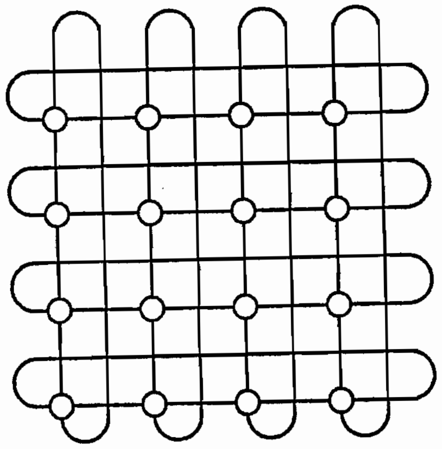
A *cube-connected-cycles* (CCC) computer (fig.4) is a combination of a CUBE and a RING [Preparata and Vuillemin (1979),(1981)]. Processors are connected along edges of a $(q-r)$-dimensional cube that has its vertices replaced by 2^r-cycles (rings). More precisely, let $n = 2^q$ be the number of processors and r be some positive integer such that $2^r \geq q-r \geq 0$. Then the q-bit address of a processor $PE(p)$, $0 \leq p < 2^q$, can be divided into two parts: a r-bit part h , $0 \leq h < 2^r$, that corresponds to the RING-type links among processors, and a $(q-r)$-bit part i , $0 \leq i < 2^{q-r}$, that corresponds to the CUBE-type links. Without loss of generality, let $PE(p) = PE(h\,2^{q-r}+i) = PE(h,i)$. Processor $PE(h,i)$, $0 \leq h < 2^r$, $0 \leq i < 2^{q-r}$, is connected precisely to three other processors, $PE((h-1) \bmod 2^r, j)$, $PE((h+1) \bmod 2^r, j)$, and $PE(h, j^{(i)})$. The first two links belong to the neighboring processors within the 2^r-cycle. The third link corresponds to an edge of the $(q-r)$-cube.

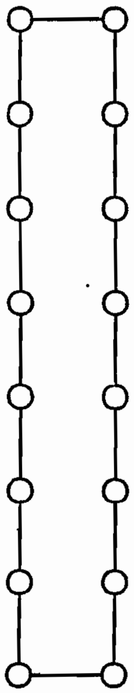
The radius of a CCC graph is $2^r-1+q-r$. A choice of r and q is valid iff conditions $m > 2^r-1+q-r$ and $2^r \geq q-r$ are satisfied. It is easy to show that $\sigma(m) > 2^{(m+1)/4}$ for $r = \left\lfloor \log \frac{m+1}{2} \right\rfloor$, $q = \left\lfloor \frac{m+1}{4} \right\rfloor + r$. In reality, $\sigma(m)$ behaves like $2^{(m+1)/2}$ for $m >> 1$ and $2^r \approx q-r$. The CCC is a feasible network.

The unbounded number of links per processor makes the CUBE architecture impractical for computers with a large number of processors. The CUBE model is, however, a practical tool for the development of algorithms for the PERFECT SHUFFLE and the CCC. As will be discussed in the following chapter, CUBE algorithms with certain restrictions are compatible with PERFECT SHUFFLE and CCC computers.

Note: An ULTRACOMPUTER, which was introduced and described with many technological details by Schwartz (1980), has links of both a PERFECT SHUFFLE and a RING.


a) MESH


b) TOROID


c) RING

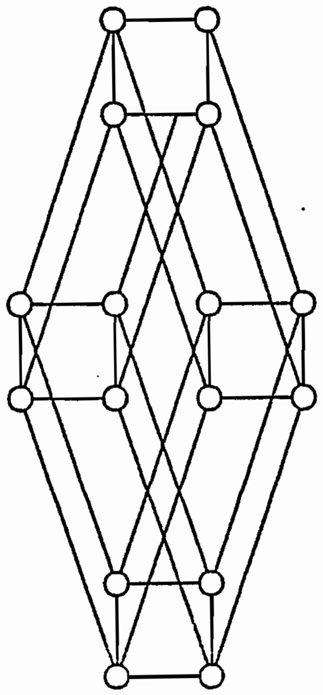

d) CUBE

Fig.2: MESH, TOROID, RING and CUBE for sixteen processors.

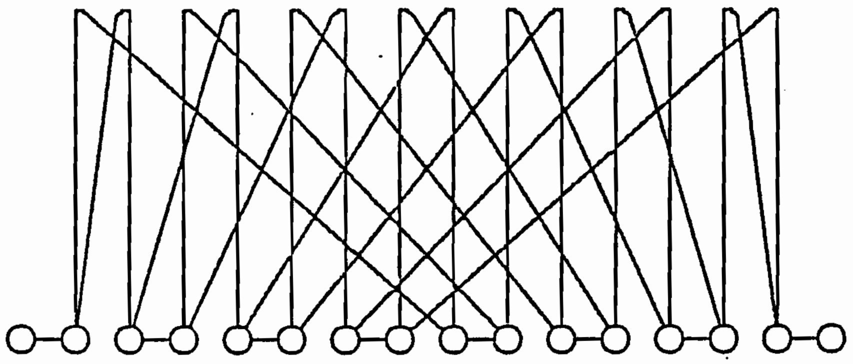


Fig.3a: PERFECT SHUFFLE for sixteen processors.

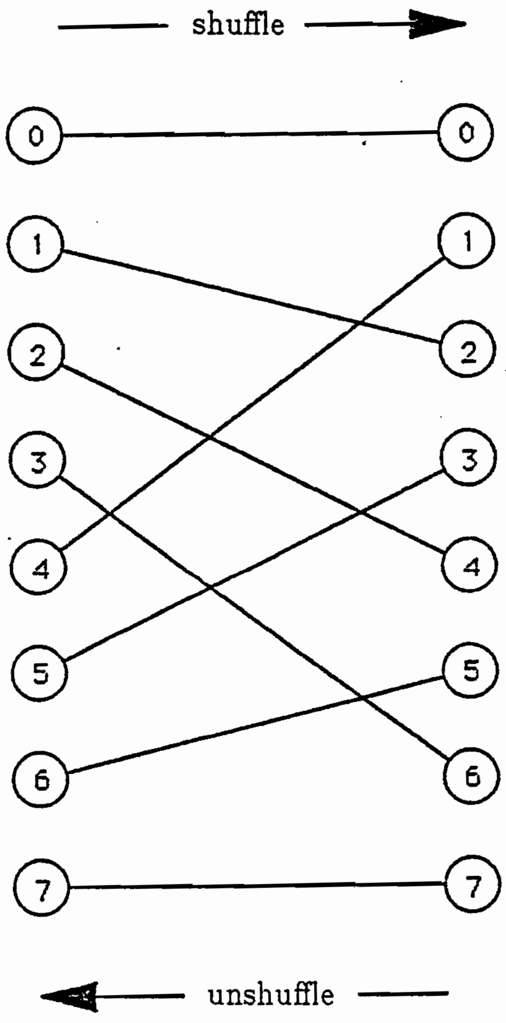


Fig.3b: Shuffle (unshuffle) mapping for eight processors.

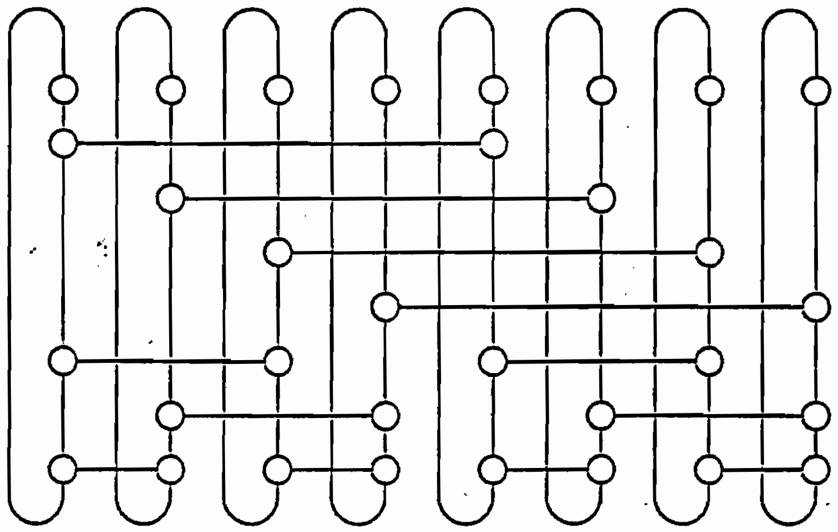


Fig.4: Cube-connected cycles for q=5 and r=2.

Chapter 5
Equivalence of CCC, PERFECT SHUFFLE and CUBE

5.1. ASCEND and DESCEND classes

For a wide class of problems there are algorithms whose data exchange patterns correspond to the links of a binary multidimensional cube. Preparata and Vuillemin (1979) proposed two dual classes of such algorithms. Assume that input data are stored in a continuous block of addresses from $[0, 2^s - 1]$.

An algorithm in the ASCEND class performs a sequence of basic operations on pairs of data with relative offsets successively $2^0, 2^1, \cdots, 2^{s-2}, 2^{s-1}$. Assuming only one register per processor, an ASCEND-type CUBE algorithm (operating on the block of the first 2^s addresses in s steps) has the following form:

for $m = 0$ **to** $s-1$ **do** $A(p) \leftarrow f\ (A(p), A(p^{(m)}); m, p),\ (0 \leq p < 2^s)$,

where $f\ (\cdot\ ; m, p)$ is some function that depends only on the address of the processor, p, and the order of the sheaf, m. Its arguments, $A(p)$, $A(p^{(m)})$, are the register contents of the processor and its neighbors in the given sheaf.

An algorithm in the DESCEND class performs a sequence of basic operations on pairs of data with relative offsets successively $2^{s-1}, 2^{s-2}, \cdots, 2^1, 2^0$, i.e. the sheaf-index m is running in the opposite direction. An algorithm from one class can be simulated by an algorithm from the other class, reversing the order of address bits by the bit reversal permutation.

Many fundamental algorithms can be decomposed into ASCEND- or DESCEND-type subroutines that use the same block of addresses. We will call them CUBE-*feasible* algorithms. Algorithms for some applications, such as *bitonic merge* or *cyclic shift*, are directly in the ASCEND or DESCEND classes. These algorithms run in $O(\log n)$ steps. Other applications, such as *permutation*, *shuffle*, *unshuffle*, *bit-reversal-permutation*, *odd-even-merge*, *Fast-Fourier-Transform*, *convolution*, or *matrix transposition*, have programs consisting of a short sequence of ASCEND- or DESCEND-type algorithms and run also in $O(\log n)$ steps. Some applications, such as *bitonic sort*, *odd-even-sort*, or calculations of *symmetric functions*, have algorithms with loops or recursive calls and have higher time bounds.

We will show that algorithms for a CUBE machine from the ASCEND and DESCEND classes (and, consequently, CUBE-feasible algorithms) can be efficiently simulated on PERFECT SHUFFLE and CCC machines of any sufficient size and give the bounds in terms of the size of the problem. In the following proofs, only one register, A , per processor is considered, since allowing more registers would not bring any significant change.

5.2. Simulation of CUBE by PERFECT SHUFFLE

Stone (1971) described the similarities between the PERFECT SHUFFLE architecture and a binary hypercube, which implicitly included the main idea of the algorithm bellow. However, the sizes and computation times were not discussed.

Theorem 5.2.1. An ASCEND-type algorithm running on the first 2^s addresses of a CUBE computer in s steps can be simulated on a PERFECT SHUFFLE computer with at least 2^s processors in $3s$ steps.

Proof. Let 2^q be the size of the PERFECT SHUFFLE for arbitrary $q \geq s$. Let $A_0(p)$ be the initial contents of register A of $PE(p)$ of the CUBE, and $A_k(p)$, $0 < k \leq s$, be the contents of the same register after k iterations of the for-loop of the simulated ASCEND-type CUBE algorithm

for $m = 0$ **to** s-1 **do** $A(p) \leftarrow f\ (A(p), A(p^{(m)}); m, p)$,

$(0 \leq p < 2^s)$,

i.e. immediately after the iteration with $m = k-1$. Let $A'_k(p)$ and $A''_k(p)$ have similar meanings for, respectively, the first and second loop of the simulating PERFECT SHUFFLE algorithm below. Let $A'_0(p) = A_0(p)$ for $0 \leq p < 2^s$. The simulating algorithm is correct iff $A''_s(p) = A_s(p)$, $0 \leq p < 2^s$.

Algorithm 5.2.1.

```
for $m =$ 0 to $s$-1 do begin
    $A(p) \leftarrow f\ (A(p), A(p^{(0)}); m, p_{<m} 2^{q-k} + p_{\geq m}),\ (P);$
    $A(p) \leftarrow A(\text{shuffle}(p));$
end;
for $m =$ 0 to $s$-1 do $A(p) \leftarrow A(\text{unshuffle}(p));$
where $P \equiv 0 \leq p_{<m} 2^{q-k} + p_{\geq m} < 2^s$.
```

Justification.

Note that $\text{unshuffle}^k(p) = p_{<m}2^{q-k}+p_{\geq m}$, $\text{unshuffle}^k(p^{(0)}) = \text{unshuffle}^k(p)^{(k)}$ and $\text{shuffle}^k(\text{unshuffle}^k(p)) = p$ for $0 \leq p < 2^q$ and $k \geq 0$. Assume that

$$A'_k(p) = A_k(\text{unshuffle}^k(p))$$

for $0 \leq \text{unshuffle}^k(p) < 2^s$ and $0 \leq k \leq l < s$.

Then

$$\begin{aligned} A'_{l+1}(\text{shuffle}(p)) &= f(A'_l(p), A'_l(p^{(0)}); l, \text{unshuffle}^l(p)) \\ &= f(A_l(\text{unshuffle}^l(p)), A_l(\text{unshuffle}^l(p)^{(l)}); l, \text{unshuffle}^l(p)) \\ &= A_{l+1}(\text{unshuffle}^l(p)) \quad \text{for} \quad 0 \leq \text{unshuffle}^l(p) < 2^s, \end{aligned}$$

i.e.

$$A'_{l+1}(p) = A_{l+1}(\text{unshuffle}^{l+1}(p)) \quad \text{for} \quad 0 \leq \text{unshuffle}^{l+1}(p) < 2^s.$$

By mathematical induction

$$A''_0(p) = A'_s(p) = A_s(\text{unshuffle}^s(p)) \text{ for } 0 \leq \text{unshuffle}^s(p) < 2^s,$$

or equivalently,

$$A''_0(\text{shuffle}^s(p)) = A_s(p) \quad \text{for} \quad 0 \leq p < 2^s.$$

Since

$$A''_{k+1}(p) = A''_k(\text{shuffle}(p)) \quad \text{for} \quad 0 \leq p < 2^q \quad \text{and} \quad 0 \leq k < s,$$

by mathematical induction

$$A''_s(p) = A''_0(\text{shuffle}^s(p)) = A_s(p) \quad \text{for} \quad 0 \leq p < 2^s.$$

It is easy to see that the algorithm runs in $3s$ steps. □

5.3. Simulation of CUBE by CCC

Preparata and Vuillemin (1979,1981) discuss in detail the simulation of an ASCEND/DESCEND CUBE algorithm on a CCC. The size of the CCC matched the size of the CUBE and had a special form 2^{2^r+r}. The computation time was given in terms of the size of the machines. We extend this result and simulate an ASCEND/DESCEND CUBE algorithm on a CCC of any sufficient size. The computation time is given in terms of the size of the problem.

Theorem 5.3.1. An ASCEND-type algorithm running on the first $2^{\hat{s}}$ addresses of a CUBE computer in $\hat{s}$ steps can be simulated on a CCC computer with at least $2^{\hat{s}}$ processors in $3s+5\cdot 2^t-t-7$ steps, where $s=\min(\hat{s}, q-r)$, $t=\max(0, \hat{s}-q+r)$, 2^q is the size of the CCC, $q \geq \hat{s}$, and 2^r, $2^r \geq q-r$, is the size of its cycles.

Proof. Let 2^q, $q \geq \hat{s}$, be the size of the CCC; the highest r bits of a processor address correspond to the RING-type connections, while the lowest $q-r$ bits correspond to the CUBE-type connections. In the CUBE being simulated only the lowest $\hat{s}$ bits of a processor address are used. Without a loss of generality, we can consider the size of the CUBE to be 2^q. If the actual size is higher, we ignore the higher bits of the processor address. If it is lower, we add (but do not use) virtual higher bits to the address. In both the CUBE and the CCC we will use separate indices, i and j, for each part of the address, i.e. $PE(p) = PE(i\,2^{q-r}+j) = PE(i,j)$, $0 \leq i < 2^r$, $0 \leq j < 2^{q-r}$. These bounds are also assumed for any further reference to i

or j in the proof. Using this notation, the ASCEND-type CUBE algorithm to be simulated reads:

```
(1) for $m =$ 0 to $s-1$ do
        $A(i,j) \leftarrow f\ (A(i,j), A(i,j^{(m)}); m, p), (P)$;
(2) for $m =$ 0 to $t-1$ do
        $A(i,j) \leftarrow f\ (A(i,j), A(i^{(m)},j); m+q-r, p), (P)$;
```

where $P \equiv (0 \leq p = i\,2^{q-r}+j < 2^{\hat{s}})$, $s = \min(\hat{s}, q-r)$ and $t = \max(0, \hat{s}-q+r)$. The simulating algorithm is divided into two parts that correspond, respectively, to the two loops of the algorithm above. Using the following lemmas, loop (1) can be simulated in $3 \cdot (s+2^t)-5$ steps; loop (2) can be simulated in $2 \cdot 2^t - t - 2$ steps. Therefore, the whole simulation runs in $3s + 5 \cdot 2^t - t - 7$ steps. □

Lemma 5.3.1. Loop (1) can be simulated in $3 \cdot (s+2^t)-5$ steps.

Proof. The following three loops on the CCC simulate loop (1) on the CUBE. Similarly to the previous proof, let $A_k(i,j)$ be the contents of register A of $PE(i,j)$ after the k-th step of (1), and $A'_k(i,j)$, $A''_k(i,j)$, $A'''_k(i,j)$ be the contents of $PE(i,j)$ after the k-th step of each of the three simulating loops, respectively. Let $A'_0(i,j) = A_0(i,j)$ for $0 \leq i < 2^t$ and $0 \leq j < 2^s$. The simulation is correct iff $A'''_{s-1}(i,j) = A_0(i,j)$ for the same range of i and j.

Algorithm 5.3.1.

```
for $m =$ 0 to $2^t$-2 do $A(i,j) \leftarrow A((i+k) \bmod 2^r, j)$;
for $m = 0$ to $s+2^t-2$ do begin
    $A(i,j) \leftarrow A((i-k) \bmod 2^r, j)$ ,   $(m>0)$;
    $A(i,j) \leftarrow f(A(i,j), A(i,j^{(i)}); i, p)$ ,   $(P)$;
end;
for $m = 0$ to $s-2$ do  $A(i,j) \leftarrow A((i+k) \bmod 2^r, j)$;
where $P \equiv (0 \leq i < 2^t$ and $0 \leq p = (i+2^t-m-1) \cdot 2^{q-r}+j < 2^{\hat{s}})$.
```

Justification.

