Lower bounds in a parallel model without bit operations[1]

(To appear in the SIAM Journal on Computing)

Ketan Mulmuley [2]
The University of Chicago
email: mulmuley@cs.uchicago.edu
web address: http://www.cs.uchicago.edu/~mulmuley

May 27, 1997

[1]Parts of this paper appeared in the proceedings of the ACM Symposium on Theory Of Computing, 1994 and 1997.
[2]The work in this paper was partially supported by David and Lucille Packard Foundation fellowship.

Contents

Abstract

We define a natural and realistic model of parallel computation called *the PRAM model without bit operations.* It is like the usual PRAM model, the main difference being that no bit operations are provided. It encompasses virtually all known parallel algorithms for (weighted) combinatorial optimization and algebraic problems. In this model we prove that, for some large enough constant b, the mincost-flow problem for graphs with n vertices cannot be solved deterministically (or with randomization) in $\sqrt{n}/b$ (expected) time using $2^{\sqrt{n}/b}$ processors; this is so even if we restrict every cost and capacity to be an integer (nonnegative if it is a capacity) of bitlength at most an for some large enough constant a. A similar lower bound is also proved for the max flow problem. It follows that these problems cannot be solved in our model deterministically (or with randomization) in $\Omega(N^c)$ (expected) time with $2^{\Omega(N^c)}$ processors, where c is an appropriate positive constant and N is the total bitlength of the input. Since these problems were known to be P-complete, this provides concrete support for the belief that P-completeness implies high parallel complexity, and for the $P \neq NC$ conjecture. Our lower bounds also extend to *the PRAM model with limited bit operations*, which provides instructions for parity and left or right shift by one bit.

Our proof is based on basic algebraic geometry. So we investigate if the algebro-geometric approach could also work for the P vs. NC problem. Our results support this possibility, and a close analysis of the limitation of our technique in this context suggests that such a proof of $P \neq NC$ should somehow use geometric invariant theory in a deep way.

Chapter 1

Overview

1.1 Introduction

One fundamental question in computer science is whether $P = NC$. It is generally believed that the answer is no. We are specifically interested in weighted combinatorial optimization problems in P, such as mincost-flow and max-flow, which have the so-called strongly polynomial time algorithms [17]–these are fast, i.e., polynomial time, sequential algorithms that do not use bit operations and the number of whose basic arithmetic steps depends polynomially only on the number of input parameters, not on the input bitlength. For these problems it is natural to ask if they also have fast parallel algorithms that do not use bit operations. Of course, if we assume that $P \neq NC$ then the answer for the mincost flow and maxflow problems is no, whether bit operations are available or not, because they are P-complete [15]. In this paper we prove this unconditionally, i.e., without assuming $P \neq NC$, when bit operations are not available. This provides concrete support for the belief that P-completeness implies high parallel complexity, and for the $P \neq NC$ conjecture itself, by proving its implications in a model that is restricted, but realistic.

The model

What do we mean by a parallel algorithm that does not use bit operations? Formally, we mean an algorithm in a *PRAM model without bit operations* (Chapter 2). It is like the usual PRAM model, the main difference being that it does not provide instructions for any bit operations, such as $\wedge$, $\vee$, or *extract-bit*. The model provides usual arithmetic $(+, -, \times)$, comparison $(=, \leq, <)$, store, indirect reference and branch operations. Each memory location can hold an integer; a rational number is represented by a pair of integers–its numerator and denominator–both of which can be accessed by the processors separately. The processors communicate as in the usual PRAM model. The model does not provide instructions for truncation or integer division with rounding. The lower bounds are proved in the conservative unit-cost model in which each operation is assigned unit cost regardless of the bitlengths of the operands. The PRAM model does not address satisfactorily the issue of interprocessor communication, so important in practice, but it is ideal for proving lower bounds since a lower bound in this model automatically applies to any other realistic parallel model of computation.

Though our model does not provide bit operations, the issue of bitlengths is not ignored. This means two things. First, the running time of an algorithm (in the unit cost model) is allowed to depend on the input bitlength. For example, the running time of Neff's parallel algorithm [38] for computing approximate roots of a polynomial–which lies in our model–depends on the input bitlength and the bit precision level desired in the output. Second, our lower bounds are expressible in terms of the total bitlength of the input and are applicable even when this bitlength is small, i.e., polynomial in the number of input parameters. In contrast, the lower bounds in the models such as the algebraic computation tree model [45, 52] are in terms of the number of input parameters and require their bitlengths to be extremely large or unbounded to be applicable.

Unlike some earlier models used for proving lower bounds, such as the constant-depth [12, 19, 51] or monotone circuit model [41, 1, 5], the PRAM model without bit operations is realistic and natural. It encompasses virtually all known parallel algorithms for weighted optimization and algebraic problems. These include fast parallel algorithms for solving linear systems [11], minimum-weight-spanning trees [31], shortest paths [31], global mincuts in weighted, undirected graphs [29, 30], blocking flows and max-flows [13, 44], approximate computation of roots of polynomials [3, 38], sorting [31], and several problems in computational geometry [42].

If in our model we require that the running time of a parallel algorithm depend only on the number of input parameters but not on their bitlengths, the resulting *the arithmetic PRAM model* is much weaker but still very interesting; many of the preceding parallel algorithms [11, 13, 29, 30, 31, 44] actually belong to this model. Our lower bounds in the PRAM model without bit operations automatically yield ones in the arithmetic PRAM model too: just ignore the bitlength restrictions in their statements.

Lower bounds

We show that, for some large enough constant b, the mincost-flow problem for networks with n nodes cannot be solved in the PRAM model without bit operations deterministically (or with randomization) in $\sqrt{n}/b$ (expected) time using $2^{\sqrt{n}/b}$ processors; this is so even if we restrict every cost and capacity to be an integer (nonnegative if it is a capacity) of bitlength at most an for some large enough constant a. We prove the same lower bound for the max flow problem, which holds even if we restrict every edge-capacity to be a nonnegative integer of bitlength at most an^2 for some large enough constant a. The restriction on the bitlengths implies that these problems cannot be solved in our model deterministically (or with randomization) in $\Omega(N^c)$ (expected) time with $2^{\Omega(N^c)}$ processors, where c is an appropriate positive constant, and N is the total bitlength of the input. Our lower bound also applies to the weighted s-t-mincut problem (for directed or undirected graphs) since it is the dual of the maxflow problem.

Our lower bounds hold for both the decision and the additive approximation versions of these problems. In the decision version, one is given an additional integer parameter w, called *threshold*, and the problem is to decide if the optimum exceeds w (thus for maxflow, the problem is to decide if the max flow value exceeds w; for mincost flow, the problem is to decide if there exists a flow with value v, specified in the input, and cost at most w.) In the additive approximation version, the goal is to compute the optimum within a small, say less than $1/8$, *additive* error.

Our lower bounds also imply some separation results of complexity theoretic nature. Let SP be the class of languages in P that have strongly polynomial time algorithms [17]; for example, the mincost flow and max flow problems belong to SP. (By a language we mean in this paper a set of integer sequences rather than boolean strings.) Let $C[t(N), p(N)]$ be the class of languages that can be recognized in the PRAM model without bit operations in $t(N)$ time (in the unit cost model) using $p(N)$ processors, where N denotes the input bit length. The randomized class $RC[t(N), p(N)]$ is defined similarly. Let $C^i = C[O(\log^i(N)), \text{poly}(N)]$ be the analogue of the class NC^i, and $C = \cup_i C^i$ the analogue of the class NC. For detailed definitions of these complexity classes see Section 2.3.

In our model an analogue of the $P \not\subseteq NC$ conjecture would be that $SP \not\subseteq C$. This is not just an analogue but also an implication, i.e., $P \not\subseteq NC$ implies that $SP \not\subseteq C$ (Proposition 2.3.1). Since the mincost flow and max flow problems belong to SP, our lower bounds imply unconditionally that $SP \not\subseteq C$; in fact, something much stronger:

$$SP \not\subseteq RC[N^c, 2^{N^c}],$$

for a suitable positive constant c.

We also prove that the hierarchies $\{C^i\}$ and $\{RC^i\}$, which are analogues of the hierarchies $\{NC^i\}$ and $\{RNC^i\}$, do not collapse. This follows from the following stronger statement: For some positive constant c,

$$C^i \not\subseteq RC^{\lfloor i/c \rfloor + 1},$$

for all i.

PRAM with limited bit operations

The $P \neq NC$ conjecture is hard essentially because of the issue of bits. There are actually two issues: (1) the running time of a parallel algorithm can depend on the input bitlength, and (2) the algorithm can access an arbitrary bit of the operand. The first issue is addressed satisfactorily in our model, the second issue is not. The higher order bits of an operand are easy to access in our model–it is the middle and lower order bits that are hard (though not impossible) to access

(Section 2.2). We can extend our lower bound to the *PRAM model with limited bit operations*, which provides instructions for parity and left or right shift by one bit; in this model even the lower order bits are easy to access. This is as far as one can go. If one allows shift by an arbitrary number of bits the model would become as powerful as the usual unrestricted PRAM-model, and that is beyond the scope of our techniques for reasons outlined in Chapter 7. Parity is a nonalgebraic operation; or, if it is considered algebraic, it has high exponential degree. So it is interesting that our techniques can handle it.

P vs. NC

Our lower bounds are proved using basic algebraic geometry (Milnor-Thom result [33], Collins' decomposition [10] and transversality techniques [16]) in conjunction with diophontine techniques over integer lattices. We also need lower bounds for the so-called parametric complexity (Chapter 3) of the mincost flow and maxflow problems; these were proved earlier in some other contexts combinatorially [37, 14, 56, 7] (see also Theorem 3.2.3). Our proof should be contrasted with the combinatorial proofs of the lower bounds for, say, constant-depth [12, 19, 51] or monotone circuits [41, 1, 5, 55]; and also with the proofs of the lower bounds in the algebraic computation tree model [2, 45, 52]–these are for sequential sorting-related problems, and the issue of bit lengths is not addressed.

In view of the effectiveness of the algebro-geometric approach in a restricted, yet realistic PRAM model of computation in this paper, we investigate its potential in the unrestricted PRAM model. Towards this end, we formulate a certain conjecture (Chapter 7), which if true would imply that $P \neq NC$. Roughly, the conjecture says that, for every positive constant a, linear pullbacks of the well known algebraic group variety $SL_m(C)$, where m satisfies some dimension constraints, cannot have a certain *separation property* that depends on the constant a. We then prove the conjecture for every large enough constant a. Our technique is insufficient for proving the conjecture in full generality, i.e., for every positive constant a–as expected, since it relies on the restricted nature of the PRAM model without bit operations. But a careful analysis of its limitation reveals what a proof of $P \neq NC$ ought to exploit: it should somehow use properties of varieties admitting good group actions–as investigated in geometric invariant theory [6, 36]–in a deep way. (In the terminology of [43], these techniques would not be "natural", but calling them so would not do justice to their depth and beauty. The framework of [43] is not meaningful in the PRAM model without bit operations in this paper just as it is not in the monotone circuit model[1].)

[1] This is because a lower bound in our model is meaningful only for a language that can be recognized by an efficient sequential algorithm that does not use bit operations (Section 2.2); for a language in P, this means it should have a strongly polynomial time algorithm. But there is no natural way to define a random language of this kind as would be required in [43] for its largeness criterion to make sense.

1.2 The issue of bitlengths

Our lower bounds for mincost flow and maxflow depend on the fact that the edge-capacities can have polynomial–i.e. $O(n)$ or $O(n^2)$–bitlengths. This is necessary: if the bitlengths were small, these problems can, in fact, be solved fast in parallel in our model. For example, consider the max flow problem. Suppose all edge-capacities have $O(\log n)$ bitlengths. Then in our model all these bits can be extracted in parallel in logarithmic time using the available instructions (Section 2.2). Once all bits in the input are extracted, the model becomes as powerful as the usual unrestricted PRAM model. One can thereafter use an RNC-algorithm for unweighted matching [24, 35] to get a polylogarithmic time algorithm. More generally, if all edge-capacities have polylog(n) bitlengths, the problem can be solved in our model in polylog(n) time using $2^{\text{polylog}(n)}$ processors; by a variation of our lower bound, this is close to optimal. The problem can also be solved in $O(n^2 \log n)$ time with $O(n)$ processors without any restriction on the bitlengths by a parallel algorithm of Shiloach and Vishkin [44], or of Goldberg and Tarjan [13]–both of which lie in our model. Our lower bound should be contrasted with these upper bounds.

Recall that a strongly polynomial time sequential algorithm [17] is a polynomial time algorithm with poly(n) basic arithmetic steps, where n is the number of input parameters, not the input bitlength. Similarly, let us define a *strongly polylogarithmic* time parallel algorithm to be an algorithm in the PRAM model without bit operations which runs in polylog(n) time. For example, the algorithms of Karger[29], and Karger and Motwani [30] for the global mincut problem for weighted undirected graphs are strongly polylogarithmic. Our lower bound implies that the mincost flow and max flow problems, which have strongly polynomial time sequential algorithms, have no analogous strongly polylogarithmic time parallel algorithms; but actually it even rules out in our model algorithms that work in $\Omega(N^c)$ (expected) time with $2^{\Omega(N^c)}$ processors, where c is a small enough positive constant and N is the total bitlength of the input. In other words, for each of these problems it rules out a fast parallel algorithm that has the main characteristic of the ellipsoid algorithm: namely, even if it does not use any bit operations, its running time can depend on the input bitlength–this may happen if the algorithm computes the optimum value or some other numerical quantities with high precision. One example of a parallel algorithm in our model that has this characteristic is Neff's algorithm [38] for computing approximate roots of polynomials.

Our lower bound also rules out in our model a fast parallel algorithm for mincost flow or max flow that computes the optimum value approximately with high precision–this is what the interior-point algorithm [22] does–and then rounds it up to get the exact value. This is because our lower bound also applies to the approximate computation of the optimum value within a small, say less than 1/8, *additive* error. Hence, even though rounding is not available in our model, the final rounding can be ignored: all that matters is that the answer just before this rounding is within a small additive error of the exact answer.

Since truncations are not available in our model, they cannot be used to control the bitsizes, as in the ellipsoid or the interior-point algorithm. But our lower bounds are proved in the unit cost model, so the algorithm need not worry about intermediate bitsizes. The ellipsoid and the interior-point algorithms also perform some basic algebraic operations, such as solving linear systems and computing approximate roots of polynomials–these can be simulated fast in parallel in our model using the algorithms in [11] and [3, 38], respectively.

1.3 Statement of the results

Now we shall formally state our main results.

Theorem 1.3.1 *The mincost-flow problem for networks with k nodes cannot solved in the PRAM model without bit operations deterministically (or with randomization) in $\sqrt{k}/b$ (expected) time using $2^{\sqrt{k}/b}$ parallel processors, even assuming that every cost and capacity is an integer with bit length at most ak for some large enough positive constants a and b.*

Similarly the max flow problem for directed or undirected networks with k nodes cannot solved in the PRAM model without bit operations deterministically (or with randomization) in $\sqrt{k}/b$ (expected) time using $2^{\sqrt{k}/b}$ parallel processors, even assuming that every capacity is a nonnegative integer of bit length at most ak^2 for some large enough positive constants a and b. The same lower bound also applies to the dual s-t-mincut problem for weighted graphs.

All lower bounds apply for the decision as well as the additive approximation versions of the problems. The lower bounds also apply in the PRAM model with limited bit operations.

The total bit size N of a network with k-nodes and $O(k)$ or $O(k^2)$-bit edge-capacities and costs is polynomial in k. This allows us to express the preceding lower bounds in terms of the input bitlength N:

Corollary 1.3.2 *The min-cost flow or the max flow problem cannot be solved in the PRAM model without bit operations (or with limited bit operations) deterministically (or with randomization) in $\Omega(N^c)$ (expected) time using $2^{\Omega(N^c)}$ processors for some positive constant c.*

Remark. The theorem implies $c = 1/6$ for min cost flow and $c = 1/8$ for max flow; for min cost flow the value of c can be reduced to $1/4$ (see Section 3.1.1).

Since the min cost flow problem belongs to the class SP, we also conclude:

Corollary 1.3.3

$$SP \not\subseteq RC[N^{1/4}/b, 2^{N^{1/4}/b}],$$

for a large enough constant b.