The first loop shifts data within each 2^r-cycle, so that in the end, sheaf 0 is accessible to data with the original address $(2^t-1) \cdot 2^{q-r}+j$, $0 \leq j < 2^s$:

$$A'_k(i,j) = A'_0((i+k) \bmod 2^r, j) \quad \text{for any } i \ , \ j \ .$$

Hence

$$A''_0(i,j) = A'_{2^t-1}(i,j) = A_0((i+2^t-1) \bmod 2^r, j)$$

for $i = 0$ or $2^r-2^t+1 \leq i < 2^r$ and $0 \leq j < 2^s$.

The second loop applies function f on appropriate data. Note that data in $PE(i,j)$ can access sheaf i only. By mathematical induction (details omitted)

$$A''_k(i,j) = \begin{cases} A_0((i+2^t-k) \bmod 2^r, j) & \text{if } P_1 \text{ and } 0 \leq j < 2^s \\ A_k((i+2^t-k) \bmod 2^r, j) & \text{if } P_2 \text{ and } 0 \leq j < 2^s \\ A_s((i+2^t-k) \bmod 2^r, j) & \text{if } P_3 \text{ and } 0 \leq j < 2^s \end{cases}$$

for $0 \leq k \leq s+2^t-1$ and

$P_1 \equiv (2^r-2^t+1 \leq i < 2^r)$,

$P_2 \equiv (\max(k-2^t,0) \leq i < \min(k,s))$,

$P_3 \equiv (s \leq i < \min(k, 2^r)$ or $0 \leq i < k-2^r)$. Hence

$$A'''_0(i,j) = A''_{s+2^t-1}(i,j) = A_s((i-s+1) \bmod 2^r, j) \quad \text{for}$$

$$s-1 \leq i < \min(s+2^t-1, 2^r) \text{ or } 0 \leq i < s+2^t-2^r-1 \quad \text{and} \quad 0 \leq j < 2^s \ .$$

The last loop shifts data back to the initial locations:

$$A'''_k(i,j) = A'''_0((i-s+1) \bmod 2^r, j) \quad \text{for any } i, j \ .$$

Hence

$$A'''_{s-1}(i,j) = A'''_0((i+s-1) \bmod 2^r, j) = A_s(i,j)$$

$$\text{for } 0 \leq i < 2^t \quad \text{and} \quad 0 \leq j < 2^s \ ,$$

and we are done. If the empty step in the second loop, " $A(i,j) \leftarrow \cdots, (m > 0)$ " for $m = 0$, is not counted, the three loops run in $3 \cdot (s+2^t) - 5$ steps. □

Lemma 5.3.2. Loop (2) can be simulated in $2 \cdot 2^t - t - 2$ steps.

Proof. An auxiliary subroutine *SHUFFLE*(x) performs the shuffle operation on consequent s^{x+1}-size blocks of processors in each 2^r-cycle of the CCC, i.e.

$$A_{after}(\cdots i_{x+1} i_x i_{x-1} \cdots i_1 i_0, j)$$

$$= A_{before}(\cdots i_{x+1} i_{x-1} i_{x-2} \cdots i_0 i_x, j) \ .$$

Algorithm 5.3.2.

```
Subroutine SHUFFLE($x$):
for $m = 1$ to $2^x-1$ do begin
    $A(i,j) \leftarrow \begin{cases} A((i+1) \bmod 2^r, j), & (i_0 = m_0 \text{ and } P); \\ A((i-1) \bmod 2^r, j), & (i_0 = \overline{m}_0 \text{ and } P); \end{cases}$
end;
```

where $P \equiv (2^x - m \leq i_{<x+1} < 2^x + m)$

Justification.

Let $B^{(m)}_{i_{\geq x+1},\, j} = \{PE(i,j)\}^{2^x+m}_{i_{<x+1}=\, 2^x-m-1}$, $0 \leq i < 2^r$, $0 \leq j < 2^{q-r}$, $1 \leq m < 2^x$, be a $(2 \cdot m + 2)$- size block of processors. After each step of the loop, each block $B^{(m)}_{i_{\geq x+1},\, j}$, $0 \leq i < 2^r$, $0 \leq j < 2^{q-r}$, is shuffled, while the remaining locations are left unchanged (see fig.5). In the end, every 2^{x+1}-size block $B^{(2^x-1)}_{i_{\geq x+1},\, j}$, $0 \leq i < 2^r$, $0 \leq j < 2^{q-r}$, is shuffled. If each iteration of the loop is counted as one step, *SHUFFLE*(x) runs in 2^{x-1} steps.

The following two loops on the CCC simulate loop (2) on the CUBE. Let $A_k(i,j)$ be the contents of $PE(i,j)$ after the k-th step of (2), and $A'_k(i,j)$, $A''_k(i,j)$ be the contents of $PE(i,j)$ after the k-th step of each of the two simulating loops, respectively. Let $A'_0(i,j) = A_0(i,j)$ for $0 \leq i < 2^t$ and $0 \leq j < 2^s$. The simulation is correct iff $A''_t(i,j) = A_0(i,j)$ for the same range of i and j.

Algorithm 5.3.3.

```
for $m = 0$ to $t-1$ do begin
    $SHUFFLE(m)$;
    $A(i,j) \leftarrow \begin{cases} f(A(i,j), A((i+1) \bmod 2^r, j); m+q-r, p), & (P \text{ and } Q); \\ f(A(i,j), A((i-1) \bmod 2^r, j); m+q-r, p), & (\overline{P} \text{ and } Q); \end{cases}$
end;
for $m = 0$ to $t-1$ do   $SHUFFLE(m)$;
```

where $P \equiv (i_0 = 0)$, $\bar{P} \equiv (i_0 = 1)$ and

$$Q \equiv (0 \leq p = i_{\geq m+1} 2^{q-r+m+1} + (i_0 i_1 \cdots i_m) \cdot 2^{q-r} + j < 2^{\hat{s}}).$$

Justification.

Note that RING-type links of the CCC can be used as a sheaf $q-r$:

$$A(i^{(0)}, j) = \begin{cases} A((i+1) \bmod 2^r, j) & \text{if } i_0 = 0, \\ A((i-1) \bmod 2^r, j) & \text{if } i_0 = 1. \end{cases}$$

The first loop uses $SHUFFLE(m)$ to bring required data to the sheaf $q-r$ and then applies function f on them. By mathematical induction (details omitted)

$$A'_k(i,j) = A_k(i_{r-1} \cdots i_k i_0 i_1 \cdots i_{k-1}, j) \quad \text{for}$$

$$0 \leq i_{\geq k} 2^k + (i_0 i_1 \cdots i_{k-1}) < 2^{\hat{s}} \quad \text{and} \quad 0 \leq j < 2^s \; , \quad 0 \leq k \leq t \; .$$

Hence

$$A'_t(i,j) = A_t(i_{r-1} \cdots i_t i_0 i_1 \cdots i_{t-1}, j)$$

for $i_{\geq t} 2^k = 0$ and $0 \leq j < 2^s$.

The second loop brings data back to the original locations:

$$A''_k(i,j) = A''_0(i_{r-1} \cdots i_k i_0 i_1 \cdots i_{k-1}, j)$$

for any i , j , and $0 \leq k \leq t$.

Hence

$$A''_t(i,j) = A''_0(i_{r-1} \cdots i_t i_0 i_1 \cdots i_{t-1}, j)$$

$$= A'_t(i_{r-1} \cdots i_t i_0 i_1 \cdots i_{t-1}, j) = A_t(i_{r-1} \cdots i_t i_{t-1} i_{t-2} \cdots i_0, j)$$

$$= A_t(i,j) \quad \text{for } 0 \leq i < 2^t \text{ and } 0 \leq j < 2^s \; ,$$

and we are done.

The two loops run in $\sum_{m=0}^{t-1} (2 \cdot 2^m - 1) = 2 \cdot 2^t - t - 2$ steps. □

5.4. Discussion

The algorithm (that simulates ASC/DESC alg's on the CCC) is very similar to that of Preparata and Vuillemin (1979, 1981). However, in their algorithm:

1. the size of the CCC matched the size of the CUBE and had a special form 2^{2^r+r},
2. the computation time was given in terms of the machine size, and
3. the algorithm was presented in the notation "foreach <condition> pardo <operation> odpar",

while in our algorithm

1. the CCC is of any sufficient size, depending only on the problem size,
2. the computation time is given in terms of the size of the problem, and
3. the algorithm is presented in the notation "<assignment>, (<condition>)".

It is our opinion that the "foreach..." notation is harder to read and verify, since it specifies the operation for a set of processors instead of each individual processor. This is especially true when nested foreach-blocks are used. (E.g. both versions of Preparata's and Vuillemin's (1979,1981) algorithm contain a hard-to-detect error in the procedure DESCEND: the condition "$a = l\,2^r + ((p+i-1) \bmod 2^r)$" in a double-nested foreach-block should read "$a = l\,2^r + ((p-i-1) \bmod 2^r)$", which can be be verified by setting $l = 0$ and $p = i = 2^r - 1$.)

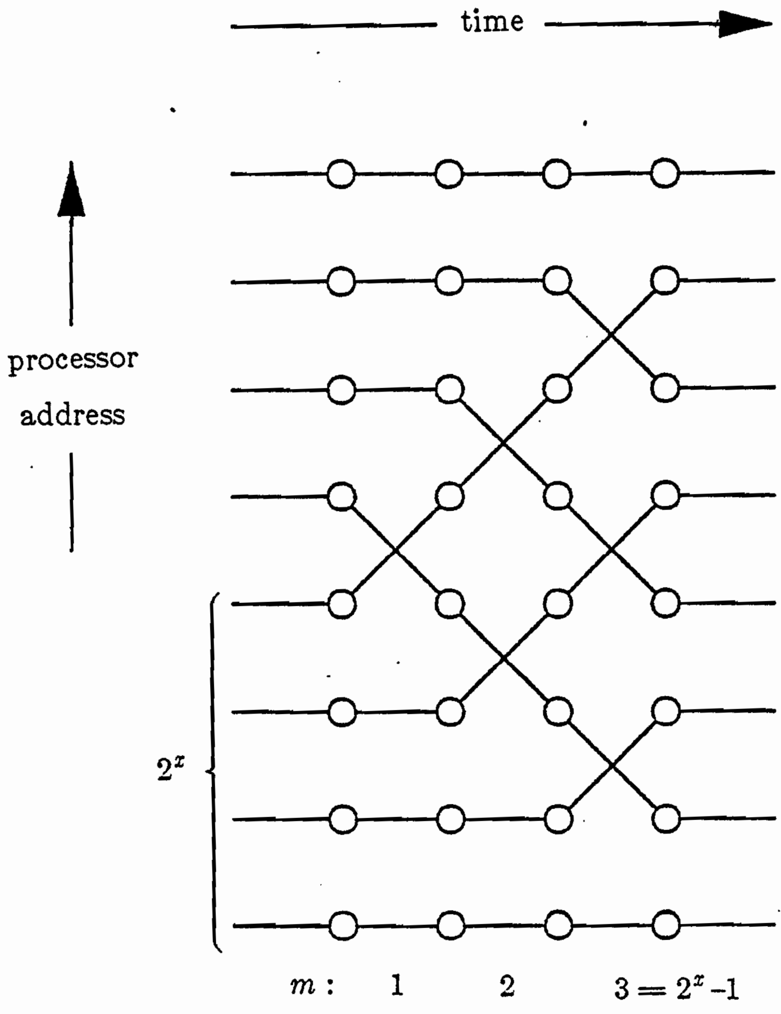


Fig.5: Data transfers by *SHUFFLE*(x) procedure in a block of 2^{x-1} addresses, $x = 2$.

Chapter 6
GCD algorithms on a CUBE

6.1. Introduction

The theoretical foundations for fast parallel computation for widely used problems of symbolic manipulation in an algebraic context are laid by Borodin, von zur Gathen and Hopcroft (1984). The problems investigated include computation of polynomial GCD's, solution of linear equations, computation of the determinant and rank of matrices. Von zur Gathen (1984) continues in the program and gives fast parallel solutions for the Extended Eucledian Scheme of two polynomials, polynomial factorization over finite fields and square free decomposition of polynomials over fields of characteristic zero and over finite fields. Recently [von zur Gathen (1986)], fast algorithms for conversion among polynomial base representations have been introduced. This includes Taylor expansion, partial fraction decomposition, the Chinese remainder algorithm, elementary symmetric functions, Padé approximation and interpolation problems.

These algorithms are designed for general models of parallel machines, such as P-RAM's or algebraic circuits. All algorithms run in time $O(\log^2 n)$, using a polynomial number of processors.

The fundamental part of these results is a fast $O(\log^2 n)$ matrix determinant algorithm that works over arbitrary fields. Borodin, von zur Gathen and Hopcroft (1984) do not give the algorithm explicitly, but rather prove its existence: A general parallelization result of Valiant, Skyum, Berkowitz and Rackoff (1981, 1983) is applied to an $O(n^5)$ sequential division-free determinant algorithm, which can be, according to Strassen (1973), derived from an ordinary $O(n^3)$ Gaussian elimination algorithm. The required number of processors is $O(n^{15})$.

Berkowitz (1984) improves this result and gives an $O(\log^2 n)$ time algorithm that uses only $O(n^{\alpha+1+\epsilon})$ processors. $O(n^\alpha)$ is the number of processors that are required for matrix multiplication in time $O(\log n)$. (Coopersmith and Winograd (1981) proved the existence of algorithms with $\alpha < 2.5$.) ϵ is an arbitrary positive constant; ϵ^{-1} acts as a multiplicative time constant. Although a part of the proof is incorrect (see the following note), the overall $t = O(\log^2 n)$, $p = O(n^{\alpha+1+\epsilon})$ bounds hold.

Note: Claim 4 of Berkowitz (1984) incorrectly states that the product of $n \times m$ and $m \times p$ lower triangular Toeplitz matrices is lower triangular and Toeplitz. An example shows that this is untrue for matrices with $n > m > p$:

$$\begin{bmatrix} 1 & & \\ 2 & 1 & \\ 3 & 2 & 1 \\ 4 & 3 & 2 \end{bmatrix} \cdot \begin{bmatrix} 1 & \\ 2 & 1 \\ 3 & 2 \end{bmatrix} = \begin{bmatrix} 1 & \\ 4 & 1 \\ 10 & 4 \\ 16 & 7 \end{bmatrix}$$

Claim 4 is the basis for the $(O(\log^2 n),\ O(n^3))$ (t, p)-bounds on the computation of the coefficients of the characteristic polynomial. Without the claim, these bounds must be replaced by $(O(\log^2 n),\ O(n^{\alpha+1}))$. These bounds are, however, still sufficient to keep the overall bounds $(O(\log^2 n),\ O(n^{\alpha+1+\epsilon}))$ of the determinant algorithm.

For more restricted domains, such as fields of characteristic zero or integer numbers, the processor bound of a determinant algorithm can be improved [Csansky (1976), Preparata, Sarwate (1978)]. The best bound is achieved by a new iteration method of Pan and Reif (1985), which requires only $O(n^{\alpha})$ processors to solve a related problem of matrix inversion in $O(\log^2 n)$ time.

In the following, we focus on the polynomial GCD algorithm, which is basic for the whole package of symbolic manipulation algorithms. Borodin, Hopcroft, von zur Gathen (1984) suggest a matrix approach to the polynomial GCD problem. Let n, m be the degrees of the polynomials. Let A_i be an $(n+m-2i)\times(n+m-2i)$ (asymmetric) matrix of a system of linear equations S_i that correspond to the polynomial equation $p = fa+gb$ with p of degree i. The algorithm then reads:

1. Compute in parallel $\det A_i$, $i \in [1,n]$;
2. Set $k = \min\{i \mid \det A_i \neq 0\}$;
3. Compute a solution (f, g) of S_k;
4. Compute $\gcd(a, b) = fa+gb$;

We reduce the size of the matrices to $(n-i)\times(n-i)$ and show that the reduced matrices are upper left principal minors of a symmetric $n \times n$ matrix. We use a modification of the Berkowitz (1984) parallelization of the Samuelson method [Samuelson (1942)] to compute characteristic polynomials (and determinants) for the system of upper left principal minors in $O(\log^2 n)$ time with

$O(n^{\alpha+1+\epsilon})$ processors. The characteristic polynomial of the largest nonsingular (upper left principal) minor is then used to invert the minor. This allows us to decrease the processor bound for the GCD algorithm by a factor of n.

The full strength of the general parallel machine models that are suggested for the algorithm is not needed. We develop a CUBE-feasible GCD algorithm that uses a simple matrix multiplication technique with $\alpha = 3$ and runs on a CUBE or equivalent (PERFECT SHUFFLE, CCC) computer with $n^{\alpha+1+\epsilon}$ processors in $O(\log^2 n)$ time.

Section 6.2. gives a new matrix formula for the GCD of two polynomials. Section 6.3. gives a matrix formula for the coefficients of the characteristic polynomials of a matrix and its upper left principal minors, which is similar to the formulas used by Berkowitz (1984). Note: Berkowitz (1984) uses incorrect indices and sizes of matrices; there is, however, no effect on the resulting bounds in the big-oh notation. Section 6.4. outlines the GCD algorithm. The exact definition is left to the next chapter, where an axiomatic system of verification is developed for the CUBE-feasible algorithms and used to verify the program.

6.2. A matrix formula for the GCD

In this chapter a matrix formula for the GCD of two polynomials is derived. The theorem 6.2.1 is based on the claim of lemma 6.2.3, which is a modification of two known properties of polynomials, stated in lemma 6.2.1 and 6.2.2. Notation: F is an arbitrary field and $F[x]$ the ring of polynomials over F. $\delta(p)$ is the degree of a polynomial p.

Definition 6.2.1. Let $a, h \in F[x]$. Then h *divides* a iff

$$(\exists a' \in F[x]) \ (a = a'h).$$

Definition 6.2.2. Let $a, b, h \in F[x]$, $a \neq 0, b \neq 0$. Then $h = \gcd(a, b)$, i.e. h is the *greatest common divisor* (GCD) of a, b, iff

h divides a, h divides b, h is monic and

$(\forall \bar{h} \in F[x]) \ ((\bar{h}$ divides a and $\bar{h}$ divides $b) \Rightarrow \bar{h}$ divides $h)$.

Note: A simple argument can show that the GCD is unique.

Lemma 6.2.1.

$$(\forall a, b \in F[x], a \neq 0, b \neq 0) (\exists f, g \in F[x]) (\gcd(a, b) = fa + gb)$$

Proof. Let $R = \{ p \in F[x], p \neq 0 \mid (\exists f, g \in F[x]) (p = fa + gb) \}$.

Note that R is non-empty (at least $a \in R$ and $b \in R$) and partially ordered by δ. Let h be a minimum of R with ordering δ. Since any nonzero scalar multiple of an element of R is also in R, h can be chosen monic. We will prove that $h = \gcd(a, b)$:

Let $a', a'', f, g \in F[x]$, such that $h = fa + gb$ and $a = a'h + a''$, where either $\delta(a'') < \delta(h)$ or $a'' = 0$. Since

$$a'' = a - a'h = (1 - a'f)a + (-g)b \in R \cup \{0\}$$

and $(\forall p \in R) \ (\delta(p) \geq \delta(h))$,

$a'' = 0$ and h divides a. Similarly, h divides b, i.e. h is a monic common divisor of a, b. Any other common divisor $\bar{h}$ of a, b, such that $a = \bar{a}\,\bar{h}$ and $b = \bar{b}\,\bar{h}$, divides h:

$$h = fa + gb = (\bar{f}\bar{a} + \bar{g}\,\bar{b})\bar{h} .$$

Therefore, h is the greatest common divisor of a , b . □

Lemma 6.2.2.

$(\forall\ a, b \in F[x],$ s.t. $a \neq 0$, $b \neq 0$, $\delta(a) \geq \delta(b)$, a is not a scalar multiple of b) $(\exists\ f, g \in F[x])$

$$(\gcd(a, b) = fa + gb \text{ and } \delta(g) < \delta(a) - \delta(\gcd(a, b))).$$

Note: When a is a scalar multiple of b , $0 = \delta(g) = \delta(a) - \delta(\gcd(a, b))$ or $g = 0$.

Proof. Let $h = \gcd(a, b)$, $a = a'h$, $b = b'h$. By lemma 1, $\exists\ \bar{f}, \bar{g} \in F[x]$, $h = \bar{f}a + \bar{g}\ b$. It is easy to see that $\bar{g}$ is not divisible by a': Suppose $\bar{g} = g'a'$ for some $g' \in F[x]$. Then $h = (\bar{f} + g'b')a$, i.e a divides h . Since h divides b and $\delta(b) \leq \delta(a)$, a must be a scalar multiple of b . This contradicts the lemma's assumption. Thus $\bar{g}$ is not divisible by a', i.e.

$$(\exists\ g \in F[x],\ g \neq 0)\ (\ \bar{g} = g'a' + g \text{ and } \delta(g) < \delta(a') = \delta(a) - \delta(h)).$$

Then

$$h = \bar{f}a'h + g'a'b'h + gb'h = fa + gb ,$$

where $f = \bar{f} + g'b'$. Lemma 6.2.2 is proved. □

Lemma 6.2.3. Let $a, b \in F[x]$, $a \neq 0$, $b \neq 0$, $\delta(a) \geq \delta(b)$ and a is not a scalar multiple of b . Let

$$R_i = \{\ p \in F[x],\ p \text{ monic } \mid \delta(p) = i \text{ and }$$
$$(\exists\ f, g \in F[x])\ (\ p = fa + gb \text{ and } \delta(g) < \delta(a) - i\ )\}$$

for $i \geq 0$. Then

$$\{\gcd(a,b)\} = R_{\min\{ i \mid R_i \neq \emptyset\}}.$$

Proof. Let

$$R'_i = \{ p \in F[x] \mid \delta(p) = i \text{ and } (\exists f, g \in F[x])\ (p = fa + gb)\}.$$

Note that $R'_i \supseteq R_i$ and $R = \bigcup_{\geq i} R'_i$, where R is the set defined in the proof of lemma 6.2.1. From the proof of lemma 6.2.1,

$$\gcd(a,b) \in R'_{\min\{ i \mid R'_i \neq \emptyset\}}.$$

By lemma 6.2.2,

$$\gcd(a,b) \in R_{\min\{ i \mid R'_i \neq \emptyset\}} \subseteq R'_{\min\{ i \mid R'_i \neq \emptyset\}}.$$

Since the GCD is unique, $\{\gcd(a,b)\} = R_{\min\{ i \mid R'_i \neq \emptyset\}}$.

Since $R_i \subseteq R'_i$, $\min\{ i \mid R'_i \neq \emptyset\} \leq \min\{ i \mid R_i \neq \emptyset\}$.

Since $\emptyset \neq R_{\min\{ i \mid R'_i \neq \emptyset\}}$,

$$\min\{ i \mid R'_i \neq \emptyset\} \geq \min\{ i \mid R_i \neq \emptyset\}.$$

Therefore

$$\{\gcd(a,b)\} = R_{\min\{ i \mid R'_i \neq \emptyset\}} = R_{\min\{ i \mid R_i \neq \emptyset\}}.$$

Lemma 3 is proved. □

Note. The polynomials f and g are unique. The argument goes as follows:

Let $h = \gcd(a,b) = fa + gb = \bar{f}a + \bar{g}\,b$, $a = a'h$ and $b = b'h$ for $a, a', b, b', f, \bar{f}, g, \bar{g}, h \in F[x]$. Then $(f - \bar{f})a' = (g - \bar{g})b'$ and $\gcd(a', b') = 1$. This is possible only in two ways. Either $f - \bar{f} = g - \bar{g} = 0$, or b' divides $f - \bar{f}$ and a' divides $g - \bar{g}$. The latter alternative is excluded by

$$\delta(g'-g) \leq \max\{\delta(g),\delta(g')\} < \delta(a') = \delta(a)-\delta(h).$$

Hence, the polynomials $f = \overline{f}$ and $g = \overline{g}$ are unique.

Notation. Let $a_k = b_k = 0$ for $k<0$. Let $i = 0,1,\ldots,n-1$. Then X_i, Y_i are upper triangular $(n-i)\times(n-i)$ Toeplitz matrices

$$X_i = \begin{bmatrix} a_n & a_{n-1} & \cdots & a_{i+1} \\ & a_n & \cdots & a_{i+2} \\ & & \cdots & \\ & & & a_n \end{bmatrix}, \qquad Y_i = \begin{bmatrix} b_n & b_{n-1} & \cdots & b_{i+1} \\ & b_n & \cdots & b_{i+2} \\ & & \cdots & \\ & & & b_n \end{bmatrix},$$

U_i, V_i are $(n-i)\times(n-i)$ Hankel matrices

$$U_i = \begin{bmatrix} a_{n-1} & a_{n-2} & \cdots & a_i \\ a_{n-2} & a_{n-3} & \cdots & a_{i-1} \\ & \cdots & & \\ a_i & a_{i-1} & \cdots & a_{2i+1-n} \end{bmatrix}, \qquad V_i = \begin{bmatrix} b_{n-1} & b_{n-2} & \cdots & b_i \\ b_{n-2} & b_{n-3} & \cdots & b_{i-1} \\ & \cdots & & \\ b_i & b_{i-1} & \cdots & b_{2i+1-n} \end{bmatrix},$$

S_i, T_i are $i\times(n-i)$ Hankel matrices

$$S_i = \begin{bmatrix} a_{i-1} & a_{i-2} & \cdots & a_{2i-n} \\ a_{i-2} & a_{i-3} & \cdots & a_{2i-n-1} \\ & \cdots & & \\ a_0 & 0 & \cdots & 0 \end{bmatrix}, \qquad T_i = \begin{bmatrix} b_{i-1} & b_{i-2} & \cdots & b_{2i-n} \\ b_{i-2} & b_{i-3} & \cdots & b_{2i-n-1} \\ & \cdots & & \\ b_0 & 0 & \cdots & 0 \end{bmatrix}.$$

Note that $V_0X_0 = \begin{bmatrix} V_i & \cdot \\ T_i & \cdot \end{bmatrix} \cdot \begin{bmatrix} X_i & \cdot \\ O & \cdot \end{bmatrix} = \begin{bmatrix} V_iX_i & \cdot \\ T_iX_i & \cdot \end{bmatrix}$; similarly for U_0Y_0. Thus, $(V_iX_i - U_iY_i) =$

[the first $n-i$ rows and columns of $(V_0X_0 - U_0Y_0)$],

and $(T_iX_i - S_iY_i) =$

[the last i rows and the first $n-i$ columns of $(V_0X_0 - U_0Y_0)$].

Theorem 6.2.1. Let $a, b \in F[x]$, where $a(\lambda) = \sum_{i\geq 0} a_i\lambda^i$, $b(\lambda) = \sum_{i\geq 0} b_i\lambda^i$, $\delta(a) = n \geq m = \delta(b)$, and a is not a scalar multiple of b . Then

$$\gcd(a,b)(\lambda) \;=\; \lambda^k + [\,\lambda^{k-1}, \lambda^{k-2}, \ldots, 1\,]$$

$$\cdot\,(T_k X_k - S_k Y_k) \cdot (V_k X_k - U_k Y_k)^{-1} \cdot [\,0, \ldots, 0, 1\,]^T ,$$

where $k = \min\{\, i \mid \det(V_i X_i - U_i Y_i) \neq 0 \,\}$.

Proof. Let $p, f, g \in F[x]$, $p(\lambda) = \sum_{i \geq 0} p_i \lambda^i$, $f(\lambda) = \sum_{i \geq 0} f_i \lambda^i$, $g(\lambda) = \sum_{i \geq 0} g_i \lambda^i$. Let R_i, $i = 0, 1, \ldots, n-1$, be the set defined in lemma 6.2.3. We will examine the sufficient and necessary conditions for $p \in R_i$. Clearly, polynomials p, f, g satisfy the conditions:

$$p \text{ is monic} \quad \text{and} \quad \delta(p) = i \quad \text{and} \quad \delta(g) < n-i \quad \text{and} \quad p = fa + gb$$

iff their coefficients satisfy the following system of linear equations:

$$\begin{bmatrix} & & a_n & & & b_m \\ & \cdots & & & & \\ a_n & \cdots & a_{m-n+i+1} & & \cdots & \\ a_{n-1} & \cdots & a_{m-n+i} & & & \\ & & & b_m & \cdots & b_{n-m+i+1} \\ & \cdots & & b_{m-1} & \cdots & b_{n-m+i} \\ & & & & \cdots & \\ a_i & \cdots & a_{2i-m+1} & b_i & \cdots & b_{2i-n+1} \\ a_{i-1} & \cdots & a_{2i-m} & b_{i-1} & \cdots & b_{2i-n} \\ & \cdots & & & \cdots & \\ a_0 & \cdots & a_{i-m+1} & b_0 & \cdots & b_{i-n+1} \end{bmatrix} \cdot \begin{bmatrix} f_0 \\ \cdots \\ f_{m-i-1} \\ g_0 \\ \cdots \\ g_{n-i-1} \end{bmatrix} = \begin{bmatrix} 0 \\ \cdots \\ 0 \\ 0 \\ \\ \cdots \\ \\ 1 \\ p_{i-1} \\ \cdots \\ p_0 \end{bmatrix}$$

Since $a_n \neq 0$ and $b_l = 0$ for $l > m$, the system above is equivalent to the system

$$\begin{bmatrix} & & a_n & & & b_n \\ & \cdots & & & \cdots & \\ a_n & \cdots & a_{i+1} & b_n & \cdots & b_{i+1} \\ a_{n-1} & \cdots & a_i & b_{n-1} & \cdots & b_i \\ & \cdots & & & \cdots & \\ a_i & \cdots & a_{2i-n+1} & b_i & \cdots & b_{2i-n+1} \\ a_{i-1} & \cdots & a_{2i-n} & b_{i-1} & \cdots & b_{2i-n} \\ & \cdots & & & \cdots & \\ a_0 & \cdots & a_{i-n+1} & b_0 & \cdots & b_{i-n+1} \end{bmatrix} \cdot \begin{bmatrix} f_0 \\ \cdots \\ f_{n-i-1} \\ g_0 \\ \cdots \\ g_{n-i-1} \end{bmatrix} = \begin{bmatrix} 0 \\ \cdots \\ 0 \\ 0 \\ \cdots \\ 1 \\ p_{i-1} \\ \cdots \\ p_0 \end{bmatrix}$$

which after reordering the first $n-i$ rows becomes

$$\begin{bmatrix} X_i & Y_i \\ U_i & V_i \\ S_i & T_i \end{bmatrix} \cdot \begin{bmatrix} F_i \\ G_i \end{bmatrix} = \begin{bmatrix} O_i \\ E_i \\ P_i \end{bmatrix} ,$$

where

$$F_i = [f_0, \ldots, f_{n-i-1}]^T ,$$

$$G_i = [g_0, \ldots, g_{n-i-1}]^T ,$$

$$E_i = [0, \ldots, 1]^T ,$$

$$P_i = [p_{i-1}, \ldots, p_0]^T ,$$

$$O_i = [0, \ldots, 0]^T .$$

F_i, G_i, E_i and O_i are $(n-i)$-vectors (or $(n-i)\times 1$ matrices); P_i is an i-vector.

We can conclude that

$$(\exists f, g \in F[x]) \ (p = fa + gb \in R_i) \text{ iff } (\exists F_i, G_i)$$

$$\left(\begin{bmatrix} X_i & Y_i \\ U_i & V_i \end{bmatrix} \cdot \begin{bmatrix} F_i \\ G_i \end{bmatrix} = \begin{bmatrix} O_i \\ E_i \end{bmatrix} \text{ and } [S_i \ T_i] \cdot \begin{bmatrix} F_i \\ G_i \end{bmatrix} = P_i \right).$$

By lemma 6.2.3 and the note, $p = \gcd(a,b)$ iff

$$(\exists \text{ unique } f, g \in F[x])$$

$$(p = fa + gb \in R_k, \quad k = \min\{ i \mid R_i \neq \emptyset \}),$$

i.e. $p = \gcd(a,b)$ iff

$$P_k = [S_k \;\; T_k] \cdot \begin{bmatrix} X_k & Y_k \\ U_k & V_k \end{bmatrix}^{-1} \cdot \begin{bmatrix} O_k \\ E_k \end{bmatrix},$$

$$k = \min\{ i \mid \det \begin{bmatrix} X_i & Y_i \\ U_i & V_i \end{bmatrix} \neq 0 \}.$$

Note that

1. The inverse of of an invertible, square, upper triangular Toeplitz matrix is invertible, square, upper triangular and Toeplitz.

2. The product of two square, upper triangular Toeplitz matrices is square, upper triangular and Toeplitz.

3. Matrix multiplication is commutative for square, upper triangular Toeplitz matrices.

Since $a_n \neq 0$, X_k is invertible. Let I_k be the $(n-k)\times(n-k)$ identity matrix. Then

$$\begin{bmatrix} X_k & Y_k \\ U_k & V_k \end{bmatrix}^{-1} \cdot \begin{bmatrix} O_k \\ E_k \end{bmatrix}$$

$$= \left(\begin{bmatrix} I_k & \\ -U_k X_k^{-1} & I_k \end{bmatrix}^{-1} \cdot \begin{bmatrix} I_k & \\ -U_k X_k^{-1} & I_k \end{bmatrix} \cdot \begin{bmatrix} X_k & Y_k \\ U_k & V_k \end{bmatrix} \right.$$

$$\left. \cdot \begin{bmatrix} X_k^{-1} & -Y_k \\ & X_k \end{bmatrix} \cdot \begin{bmatrix} X_k^{-1} & -Y_k \\ & X_k \end{bmatrix}^{-1} \right)^{-1} \cdot \begin{bmatrix} O_k \\ E_k \end{bmatrix}$$

$$= \left(\begin{bmatrix} I_k & \\ -U_k X_k^{-1} & I_k \end{bmatrix}^{-1} \cdot \begin{bmatrix} I_k & \\ & V_k X_k - U_k Y_k \end{bmatrix} \cdot \begin{bmatrix} X_k^{-1} & -Y_k \\ & X_k \end{bmatrix}^{-1} \right)^{-1} \cdot \begin{bmatrix} O_k \\ E_k \end{bmatrix}$$

$$= \begin{bmatrix} X_k^{-1} & -Y_k \\ & X_k \end{bmatrix} \cdot \begin{bmatrix} I_k & \\ & V_k X_k - U_k Y_k \end{bmatrix}^{-1} \cdot \begin{bmatrix} I_k & \\ -U_k X_k^{-1} & I_k \end{bmatrix} \cdot \begin{bmatrix} O_k \\ E_k \end{bmatrix}$$

$$= \begin{bmatrix} X_k^{-1} & -Y_k \\ & X_k \end{bmatrix} \cdot \begin{bmatrix} I_k & \\ & (V_k X_k - U_k Y_k)^{-1} \end{bmatrix} \cdot \begin{bmatrix} O_k \\ E_k \end{bmatrix}$$

$$= \begin{bmatrix} -Y_k \\ X_k \end{bmatrix} \cdot (V_k X_k - U_k Y_k)^{-1} \cdot E_k \ ,$$

Since .

$$\det \begin{bmatrix} X_i & Y_i \\ U_i & V_i \end{bmatrix}$$

$$= \det \begin{bmatrix} I_i & \\ U_i X_i^{-1} & I_i \end{bmatrix} \cdot \det \begin{bmatrix} I_i & \\ & V_i X_i - U_i Y_i \end{bmatrix} \cdot \det \begin{bmatrix} X_i & Y_i \\ & X_i^{-1} \end{bmatrix}$$

$$= \det(V_i X_i - U_i Y_i) \ \text{and} \ [S_k \ \ T_k] \cdot [-Y_k \ \ X_k]^T = T_k X_k - S_k Y_k \ ,$$

$p = \gcd(a, b)$ iff

$$P_k = (T_k X_k - S_k Y_k) \cdot (V_k X_k - U_k Y_k)^{-1} \cdot E_k \ ,$$

$$k = \min\{ i \mid \det(V_i X_i - U_i Y_i) \neq 0 \}.$$

Since $p(\lambda) = \lambda^k + [\lambda^{k-1}, \lambda^{k-2}, \ldots, 1] \cdot P_k$, $k = \delta(p)$,

$$\gcd(a, b)(\lambda) = \lambda^k + [\lambda^{k-1}, \lambda^{k-2}, \ldots, 1]$$

$$\cdot (T_k X_k - S_k Y_k) \cdot (V_k X_k - U_k Y_k)^{-1} \cdot E_k \ ,$$

$$k = \min\{ i \mid \det(V_i X_i - U_i Y_i) \neq 0 \}.$$

The theorem is proved. □

Lemma 6.2.4. $(V_0 X_0 - U_0 Y_0)$ is symmetric.

Note: Since all upper left principal minors of a symmetric matrix are symmetric, matrices $(V_i X_i - U_i Y_i)$, $i \in [0, n-1]$ are symmetric.