We shall also prove that the $\{C^i\}$ and $\{RC^i\}$ hierarchies do not collapse; these are analogues of the conjectures that the $\{NC^i\}$ and $\{RNC^i\}$ hierarchies do not collapse.

Theorem 1.3.4 *$C^i \not\subseteq RC^{i/c}$, for some positive constant c.*

It follows that:

Corollary 1.3.5 *The $\{C^i\}$ and $\{RC^i\}$ hierarchies do not collapse.*

For the P vs. NC problem and related results, see Chapter 7.

1.4 Basic idea of the proof

We shall actually prove a general result (Theorem 3.1.1) which says that if the so-called parametric complexity of the problem is high, then it is hard to parallelize in our model. This implies our lower bounds for the mincost flow and max flow problems because their parametric complexity is exponential, as was shown by Carstensen and Zadeh [7, 8, 56]. Our proof has two steps. In the first step we show that if the problem with high parametric complexity had a fast parallel algorithm in our model then there would exist a certain low-degree algebraic decomposition of a small subset of the three dimensional integer lattice. In the second step–the heart of the proof–we show that such algebraic decomposition cannot exist. We shall now elaborate this a bit more.

Let n denote the number of integer parameters in the input and N the total bit length of the input. Let Z denote the set of integers. Suppose, to the contrary, that there is a (nonuniform) machine M in the PRAM model without bit operations that efficiently solves a given combinatorial optimization problem with high parametric complexity for all inputs with number of parameters n and bit length at most N. Let $L \subseteq Z^n$ denote the language corresponding to the problem; i.e., L is the set of all points in Z^n for which the answer is yes. For reasons that will become clear soon, we shall parametrize input to this machine using a constant d number of integer parameters. The parametrization is given by an integral linear map $I : Z^d \to Z^n$. Thus any integer point $z \in Z^d$ corresponds to the input $I(z)$ to the machine. For example, if L is the max flow language, one can choose a suitable flow network whose edge-capacities are integral linear forms in two integer parameters z_1 and z_2, and let z_3 denote the threshold. This specifies a linear map from $z = (z_1, z_2, z_3)$ to the input $I(z)$ consisting of a flow network and a threshold.

Since the machine works correctly only for inputs of bit length at most N, we shall only be interested in those $z \in Z^d$ such that the bit length of $I(z)$ is at most N–such z will be called *permissible.* Let us colour a permissible point $z \in Z^d$ green if $I(Z)$ belongs to the language L; otherwise colour it red. The points that are not permissible are colourless. The first step in our proof (Theorems 4.1.4 and 5.1.6) is to show that if M recognizes L efficiently then the green and the red points in Z^d can be *separated* by a set of algebraic (hyper)surfaces of "small" total degree. Here the total degree depends exponentially on d; so we need d to be as small as possible–in fact, we shall be able to choose $d = 3$ in our parametrization. This first step uses the Milnor-Thom result [33] from algebraic geometry, which has also been used earlier [2, 45] in the algebraic computation tree model.

When the parametric complexity of the problem is high, one can choose a parametrization I so that the green and the red points are distributed badly in some sense. The heart of our proof (Theorem 5.3.1) lies in showing that when the permissible points are badly coloured, the red and the green points cannot be separated by any set of algebraic surfaces of small degree. The algebraic surfaces that arise in our proof can be highly singular, and they can intersect badly. So, without disturbing the separation property, first we smoothen the surfaces, and ensure that their intersections and "silhouettes" are also smooth using transversality techniques from differential geometry [16]. Next we refine the partition formed by these surfaces further by Collins' decomposition method [10] so that all regions in the resulting partition are "cylinders" of simple shape. Then using diophontine techniques, especially the pigeonhole principle, we show that if this partition were to separate the green and the red points then some region in it must be too warped. However, since all surfaces have small degree, this cannot happen–a contradiction.

It turns out that the parametric complexity of a combinatorial optimization problem is essentially the maximum number of pivot steps the simplex algorithm would need using the so-called Gass-Saaty pivot rule. It was studied in this and related contexts in operations research. Klee and Minty [26] had shown in their classic paper that the simplex algorithm can take exponentially many steps for some simple pivot rules. The same was shown for the Gass-Saaty pivot rule by Goldfarb[14] and Murty[37] for general linear programming, by Zadeh[56] for min cost flow, and by Carstensen[7, 8] for max flow. This provides exponential lower bounds for the parametric complexity of these problems. Lower bounds on their parallel complexity in our model then follow from our general lower bound (Theorem 3.1.1) mentioned above.

1.5 Organization of the paper

In Chapter 2 we describe the PRAM model without bit operations formally. In Chapter 3 we explain the connection between parametric and parallel complexity. In Chapter 4 we first prove our general lower bound in a restricted *Linear PRAM model without bit operations*; this model does not provide general multiplication. The proof here is elementary and shows the basic ideas in a very simple setting. Moreover, we even get a stronger lower bound that is sensitive to processor-time tradeoff. In Chapter 5 the proof is extended to the general PRAM model without bit operations. In Chapter 6 we extend the lower bound to randomized algorithms and to the PRAM model with limited bit operations. In Chapter 7 we formulate an algebro-geometric conjecture that would imply $P \neq NC$ and prove its special case. A proof of $P \neq NC$ may not go via such conjecture; it has been formulated to bring out what is lacking in our current approach and what more is needed.

Chapter 2

The model

Here we shall formally define *the PRAM model without bit operations*. This model is like the usual PRAM model [23], the essential difference being that the instruction set of processors does not contain any bit operations.

Input to the machine is supposed to be a sequence of integer, rather than boolean, numbers. In other words, each integer in the input is used as a specification for itself, each rational number is specified in terms of its numerator and denominator, and the remaining information, such as graph specification, is encoded as an integer sequence of zeroes and ones. The input is supposed to be divided in two parts: nonnumeric data, and numeric data. For the max flow problem, the nonnumeric data would specify the underlying graph, and the numeric data would specify the nonnegative edge-capacities. Let n denote the cardinality of the input, i.e., the total number of its integer parameters, and N its the total bitlength.

2.1 PRAM without bit operations

First, we shall define the model in the nonuniform setting. A nonuniform machine for cardinality n and bitsize N is supposed to work correctly on all inputs of cardinality n and bitlength at most N. Our machine will consist of $p(n, N)$ processors ordered in some fashion, where $p(n, N)$ is some function of n and N. The processors have private memories, and they communicate through a shared global memory. EREW, CREW, and CRCW modes of communication are defined as usual. Each memory location can contain an integer of any size that can arise in the course of execution. Accordingly, instructions can be assigned costs that depend on the bitlengths of the operands. This is important when one proves an upper bound on the running time of an algorithm. However, our lower bounds also work in a stronger sense, when one assigns unit cost to each instruction regardless of the bitsizes of the operands; since the actual cost of any instruction is $\Omega(1)$, such lower bounds hold in any cost model. For the purpose of lower bounds, we shall assume this unit-cost model in the paper.

In our machine, a rational number is represented as a pair of its denominator and numerator, which need not be relatively prime, and each of which can be accessed separately by a processor. Rational operations, including division, can then be simulated by integer operations. It is important that the machine can access the numerator and the denominator separately; if we were to treat a rational number as an abstract data type (i.e. a black box) so that only its value is visible to the outside world, then the model would be far weaker. Initially, the shared memory will contain n, the cardinality of the input, in its first location, followed by the nonnumeric data, followed by the numeric data. The remaining memory locations are initialized to zero. At the end of execution, the first location in the shared memory should contain the output value.

In the nonuniform setting, it is best to think that the program of each processor has been unfolded in the form of a computation tree, quite like in the algebraic-computation-tree model [2, 45]. This means the length of a program can depend on n and N. Each node in this tree has a label and an instruction associated with it. No instructions for bit manipulations are provided. The instructions are of the following kinds:

1. $w = u \circ v$, where w denotes a memory location, u and v denote either memory locations or constants, which in the nonuniform setting can depend on n and N. Finally, $\circ$ denotes a binary operation $+, -$ or $\times$. In the nonuniform setting the memory locations can depend on n and N.