Proof. We will omit the index 0 of the matrices X_0, Y_0, U_0, V_0. M_{ij} is the i, j-th element of a matrix M, $i, j \in [0, n-1]$. Let $\sum^* \equiv \sum\limits_{x+y=2n-1-i-j}$. Since

$$U_{ik} = \begin{cases} a_{n-1-i-k} & , \ k \le n-1-i \\ 0 & , \ \text{else} \end{cases} \quad \text{and} \quad Y_{kj} = \begin{cases} b_{n+k-j} & , \ k \le j \\ 0 & , \ \text{else} \end{cases} ,$$

the i, j-th element of matrix $(U \cdot Y)$ is

$$(UY)_{ij} = \sum_{k=1}^{n} U_{ik} Y_{kj} = \sum_{k=1}^{\min(n-1-i,\, j)} a_{n-1-i-k}\, b_{n+k-j}$$

$$= \sum_{\substack{x+y=2n-1-i-j \\ 0 \le x \le n-i-1 \\ n-1-j \le y \le n}} a_x b_y = \sum\nolimits^*_{\substack{0 \le x \le n-i-1 \\ n-1-j \le y \le n}} a_x b_y .$$

Similarly,

$$(VX)_{ij} = \sum\nolimits^*_{\substack{n-1-j \le x \le n \\ 0 \le y \le n-i-1}} a_x b_y .$$

Then

$$(VX - UY)_{ij} - (VX - UY)_{ji}$$

$$= \Big(\sum\nolimits^*_{\substack{n-1-j \le x \le n \\ 0 \le y \le n-i-1}} a_x b_y - \sum\nolimits^*_{\substack{0 \le x \le n-i-1 \\ n-1-j \le y \le n}} a_x b_y \Big)$$

$$- \Big(\sum\nolimits^*_{\substack{n-1-i \le x \le n \\ 0 \le y \le n-j-1}} a_x b_y - \sum\nolimits^*_{\substack{0 \le x \le n-j-1 \\ n-1-i \le y \le n}} a_x b_y \Big)$$

$$= \Big(\sum\nolimits^*_{\substack{n-1-j \le x \le n \\ 0 \le y \le n-i-1}} a_x b_y + \sum\nolimits^*_{\substack{0 \le x \le n-j-1 \\ n-1-i \le y \le n}} a_x b_y \Big)$$

$$- \Big(\sum\nolimits^*_{\substack{n-1-i \le x \le n \\ 0 \le y \le n-j-1}} a_x b_y + \sum\nolimits^*_{\substack{0 \le x \le n-i-1 \\ n-1-j \le y \le n}} a_x b_y \Big)$$

$$= \sum\nolimits^*_{\substack{0 \le x \le n \\ 0 \le y \le n}} a_x b_y - \sum\nolimits^*_{\substack{0 \le x \le n \\ 0 \le y \le n}} a_x b_y = 0 ,$$

i.e. $(VX-UY)_{ij} = (VX-UY)_{ji}$. □

6.3. Characteristic polynomials of upper left principal minors

In this chapter a matrix formula for all upper left principal minors is given. The main theorem is based on lemmas 6.3.1-3, which correspond, respectively, to claim 1, claim 2 and a part of theorem 5 of Berkowitz (1984) parallelization scheme of the Samuelson method [Samuelson (1942)] for determining the coefficients of characteristic polynomials. The notation and the structure of the matrices has been, however, changed to support our needs.

Notation. Let M be an $(l+1)\times(l+1)$ matrix and N its $l\times l$ upper left principal minor, $l>1$.

$$M=\begin{bmatrix} N & S \\ R & a \end{bmatrix}, \quad M\in F^{l\times l}, \quad R\in F^{1\times l}, \quad S\in F^{l\times 1}, \quad a\in F .$$

Let p, q be their characteristic polynomials, respectively.

$$p(\lambda) = \sum_{i\geq 0} p_i\lambda^i = \det(M-\lambda I), \qquad q(\lambda) = \sum_{i\geq 0} q_i\lambda^i = \det(N-\lambda I).$$

Lemma 6.3.1. $\det(M) = a\cdot\det(N) + R\cdot\operatorname{adj}(N)\cdot S$.

Note: The adjoint of a one-element matrix is the identity matrix.

Proof. Expand $\det(M)$ by the cofactor expansion on the last row and then on the last column. □

Lemma 6.3.2. $\operatorname{adj}(N-\lambda I) = -\sum_{i=0}^{l-1}\sum_{j=0}^{l-i-1} N^j q_{i+1+j}\lambda^i$.

Proof. Multiply both sides of the equation by $(N-\lambda I)$. The left-hand side is then equal to $q(\lambda)\cdot I$. The right-hand side is

$$\left(-\sum_{i=0}^{l-1} \sum_{j=0}^{l-i-1} N^j q_{i+1+j} \lambda^i \right) \cdot (-\lambda I + N)$$

$$= \sum_{i=0}^{l-1} \sum_{j=0}^{l-i-1} N^j q_{i+1+j} \lambda^{i+1} - \sum_{i=0}^{l-1} \sum_{j=0}^{l-i-1} N^{j+1} q_{i+1+j} \lambda^i$$

$$= \sum_{i=1}^{l} \sum_{j=0}^{l-i} N^j q_{i+j} \lambda^i - \sum_{i=0}^{l-1} \sum_{j=1}^{l-i} N^j q_{i+j} \lambda^i$$

$$= \sum_{i=l}^{l} \sum_{j=0}^{l-i} N^j q_{i+j} \lambda^i + \sum_{i=1}^{l-1} \sum_{j=0}^{0} N^j q_{i+j} \lambda^i$$

$$+ \sum_{i=1}^{l-1} \sum_{j=1}^{l-i} (N^j q_{i+j} \lambda^i - N^j q_{i+j} \lambda^i)$$

$$= I\, q_l \lambda^l + \sum_{i=1}^{l-i} I\, q_i \lambda^i - \sum_{j=1}^{l} N^j q_j$$

$$= q(\lambda) \cdot I - \sum_{j=0}^{l} N^j q_j = q(\lambda) \cdot I .$$

(Note that $\sum_{j=0}^{l} N^j q_j = 0$ by the Caley-Hamilton theorem.) The lemma is proved for $q(\lambda) \neq 0$. Since the matrix coefficients of both sides of the equation are polynomials in λ, the lemma holds for any λ. □

Lemma 6.3.3. $[p_0, \ldots, p_{l+1}]^T = C \cdot [q_0, \ldots, q_l]^T$,

where

$$C_{ij} = \begin{cases} 0 & , \ j-i < -1 \\ -1 & , \ j-i = -1 \\ a & , \ j-i = 0 \\ -RN^{j-i-1}S & , \ j-i > 0 \end{cases}$$

i.e.

$$\begin{bmatrix} p_0 \\ \\ \cdots \\ \\ p_{l+1} \end{bmatrix} = \begin{bmatrix} a & -RS & \cdots & -RN^{l-1} \\ -1 & a & \cdots & -RN^{l-2} \\ & & \cdots & \\ 0 & 0 & \cdots & a \\ 0 & 0 & \cdots & -1 \end{bmatrix} \cdot \begin{bmatrix} q_0 \\ \\ \cdots \\ \\ q_l \end{bmatrix}$$

Proof. For $l = 0$ the claim obviously holds. Let $l > 0$. By lemma 1 and lemma 2,

$$p(\lambda) = [1, \lambda, \ldots, \lambda^{l+1}] \cdot [p_0, \ldots, p_{l+1}]^T = \det(M - \lambda I)$$
$$= (a - \lambda) \cdot \det(N - \lambda I) + R \cdot \operatorname{adj}(N - \lambda I) \cdot S$$
$$= (a - \lambda) \cdot \sum_{i=0}^{l} q_i \lambda^i + R \cdot \Big(\sum_{i=0}^{l-1} \sum_{j=0}^{l-i-1} N^j q_{i+1+j} \lambda^i \Big) \cdot S$$
$$= \sum_{i \geq 0} \Big(-q_{i-1} + a q_i - \sum_{j=i+1}^{l} R N^{j-i-1} S q_j \Big) \lambda^i$$
$$= \sum_{i \geq 0} \sum_{j=0}^{l} C_{ij} \, q_j \, \lambda^i = \sum_{i=0}^{l+1} \sum_{j=0}^{l} C_{ij} \, q_j \, \lambda^i$$
$$= [1, \lambda, \ldots, \lambda^{l+1}] \cdot C \cdot [q_0, \ldots, q_l]^T .$$

The lemma is proved. □

Notation. Let $A = [A_{ij}]_{\substack{0 \leq i < n \\ 0 \leq j < n}}$ be an $n \times n$ matrix. Let

$${}_lM = [A_{ij}]_{\substack{0 \leq i < l \\ 0 \leq j < l}}, \quad {}_lR = [A_{ij}]_{\substack{i = l \\ 0 \leq j < l}}, \quad {}_lS = [A_{ij}]_{\substack{0 \leq i < l \\ j = l}}$$ and

${}_l\alpha = A_{ll}$, $l \in [0, n]$, i.e.

$${}_{l+1}M = \begin{bmatrix} {}_lM & {}_lS \\ {}_lR & {}_l\alpha \end{bmatrix}, \quad {}_nM = A .$$

Let ${}_l\overline{C}$ and ${}_lC$, $l \in [0, n-1]$, be, respectively, $(l+2) \times (l+1)$ and $n \times n$ matrices:

$${}_l\overline{C}_{ij} = \begin{cases} 0 & , \; j - i < -1 \\ -1 & , \; j - i = -1 \\ {}_l\alpha & , \; j - i = 0 \\ -{}_lR \, {}_lM^{\,j-i-1} \, {}_lS & , \; j - i > 0 \end{cases} ,$$

$$ {}_l C_{ij} = \begin{cases} {}_l \bar{C}_{ij} & , \ i \leq l+1 \text{ and } j \leq l \\ 0 & , \text{ else} \end{cases} . $$

Let ${}_l c_i$, $l \in [0, n-1]$, $i \in [0, l+1]$, be the i-th coefficient of the characteristic polynomial of matrix ${}_{l+1}M$, i.e.

$$ \sum_{i=0}^{l+1} {}_l c_i \lambda^i = \det({}_{l+1}M - \lambda I) , $$

where I is the $(l+1)\times(l+1)$ identity matrix. The operator $\prod$ denotes matrix multiplication "from the left":

$$ \prod_{i=a}^{b} N_i \equiv N_b \cdot N_{b-1} \cdot \ \cdots \ \cdot N_a $$

Theorem 6.3.1.

$$ (\forall\, l \in [0, n-1]) \ (\ [{}_l c_0, \ldots, {}_l c_{l+1}]^T = \prod_{m=0}^{l} {}_m \bar{C} \ ) . $$

Proof. The lemma holds for $l = 0$:

$$ [{}_0 c_0, {}_0 c_1]^T = \begin{bmatrix} {}_0\alpha \\ -1 \end{bmatrix} = {}_0 \bar{C} = \prod_{m=0}^{0} {}_m \bar{C} \ . $$

By lemma 3,

$$ [{}_l c_0, \ldots, {}_l c_{l+1}]^T = {}_l \bar{C} \cdot [{}_{l-1} c_0, \ldots, {}_{l-1} c_l]^T $$

for $l > 0$. The claim follows by mathematical induction. □

Lemma 6.3.4. $(\forall\, i, j, l \in [0, n-1])$

$$ (\prod_{m=0}^{l} {}_m C)_{ij} = \begin{cases} {}_l c_j & , \ i = 0 \leq j \leq l+1 \leq n \text{ and } j < n \\ 0 & , \text{ else} \end{cases} $$

Proof. It is easy to see that

$$\prod_{m=0}^{l} {}_mC = \begin{bmatrix} \prod_{m=0}^{l} {}_m\bar{C} & O \\ O & O \end{bmatrix} \quad \text{for} \quad l = 0, 1, \ldots, n-2 ,$$

where O is the zero matrix of appropriate size. The claim follows from the theorem applied to the first column of the matrix $\prod_{m=0}^{l} {}_mC$. When $l = n-1$, the last element (row) of $\prod_{m=0}^{l} {}_m\bar{C}$ is not used. □

6.4. GCD algorithm

We assume that $\delta(a) = n = 2^q \geq \delta(b)$. This restriction is not critical: If $\delta(a)$ is not a power of 2, both polynomials a , b can be multiplied by $x^{2^q - \delta(a)}$, where $q = \lceil \log \delta(a) \rceil$. Then the resulting gcd is divided by the same amount. This can be easily achieved by a simple shift in the arrays of coefficients.

The matrix GCD formula of theorem 6.2.1 implies the following algorithm for the $\gcd(a, b)$:

Algorithm *GCD* 1:

1. Construct $V_0X_0 - U_0Y_0$.
2. Compute $\det(V_iX_i - U_iY_i)$, $i \in [0, n-1]$.
3. Find $k = \min\{ i \mid \det(V_iX_i - U_iY_i) \neq 0 \}$.
4. Compute $(V_kX_k - U_kY_k)^{-1}$.
5. Compute $[0, \ldots, 0, 1, p_{k-1}, \ldots, p_0]^T =$
   [the first $n-k$ columns of $(V_0X_0 - U_0Y_0)$] $\cdot$
   [the last column of $(V_kX_k - U_kY_k)^{-1}$] .

Note that the algorithm is similar to that of Borodin-von zur Gathen-Hopcroft (1984). However, the structure of the matrices is different. The last step of the algorithm yields the coefficients of the $\gcd(a,b)$:

$$\gcd(a,b)(\lambda) = [\lambda^{n-1}, \lambda^{n-2}, \ldots, 1] \cdot [0, \ldots, 0, 1, p_{k-1}, \ldots, p_0]^T .$$

Note: If required, the polynomials f and g, $\gcd(a,b) = fa + gb$, can be computed as

$$f(\lambda) = [\lambda^{n-k}, \lambda^{n-k-1}, \ldots, 1] \cdot (-Y_k) \cdot (V_k X_k - U_k Y_k)^{-1} \cdot E_k \quad \text{and}$$

$$g(\lambda) = [\lambda^{n-k}, \lambda^{n-k-1}, \ldots, 1] \cdot X_k^{-1} \cdot (V_k X_k - U_k Y_k)^{-1} \cdot E_k .$$

Let $A = {}_nM = (V_0 X_0 - U_0 Y_0)$. Using the notation from section 6.3, the algorithm reads:

Algorithm *GCD* 1:

1. Construct $A = {}_nM$.
2. Compute $\det {}_{l+1}M = {}_lc_0, \; l \in [0, n-1]$.
3. Find $l^* = \max\{ l \mid {}_lc_0 \neq 0 \}$.
4. Compute ${}_{l^*+1}M^{-1}$.
5. Compute $[0, \ldots, 0, 1, p_{n-l^*-1}, \ldots, p_0]^T = [A_{ij}]_{\substack{0 \le i < n \\ 0 \le j \le l^*}} \cdot [{}_{l^*+1}M_{ij}^{-1}]_{\substack{0 \le i \le l^* \\ j = l^*}}$.

Using the matrix formula for characteristic polynomials from theorem 6.3.1 and lemma 6.3.4, we replace step 2 by

2'.1 Compute ${}_lR \, {}_lM^k \, {}_lS$, $k \in [0, l-2]$, $l \in [0, n-1]$.

2'.2 Construct ${}_lC_{ij}$, $i \in [0, l+1]$, $j \in [0, l]$, $l \in [0, n-1]$.

2'.3 Compute ${}_lc_i = (\prod_{m=0}^{l} {}_mC)_i$, $i \in [0, l+1]$, $l \in [0, n-1]$.

Coefficients ${}_{l^*}c_i$, $i \in [0, l^*+1]$, are then used to compute ${}_{l^*+1}M^{-1}$ by the Caley-Hamilton theorem:

$$
{}_{l^*}c_0 \cdot {}_{l^*+1}M^{-1} = \sum_{m=1}^{l^*+1} {}_{l^*}c_m \cdot {}_{l^*+1}M^{m-1} = \sum_{m=0}^{l^*} {}_{l^*}c_{m+1} \cdot {}_{l^*+1}M^{m} \ .
$$

Let $N = {}_{l^*+1}M$, $W = [A_{ij}]_{\substack{l^* < i < n \\ 0 \le j \le l^*}}$. Then ($\forall\ k \ge 0$)

$$
\begin{bmatrix} N & O' \\ W & O'' \end{bmatrix}^{k+1} = \begin{bmatrix} N^{k+1} & O' \\ WN^k & O'' \end{bmatrix},
$$

where O', O'' are the zero matrices of appropriate sizes. Thus, steps 4 and 5 can be replaced by

4'. Compute $Z^{k+1} = \begin{bmatrix} N & O' \\ W & O'' \end{bmatrix}^{k+1}$, $k \in [0, l^*]$.

5'. Compute $[\,0, \ldots, 0, 1, p_{n-l^*-1}, \ldots, p_0]^T$ =

[the l^*-th column of $\sum_{m=0}^{l^*} -({}_{l^*}c_{m+1} / {}_{l^*}c_0)\, Z^{m+1}$].

Step 2'.1 seems to require $O(n^2)$ matrix multiplications, i.e. $O(n^{\alpha+2})$ processors. However, a simple divide-and-conquer technique can be used to reduce the required number of processors. The basic idea for the "conquer" step is to compute $\{{}_lM^k\, {}_lS\}_{0 \le k < x^2}$ by the block-matrix formula:

$$
[{}_lM^{ix+j}\, {}_lS]_{\substack{0 \le i < x \\ 0 < j \le x}} = [{}_lM^{ix}]_{\substack{0 \le i < x \\ 0 = k}} \cdot [{}_lM^{j}\, {}_lS]_{\substack{0 = k \\ 0 < j \le x}} \ .
$$

Each "conquer" step quadratically decreases required number of processors but increases computation time. It is easy to see that the tradeoff can be controlled by a constant $\epsilon > 0$:

$$
\frac{n^{\epsilon}}{\epsilon} \cdot \frac{TIME}{PROCESSORS} \approx \text{const.}
$$

```
$MAX$ ; { step 3 }
$INV$ ; { step 4' }
$RES$ ; { step 5' }
```

where

$$\{a_i\}_i,\ \{b_i\}_i \xrightarrow{INIT} A \longrightarrow \{{}_lM\}_l$$

$$\{{}_lM\}_l \xrightarrow{PWRA} \{{}_lM^h\}_{h,l}$$

$$\{{}_lM^{h\,2^{rc-r}}\}_{h,l} \xrightarrow{PWRB} \{{}_lM^{h\,2^{rc}}\}_{h,l}$$

$$\{{}_lM^{h\,2^{rc}}\}_{h,l},\ \{{}_lM^{k<_{rc}}\,S\}_{k,l} \xrightarrow{DAC} \{{}_lM^{k<_{rc+r}}\,S\}_{k,l}$$

$$\{{}_lM^k\ {}_lS\}_{k,l} \xrightarrow{CIJ} \{{}_lC\}_l$$

$$\{{}_lC\}_l \xrightarrow{CI} \{\prod_{m=0}^{l} {}_mC\}_l \longrightarrow \{{}_lc_k\}_{k,l}$$

$$\{{}_lc_0\}_l \xrightarrow{MAX} l^*$$

$$\{{}_lM\}_l \xrightarrow{PWRC} \{Z^k\}_k$$

$${}_nM,\ \{{}_{l^*}c_k\}_k,\ \{Z^k\}_k \xrightarrow{RES} \{p_l\}_l\,.$$

The implementation and verification of the subroutines is shown in chapter 7. It is easy to see that all subroutines run in $O(\log^2 n)$ time. Since $h \in [0, 2^r]$ and $i, j, k, l \in [0, 2^q]$, the address of a processor contains $r+i+j+k+l$ bits. Therefore, the algorithm requires $2^{4q+r} = n^{4+\epsilon}$ processors, where $r = \lceil \epsilon q \rceil$.

Chapter 7
Subroutines

7.1. Introduction

In this chapter we present CUBE-feasible version of the subroutines that were introduced in the end of section 6.4. We develop an axiomatic verification system for CUBE-feasible algorithms and apply it to our subroutines.

Suppose we have a CUBE of size $n^{4+\epsilon} = 2^{4q+r}$. The processor address consists of five fields, h, i, j, k, l , with r, q, q, q, q bits, respectively. We will omit the separating commas in the address. When not specified otherwise, indexes have their the default ranges, which are $[0, 2^r-1]$ for h , and $[0, 2^q-1]$ for i, j, k, l . All subroutines use two registers, A , B , per processor.