2. **go to** l, where l is a label of some instruction.

3. **if** $u : 0$ **go to** l, where u denotes a memory location and $:$ denotes either $<, \leq$ or $=$.

4. $u := v$: store contents of the memory location v into the location u.

5. $u :=\uparrow v$ (Indirect reference): Treat the value in the memory location v as a pointer. Store into the memory location u the value in the pointed memory location. We assume that the value in v is a valid pointer in a sense specified below.

6. **stop**.

We assume that the pointer involved in an indirect reference is not some numeric argument in the input or a quantity that depends on it. For example, in the max flow problem the algorithm should not use an edge-capacity as a pointer–which is a reasonable condition. To enforce this restriction, one initially puts an *invalid-pointer* tag on every numeric argument in the input. During the execution of an arithmetic instruction, the same tag is also propagated to the result if any operand has that tag. Trying to use a memory value with invalid-pointer tag results in error.

To allow randomization, we provide an additional instruction: *random-branch* l. When it is encountered, the processor flips a fair coin. If the toss is head, it branches to the instruction labeled l; otherwise, it proceeds to the next instruction. We allow randomized algorithms two sided errors.

In this model the higher order bits of an operand can be easily accessed using available instructions, but the lower order bits are hard to access (Section 2.2): more precisely, the l-th bit from the left can be extracted in $O(l)$ operations. Our model does not provide integer division with rounding, because if it did arbitrary bit in the input could be extracted fast, and the model would become as powerful as the usual PRAM model. Nevertheless, small roundings, in which the rounded quotient is guaranteed to have small, say polylogarithmic, bit size, can be simulated efficiently using available operations. Since general truncations are not provided in our model, one can not use them to control bitsizes as in the ellipsoid or the interior-point algorithm. This does not matter, however, because our lower bounds are proved in the unit cost model, so the algorithm need not worry about intermediate bitsizes.

The model just described was nonuniform. In its uniform version we assume that there is a Turing-machine that, given n and N, generates programs for all $p(n, N)$ processors using space that is logarithmic in their total size. In the uniform version we also assume that the size of each processor's program is constant; in other words, it is not meant to be unfolded in the form of a computation tree. Since we have provided indirect reference, unfolding is not necessary.

2.1.1 Arithmetic PRAM model

It is the model obtained when one restricts the PRAM model without bit operations by requiring the running time $t(n, N)$ and the number of processors $p(n, N)$ to depend only on n, but not on the bitlength N. In this model the issue of bitlengths is ignored just as in Yao's algebraic computation tree model for integer inputs [52]. Nevertheless it is interesting because many parallel algorithms that have been designed for (weighted) combinatorial optimization or algebraic problems lie in this model. If one is only interested in showing that a given problem does not have a strongly polylogarithmic time parallel algorithm, as defined in Section 1.2, then this is the model to be used. All our lower bounds automatically hold in this weaker model–just ignore the bit-size restrictions in their statements.

2.1.2 Linear PRAM model

This is the model that results if we restrict the PRAM model without bit operations by requiring that at least one operand of the $\times$ instruction be a constant. If in addition we require that that the running time $t(n, N)$ and the number of processors $p(n, N)$ depend only on n, but not on the bit length N, then the resulting model is called *the arithmetic linear PRAM model.*

2.1.3 Examples

We now list various versions of our models in the increasing order of power. For each model we give examples of algorithms which lie in that model but not in the weaker model preceding it.

Arithmetic linear PRAM model: Contains parallel algorithms for many weighted combinatorial optimization problems, such as shortest paths [31], minimum weight spanning trees [31], max-flows and blocking flows [44, 13], global mincuts in undirected weighted graphs [30], and many problems in computational geometry [42].

Arithmetic PRAM model: Also contains parallel algorithms for solving linear systems over rationals and other problems in linear algebra, due to Csanky [11] and others (e.g. [21]), and for evaluating arithmetic circuits (straight line codes) over rationals [54]. In Csanky's algorithm [11], as it is, the bitsizes of intermediate operands can become superpolynomial in the bitlength of the input. This is fine, since we are using the unit cost model. (Of course, one can keep the bitsizes under control, but this requires integer division with rounding, an operation not available in our model.)

PRAM model without bit operations: Also contains Neff's algorithm for computing approximate roots of polynomials [38], the earlier algorithm of Ben-Or et al. [3] for a restricted class of polynomials, the numerical algorithm of Pan and Reif [39] for solving linear systems iteratively, and numerical parallel algorithms for several other problems (see the book [4] for a survey). These algorithms use truncations, which are not available in our model, to keep intermediate bitsizes under control; but, again, this is not necessary in our unit-cost model, as we already pointed out.

We do not know a natural example that is in the linear PRAM model but not in the arithmetic linear PRAM model–in this paper the linear PRAM model is used more as a medium for showing basic ideas of the proof in an elementary setting.

As one nontrivial example in weighted combinatorial optimization, let us explain why Karger and Motwani's NC-algorithm [30] for computing global mincuts in undirected weighted graphs falls in our model–in fact, it lies in the arithmetic PRAM model. It begins by normalizing weights in the graph so that they have polynomial magnitudes, i.e., logarithmic bitsizes. This requires rounding a ratio of two integers to the nearest integer, where the ratio is guaranteed to have logarithmic bitsize. Such small roundings can be carried out efficiently in our model in logarithmic time using the available operations. After this, Karger and Motwani give an NC-algorithm to compute all approximate global mincuts in the resulting normalized graph. But since all bits of the edge-weights in the normalized graph are now known to our algorithm, it can simulate any algorithm in the unrestricted PRAM model hereafter. So the subsequent NC-computation on the normalized graph can be simulated in our model without any problem. Once all approximate global mincuts in the normalized graph are computed–which, by Karger's result [29], are polynomial in number–Karger and Motwani's algorithm computes the weights of these cuts in the original graph, and selects one with minimum weight. This too poses no problem in our model.

2.1.4 Algebraic operations

Since the parallel algorithms in [3, 38] for computing of approximate roots of polynomials lie in the PRAM model without bit operations, this operation can be simulated efficiently. So all our lower bounds also hold, with appropriate modifications to take into account the cost of simulation, even if we were to allow this approximate algebraic operation in the model. In the sequential setting it plays a crucial role in the ellipsoid and interior-point algorithms.

2.1.5 PRAM with limited bit operations

This is obtained by extending the PRAM model without bit operations by adding to the instruction set operations for left or right shift by one bit and parity–or equivalently an assignment operation $u := \lfloor u/2 \rfloor$, or, more generally, an assignment $u := \lfloor u/p \rfloor$ for any constant p. If we were to allow shift by arbitrary number of bits, the model would become as strong as the unrestricted PRAM model.

2.2 Bit extraction

In the arithmetic PRAM model the running time of an algorithm is required to depend only on the cardinality n but not on the total bitlength N, so extracting arbitrary bit of the operand is, in general, impossible. The situation here is then like in the monotone circuit model [41], where the *not* operation is impossible, or in Yao's algebraic computation tree model for integer inputs [52], where bit extraction is impossible.

In the PRAM model without bit operations, bits can be extracted using the available instructions–but this can be hard, though not impossible. More precisely, all bits of an operand x with bitlength l can be extracted in $O(l)$ time sequentially as follows. First estimate the bit length l in $O(\log l)$ operations, starting with 2 and squaring repeatedly until the value of x is exceeded. After this the bits of x can be extracted one by one, starting at the most significant bit, using available arithmetic and comparison operations. More generally, all bits can be extracted in $O(\sqrt{l})$ time using $2^{\sqrt{l}}$ processors as follows. Conceptually divide x into $\sqrt{l}$ blocks of equal length, and extract the blocks one by one, starting with the most significant block. For example, to extract the most significant block, one can guess in parallel all of its possible values. Whether a guessed value is the correct one can be verified easily using the available operations.

However if l is large, bit extraction is hard. In fact, provably so (Proposition 2.2.1). Of course, a lower bound for bit extraction in our model does not have much meaning as a lower bound statement–saying that bit extraction is hard in the PRAM model without bit operations is analogous to saying that the *not* operation is impossible in the monotone circuit model, or that bit extraction is impossible in the algebraic computation tree model or in the arithmetic PRAM model. Just as a lower bound in the monotone circuit model is interesting only if the problem is monotone, a lower bound in the PRAM model without bit operations is interesting only if the problem has an efficient sequential algorithm that does not use bit operations: for a problem in P, this means it should have a strongly polynomial time algorithm [17] (see also Section 2.3 for more discussion of this issue). This is true of all combinatorial optimization problems we consider. One trivial example of a problem that does not have a strongly polynomial time algorithm is bit extraction itself (because such an algorithm for bit extraction must work within $O(1)$ arithmetic and comparison operations, which is impossible).

Proposition 2.2.1 *The lowest order bit of an n-bit operand can not be extracted in the PRAM model without bit operations in $\sqrt{n}/a$ time using $2^{\sqrt{n}/a}$ processors, for some large enough constant a.*

Proof. Suppose, to the contrary, that there is a parallel machine M in the model that outputs the value of the lowermost bit in $t(n) = \sqrt{n}/a$ time using $p(n) = 2^{t(n)} = 2^{\sqrt{n}/a}$ processors; here a is a large enough constant to be chosen later. Given a nonnegative integer t, call two inputs x and $\bar{x}$ t-equivalent if the behaviour of the machine M up to time t on the input x is the same as that on the input $\bar{x}$; more formally, the branches taken by all processors in M up to time t are the same for both x and $\bar{x}$. We say that x and $\bar{x}$ are equivalent if they are $t(n)$-equivalent.

Now assume that the input x to the machine is guaranteed to be in a fixed t-equivalence class σ. This fixes all branches taken by every processor in M up to time t; then each memory location is a certain fixed polynomial in the input x, and the degree of this polynomial is at most 2^t. Now consider all comparisons, at most $p(n)$ in number, that M would execute at time $t+1$. Since the value of each memory location at time $t+1$ is a predetermined polynomial in x, once we are told that $x \in \sigma$, each such comparison amounts to evaluating the sign of some polynomial, which is $+$, $-$, or 0. Let $f_1(x), \ldots f_p(x)$, where $p \leq p(n)$, be these polynomials. Each $f_i(x)$ has degree at most 2^t and hence at most 2^t real roots. Consider the set of all roots of all polynomials $f_i(x)$, $i \leq p$. They divide the real line into at most $2^t p(n) + 1$ intervals. If $x_1, x_2 \in \sigma$ lie in the same open interval, or if they are equal and coincide with some root, then the signs of $f_i(x_1)$ and $f_i(x_2)$ coincide, for each i, and consequently all processors in M would take the same branching path at time $t+1$. This means that any t-equivalence class σ is split into at most

$$2 \cdot 2^t p(n) + 1 \leq 2 \cdot 4^{t(n)} + 1$$

$(t+1)$-equivalence classes. By induction, it follows that the total number of equivalence classes, i.e., $t(n)$-equivalence classes is at most $(2 \cdot 4^{t(n)} + 1)^{t(n)} < 5^{t(n)^2}$.

Now consider a fixed equivalence class ζ. Once ζ is fixed, the branches taken by all processors during the entire execution of M are fixed, and the comparison carried out at every such branch corresponds to evaluating the sign of a fixed polynomial in x of degree at most $2^{t(n)}$. Let $P(\zeta)$ denote the set of all these polynomials that correspond to all branches of all processors during the entire execution. The size of $P(\zeta)$ is at most $t(n)p(n)$. Let P be the union of all $P(\zeta)$, as ζ ranges over all equivalence classes, at most $5^{t(n)^2}$ in number. The size of P is at most $t(n)p(n)5^{t(n)^2}$, and each polynomial in P has degree at most $2^{t(n)}$. Consider the set R of real roots of all polynomials in P; its size is at most $t(n)p(n)5^{t(n)^2}2^{t(n)}$. These roots divide the real line into at most

$$t(n)p(n)5^{t(n)^2}2^{t(n)} + 1 < 6^{t(n)^2} = 6^{n/a^2}$$

intervals. If two inputs x_1 and x_2 lie in the same open interval, then the behaviour of all processors in M on the input x_1 is the same as that on x_2; so the output would be the same too–in other words, the lowermost bits of x_1 and x_2 must be the same. However, we have already seen that the number of intervals is at most $6^{n/a^2}$. If a is large enough, this number is much less than $2^n/2$. But then, by the pigeonhole principle, at least one such open interval would contain two integers x_1 and x_2 with the same lowermost bit–a contradiction. $\square$

More generally, the same method also proves that extracting the i-th bit from the left is hard for every "large" i. In the PRAM model with limited bit operations, one can prove similarly that the middle bits are hard to extract.

The essential weakness of a machine in our model lies in accessing bits. But if the language is boolean, i.e., if all its input parameters are either 0 or 1, then all bits of the input are available to our machine right at the outset. In this case, our model is as powerful as the unrestricted PRAM model. So proving a lower bound for a boolean language in our model is equivalent to proving it in the unrestricted PRAM model. On the other hand, using our model for proving a lower bound for a boolean language hardly makes sense: if one wants to use an algebraic approach for proving lower bounds, one should choose a language for which the approach makes sense.

2.3 Complexity Classes

We shall now formally define, both in the sequential and the parallel setting, complexity classes of problems having efficient algorithms that do not use bit operations; but their running times may or may not depend on the input bit lengths and we need to distinguish between these two cases carefully.

First consider the sequential setting. If we allow the running time to depend on the bit length then one needs to be very careful. One can not simply define a sequential algorithm that does not use bit operations to be an algorithm in the restricted RAM model without bit operations. This is because all bits in the input of bit length N can be extracted in this model sequentially with $O(n)$ arithmetic and comparison operations. After this, any algorithm in the unrestricted RAM model can be simulated in the restricted RAM model without bit operations. This means a polynomial time algorithm in the restricted RAM model without bit operations is as powerful as a polynomial time algorithm in the unrestricted RAM model. So it is somewhat tricky to define a fast sequential algorithm whose running time can depend on the input bit length, but which does not use bit operations in a "real sense". Intuitively, such an algorithm should be like the the ellipsoid or the interior point algorithm for linear programming [22, 25] (the truncations that occur in these algorithms for controlling bit lengths are not bit operations in "a real sense"). This can be made precise and formal, but we shall not do so here.

If we do not allow the running time to depend on the input bit length, the matter becomes much simpler. A sequential algorithm that does not use bit operations and whose running time does not depend on the input bit length either is nothing but a *strongly polynomial time* algorithm [17]: by this we mean that (1) the running time of the algorithm in the unit cost model is polynomial in the number of integer parameters in the input, and (2) the bit size of any intermediate operand that arises in the course of execution is bounded by a polynomial in the total bit length of the input.

Let $SP \subseteq P$ (Strongly Polynomial) be the class of languages in P which can be recognized by strongly polynomial time algorithms. (Recall that a language in this paper means a set of integer sequences rather than boolean strings.) The class SP is quite rich. It contains decision problems associated with solving linear systems, network flow problems such as min-cost flow and max-flow, minimum-weight matching, *combinatorial* linear programming [46], ϵ-approximate knapsack problem [20], 3/2-approximate travelling salesman problem [9], and large-enough-constant-factor approximation of weighted MAX-SNP problems [40].

Next, let us turn to the parallel setting. Here an analogue of the class SP is the class SNC (Strong-NC). It is the class of problems that have *strongly polylogarithmic time algorithms*, i.e., the algorithms in the PRAM model without bit operations that work in polylog(n) time using poly(n) processors, where n denotes the number of integer parameters in the input. Since we are following the unit cost model, we are not requiring the bit lengths of intermediate operands to remain polynomial in the input bit length. So strictly speaking SNC need not be a subclass of NC. The class SNC is also quite rich. Some problems in this class are: solving linear systems over rationals [11], minimum-weight-spanning trees [31], shortest paths [31], global mincuts in undirected graphs [30], and several problems in computational geometry such as constructing convex hulls [42].