We extend the notion of CUBE-feasibility to algorithms decomposable into subroutines that are of type ASCEND or DESCEND on the sheaves of h, i, j, k or l , or that access no sheaves at all. Since the majority of the loops work for both ASCEND and DESCEND orders, we replace "**for** $m = 0$ **to** 2^q-1 **do**" or "**for** $m = 2^q$-1 **downto** 0 **do**" by "**for** $m \in [0, q-1]$ **do** ", whenever possible. For practical reasons, we also allow an access of a single sheaf between two for-loops. It should be clear, however, that the algorithms still can be efficiently simulated by a CCC or a PERFECT SHUFFLE.

Each subroutine is presented in three parts:

1. The algorithm, which is a sequence of steps or a loop.

2. The set of input (P), output (Q) and intermediate ($\{AS_i\}_{i \geq 0}$) assertions. Each assertion is a predicate about the contents of processor registers.

3. The proof of correctness, which is an application of the basic axioms and lemmas to the set of assertions.

7.2. The method of program verification

The standard Hoare's formalism [Hoare (1969)] is used here to verify the CUBE algorithms. The verification of a statement S has the form:

$$\{\text{ precondition }\}\ S\ \{\text{ postcondition }\}.$$

The program is correct iff

$$\{\text{ input condition }\}\ \text{program}\ \{\text{ output condition }\}.$$

To infer the program correctness, two rules of inference are used.

$$\text{Sequencing:}\quad \frac{\{P\}\ S\ \{R\},\ \{R\}\ T\ \{Q\}}{\{P\}\ S;T\ \{Q\}}\ .$$

$$\text{Looping:}\quad \frac{P \Longrightarrow R_0,\ \{R_i\}\ S_i\ \{R_{i+1}\},\ R_d \Longrightarrow Q}{\{P\}\ \textbf{for}\ i = 0\ \textbf{to}\ d\text{-}1\ \textbf{do}\ S_i\ \{Q\}}\ .$$

7.2.1. Axioms

The following two axioms suffice for our proofs.

A1. (Assignment).

$$\{\ A(p) = x(p), B(p) = y(p),\ \ldots\ \}$$

$$A(p) \leftarrow c\ (p, A(p), A(p^{(m)}), B(p), B(p^{(m)}),\ \ldots)$$

$$\{\ A(p) = c\ (p, x(p), x(p^{(m)}), y(p), y(p^{(m)}),\ \ldots)\ \}$$

Justification. The new contents of register A of $PE(p)$ is the result of the function $c(\cdot)$ applied to the previous contents of the registers of $PE(p)$ and its neighbors. □

A2. (Data transfer).

$\{ A(p) = x(p) \}$

$A(p) \leftarrow A(p^{(m)}),\ (\epsilon_m(p) = 0)$

$\{ A(p) = x(p_{\geq m+1}2^{m+1} + (p_m \oplus \overline{\epsilon}_m(p))2^m + p_{<m}) \}$.

Justification. If $\epsilon_m(p) = 0$, data are transferred along sheaf m , i.e. from $PE(p')$ to $PE(p)$, where $p' = p^{(m)} = p_{\geq m+1}2^{m+1} + (p_m \oplus \overline{\epsilon}_m(p))2^m + p_{<m}$. If $\epsilon_m(p) = 1$, data do not move, i.e. are "transferred" from $PE(p')$ to $PE(p)$, where $p' = p = p_{\geq m+1}2^{m+1} + (p_m \oplus \overline{\epsilon}_m(p))2^m + p_{<m}$. (Note: $x \oplus y = (x+y)_{<2}$ for $x, y \in [0,1]$). □

7.2.2. Lemmas

The following technical lemmas describe the results of some common assignments and loops. Lemmas 1-8 are straightforward. Lemma 9 describes a shift of adrresses by a constant d . Lemma 9 is equivalent to lemma 3, which uses a different notation that simplifies the proof. Lemmas 1-2 are used by lemma 3.

L1. Let $d \geq 0$.

$\{ A(p) = x(p) \}$

$A(p) \leftarrow A(p^{(m)}),\ (p_m = \overline{d}_{m-t})$

$$\{\ A(p)=x(p_{\geq m+1}2^{m+1}+\bar{d}_{m-t}\,2^m+p_{<m})\ \}$$

Proof. By A2 with $\epsilon_m(p) = p_m \oplus \bar{d}_{m-t}$. □

L2. Let $d \geq 0$.

$$\{\ A(p)=x(p)\ \}$$

$$\textbf{for } m \in [t, s+t-1] \textbf{ do } A(p) \leftarrow A(p^{(m)}),\ (p_m = \bar{d}_{m-t})$$

$$\{\ A(p)=x(p_{\geq s+t}\,2^{s+t}+\bar{d}_{<s}\,2^t+p_{<t})\ \}$$

Proof. From L1 by mathematical induction. □

L3. Let k be any integer. Then

$$\{\ A(p)=x(p)\ \}$$

$$A(p) \leftarrow A(p^{(m)}),\ (p_m = \bar{p}_{m+k})$$

$$\{\ A(p)=x(p_{\geq m+1}2^{m+1}+p_{m+k}\,2^m+p_{<m})\ \}$$

Proof. By A2 with $\epsilon_m(p) = p_m \oplus \bar{p}_{m+k}$. □

L4. Let $k \notin [0, s-1]$. Then

$$\{\ A(p)=x(p)\ \}$$

$$\textbf{for } m \in [t, s+t-1] \textbf{ do } A(p) \leftarrow A(p^{(m)}),\ (p_m = \bar{p}_{m+k})$$

$$\{\ A(p)=x(p_{\geq s+t}\,2^{s+t}+(p_{\geq t+k})_{<s}\,2^t+p_{<t})\ \}$$

Proof. From L3 by mathematical induction. □

L5.

$$\{\ A(p)=x(p)\ \}$$

$$A(p) \leftarrow A(p^{(m)})$$

$$\{\ A(p)=x(p_{\geq m+1}2^{m+1}+\bar{p}_m\,2^m+p_{<m})\ \}$$

Proof. By A2 with $\epsilon_m(p) = 0$. □

L6.

```
{ $A(p) = x(p)$ }
        for $m \in [t, s+t-1]$ do $A(p) \leftarrow A(p^{(m)})$
{ $A(p) = x(p_{\geq s+t}\, 2^{s+t} + (2^m - 1 - (p_{\geq t})_{<s})\, 2^t + p_{<t})$ }
```

Proof. From L5 and the equation $\bar{p}_m\, 2^m + 2^m - 1 - p_{<m} = 2^{m+1} - 1 - p_{<m+1}$ by mathematical induction. □

L7. Let $\bigcirc$ be an operation . Then

```
{ $A(p) = x(p)$ }
      $A(p) \leftarrow A(p) \bigcirc A(p^{(m)})$
{ $A(p) = x(p) \bigcirc x(p^{(m)})$ }
```

Proof. By A1 with $c\,(p, A(p), A(p^{(m)})) = A(p) \bigcirc A(p^{(m)})$. □

L8. Let $\bigcirc$ be an associative operation . Then

```
{ $A(p) = x(p)$ }
        for $m \in [t, s+t-1]$ do $A(p) \leftarrow A(p) \bigcirc A(p^{(m)})$
{ $A(p) = \bigcirc_{i \in [0, 2^s - 1]} x(p_{\geq s+t}\, 2^{s+t} + i\, 2^t + p_{<t})$ }
```

Proof. From L7 by mathematical induction. □

L9. Let $d \geq 0$.

```
{ $A(p) = x(p)$ }
        for $m = t$ to $s+t-1$ do
      $A(p) \leftarrow A(p^{(m)}), ((p_{<m})_{\geq t} < d_{<m+1-t} \leq 2^m + (p_{<m})_{\geq t})$
```

$$\{ A(p) = x(p_{\geq s+t} 2^{s+t} + ((p_{\geq t})_{<s} - d) \bmod 2^s) 2^t + p_{<t}) \}$$

Note:

$$(p_{<m})_{\geq t} < d_{<m+1-t} \leq 2^m + (p_{<m})_{\geq t}$$

is equivalent to

$$d_{m-t} = 0 \text{ and } (p_{<m})_{\geq t} < d_{<m-t} \quad \text{or} \quad d_{m-t} = 1 \text{ and } (p_{<m})_{\geq t} \geq d_{<m-t} \ ,$$

which is equivalent to

$$d_{m-t} \oplus \gamma_m = 1 \, , \; \gamma_{m-t} = \begin{cases} 1 \ , & (p_{<m})_{\geq t} < d_{<m-t} \\ 0 \ , & (p_{<m})_{\geq t} \geq d_{<m-t} \end{cases}$$

Proof. L9 is equivalent to the lemma 3 with

$$i = p_{\geq s+t} \ , \qquad j = (p_{\geq t})_{<s} \ , \qquad k = p_{<t} \ . \qquad \square$$

The only purpose of the following lemmas is to provide a proof for L9.

Lemma 1. Let $d, j, m \geq 0$. Then $(j - d) \bmod 2^m = (j_{<m} - d_{<m}) \bmod 2^m$.

Proof. $(j - d) \bmod 2^m$

$$= ((j_{\geq m} - d_{\geq m}) 2^m + j_{<m} - d_{<m}) \bmod 2^m$$

$$= (((j_{\geq m} - d_{\geq m}) 2^m) \bmod 2^m + (j_{<m} - d_{<m}) \bmod 2^m) \bmod 2^m$$

$$= (j_{<m} - d_{<m}) \bmod 2^m \qquad \square$$

Lemma 2. Let $d, j, m \geq 0$. Then

$$(j - d) \bmod 2^{m+1} = (j_m \oplus d_m \oplus \gamma_m) 2^m + (j - d) \bmod 2^m \, ,$$

$$\text{where} \quad \gamma_m = \begin{cases} 1 \ , & j_{<m} < d_{<m} \\ 0 \ , & j_{<m} \geq d_{<m} \end{cases}$$

Proof. Using lemma 1,

$$\begin{aligned}
&(j-d) \bmod 2^{m+1} \\
&= (j_{<m+1} - d_{<m+1}) \bmod 2^{m+1} \\
&= (((j_m - d_m)2^m) \bmod 2^{m+1} + (j_{<m} - d_{<m}) \bmod 2^{m+1}) \bmod 2^{m+1} \\
&= (((j_m - d_m) \bmod 2)2^m + \gamma_m 2^m + (j_{<m} - d_{<m}) \bmod 2^m) \bmod 2^{m+1} \\
&= (j_m \oplus d_m \oplus \gamma_m)2^m + (j-d) \bmod 2^m \ .
\end{aligned}$$

□

Lemma 3. Let $d, i, k \geq 0$, $j \in [0, 2^s - 1]$. Then

$\{ A(i,j,k) = x(i,j,k)) \}$

for $m = 0$ **to** $s-1$ **do** $A(i,j,k) \leftarrow A(i, j^{(m)}, k)$, $(d_m \oplus \gamma_m = 1)$

$\{ A(i,j,k) = x(i,(j-d) \bmod 2^s, k) \}$,

where $\gamma_m = \begin{cases} 1 \ , & j_{<m} < d_{<m} \\ 0 \ , & j_{<m} \geq d_{<m} \end{cases}$

Proof. For a given step, assume

$$A_{before}(i,j,k) = x(i, j_{\geq m} 2^m + (j-d) \bmod 2^m, k) \ .$$

By A2 with $\epsilon_m(p) = d_m \oplus \bar{\gamma}_m$,

$$\begin{aligned}
&A_{after}(i,j,k) \\
&= A_{before}(i, j_{\geq m+1} 2^{m+1} + (j_m \oplus d_m \oplus \gamma_m)2^m + j_{<m}, k) \\
&= x(i, j_{\geq m+1} 2^{m+1} + (j_m \oplus d_m \oplus \gamma_m)2^m + j_{<m}, k)
\end{aligned}$$

By lemma 2,

$$= x(i, j_{\geq m+1} 2^{m+1} + (j-d) \bmod 2^{m+1}, k) \ .$$

The claim follows by mathematical induction. □

7.3. Program GCD

Using intermediate assertions, program GCD from section 6.4 reads:

Algorithm 7.3.1.

```
Program $GCD$ :
$\{AS_0\}$
    $INIT$ ;
$\{AS_1\}$
    for $c=0$ to $\lceil q/r \rceil - 1$ do begin
$\{AS_{2,c}\}$
        if $c=0$ then $PWRA$ else $PWRB$ ;
$\{AS_{3,c}\}$
        $DAC$ ;
$\{AS_{4,c}\}$
    end ;
$\{AS_5\}$
    $CIJ$ ; $\{AS_6\}$ $CI$ ; $\{AS_7\}$ $MAX$ ; $\{AS_8\}$ $PWRC$ ; $\{AS_9\}$ $RES$ ;
$\{AS_{10}\}$
```

Assertions.

$$P \equiv \begin{cases} A(hijkl) = a_{k\,2^q+l} & , \ (h=i=j=0 \text{ and } 0 \leq k2^q+l \leq 2^q = n) \\ B(hijkl) = b_{k\,2^q+l} & , \ (h=i=j=0 \text{ and } 0 \leq k2^q+l \leq 2^q = n) \end{cases}$$

$$Q \equiv A(hijkl) = p_{h\,2^{4q} + i\,2^{3q} + j\,2^{2q} + k\,2^q + l}$$

Assertions $AS_0 \cdots AS_{10}$ are the preconditions (postconditions) of the procedures that follow (precede) them.

Theorem 7.3.1. $\{P\}\ GCD\ \{Q\}$.

Proof. Let "$\{INIT\}$", "$\{PWRA\}$", ... be an abbreviation for "Theorem 7.4.1", "Theorem 7.5.1", ... Then

$$P = AS_0 \overset{\{INIT\}}{\Longrightarrow} AS_1 \Longrightarrow AS_{2,0}\,;$$

$$AS_{2,0} \overset{\{PWRA\}}{\Longrightarrow} AS_{3,0} \overset{\{DAC\}}{\Longrightarrow} AS_{4,0} = AS_{2,1}\,;$$

$$(\forall\ c \in [1, \lceil q/r \rceil -1]\,)\ \ AS_{2,c} \overset{\{PWRB\}}{\Longrightarrow} AS_{3,c} \overset{\{DAC\}}{\Longrightarrow} AS_{4,c} = AS_{2,c+1}\,;$$

$$AS_{2,\lceil q/r \rceil} \Longrightarrow AS_5 \overset{\{CIJ\}}{\Longrightarrow} AS_6 \overset{\{CI\}}{\Longrightarrow} AS_7 \overset{\{MAX\}}{\Longrightarrow} AS_8 \overset{\{PWRC\}}{\Longrightarrow} AS_9 \overset{\{RES\}}{\Longrightarrow}$$

$$AS_{10} = Q\ .$$

Using the rules of inference from section 7.2.,

$$\{P\}\ GCD\ \{Q\}. \qquad \square$$

7.4. Subroutine INIT

Subroutine *INIT* computes entries of matrix $(V_0X_0 - U_0Y_0)$. The input consists of the coefficients of polynomials a, b loaded in the lowest 2^q addresses. The output consists of values $(V_0X_0 - U_0Y_0)_{ij}$. These two conditions are formalized in assertions P and Q, respectively. Let $a_x = b_x = 0$ for x out of range, i.e. $x \notin [0, 2^q]$.

Algorithm 7.4.1.

Subroutine *INIT* :

```
{$AS_0$}

    $A(hijkl) \leftarrow 0 \quad , \quad (k\,2^q + l > 2^q)$ ;
    $B(hijkl) \leftarrow 0 \quad , \quad (k\,2^q + l > 2^q)$ ;

{$AS_1$}

    for $m \in [0, q-1]$ do $\begin{cases} A(hijkl) \leftarrow A(hij^{(m)}kl) & , \ (j_m = 1)\ ; \\ B(hijkl) \leftarrow B(hij^{(m)}kl) & , \ (j_m = 1)\ ; \end{cases}$

    for $m \in [0, q-1]$ do $\begin{cases} A(hijkl) \leftarrow A(hi^{(m)}jkl) & , \ (i_m = 1)\ ; \\ B(hijkl) \leftarrow B(hi^{(m)}jkl) & , \ (i_m = 1)\ ; \end{cases}$

    for $m \in [0, r-1]$ do $\begin{cases} A(hijkl) \leftarrow A(h^{(m)}ijkl) & , \ (h_m = 1)\ ; \\ B(hijkl) \leftarrow B(h^{(m)}ijkl) & , \ (h_m = 1)\ ; \end{cases}$

{$AS_2$}

    for $m = 0$ to $q-1$ do $\begin{cases} A(hijkl) \leftarrow A(hijkl^{(m)}) & , \ (l_{<m} < i_{<m+1} \leq 2^m + l_{<m})\ ; \\ B(hijkl) \leftarrow B(hijkl^{(m)}) & , \ (l_{<m} < i_{<m+1} \leq 2^m + l_{<m})\ ; \end{cases}$

{$AS_3$}

    $A(hijkl) \leftarrow A(hijk^{(0)}l) \quad , \quad (l < i)$ ;
    $B(hijkl) \leftarrow B(hijk^{(0)}l) \quad , \quad (l \geq i)$ ;

{$AS_4$}

    for $m \in [0, q-1]$ do $\begin{cases} A(hijkl) \leftarrow A(hij^{(m)}kl) & , \ (j_m = \bar{j}_m \text{ and } k = 1)\ ; \\ B(hijkl) \leftarrow B(hij^{(m)}kl) & , \ (j_m = \bar{j}_m \text{ and } k = 0)\ ; \end{cases}$

{$AS_5$}

    for $m \in [0, q-1]$ do $\begin{cases} A(hijkl) \leftarrow A(hijkl^{(m)}) & , \ (k = 0)\ ; \\ B(hijkl) \leftarrow B(hijkl^{(m)}) & , \ (k = 1)\ ; \end{cases}$

{$AS_6$}
```

$$A(hijkl) \leftarrow \begin{cases} A(hijkl) \cdot B(hijkl) & , \ (k=0)\ ; \\ B(hijkl) \cdot A(hijkl) & , \ (k=1)\ ; \end{cases}$$

$\{AS_7\}$

$$A(hijkl) \leftarrow A(hijk^{(0)}l) - A(hijkl)\ ;$$

$\{AS_8\}$

for $m \in [0, q-1]$ **do** $A(hijkl) \leftarrow A(hijkl^{(m)}) + A(hijkl)\ ;$

$\{AS_9\}$

for $m \in [0, q-1]$ **do** $A(hijkl) \leftarrow A(hijk^{(m)}l)\ ,\ (k_m = 1)\ ;$

$\{AS_{10}\}$

Assertions.

$$P \ \equiv \ \begin{cases} A(hijkl) = a_{k\,2^q+l} & , \ (h=i=j=0 \text{ and } 0 \leq k2^q+l \leq 2^q = n) \\ B(hijkl) = b_{k\,2^q+l} & , \ (h=i=j=0 \text{ and } 0 \leq k2^q+l \leq 2^q = n) \end{cases}$$

$$Q \ \equiv \ A(hijkl) \ = \ (V_0 X_0 - U_0 Y_0)_{ij}$$

$$AS_0 \ \equiv \ P$$

$$AS_1 \ \equiv \ \begin{cases} A(hijkl) = a_{k\,2^q+l} & , \ (h=i=j=0) \\ B(hijkl) = b_{k\,2^q+l} & , \ (h=i=j=0) \end{cases}$$

$$AS_2 \ \equiv \ \begin{cases} A(hijkl) = a_{k\,2^q+l} \\ B(hijkl) = b_{k\,2^q+l} \end{cases}$$

$$AS_3 \ \equiv \ \begin{cases} A(hijkl) = a_{k\,2^q+(l-i) \bmod 2^q} \\ B(hijkl) = b_{k\,2^q+(l-i) \bmod 2^q} \end{cases}$$

$$AS_4 \ \equiv \ \begin{cases} A(hijkl) = \begin{cases} a_{l-i} & , \ (k=0) \\ a_{2^q+l-i} & , \ (k=1) \end{cases} \\ B(hijkl) = \begin{cases} b_{2^q+l-i} & , \ (k=0) \\ b_{l-i} & , \ (k=1) \end{cases} \end{cases}$$

$$AS_5 \equiv \begin{cases} A(hijkl) = \begin{cases} a_{l-i} & ,\ (k=0) \\ a_{2^q+l-j} & ,\ (k=1) \end{cases} \\ B(hijkl) = \begin{cases} b_{2^q+l-j} & ,\ (k=0) \\ b_{l-i} & ,\ (k=1) \end{cases} \end{cases}$$

$$AS_6 \equiv \begin{cases} A(hijkl) = \begin{cases} a_{2^q-1-l-i} & ,\ (k=0) \\ a_{2^q+l-j} & ,\ (k=1) \end{cases} \\ B(hijkl) = \begin{cases} b_{2^q+l-j} & ,\ (k=0) \\ b_{2^q-1-l-i} & ,\ (k=1) \end{cases} \end{cases}$$

$$AS_7 \equiv A(hijkl) = \begin{cases} a_{2^q-1-l-i}\, b_{2^q+l-j} & ,\ (k=0) \\ b_{2^q-1-l-i}\, a_{2^q+l-j} & ,\ (k=1) \end{cases}$$

$$AS_8 \equiv A(hijkl) = a_{2^q-1-l-i}\, b_{2^q+l-j} - b_{2^q-1-l-i}\, a_{2^q+l-j}\ ,\ (k=0)$$

$$AS_9 \equiv A(hijkl) = \sum_{x=0}^{2^q-1} a_{2^q-1-x-i}\, b_{2^q+x-j} - b_{2^q-1-x-i}\, a_{2^q+x-j}$$

$$= (V_0X_0 - U_0Y_0)_{ij}\ ,\ (k=0)$$

$$AS_{10} \equiv A(hijkl) = (V_0X_0 - U_0Y_0)_{ij}$$

Theorem 7.4.1. $\{P\}$ *INIT* $\{Q\}$.

Proof.. $P = AS_0 \overset{\text{A1}}{\Longrightarrow} AS_1 \overset{\text{L2}}{\Longrightarrow} AS_2 \overset{\text{L3}}{\Longrightarrow} AS_3 \overset{\text{A2}}{\Longrightarrow} AS_4 \overset{\text{L5}}{\Longrightarrow} AS_5$
$\overset{\text{L4}}{\Longrightarrow} AS_6 \overset{\text{A1}}{\Longrightarrow} AS_7 \overset{\text{L7}}{\Longrightarrow} AS_8 \overset{\text{L8}}{\Longrightarrow} AS_9 \overset{\text{L2}}{\Longrightarrow} AS_{10} = Q$. □

7.5. Subroutine PWRA

Subroutine *PWRA* computes powers of the (symmetric) matrices $\{{}_lM\}_l$. The input consists of the coefficients ${}_lM_{ij}$, $(i, j < l)$, the output consists of the coefficients ${}_lM^h_{ij}$, $(i, j < l)$. Note that for ${}_nM = (V_0X_0 - U_0Y_0)$ and $i, j < l$ the output assertion of subroutine *INIT* is equivalent to the input assertion of

PWRA.

Algorithm 7.5.1.

```
Subroutine $PWRA$ :
$\{AS_0\}$
    $B(hijkl) \leftarrow A(hijkl)$ ;
$\{AS_1\}$
    for $d = 0$ to $r-1$ do begin
$\{AS_{2,d}\}$
        for $m \in [0, r-1]$ do $B(hijkl) \leftarrow B(h^{(m)}ijkl)$ , $(m < d$ and $h_m = 0)$ ;
$\{AS_{3,d}\}$
        for $m \in [0, q-1]$ do $B(hijkl) \leftarrow B(hi^{(m)}jkl)$ , $(i_m = \bar{k}_m)$ ;
$\{AS_{4,d}\}$
        $B(hijkl) \leftarrow \begin{cases} A(hijkl) \cdot B(hijkl) & , (j < l); \\ 0 & , (j \geq l); \end{cases}$
$\{AS_{5,d}\}$
        for $m \in [0, q-1]$ do $B(hijkl) \leftarrow B(hijkl) + B(hij^{(m)}kl)$ ;
$\{AS_{6,d}\}$
        for $m \in [0, q-1]$ do $B(hijkl) \leftarrow B(hijk^{(m)}l)$ , $(k_m = \bar{j}_m)$ ;
$\{AS_{7,d}\}$
        $A(hijkl) \leftarrow B(hijkl)$ , $(h_d = 1$ and $i, j < l)$ ;
$\{AS_{8,d}\}$
```

```
sh 2 _ 7 5
    end ;
{$AS_9$}
    for $m = 0$ to $r$-1 do $A(hijkl) \leftarrow A(h^{(m)}ijkl)$ , $(h_{<m} = 0)$ ;
    (* Note: $h_{<m} = 0 \equiv h_{<m} < 1_{<m+1} \leq 2^m + h_{<m}$ *)
{$AS_{10}$}
    $A(hijkl) \leftarrow \begin{cases} 1 , & (i = j \text{ and } h = 0 \text{ and } i, j < l) ; \\ 0 , & (i \neq j \text{ and } h = 0 \text{ and } i, j < l) ; \end{cases}$
{$AS_{11}$}
```

Assertions.

Note: Predicates about values that (trivially) do not change in the assignments are omitted from the intermediate assertions.

$P \equiv A(hijkl) = {}_lM_{ij} \ , (i, j < l)$

$$Q \equiv A(hijkl) = \begin{cases} {}_lM_{ij}^h & , (i, j < l) \\ \text{unchanged} & , \text{else} \end{cases}$$

$AS_0 \equiv P$

$AS_1 \equiv A(hijkl) = B(hijkl) = {}_lM_{ij} \ , (i, j < l)$

$$AS_{2,d} \equiv \begin{cases} A(hijkl) = {}_lM_{ij}^{h_{<d}+1} & , (i, j < l) \\ B(hijkl) = {}_lM_{ij}^{2^d} & , (h_{<d} = 2^d\text{-}1 \text{ and } i, j < l) \end{cases}$$

$AS_{3,d} \equiv B(hijkl) = {}_lM_{ij}^{2^d} \ , (i, j < l)$

$AS_{4,d} \equiv B(hijkl) = {}_lM_{kj}^{2^d} = {}_lM_{jk}^{2^d} \ , (j, k < l)$

$$AS_{5,d} \equiv B(hijkl) = \begin{cases} {}_lM_{ij}^{h_{<d}+1} \cdot {}_lM_{jk}^{2^d} & , (i,k<l) \\ 0 & , (j \geq l) \end{cases}$$

$$AS_{6,d} \equiv B(hijkl) = \sum_{x=0}^{l-1} {}_lM_{ix}^{h_{<d}+1} \cdot {}_lM_{xk}^{2^d} = {}_lM_{ij}^{2^d+h^{<d}+1} , (i,k<l)$$

$$AS_{7,d} \equiv B(hijkl) = {}_lM_{ij}^{2^d+h^{<d}+1} , (i,j<l)$$

$$AS_{8,d} \equiv$$

$$\begin{cases} A(hijkl) = {}_lM_{ij}^{2^d+h^{<d+1}+1} & , (i,j<l) \\ B(hijkl) = {}_lM_{ij}^{2^d+h^{<d+1}+1} = {}_lM_{ij}^{2^{d+1}} & , (h_{<d+1} = 2^{d+1}\text{-}1 \text{ and } i,j<l) \end{cases}$$

$$AS_9 \equiv A(hijkl) = {}_lM_{ij}^{h+1} , (i,j<l)$$

$$AS_{10} \equiv A(hijkl) = {}_lM_{ij}^{(h-1) \bmod 2^r+1} , (i,j<l)$$

$$AS_{11} \equiv A(hijkl) = \begin{cases} {}_lI_{ij} & , (h=0 \text{ and } i,j<l) \\ {}_lM_{ij}^h & , (h>0 \text{ and } i,j<l) \end{cases}$$

Theorem 7.4.1. $\{P\}$ *PWRA* $\{Q\}$.

Proof. $P = AS_0 \xrightarrow{A1} AS_1 \Longrightarrow AS_{2,0}$;

$$(\forall\, d \in [0, r\text{-}1])\ AS_{2,d} \xrightarrow{L2} AS_{3,d} \xrightarrow{L4} AS_{4,d} \xrightarrow{A1} AS_{5,d} \xrightarrow{L8}$$

$$AS_{6,d} \xrightarrow{L4} AS_{7,d} \xrightarrow{A1} AS_{8,d} = AS_{2,d+1} ;$$

$$AS_{2,r} \Longrightarrow AS_9 \xrightarrow{L9} AS_{10} \xrightarrow{A1} AS_{11} = Q .$$ □

7.5. Subroutine PWRB

Subroutine *PWRB* computes powers of the (symmetric) matrices $\{{}_lM^{2^{rc}}\}_l$. from powers of the matrices $\{{}_lM^{2^{rc-r}}\}_l$. The input consists of the coefficients ${}_lM_{ij}^{h\,2^{rc-r}}$, $(i,j<l)$, the output consists of the coefficients ${}_lM_{ij}^{h\,2^{rc}}$, $(i,j<l)$.

Note that $PWRB$ does not alter $A(hijkl)$ for $i \geq l$ or $j \geq l$.

Algorithm 7.5.1.

Subroutine $PWRB$:

```
{$AS_0$}
    $B(hijkl) \leftarrow A(hijkl)$ ;
{$AS_1$}
    for $d = 0$ to $r-1$ do begin
{$AS_{2,d}$}
        for $m \in [0, q-1]$ do $B(hijkl) \leftarrow B(hi^{(m)}jkl)$ , $(i_m = \bar{k}_m)$ ;
{$AS_{3,d}$}
        $B(hijkl) \leftarrow \begin{cases} A(hijkl) \cdot B(hijkl) & , (j < l); \\ 0 & , (j \geq l); \end{cases}$
{$AS_{4,d}$}
        for $m \in [0, q-1]$ do $B(hijkl) \leftarrow B(hijkl) + B(hij^{(m)}kl)$ ;
{$AS_{5,d}$}
        for $m \in [0, q-1]$ do $B(hijkl) \leftarrow B(hijk^{(m)}l)$ , $(k_m = \bar{j}_m)$ ;
{$AS_{6,d}$}
        $A(hijkl) \leftarrow B(hijkl)$ , $(h_d = 1$ and $i, j < l)$ ;
{$AS_{7,d}$}
    end ;
{$AS_8$}
```

Assertions.

$$P \equiv A(hijkl) = {}_lM_{ij}^{h\,2^{rc-r}} \;, (i, j < l)$$

$$Q \equiv A(hijkl) = \begin{cases} {}_lM_{ij}^{h\,2^{rc}} & , (i, j < l) \\ \text{unchanged} & , \text{else} \end{cases}$$

$$AS_0 \equiv P$$

$$AS_1 \equiv A(hijkl) = B(hijkl) = {}_lM_{ij}^{h\,2^{rc-r}} \;, (i, j < l)$$

$$AS_{2,d} \equiv A(hijkl) = B(hijkl) = {}_lM_{ij}^{h\,2^{rc-r+d}} \;, (i, j < l)$$

$$AS_{3,d} \equiv B(hijkl) = {}_lM_{kj}^{h\,2^{rc-r+d}} = {}_lM_{jk}^{h\,2^{rc-r+d}} \;, (j, k < l)$$

$$AS_{4,d} \equiv B(hijkl) = {}_lM_{ij}^{h\,2^{rc-r+d}} \cdot {}_lM_{jk}^{h\,2^{rc-r+d}} \;, (i, j, k < l)$$

$$AS_{5,d} \equiv B(hijkl) = {}_lM_{ik}^{h\,2^{rc-r+d+1}} \;, (i, k < l)$$

$$AS_{6,d} \equiv B(hijkl) = {}_lM_{ij}^{h\,2^{rc-r+d+1}} \;, (i, j < l)$$

$$AS_{7,d} \equiv A(hijkl) = B(hijkl) = {}_lM_{ij}^{h\,2^{rc-r+d+1}} \;, (i, j < l)$$

$$AS_8 \equiv A(hijkl) = {}_lM_{ij}^{h\,2^{rc}} \;, (i, j < l)$$

Theorem 7.5.1. $\{P\}\; PWRB\; \{Q\}$.

Proof. $P = AS_0 \overset{\text{A1}}{\Longrightarrow} AS_1 = AS_{2,0}$;

$$(\forall\, d \in [0, r-1])\; AS_{2,d} \overset{\text{L4}}{\Longrightarrow} AS_{3,d} \overset{\text{A1}}{\Longrightarrow} AS_{4,d} \overset{\text{L8}}{\Longrightarrow} AS_{5,d} \overset{\text{L4}}{\Longrightarrow}$$

$$AS_{6,d} \overset{\text{A1}}{\Longrightarrow} AS_{7,d} = AS_{2,d+1} ;$$

$$AS_{2,r} \Longrightarrow AS_8 = Q .$$

□

7.7. Subroutine DAC

Subroutine *DAC* multiplies matrices $\{{}_lM^{h\,2^{rc}}\}_{h,l}$ by vectors $\{{}_lM^{k<rc}\,{}_lS\}_{k,l}$, for any l. The input consists of the coefficients ${}_lM_{ij}^{h\,2^{rc}}$, $(i,j<l)$, and $({}_lM^{k<rc}\,{}_lS)_i$, $(i<l)$. The output consists of the coefficients $({}_lM^{k<rc+r}\,{}_lS)_i$, $(i<l)$.

Algorithm 7.7.1.

```
Subroutine $DAC$:
{$AS_0$}
    $B(hijkl) \leftarrow A(hijkl)$ ;
{$AS_1$}
    for $m \in [0, q-1]$ do $B(hijkl) \leftarrow B(hij^{(m)}kl)$ , $(j_m = \bar{l}_m)$ ;
{$AS_2$}
    $B(hijkl) \leftarrow \begin{cases} A(hijkl) \cdot B(hijkl) & , (i,j<l) ; \\ 0 & , \text{else} ; \end{cases}$
{$AS_3$}
    for $m \in [0, q-1]$ do $B(hijkl) \leftarrow B(hijkl) + B(hi^{(m)}jkl)$ ;
{$AS_4$}
    for $m \in [0, r-1]$ do $B(hijkl) \leftarrow B(h^{(m)}ijkl)$ , $(h_m = \bar{k}_{rc+m})$ ;
{$AS_5$}
    for $m \in [0, q-1]$ do $B(hijkl) \leftarrow B(hij^{(m)}kl)$ , $(j_m = \bar{i}_m)$ ;
{$AS_6$}
```

$A(hijkl) \leftarrow B(hijkl)$, ($j < l$ and $j = l$) ;

$\{AS_7\}$

Assertions.

$$P \equiv A(hijkl) = \begin{cases} {}_lM_{ij}^{h\,2^{rc}} = {}_lM_{ji}^{h\,2^{rc}} & , \ (i, j < l) \\ ({}_lM^{k_{<rc}}\,{}_lS)_i & , \ (i < l \text{ and } j = l) \end{cases}$$

$$Q \equiv A(hijkl) = \begin{cases} ({}_lM^{k_{<rc+r}}\,{}_lS)_i & , \ (i < l \text{ and } j = l) \\ \text{unchanged} & , \ \text{else} \end{cases}$$

$$AS_0 \equiv P$$

$$AS_1 \equiv B(hijkl) = ({}_lM^{k_{<rc}}\,{}_lS)_i \ , (i < l \text{ and } j = l)$$

$$AS_2 \equiv B(hijkl) = ({}_lM^{k_{<rc}}\,{}_lS)_i \ , (i < l)$$

$$AS_3 \equiv B(hijkl) = \begin{cases} {}_lM_{ji}^{h\,2^{rc}} \cdot ({}_lM^{k_{<rc}}\,{}_lS)_i & , \ (i, j < l) \\ 0 & , \ \text{else} \end{cases}$$

$$AS_4 \equiv B(hijkl) = ({}_lM^{h\,2^{rc} + k_{<rc}}\,{}_lS)_j \ , (j < l)$$

$$AS_5 \equiv B(hijkl) = ({}_lM^{(k_{<rc+r})_{\geq rc}\,2^{rc} + k_{<rc}}\,{}_lS)_j = ({}_lM^{k_{<rc+r}}\,{}_lS)_j \ , (j < l)$$

$$AS_6 \equiv B(hijkl) = ({}_lM^{k_{<rc+r}}\,{}_lS)_i \ , (i < l)$$

$$AS_7 \equiv A(hijkl) = ({}_lM^{k_{<rc+r}}\,{}_lS)_i \ , (i < l \text{ and } j = l)$$

Theorem 7.7.1. $\{P\}\ DAC\ \{Q\}$.

Proof. $P = AS_0 \overset{\text{A1}}{\Longrightarrow} AS_1 \overset{\text{L4}}{\Longrightarrow} AS_2 \overset{\text{A1}}{\Longrightarrow} AS_3 \overset{\text{L8}}{\Longrightarrow} AS_4 \overset{\text{L4}}{\Longrightarrow} AS_5$
$\overset{\text{L4}}{\Longrightarrow} AS_6 \overset{\text{A1}}{\Longrightarrow} AS_7 = Q$. □

7.8. Subroutine CIJ

Subroutine *CIJ* computes entries of the matrices $\{{}_lC\}_l$. The input consists of $\{{}_lM^k{}_lS\}_{k,l}$. The output consists of $\{{}_lC\}_l$ for $h=0$, and ${}_nM = V_0X_0 - U_0Y_0$ for $h=0$.

Algorithm 7.8.1.