Now let us allow the running time of a parallel algorithm to depend on the input bit length; but it should not use any bit operations. Formally, let C be the class of problems that can be solved in the PRAM model without bit operations in polylog(N) time using poly(N) processors, where N denotes the input bit length. Again, there is no restriction on the bit lengths of intermediate operands, so C need not be a subclass of NC. An important problem that is in the class C but not in SNC is that of computing approximate roots of real polynomials up to a specified bit-precision level [3, 38]. Neff's algorithm [38] uses truncations, which are not available in our model, to keep bit sizes of intermediate operands under control. But they can be ignored. Then bit sizes of intermediate operands can become superpolynomial, which is fine since we are following the unit cost model. This also explains why we did not impose any restriction on the bit sizes of intermediate operands.

More generally, let $C[t(n,N), p(n,N)]$ be the class of languages that can be recognized in the PRAM model without bit operations in $t(n,N)$ time in the unit cost model using $p(n,N)$ processors, where n is the number of parameters in the input and N is the total bit length of the input. Let $C^i = C[O(\log^i(N)), \mathrm{poly}(N)]$ be the analogue of the class NC^i. One can define the randomized class RC, or more generally, $RC[t(n,N), p(n,N)]$ similarly. The class RC is the analogue of the class RNC.

In our model, an analogue of the $P \not\subseteq NC$ conjecture would be that $SP \not\subseteq SNC$, or more strongly, that $SP \not\subseteq C$. This is not just an analogue but also an implication:

Proposition 2.3.1 *$P \not\subseteq NC$ implies $SP \not\subseteq C$*

Proof. Suppose $P \not\subseteq NC$. Consider the restricted linear programming problem in which all integer parameters in the input have $O(1)$ bit sizes. It is known to be P-complete, so it cannot belong to NC. If it belonged C it would also belong to NC, because when all integer parameters in the input have $O(1)$ bit sizes the PRAM model without bit operations is as powerful the unrestricted PRAM model (Section 2.2). $\square$

Our results in this paper imply, unconditionally, that $SP \not\subseteq C$ and much more (Corollary 1.3.3). They also imply that the hierarchies $\{C^i\}$ and $\{RC^i\}$ do not collapse.

Chapter 3

Parametric vs. parallel complexity

All our lower bounds follow from a general lower bound (Theorem 3.1.1) on the parallel complexity of a general optimization problem in our model in terms of its parametric complexity. In this chapter we explore the relationship between parametric and parallel complexity in detail. In Section 3.1 we state our general lower bound and derive from it our main results stated in Section 1.3. In Section 3.2 we give a simpler proof of Carstensen's exponential lower bound [8] for the parametric complexity of the max flow and the dual weighted s-t-mincut problem. In Section 3.3 we prove a polynomial upper bound on the parametric complexity of the weighted global mincut problem for undirected graphs. This should explain why the weighted s-t-mincut problem for undirected graphs has no fast parallel algorithm in our model, whereas the global mincut problem has: e.g. the one in [29] or [30].

3.1 A general lower bound

First, let us define parametric complexity. For the maxflow problem it is defined as follows. Consider a graph with n vertices whose edge-capacities are linear functions of a parameter λ with rational coefficients of bitsize at most $\beta(n)$ for some function $\beta(n)$. We also require that all capacities remain nonnegative as λ varies over reals in some interval $[p, q]$ of definition. We are using real capacities only for the sake of defining parametric complexity; the actual capacities in the input will always be nonnegative integers. Let $f(\lambda)$ be the max flow value for a particular $\lambda \in [p, q]$. Let $\rho(f)$ be the number of breakpoints, i.e., slope changes in the function graph of f in the interval $[p, q]$. Parametric complexity $\phi(n, \beta(n))$ of the max flow problem for cardinality n and bitsize $\beta(n)$ is defined to be the maximum possible value of $\rho(f)$ over all valid edge-capacity parametrizations, as above, of graphs with n vertices.

One can similarly define parametric complexity of an arbitrary homogeneous optimization problem. So consider a general combinatorial optimization problem, where the input consists of two parts: nonnumeric data, and numeric data. Given an input I, the problem is to compute the optimum value of some quantity; we shall denote this optimum value for I by $F(I)$. For example, in the max flow problem the nonnumeric data would specify of the network, and the numeric data would specify the edge-capacities. Given an input I, $F(I)$ would be the max flow value. In the decision version of the optimization problem one is given an additional integer parameter w, called *threshold*, and the problem is decide if $w \leq F(I)$.

We will mainly be interested in the integral version of the optimization problem, wherein numeric parameters are integers. But for the sake of defining parametric complexity we assume that the quantity $F(I)$ is well defined even when the numeric parameters are reals, possibly with some constraints–for example, we can require them to be nonnegative. We also assume that the problem is homogeneous: this means if we multiply all numeric parameters in I by a positive number α, then the optimum value $F(I)$ also gets multiplied by α. For example, the max flow problem is well defined even when all capacities are nonnegative reals and it is homogeneous too.

Definition. By a linear parametrization for cardinality n, we mean a map $\mathcal{P}$ that associates with each real number λ in some interval $[p, q]$ an input $I(\lambda)$ such that:

1. The nonnumeric part of the input $I(\lambda)$ remains the same for every $\lambda \in [p, q]$.
2. The number of numeric parameters in $I(\lambda)$ is n for every $\lambda \in [p, q]$.
3. Each numeric parameter is a linear function of λ with rational coefficients and it satisfies any required constraints in the problem (such as nonnegativity) for every $\lambda \in [p, q]$.
4. The function graph of $F(\lambda) = F(I(\lambda))$ as λ varies over $[p, q]$ is piecewise linear and convex; we shall denote it by $G(\mathcal{P})$.

The interval $[p, q]$ is called the interval of definition. The bit size of $\mathcal{P}$ is defined to be the maximum among the bit sizes of the coordinates of the vertices of $G(\mathcal{P})$ and of the coefficients of the linear functions in $\mathcal{P}$ specifying numeric parameters. These coordinates and coefficients are in general rational. We define the bit size of a rational to be sum of the bit sizes of its denominator and numerator. We shall denote the bit size of $\mathcal{P}$ by $\beta(\mathcal{P})$. The number of bounded linear segments in $G(\mathcal{P})$ is called the *complexity* of $\mathcal{P}$; it will be denoted by $\rho(\mathcal{P})$.

For example, in the max flow problem, we can fix a graph with n vertices and let the capacities of its edges be linear functions of λ so that as λ varies over some interval $[p, q]$ all these capacities remain nonnegative–this specifies a linear parametrization. It is well known that the optimum function of such parametrization is piecewise linear and convex.

Definition. Fix a homogeneous optimization problem. Its parametric complexity for cardinality n is defined to the maximum of $\rho(\mathcal{P})$ as $\mathcal{P}$ ranges over all linear parametrizations for cardinality n; we denote it by $\phi(n)$.

More generally, parametric complexity for cardinality n and bit size $\beta(n)$ is defined to the maximum of $\rho(\mathcal{P})$ as $\mathcal{P}$ ranges over all linear parametrizations for cardinality n with bit size $\beta(n)$; we denote it by $\phi(n, \beta(n))$.

So far, n has denoted the number of numeric parameters in the input. We shall abuse the notation on some occasions and let n stand for something different but similar. For example, in the max flow problem we shall let n denote the number of nodes in the flow network. The precise meaning of n in the given context should be clear.

We shall show that if the parametric complexity of a problem is high then it is hard to parallelize in our model:

Theorem 3.1.1 Fix a homogeneous optimization problem. Let $\phi(n, \beta(n))$ be its parametric complexity for input cardinality n and bitsize $\beta(n)$. Then there exists a large enough constant b such that the decision version of the problem can not be solved in the PRAM model without bit operations in $\sqrt{\log[\phi(n, \beta(n))]}/b$ time using $2^{\sqrt{\log[\phi(n,\beta(n))]}/b}$ processors; this is so even if we restrict every numeric parameter in the input to be an integer with bitlength at most $a\beta(n)$ for a large enough constant a.

We shall prove this result in Chapters 4 and 5. It will be extended to the randomized setting in Chapter 6.