Subroutine *CIJ*:

```
{$AS_0$}
    for $m \in [0, q-1]$ do $A(hijkl) \leftarrow A(hijkl^{(m)})$ , $(h > 0$ and $l_m = 1)$ ;
{$AS_1$}
    for $m \in [0, q-1]$ do $A(hijkl) \leftarrow A(hij^{(m)}kl)$ , $(h = 0$ and $j_m = \overline{l}_m)$ ;
{$AS_2$}
    $B(hijkl) \leftarrow A(hijkl)$ ;
{$AS_3$}
    for $m \in [0, q-1]$ do $B(hijkl) \leftarrow B(hijk^{(m)}l)$ , $(k_m = 1)$ ;
{$AS_4$}
    $B(hijkl) \leftarrow \begin{cases} B(hijkl) \cdot A(hijkl) & , (i < j); \\ 0 & , (i \geq j); \end{cases}$
{$AS_5$}
    for $m \in [0, q-1]$ do $B(hijkl) \leftarrow B(hijkl) + B(hi^{(m)}jkl)$ ;
{$AS_6$}
    $B(hijkl) \leftarrow 0$ , $(k \neq j-i-1)$ ;
```

$\{AS_7\}$

for $m \in [0, q-1]$ **do** $B(hijkl) \leftarrow B(hijkl) + B(hijk^{(m)}l)$;

$\{AS_8\}$

for $m \in [0, q-1]$ **do** $A(hijkl) \leftarrow A(hi^{(m)}jkl)$, $(h = 0 \text{ and } i_m = \overline{l}_m)$;

$\{AS_9\}$

$$A(hijkl) \leftarrow \begin{cases} -1 & , (j-i = -1 \text{ and } i \leq l+1 \text{ and } j \leq l \text{ and } h = 0) \\ -B(hijkl) & , (j-i > 0 \text{ and } i \leq l+1 \text{ and } j \leq l \text{ and } h = 0) \\ 0 & , ((j-i < -1 \text{ or } i > l+1 \text{ or } j > l) \text{ and } h = 0) \end{cases}$$

$\{AS_{10}\}$

Assertions.

$$P \equiv A(hijkl) = \begin{cases} {}_nM_{ij} & , (l = 0) ; \\ ({}_lM^k\,{}_lS)_i & , (i < l \text{ and } j = l) ; \\ {}_l\alpha & , (i, j = l) ; \end{cases}$$

$$Q \equiv A(hijkl) = \begin{cases} {}_nM_{ij} & , (h > 0) ; \\ {}_lC_{ij} & , (h = 0) ; \end{cases}$$

$$AS_0 \equiv P$$

$$AS_1 \equiv A(hijkl) = \begin{cases} {}_nM_{ij} & , (h > 0) \\ ({}_lM^k\,{}_lS)_i & , (i < l \text{ and } j = l \text{ and } h = 0) \\ {}_l\alpha & , (i, j = l \text{ and } h = 0) \end{cases}$$

$$AS_2 \equiv A(hijkl) = \begin{cases} ({}_lM^k\,{}_lS)_i & , (i < l \text{ and } h = 0) \\ {}_l\alpha & , (i = l \text{ and } h = 0) \end{cases}$$

$$AS_3 \equiv B(hijkl) = \begin{cases} ({}_lM^k\,{}_lS)_i & , (i < l \text{ and } h = 0) \\ {}_l\alpha & , (i = l \text{ and } h = 0) \end{cases}$$

$$AS_4 \equiv B(hijkl) = {}_lS_i = {}_lR_i \;, (i < l \text{ and } h = 0)$$

$$AS_5 \equiv B(hijkl) = \begin{cases} {}_lR_i \cdot ({}_lM^k\,{}_lS)_i & , (i < l \text{ and } h = 0) \\ 0 & , (i \geq l \text{ and } h = 0) \end{cases}$$

$$AS_6 \equiv B(hijkl) = {}_lR\,{}_lM^k\,{}_lS \ , (h = 0)$$

$$AS_7 \equiv B(hijkl) = \begin{cases} {}_lR\,{}_lM^{j-i-1}{}_lS & , (k = j-i-1 \text{ and } h = 0) \\ 0 & , (k \neq j-i-1 \text{ and } h = 0) \end{cases}$$

$$AS_8 \equiv B(hijkl) = {}_lR\,{}_lM^{j-i-1}{}_lS \ , (h = 0)$$

$$AS_9 \equiv A(hijkl) = {}_l\alpha \ , (h = 0)$$

$$AS_{10} \equiv A(hijkl) = {}_lC_{ij} \ , (h = 0)$$

Theorem 7.8.1. $\{P\}$ *CIJ* $\{Q\}$.

Proof. $P = AS_0 \overset{\text{L2}}{\Longrightarrow} AS_1 \overset{\text{L4}}{\Longrightarrow} AS_2 \overset{\text{A1}}{\Longrightarrow} AS_3 \overset{\text{L2}}{\Longrightarrow} AS_4 \overset{\text{A1}}{\Longrightarrow} AS_5$
$\overset{\text{L8}}{\Longrightarrow} AS_6 \overset{\text{A1}}{\Longrightarrow} AS_7 \overset{\text{L8}}{\Longrightarrow} AS_8 \overset{\text{L4}}{\Longrightarrow} AS_9 \overset{\text{A1}}{\Longrightarrow} AS_{10} = Q$. □

7.9. Subroutine CI

Subroutine *CI* computes the characteristic polynomials of matrices $\{{}_lM\}_l$. The input consists of the (asymmetric) matrices $\{{}_lC\}_l$. The output consists of the coefficients $\{{}_lc_i\}_{i,l}$.

Algorithm 7.9.1.

```
Subroutine CI :
{$AS_0$}
    for $d = 0$ to $r-1$ do begin
{$AS_{1,d}$}
        $B(hijkl) \leftarrow A(hijkl)$ ;
```

```
$\{AS_{2,d}\}$

        for $m \in [0, q-1]$ do $B(hijkl) \leftarrow B(hijkl^{(m)})$ ,
        $((m < d$ and $l_m = 0$ or $m = d$ and $l_m = 1)$ and $h = 0)$ ;

$\{AS_{3,d}\}$

        for $m \in [0, q-1]$ do $A(hijkl) \leftarrow A(hij^{(m)}kl)$ ,
        $(j_m = \overline{k}_m$ and $l_d = 1$ and $h = 0)$ ;

$\{AS_{4,d}\}$

        for $m \in [0, q-1]$ do $B(hijkl) \leftarrow B(hi^{(m)}jkl)$ , $(i_m = \overline{k}_m$ and $h = 0)$ ;

$\{AS_{5,d}\}$

        $A(hijkl) \leftarrow A(hijkl) \cdot B(hijkl)$ , $(l_d = 1$ and $h = 0)$ ;

$\{AS_{6,d}\}$

        for $m \in [0, q-1]$ do $A(hijkl) \leftarrow A(hijkl) + A(hijk^{(m)}l)$ ,
        $(l_d = 1$ and $h = 0)$ ;

$\{AS_{7,d}\}$

    end ;

$\{AS_8\}$

    for $m \in [0, q-1]$ do $A(hijkl) \leftarrow A(hij^{(m)}kl)$ , $(j_m = 1$ and $h = 0)$ ;

$\{AS_9\}$
```

Assertions.

$P \;\equiv\; A(hijkl) = {}_lC_{ij}$, $(h = 0)$

$$Q \;\equiv\; \begin{cases} A(hijkl) = {}_lc_i & , \ (h = 0) \\ \text{unchanged} & , \ (h > 0) \end{cases}$$

$$AS_0 \equiv P$$

$$AS_{1,d} \equiv A(hijkl) = \left(\prod_{m=0}^{l_{<d}} {}_{l_{\geq d}2^d+m}C \right)_{ij} \;,\; (h = 0)$$

$$AS_{2,d} \equiv B(hijkl) = \left(\prod_{m=0}^{l_{<d}} {}_{l_{\geq d}2^d+m}C \right)_{ij} \;,\; (h = 0)$$

$$AS_{3,d} \equiv B(hijkl) = \left(\prod_{m=0}^{2^d-1} {}_{l_{\geq d}2^d+m}C \right)_{ij} \;,\; (h = 0)$$

$$AS_{4,d} \equiv A(hijkl) = \begin{cases} \left(\prod_{m=0}^{l_{<d+1}} {}_{l_{\geq d+1}2^{d+1}+m}C \right)_{ij} & ,\; (l_d = 0 \text{ and } h = 0) \\ \left(\prod_{m=2^d}^{l_{<d+1}} {}_{l_{\geq d+1}2^{d+1}+m}C \right)_{ik} & ,\; (l_d = 1 \text{ and } h = 0) \end{cases}$$

$$AS_{5,d} \equiv B(hijkl) = \left(\prod_{m=0}^{2^d-1} {}_{l_{\geq d}2^d+m}C \right)_{kj} \;,\; (h = 0)$$

$$AS_{6,d} \equiv A(hijkl) =$$

$$\begin{cases} \left(\prod_{m=0}^{l_{<d+1}} {}_{l_{\geq d+1}2^{d+1}+m}C \right)_{ij} & ,\; (l_d = 0 \text{ and } h = 0) \\ \left(\prod_{m=2^d}^{l_{<d+1}} {}_{l_{\geq d+1}2^{d+1}+m}C \right)_{ik} \cdot \left(\prod_{m=0}^{2^d-1} {}_{l_{\geq d+1}2^{d+1}+m}C \right)_{kj} & ,\; (l_d = 1 \text{ and } h = 0) \end{cases}$$

$$AS_{7,d} \equiv A(hijkl) = \left(\prod_{m=0}^{l_{<d+1}} {}_{l_{\geq d+1}2^{d+1}+m}C \right)_{ij} \;,\; (h = 0)$$

$$AS_8 \equiv A(hijkl) = \left(\prod_{m=0}^{l} {}_{m}C \right)_{ij} \;,\; (h = 0)$$

$$AS_9 \equiv A(hijkl) = \left(\prod_{m=0}^{l} {}_{m}C \right)_{i0} = {}_{l}c_i \;,\; (h = 0)$$

Theorem 7.9.1. $\{P\}\ CI\ \{Q\}$.

Proof. $P = AS_0 = AS_{1,0}$;

$$(\forall\, d \in [0, r-1])\ AS_{1,d} \overset{\text{A1}}{\Longrightarrow} AS_{2,d} \overset{\text{L2}}{\Longrightarrow} AS_{3,d} \overset{\text{L4}}{\Longrightarrow} AS_{4,d} \overset{\text{L4}}{\Longrightarrow}$$

$$AS_{5,d} \overset{\text{A1}}{\Longrightarrow} AS_{6,d} \overset{\text{L8}}{\Longrightarrow} AS_{7,d} = AS_{1,d+1} ;$$

$$AS_{1,r} = AS_8 \overset{\text{L2}}{\Longrightarrow} AS_9 = Q \ . \qquad \square$$

7.10. Subroutine MAX

Subroutine *MAX* finds the index l^* of the largest nonsingular matrix ${}_lM$. The input consists of the coefficients $\{{}_lc_i\}_{i,l}$. The output consists of $\{-{}_{l^*}c_{k+1}/{}_{l^*}c_0\}_k$ and l^*.

Algorithm 7.10.1.

Subroutine *MAX*:

```
{$AS_0$}
    $B(hijkl) \leftarrow \begin{cases} l \ , \ (h,i=0 \text{ and } A(hijkl) \neq 0) ; \\ 0 \ , \ \text{else}) ; \end{cases}$
{$AS_1$}
    for $m \in [0,q-1]$ do $B(hijkl) \leftarrow \max(B(hijkl), B(hijk^{(m)}l))$ ;
{$AS_2$}
    for $m \in [0,q-1]$ do $B(hijkl) \leftarrow B(hi^{(m)}jkl)$ , $(i_m = 1)$ ;
    for $m \in [0,r-1]$ do $B(hijkl) \leftarrow B(h^{(m)}ijkl)$ , $(h_m = 1)$ ;
{$AS_3$}
    $B(hijkl) \leftarrow A(hijkl)$ , $(h = 0)$ ;
{$AS_4$}
    for $m \in [0,q-1]$ do $B(hijkl) \leftarrow B(hi^{(m)}jkl)$ , $(i_m = 1 \text{ and } h = 0)$ ;
```

$\{AS_5\}$

$A(hijkl) \leftarrow -A(hijkl)/B(hijkl)$, $(h = 0)$;

$\{AS_6\}$

$B(hijkl) \leftarrow B(h^{(m)}ijkl)$, $(h = 0)$;

$\{AS_7\}$

for $m = 0$ **to** q-1 **do** $A(hijkl) \leftarrow A(hi^{(m)}jkl)$, $(i_{<m} = 2^m\text{-}1$ and $h = 0)$;

(* Note: $i_{<m} = 2^m\text{-}1 \equiv i_{<m} < (2^q\text{-}1)_{<m+1} \leq 2^m + i_{<m}$ *)

$\{AS_8\}$

$$A(hijkl) \leftarrow \begin{cases} 1 , & (i = 2^q\text{-}1 \text{ and } l = 2^q\text{-}1 \text{ and } h = 0) ; \\ 0 , & (i = 2^q\text{-}1 \text{ and } l < 2^q\text{-}1 \text{ and } h = 0) ; \end{cases}$$

$\{AS_9\}$

$A(hijkl) \leftarrow 0$, $(l \neq B(hijkl)$ and $h = 0$ or $l > B(hijkl)$ and $h > 0)$;

$\{AS_{10}\}$

for $m \in [0, q\text{-}1]$ **do** $A(hijkl) \leftarrow A(hijkl) + A(hijkl^{(m)})$, $(h = 0)$;

$\{AS_{11}\}$

for $m \in [0, q\text{-}1]$ **do** $A(hijkl) \leftarrow A(hi^{(m)}jkl)$, $(i_m = \overline{k}_m$ and $h = 0)$;

$\{AS_{12}\}$

Assertions.

$$P \equiv A(hijkl) = \begin{cases} {}_nM_{ij} , & (h > 0) ; \\ {}_lc^{\,i} , & (h = 0) ; \end{cases}$$

$$Q \equiv \begin{cases} A = \begin{cases} {}_nM_{ij} & , (j \leq l^* \text{ and } h > 0) \\ 0 & , (j > l^* \text{ and } h > 0) \\ -{}_{l^*}c_{k+1}/{}_{l^*}c_0 & , (h = 0) \end{cases} \\ B = l^* , (h = 0) \end{cases}$$

$$AS_0 \equiv P$$

$$AS_1 \equiv A(hijkl) = \begin{cases} l & , (h, i = 0 \text{ and } {}_lc_0 \neq 0) \\ 0 & , \text{else} \end{cases}$$

$$AS_2 \equiv B(hijkl) = \max_{l \in [0, 2^q - 1]} \{ l \mid {}_lc_0 \neq 0 \} = l^* \ , (h, i = 0)$$

$$AS_3 \equiv B(hijkl) = l^*$$

$$AS_4 \equiv A(hijkl) = \begin{cases} l^* & , (h > 0) \\ {}_lc_i & , (h = 0) \end{cases}$$

$$AS_5 \equiv A(hijkl) = \begin{cases} l^* & , (h > 0) \\ {}_lc_0 & , (h = 0) \end{cases}$$

$$AS_6 \equiv A(hijkl) = \begin{cases} {}_nM_{ij} & , (h > 0) \\ -{}_lc_i / {}_lc_0 & , (h = 0) \end{cases}$$

$$AS_7 \equiv B(hijkl) = l^*$$

$$AS_8 \equiv A(hijkl) = \begin{cases} {}_nM_{ij} & , (h > 0) \\ -{}_lc_{i+1} / {}_lc_0 & , (h = 0 \text{ and } i < 2^q - 2) \end{cases}$$

$$AS_9 \equiv A(hijkl) = \begin{cases} {}_nM_{ij} & , (h > 0) \\ -{}_lc_{i+1} / {}_lc_0 & , (h = 0) \end{cases}$$

$$AS_{10} \equiv A(hijkl) = \begin{cases} {}_nM_{ij} = Z_{ij} & , (h > 0 \text{ and } i, j \leq l^*) \\ 0 = Z_{ij} & , (h > 0 \text{ and } i \leq l^* \text{ and } j > l^*) \\ -{}_{l^*}c_{i+1} / {}_{l^*}c_0 & , (h = 0 \text{ and } l = l^*) \\ 0 & , (h = 0 \text{ and } l \neq l^*) \end{cases}$$

$$AS_{11} \equiv A(hijkl) = -{}_{l^*}c_{i+1} / {}_{l^*}c_0 \ , (h = 0)$$

$$AS_{12} \equiv A(hijkl) = -{}_{l^*}c_{k+1} / {}_{l^*}c_0 \ , (h = 0)$$

Theorem 7.10.1. $\{ P \}$ *MAX* $\{ Q \}$.

Proof. $P = AS_0 \overset{\text{A1}}{\Longrightarrow} AS_1 \overset{\text{L8}}{\Longrightarrow} AS_2 \overset{\text{L2}}{\Longrightarrow} AS_3 \overset{\text{A1}}{\Longrightarrow} AS_4 \overset{\text{L2}}{\Longrightarrow} AS_5 \overset{\text{A1}}{\Longrightarrow} AS_6 \overset{\text{L1}}{\Longrightarrow} AS_7 \overset{\text{L9}}{\Longrightarrow} AS_8 \overset{\text{A1}}{\Longrightarrow} AS_9 \overset{\text{A1}}{\Longrightarrow} AS_{10} \overset{\text{L8}}{\Longrightarrow} AS_{11} \overset{\text{L4}}{\Longrightarrow} AS_{12} = Q$. □

7.11. Subroutine PWRC

Subroutine $PWRC$ computes powers of the matrix Z. The structure of the subroutine is similar to that of $PWRA$. Unlike matrices $\{{}_lM\}_l$, matrix Z is not symmetric.

Algorithm 7.11.1.

```
Subroutine $PWRC$:
$\{AS_0\}$
    $B(hijkl) \leftarrow A(hijkl)$ , $(h > 0)$ ;
$\{AS_1\}$
    **for** $d = 0$ **to** $q-1$ **do begin**
$\{AS_{2,d}\}$
        **for** $m \in [0, q-1]$ **do** $B(hijkl) \leftarrow B(hijk^{(m)}l)$ ,
        $(m < d$ and $k_m = 0$ and $h > 0)$ ;
$\{AS_{3,d}\}$
        **for** $m \in [0, q-1]$ **do** $A(hijkl) \leftarrow A(hij^{(m)}kl)$ ,
        $(j_m = \bar{l}_m$ and $k_d = 1$ and $h > 0)$ ;
$\{AS_{4,d}\}$
```

```
        for $m \in [0, q-1]$ do $B(hijkl) \leftarrow B(hi^{(m)}jkl)$ , $(i_m = \overline{l}_m$ and $h > 0)$ ;
$\{AS_{5,d}\}$
        $B(hijkl) \leftarrow A(hijkl) \cdot B(hijkl)$ , $(h > 0)$ ;
$\{AS_{6,d}\}$
        for $m \in [0, q-1]$ do $B(hijkl) \leftarrow B(hijkl) + B(hijkl^{(m)})$ , $(h > 0)$ ;
$\{AS_{7,d}\}$
        $A(hijkl) \leftarrow B(hijkl)$ , $(k_d = 1$ and $h > 0)$ ;
$\{AS_{8,d}\}$
    end ;
$\{AS_9\}$
```

Assertions.