Corollary 3.1.2 *The lower bound in Theorem 3.1.1 also applies to the additive approximation version of the problem, in which the goal is to compute the optimum within a small, say less than* $1/8$, additive *error.*

Proof. Suppose there is a fast parallel algorithm in our model for the additive approximation version. This will yield a fast parallel algorithm for the decision version as follows. First we compute the approximate optimum v with additive error less than, say, $1/8$. Since we are assuming that the input is integral, we know that the exact optimum is integral. So we compare v with the integral threshold w in the input; w is less than or equal to the exact optimum iff $w \leq v + 1/8$. $\square$

Theorem 3.1.3

1. *(Murty [37]): For combinatorial linear programming,* $\phi(n, O(n)) = 2^{\Omega(n)}$, *where* n *denotes the total number of variables and constraints.*
2. *(Zadeh [56]): For the mincost flow problem,* $\phi(n, O(n)) = 2^{\Omega(n)}$, *where* n *denotes the number of nodes in the flow network.*
3. *(Carstensen [8] and Theorem 3.2.3): For the maxflow problem for directed or undirected networks* $\phi(n, O(n^2)) = 2^{\Omega(n)}$, *where* n *denotes the number of nodes in the flow network.*

By combinatorial linear programming, we mean restricted linear programming problem of the form "maximize $\{cx \mid Ax \leq b\}$", where all entries of A have logarithmic bit lengths; it has a strongly polynomial time algorithm [46]. The result of Murty [37] is subsumed by the later results [56, 7] since the max-flow and mincost-flow are instances of linear programming; however, Murty's proof is much simpler than these later ones.

3.1.1 Applications

We shall now derive the main results stated in Section 1.3 from Theorem 3.1.1.

In the deterministic setting, our lower bounds for the mincost flow and max flow (Theorem 1.3.1) follow directly from Theorem 3.1.1, Corollary 3.1.2 and Theorem 3.1.3. For randomized algorithms and the PRAM model with limited bit operations, see Chapter 6. (For the mincost flow problem the improvement in the constant c mentioned after the statement of Corollary 1.3.2 follows from the fact that the total bit size of Zadeh's [56] parametrized flow network with k nodes is only $O(k^2)$ even though some its edges may have $O(k)$-bit costs.)

We now proceed to the proof of Theorem 1.3.4. Fix a positive integer j. From the usual maxflow problem, we shall first construct a *padded* language L that belongs to the class C. A valid input instance of L for cardinality n has the following form: It contains a flow network G on nodes $1, \ldots, \lfloor \log^j n \rfloor$, with edge capacities of bitlength at most $a \log^{2j} n$, for a suitable constant a to be specified later. The remaining nodes are dummy and isolated. The input also specifies the threshold w.

The input is accepted iff the maxflow value for G is less than or equal to w. Let us show that $L \in C^{bj}$, for a suitable constant b. It is easy to check fast in parallel if the input is valid, i.e., if the nodes $\lfloor \log^j n \rfloor + 1, \ldots, n$ are indeed dummy and isolated, and if all edge-capacities have bitlengths at most $a \log^{2j} n$. If the input is not valid it is rejected right away. Otherwise dummy nodes are discarded. After this one is left with a small network G with $\lfloor \log^j n \rfloor$ nodes. The maxflow problem for G can be solved by just one processor using, say, a version of Dinits' strongly polynomial time algorithm [47] in $O(\log^{3j} n)$ time. If one uses the parallel algorithm of Shiloach and Vishkin [44] or Goldberg and Tarjan [13] it can be solved in $O(\log^{2j} n \log\log n)$ time using $O(\log^j n)$ processors. The total bitlength N of a valid input instance in L is $O(n + \log^{4j} n)$. Hence $L \in C^{bj}$, for a suitable constant b.

Now we shall get a lower bound on the parametric complexity of the padded problem. For this we use Carstensen's [8] or our (Section 3.2) parametric construction with padding. This means we form a padded graph with n nodes, in which all but $k = \log^j n$ nodes are dummy. The edge-capacities of the network G on the remaining k nodes are then linear functions of the parameter λ with coefficients of $O(k)$ bit-size as in [8] or Section 3.2. It follows from Theorem 3.1.3 (third statement) that the parametric complexity $\phi(n, O(k^2))$ of the padded maxflow problem is $2^{\Omega(k)}$, where $k = \log^j n$. So, by Theorem 3.1.1, it cannot be solved in the PRAM model without bit operations in $\log^{j/2} n / a'$ time using $2^{\log^{j/2} n / a'}$ processors for a large enough constant a', even assuming that every edge-capacity in the input has at most $a \log^{2j} n$ bitlength for a suitable large enough constant a. Thus L does not belong to $RC^{j/2-1}$. But we have already seen that $L \in C^{bj}$, for a suitable positive constant b. Thus it follows that $C^i \not\subseteq RC^{i/c}$, for a suitable positive constant c.

The role of the maxflow problem in this proof can be played by any problem in SP whose parametric complexity has a good exponential lower bound. For example, we could as well have used the mincost flow problem or the general combinatorial linear programming problem. Then we have to use Zadeh's [56] or Murty's [37] parametric construction.

3.2 s-t-mincuts in weighted undirected graphs

Carstensen [8] proved that the parametric complexity of the s-t-mincut problem for weighted undirected graphs with n vertices is $2^{\Omega(n)}$. The same lower bound also applies to the dual maxflow problem for undirected graphs. Carstensen's proof is very intricate though. In this section we give a much simpler proof using similar construction.

Let G be a weighted undirected graph with two distinguished vertices s and t. Each edge has nonnegative weight. By an s-t-cut (U, V), we shall mean a disjoint partition (U, V) of the vertex set of G such that U contains s and V contains t. The weight $w(U, V)$ of this cut is the total weight of all edges going from U to V. By an *s-t*-mincut in G, we mean an s-t-cut (U, V) with minimum weight. We say that an s-t-cut $J = (U, V)$ contains a vertex u, if $u \in U$.

For each positive integer n, we shall construct an weighted undirected graph G_n with $2n + 2$ vertices whose weights are linear functions of a parameter λ such that

1. In the interval $(-T, T)$, where $T = 2^{n+1}$, all edge-weights are nonnegative.
2. Let $g(\lambda)$ denotes the weight of a mincut in G for a particular λ. Then the function graph of $g(\lambda)$ has at least $2^n - 1$ breakpoints, i.e., slope changes in the interval $(-T, T)$.
3. The coefficients of the vertices of this function graph are rationals with $O(n^2)$ bit lengths.

The graph G_n is defined as follows. The vertices of G_n consist of s, t, the vertices labeled $1, \ldots, n$ and the vertices labeled $\bar{1}, \ldots, \bar{n}$. There are edges connecting s to every vertex i and $\bar{i}$, and edges connecting every vertex i and $\bar{i}$ to t. For all $i \neq j$, there are also edges connecting the vertices i and $\bar{i}$ to the vertices j and $\bar{j}$. Let $w(u, v) = w(v, u)$ denote the weight of the edge between u and v. Let w_i, for $1 \leq i \leq n$, be positive integers such that $w_n = 1$, and for $i < n$

$$w_i > 2T \sum_{i<l\leq n} w_l. \tag{3.1}$$

The choice $w_i = (2nT)^{n-i}$, for $i \leq n$, would suffice. Let $t_j = 2^{n-j+1}$. The edge-weights are then defined as follows.

1. For all i, $w(s,i) = w(s,\bar{i}) = w_i T$.
2. For all i, $w(i,t) = w_i(T+\lambda)$, and $w(\bar{i},t) = w_i(T-\lambda)$
3. For all $i > j$, $w(i,j) = w(\bar{i},\bar{j}) = w_i(T - t_j)$, and $w(i,\bar{j}) = w(\bar{i},j) = w_i T$.

Note that all edge-weights are positive in the interval $(-T,T)$ of interest to us.

Theorem 3.2.1 *The graph of the mincut-cost-function $g(\lambda)$ for G_n has at least $2^n - 1$ breakpoints in the interval $(-T,T)$.*

Before we prove the theorem, we shall prove a lemma. Let part(0) denote the interval $(-T,T)$. For $i \geq 1$, define part(i) to be the partition obtained by splitting each interval of part($i-1$) at its midpoint. Thus part(i) contains 2^i intervals of length 2^{n+2-i}. Consider a complete binary tree of depth n whose nodes at depth i from the root are labelled with the intervals of part(i). Any sequence σ of $+1$s and -1s with length $i \leq n$ denotes a unique node of this tree. Let $J(\sigma)$ denote the corresponding interval; it is the unique interval of part(i) containing the point $p(\sigma)$ with coefficient $t(\sigma) = \sum_{j \leq i} \sigma(j) t_j$, where $\sigma(j)$ denotes the jth symbol of σ. By convention $t(\sigma) = 0$ and $J(\sigma)$ denotes the interval $(-T,T)$ if σ is empty. Let $I(\sigma)$ be the interval of positive length $2^{n+2-i} - 2$ obtained by removing from both ends of $J(\sigma)$ intervals of length 1.