$P \equiv A(hijkl) = Z_{ij} \ , (h > 0)$

$$Q \equiv A(hijkl) = \begin{cases} Z_{ij}^{k+1} & , \ (h > 0) \\ \text{unchanged} & , \ \text{else} \end{cases}$$

$AS_0 \equiv P$

$AS_1 \equiv A(hijkl) = B(hijkl) = Z_{ij} \ , (h > 0)$

$$AS_{2,d} \equiv \begin{cases} A(hijkl) = Z_{ij}^{k_{<d}+1} & , \ (h > 0) \\ B(hijkl) = Z_{ij}^{2^d} & , \ (k_{<d} = 2^d-1 \text{ and } h > 0) \end{cases}$$

$AS_{3,d} \equiv B(hijkl) = Z_{ij}^{2^d} \ , (h > 0)$

$$AS_{4,d} \equiv A(hijkl) = \begin{cases} Z_{ij}^{k_{<d+1}+1} & , \ (k_d = 0 \text{ and } h > 0) \\ Z_{il}^{k_{<d+1}-2^d+1} & , \ (k_d = 1 \text{ and } h > 0) \end{cases}$$

$$AS_{5,d} \equiv B(hijkl) = Z_{lj}^{2^d} \text{ , } (h > 0)$$

$$AS_{6,d} \equiv B(hijkl) = Z_{il}^{k_{<d+1}-2^d+1} \cdot Z_{lk}^{2^d} \text{ , } (k_d = 1 \text{ and } h > 0)$$

$$AS_{7,d} \equiv B(hijkl) = Z_{ij}^{k_{<d+1}+1} \text{ , } (k_d = 1 \text{ and } h > 0)$$

$$AS_{8,d} \equiv \begin{cases} A(hijkl) = Z_{ij}^{k_{<d+1}+1} & , \ (h > 0) \\ B(hijkl) = Z_{ij}^{k_{<d+1}+1} = Z_{ij}^{2^{d+1}} & , \ (k_{<d+1} = 2^{d+1}\text{-}1 \text{ and } h > 0) \end{cases}$$

$$AS_9 \equiv A(hijkl) = Z_{ij}^{k+1} \text{ , } (h > 0)$$

Theorem 7.11.1. $\{P\}\ PWRC\ \{Q\}$.

Proof. $P = AS_0 \overset{\text{A1}}{\Longrightarrow} AS_1 \Longrightarrow AS_{2,0}$;

$(\forall\ d \in [0, q\text{-}1]\ )\ AS_{2,d} \overset{\text{L2}}{\Longrightarrow} AS_{3,d} \overset{\text{L4}}{\Longrightarrow} AS_{4,d} \overset{\text{L4}}{\Longrightarrow} AS_{5,d} \overset{\text{A1}}{\Longrightarrow}$

$AS_{6,d} \overset{\text{L8}}{\Longrightarrow} AS_{7,d} \overset{\text{A1}}{\Longrightarrow} AS_{8,d} = AS_{2,d+1}$;

$AS_{2,q} \Longrightarrow AS_9 = Q$. □

7.12. Subroutine RES

Subroutine *RES* computes the GCD coefficients. The input consists of powers of the matrix Z and the coefficients $\{ {}_{l^*}c_k \}_k$. The output consists of the GCD coefficients stored in the lowest $(n\text{-}l^*)$ addresses. The remaining registers are set to zero.

Algorithm 7.12.1.

Subroutine *RES* :

$\{AS_0\}$

```
    $B(hijkl) \leftarrow B(h^{(0)}ijkl)$ , $(h_0 = 1)$ ;
$\{AS_1\}$
    $B(hijkl) \leftarrow B(h^{(0)}ijkl)$ , $(h_0 = 0)$ ;
$\{AS_2\}$
    $A(hijkl) \leftarrow A(hijkl) \cdot B(hijkl)$ , $(h = 0)$ ;
$\{AS_3\}$
    for $m \in [0, q-1]$ do $A(hijkl) \leftarrow A(hijkl) + A(hijk^{(m)}l)$ , $(h = 0)$ ;
$\{AS_4\}$
    $B(hijkl) \leftarrow B(h^{(0)}ijkl)$ , $(h_0 = 0)$ ;
$\{AS_5\}$
    $A(hijkl) \leftarrow 0$ , $(B(hijkl) \neq 0$ or $h \neq 0)$ ;
$\{AS_6\}$
    for $m \in [0, q-1]$ do $A(hijkl) \leftarrow A(hijkl) + A(hi^{(m)}jkl)$ ;
$\{AS_7\}$
    for $m \in [0, q-1]$ do $A(hijkl) \leftarrow A(hi^{(m)}jkl)$ , $(i_m = \bar{l}_m)$ ;
$\{AS_8\}$
```

Assertions.

$$P \equiv A(hijkl) = \begin{cases} A(hijkl) = \begin{cases} Z_{ij}^{k+1} & , (h > 0) \\ -{}_{l^*}c_{k+1} / {}_{l^*}c_0 & , (h = 0) \end{cases} \\ B(hijkl) = l^* \ , (h = 0) \end{cases}$$

$$Q \equiv A(hijkl) = p_{h\,2^{4q} + i\,2^{3q} + j\,2^{2q} + k\,2^q + l}$$

$$AS_0 \equiv P$$

$$AS_1 \equiv B(hijkl) = l^* \ , (h = 1)$$

$$AS_2 \equiv B(hijkl) = Z_{ij}^{k+1} \ , (h = 0)$$

$$AS_3 \equiv A(hijkl) = -{}_{l^*}c_{k+1} / {}_{l^*}c_0 \cdot Z_{ij}^{k+1} \ , (h = 0)$$

$$AS_4 \equiv A(hijkl) = -\sum_{x=0}^{l^*} {}_{l^*}c_{x+1} / {}_{l^*}c_0 \cdot Z_{ij}^{x+1} \ , (h = 0)$$

$$AS_5 \equiv B(hijkl) = l^*$$

$$AS_6 \equiv A(hijkl) = \begin{cases} p_i & , (j = l^* \text{ and } h = 0) \\ 0 & , \text{else} \end{cases}$$

$$AS_7 \equiv A(hijkl) = \begin{cases} p_i & , (h = 0) \\ 0 & , \text{else} \end{cases}$$

$$AS_8 \equiv A(hijkl) = \begin{cases} p_l & , (h = 0) \\ 0 & , \text{else} \end{cases}$$

Theorem 7.12.1. $\{P\}\ RES\ \{Q\}$.

Proof. $P = AS_0 \overset{\text{L1}}{\Longrightarrow} AS_1 \overset{\text{L1}}{\Longrightarrow} AS_2 \overset{\text{A1}}{\Longrightarrow} AS_3 \overset{\text{L8}}{\Longrightarrow} AS_4 \overset{\text{L1}}{\Longrightarrow} AS_5$
$\overset{\text{A1}}{\Longrightarrow} AS_6 \overset{\text{L8}}{\Longrightarrow} AS_7 \overset{\text{L4}}{\Longrightarrow} AS_8 = Q$. □

CONCLUSIONS

As chapters 2 and 3 showed, there is a variety of models of parallel machines. The differences among them are deeper than the differences among sequential models. Unlike the sequential models, general theoretical parallel models are much more powerful than the existing real parallel computers. As a result of this situation, the parallel algorithms designed for general parallel models, such as P-RAM, usually do not have the same potential for practical use as their sequential counterparts.

We discussed architectures of some more practical network machines and defined two classes of models, σ-polynomial and σ-exponential. We conjecture that σ-exponential models are more powerful than σ-polynomial models. We also presented algorithms that simulate ASCEND/DESCEND CUBE algorithms on the PERFECT SHUFFLE and the CCC.

The algorithm of Borodin-von zur Gathen-Hopcroft (1984) provided the main idea for our CUBE-feasible polynomial GCD algorithm that works over arbitrary fields, runs in $O(\log^2 n)$ time and uses $n^{\alpha+1+\epsilon}$, $\alpha = 3$, processors. We presented a new matrix formula for the GCD of two polynomials and modified Berkowitz (1984) - Samuelson's (1942) formula for characteristic polynomials. The combination of both results allowed us to decrease the required number of processors by $O(n)$.

We developed an axiomatic verification system for CUBE-feasible algorithms and used it to prove the correctness of our routines.

The processor bound $n^{4+\epsilon}$ seems to be unrealistic. It is not. Restricted, regular architectures of some network machines allow parallel computer with a relatively large number of processors to be built with current technology. Currently, 2^{20} processors on a CCC is implementable; a 2^{30} processor machine is considered to be feasible [Wagner (1983), Duval, Wagner, Han and Loveland (1986)]. It is, however, desirable to decrease the required number of processors at least by another $O(n)$. The following directions seem to be promising.

1. Upper left principal minors of a matrix have closely related structures. Their simultaneous powering "should" require less than $O(n^{\alpha+1+\epsilon})$ processors. Indeed, Berkowitz (1984) conjectured that only $O(n^{\alpha})$ processors are necessary.

2. The matrix $(V_0X_0 - U_0Y_0)$ has a special structure: The (+)-displacement rank of the matrix with the reversed row order is 2. The (+)-displacement rank of any matrix is equal to the the (–)-displacement rank of its inverse [Kaliath, Kung and Morf (1979)]. This allows us to construct a sequential algorithm for the inversion of matrices with low displacement ranks [Bitmead, Anderson (1980)]. Some of these results might be applicable in parallel.

3. Bini (1984) showed that matrices from a certain class, including upper triangular Toeplitz matrices, can be inverted in $O(\log n)$ time using $O(n^2)$ processors. The structure of our matrices is similar to that class, suggesting that similar techniques might be found for it.

REFERENCES

Aho, A. V., Hopcroft, J. and Ullman, J. D. (1984), *The Design and Analysis of Computer Algorithms*, Addisson-Wesley, Reading, MA .

Akl, S. (1985), *Parallel Sorting Algorithms*, Academic Press.

Basu, A.(1984), *A classification of parallel processing systems*, Proc. IEEE Intl. Conf. Comp. Design: VLSI in Computers ICCD'84, pp. 222-225, (1984).

Batcher, K. E. (1968), *Sorting Networks and Their Applications*, AFIPS Spring Joint Computer Conference, pp. 308-314.

Berkowitz, S. J. (1984), *On Computing the Determinant in Small Parallel Time Using a Small Number of Processors*, Inf. Proc. Letters 18(3), pp. 147-150.

Bini, D. (1984), *Parallel Solution of Certain Toeplitz Linear Systems*, SIAM J. Comput. 13(2), pp. 268-276.

Bitmead, R. R., Anderson, B. D. O. (1980), *Assymptotically Fast Solution of Toeplitz and Related Systems of Linear Equations*, Lin. Alg. Appl. 34, pp. 103-116.

Borodin, A. (1977), *On relating time and space to size and depth*, SIAM J. Comput. 6(4), pp. 733-744.

Borodin, A., von zur Gathen, J., Hopcroft, J. (1984), *Fast Parallel Matrix and Gcd Computations*, Inf. Contr. 52(3), pp. 241-256.

Brent, R. P., Kung, H. T. (1983), *Some linear-time algorithms for systolic arrays*, Inf. Processing '83 (R.Mason, editor).

Chandra, A. K., Stockmeyer, L. J. (1976), *Alternation*, Proc. 17th FOCS, pp. 338-345.

Chandra, A. K., Kozen, D. C., Stockmeyer, L. J. (1981), *Alternation*, J. ACM 28(1), pp. 114-133.

Cook, S. A. (1981), *Towards a Complexity Theory of Synchronous Parallel Computation*, in L'enseignement mathematique, Ser.II, Tome XXVII, fasc.1-2.

Cook, S. A. (1985), *A Taxonomy of Problems with Fast Parallel Algorithms*, Inf. Contr. 64(1-3), pp. 2-22.

Coopersmith, D., Winograd, S. (1981), *On the Asymptotic Complexity of Matrix Multiplication*, Proc. 22nd FOCS, pp. 82-90.

Csansky, L. (1976), *Fast Parallel Matrix Inversion Algorithms*, SIAM J. Comput. 5, pp. 618-623.

Duval, L. D., Wagner, R. A., Han, Y., Loveland, D. W. (1986), *Finding Test-and-treatment Procedures Using Parallel Computation*, Proc. Intl. Conf. Parallel Processing, pp. 688-690.

Dymond, P. (1980), *Simultaneous Resource Bounds and Parallel Computation*, Ph.D. thesis, U. of Toronto, Dept. of Comp. Sci.

Dymond, P. W., Cook, S. A. (1980), *Hardware Complexity and Parallel Computation*, Proc. 21st FOCS, pp. 360-372.

Dymond, P., Tompa, M. (1983), *Speedups of Deterministic Machines by Synchronous Parallel Machines*, Proc. 15th STOC, pp. 336-343.

Eberly, W. (1984), *Very Fast Parallel Matrix and Polynomial Arithmetic*, Proc. 25th FOCS, pp. 21-31.

Flynn, M. J. (1972), *Some Computer Organizations and Their Effectiveness*, IEEE T. Computers C-21(9), pp. 948-960.

Fortune, S., Wyllie, J. (1978), *Parallelism in Random Access Machines*, Proc. 10th STOC, pp. 114-118.

Galil, Z., Paul, W. J. (1983), *An Efficient General-Purpose Parallel Computer*, J.ACM 30(2), pp. 360-387.

Gentleman, W. M. (1978), *Some Complexity Results for Matrix Computations on Parallel Processors*, J. ACM 25(1), pp. 112-115.

Goldschlager, L. M. (1978), *A Unified Approach to Models of Synchronous Parallel Machines*, Proc. 10th STOC, pp. 89-94.

Gustavson, J. L., Hawkinson, S., Scott, K. (1986), *The Architecture of a Homogenous Vector Supertcomputer*, Proc. Intl. Conf. Parallel Processing, pp. 649-652.

Hillis, W. D. (1985), *The Connection Machine*, MIT Press.

Hoare, C. A. R. (1969), *An Axiomatic Basis of Computer Programming*, CACM 12(10), pp. 576-580.

Hopcroft, J. E., Ullman, J. D. (1979), *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley.

Hopcroft, J., Paul, W. J., Valiant, L. J. (1975), *On Time Versus Space and Related Problems*, Proc. 16th FOCS, pp. 57-84.

Hopcroft, J., Paul, W. J., Valiant, L. J. (1977), *On Time Versus Space*, J.ACM 24, pp. 332-337.

Intel Corp. (1986), *iPSC System Overview*,

Kaliath, T., Kung, S.-Y., Morf, M. (1979), *Displacement Ranks of Matrices and Linear Equations*, J. Math. Anal. Appl. 68(2), pp. 395-659.

Knuth, D. E. (1973), *The Art of Computer Programming: Fundamental Algorithms*, Vol.1, Addison-Wesley.

Kozen, D. (1976), *On Parallelism in Turing Machines*, Proc. 17th FOCS, pp. 89-97.

Kung, H. T. (1979), *New Algorithms and Lower Bounds for the Parallel Evaluation of Certain Rational Expressions and Recurrences*, J. ACM 23(2), pp. 252-261.

Kung, H. T. (1980), *The Structure of Parallel Algorithms*, in Advances in Computers 19, New York: Academic.

Kung, H. T., Leiserson, C. L. (1980), *Algorithms for VLSI Processor Arrays*, in Introduction to VLSI Systems, Addison-Wesley, pp. 271-292.

Levitt, K. N., Kautz, W. H. (1972), *Cellular Arrays for the Solution of Graph Problems*, CACM 15(9), pp. 789-801.

MacWilliams, F. J. (1977), *The Theory of Error Correcting Codes*, North-Holland, Amsterodam.

Nassimi, D., Sahni, S. (1979), *Bitonic Sort on a Mesh-connected Parallel Computer*, IEEE T. Computers C-28(1), pp. 2-7.

NCUBE Corp. (1986), *NCUBE Handbook*, Version 1.0, Beaverton, Ore.

Pan, V., Reif, J. (1985), *Fast and Efficient Parallel Solution of Linear Systems*, Proc. 17th STOC, pp. 143-152.

Parberry, I. (1985), *Some Practical Simulations of Impractical Parallel Machines*, VLSI: Algorithms and Architectures.

Pease, M. C. (1977), *The Indirect Binary n-cube Microprocessor Array*, IEEE T. Computers C-26(5), pp. 458-473.

Pippinger, N. (1979), *On Simultaneous Resource Bounds*, Proc. 20th FOCS, pp. 307-311.

Preparata, F. P., Sarwate (1978), *An Improved Parallel Process-bound in Fast Matrix inversion*, Inf. Proc. Letters 7, pp. 148-151.

Preparata, F. P., Vuillemin, J. (1979), *The Cube-Connected Cycles: a versatile network for Parallel Computation*, Proc. 20th FOCS, pp. 140-147.

Preparata, F. P., Vuillemin, J. (1981), *The Cube-Connected Cycles: a versatile network for Parallel Computation*, CACM 24(5), pp. 300-309.

Ruzzo, W. L. (1979), *On Uniform Circuit Complexity*, Proc. 20th FOCS, pp. 312-318.

Ruzzo, W. L. (1981), *On Uniform Circuit Complexity*, J. Comp. Sys. Sci. 22(3), pp. 365-383.

Savitch, W., Stimson, M., (1979), *Time Bounded Random Access Machines with Parallel Processing*, J. ACM 26, pp. 103-118.

Schönhage (1979) *Storage Modification Machines*, Tech. Rep., Math. Inst. Univ. Tubingen, Germany.

Schwartz, J. T. (1980), *Ultracomputers*, ACM T. Prog. Lang. Sys. 2(4), pp. 484-521.

Seitz, C. L. (1985), *The Cosmic Cube*, CACM 28, pp. 22-23.

Shannon, C. E. (1949), *The Synthesis of Tho Terminal Switching Circuits*, BSTJ 28, pp. 59-98.

Squire, J. S. , Palais, S. M. (1962), *Physical an Logical Design of A Highly Parallel Computer*, Tech. Note, Dept. of Elec. Eng., U. of Michigan.

Squire, J. S. , Palais, S. M. (1963), *Programming and Design Considerations for a Highly Parallel Computer*, Proc. Spring Joint Comput. Conf., pp. 395-400.

Stockmeyer, L, Vishkin U. (1984), *Simulation of Practical Random Access Machines by Circuits*, SIAM J. Comput. 13(2), pp. 409-422.

Stone, H. S. (1971), *Parallel Processing with the Perfect Shuffle*, IEEE T. Computers C-20(2), pp. 161-163

Strassen, V., (1973). *Vermeidung von Divisionen*, J. Reine Angew. Math. 264, 184-202.

Upfal, E. (1984), *A Probabilistic Relation Between Desirable and Feasible Models for Parallel Computation*, Proc. 16th STOC, pp. 258-265.

Valiant, L., Skyum, S., Berkowitz, S., Rackoff, C. (1983), *Fast Parallel Computation of Polynomials Using Few Processors*, SIAM J. Comput. 12(3), pp. 641-644.

Von zur Gathen, J. (1984), *Parallel Algorithms for Algebraic Problems*, SIAM J. Comput. 13, pp. 802-824.

Von zur Gathen, J. (1986), *Representations and Parallel Computations for Rational Functions*, SIAM J. Comput. 15, pp. 432-452.

Wagner, R. A. (1983), *The Boolean Vector Machine [BVM]*, Proc. IEEE Intl. Symp. Comp. Arch., pp. 59-66.