Lemma 3.2.2 *Fix any $i \leq n$ and a sequence σ of length i. Fix any $\lambda \in I(\sigma)$. Let $C(\lambda) = (U(\lambda), V(\lambda))$ be a mincut in G_n at this λ; it need not be unique. Then, for every $j \leq i$,*

1. *$U(\lambda)$ contains j iff $\sigma(j) = -1$, and*
2. *$U(\lambda)$ contains $\bar{j}$ iff $\sigma(j) = +1$.*

Proof. Notice that the weighted graph G_n remains invariant if we switch all pairs $(i,\bar{i})$ of vertices and replace λ by $-\lambda$. Hence, the first statement is equivalent to the second. So we shall only prove the former. The proof proceeds by induction on i.
Basis case, $i = 1$:

Consider the case when $\sigma(1) = +1$. Suppose, to the contrary, that $U(\lambda)$ contains the vertex 1. If we move 1 from $U(\lambda)$ to $V(\lambda)$, the value of the cut would decrease by

$$w(1,t) + \textstyle\sum_{j>1: j \in V(\lambda)} w(j,1) + \sum_{j>1: \bar{j} \in V(\lambda)} w(\bar{j},1)$$

$$-w(s,1) - \textstyle\sum_{j>1: j \in U(\lambda)} w(j,1) - \sum_{j>1: \bar{j} \in U(\lambda)} w(\bar{j},1)$$

$$= w_1(T+\lambda) + \textstyle\sum_{j>1: j \in V(\lambda)} w_j(T - t_1) + \sum_{j>1: \bar{j} \in V(\lambda)} w_j T$$

$$-w_1 T - \textstyle\sum_{j>1: j \in U(\lambda)} w_j(T - t_1) - \sum_{j>1: \bar{j} \in U(\lambda)} w_j T)$$

$$= w_1(\lambda) + h(\lambda, 1),$$

where $h(\lambda, 1)$ is the lower order term

$$\sum_{j>1: j \in V(\lambda)} w_j(T-t_1) + \sum_{j>1: \bar{j} \in V(\lambda)} w_j T - \sum_{j>1: j \in U(\lambda)} w_j(T-t_1) - \sum_{j>1: \bar{j} \in U(\lambda)} w_j T);$$

it is bounded in absolute value by $2T \sum_{j>1} w_j$, which, by eq.(3.1), is strictly less than w_1. It follows that in in the interval $I(\sigma) = [1, T-1]$ the decrease in the weight of the mincut is positive. This contradicts the fact that J is a mincut. So $U(\lambda)$ can not contain 1. It follows similarly that if $\sigma(1) = -1$ and $V(\lambda)$ contained the vertex 1, then moving it from $V(\lambda)$ to $U(\lambda)$ would decrease the weight of the cut by

$$-w_1\lambda - h(\lambda, 1)$$

which is positive if $\lambda \in I(\sigma) = (-T, -1)$. So $V(\lambda)$ can not contain 1 if $\sigma(1) = -1$. It follows that $U(\lambda)$ contains 1 iff $\sigma(1) = -1$.

Now let us turn to the general case. In general, let $\hat{\sigma}$ be the string obtained from σ by removing its last, i.e., the ith symbol. Clearly, $I(\sigma) \subset I(\hat{\sigma})$. Hence, by the inductive hypothesis applied to $\hat{\sigma}$, $U(\lambda)$ at any $\lambda \in I(\sigma)$ must contain, for every $j < i$, the vertex j iff $\sigma(j) = -1$, and the vertex $\bar{j}$ iff $\sigma(j) = 1$. It remains to show that $U(\lambda)$ contains i iff $\sigma(i) = -1$.

First, consider the case when $\sigma(i) = 1$. Suppose, to the contrary, that $U(\lambda)$ contains i. Then moving i from $U(\lambda)$ to $V(\lambda)$ would decrease the weight of the cut by

$$\begin{aligned} & w(i,t) - w(s,i) + \textstyle\sum_{j \neq i: j, \bar{j} \in V(\lambda)} w(i,j) + w(i,\bar{j}) \\ & \quad - \textstyle\sum_{j \neq i: j, \bar{j} \in U(\lambda)} [w(j,i) - w(\bar{j}, i)] \\ & = w_i\lambda + \textstyle\sum_{j<i: j, \bar{j} \in V(\lambda)} w(i,j) + w(i,\bar{j}) \\ & \quad - \textstyle\sum_{j<i: j, \bar{j} \in U(\lambda)} [w(i,j) - w(i,\bar{j})] + h(\lambda, i), \end{aligned} \tag{3.2}$$

where $h(\lambda, i)$ is the lower order term

$$\sum_{j>i: j, \bar{j} \in V(\lambda)} w(j,i) + w(\bar{j}, i) - \sum_{j>i: j, \bar{j} \in U(\lambda)} [w(j,i) - w(\bar{j}, i)];$$

it is bounded in absolute value by $2T \sum_{j>i} w_j$, which, by eq.(3.1), is strictly less than w_i. By the inductive hypothesis, the decrease given by eq.(3.2) is equal to

$$w_i\lambda + \sum_{j<i:\sigma(j)=1} w(i,j) + \sum_{j<i:\sigma(j)=-1} w(i,\bar{j})$$

$$-\sum_{j<i:\sigma(j)=-1} w(i,j) - \sum_{j<i:\sigma(j)=1} w(i,\bar{j}) + h(\lambda,i)$$

$$= w_i\lambda + \sum_{j<i:\sigma(j)=1} w_i(T-t_j) + \sum_{j<i:\sigma(j)=-1} w_iT$$

$$-\sum_{j<i:\sigma(j)=-1} w_i(T-t_j) - \sum_{j<i:\sigma(j)=1} w_iT + h(\lambda,i)$$

$$= w_i\lambda - \sum_{j<i:\sigma(j)=1} w_it_j + \sum_{j<i:\sigma(j)=-1} w_it_j + h(\lambda,i)$$

$$= w_i\lambda - w_i(\sum_{j<i} \sigma(j)t_j) + h(\lambda,i),$$

$$= w_i(\lambda - t(\hat{\sigma})) + h(\lambda,i),$$

where $t(\hat{\sigma}) = \sum_{j<i} \hat{\sigma}(j)t_j = \sum_{j<i} \sigma(j)t_j$. If $\lambda \in I(\sigma)$ and $\sigma(i) = 1$ then $\lambda - t(\hat{\sigma}) \geq 1$. Since the absolute value of $h(\lambda,i)$ is less than w_i, this decrease is positive in that case. Hence moving i from $U(\lambda)$ to $V(\lambda)$ would strictly decrease the weight of the cut, which contradicts the fact the $C(\lambda)$ is a mincut. So $U(\lambda)$ can not contain i if $\sigma(i) = 1$.

Now consider the case when $\sigma(i) = -1$. Suppose to the contrary that $V(\lambda)$ contains i. Then it follows similarly that moving i from $V(\lambda)$ to $U(\lambda)$ would decrease the weight of the cut by

$$w_i(t(\hat{\sigma}) - \lambda) - h(\lambda,i),$$

If $\lambda \in I(\sigma)$ and $\sigma(i) = -1$ then $t(\hat{\sigma}) - \lambda \geq 1$. Hence this decrease is strictly positive, which contradicts the fact that $C(\lambda)$ is a mincut. It follows that $V(\lambda)$ can not contain i if $\sigma(i) = -1$.

Thus $U(\lambda)$ contains i iff $\sigma(i) = -1$. $\square$

If σ is any sequence of 1 and -1 of length n, then it follows from the preceding lemma that throughout the interval $I(\sigma)$ the mincut remains the same–let us call it $C(\sigma) = (U(\sigma), V(\sigma))$. It also follows that $U(\sigma)$ contains i iff $\sigma(i) = -1$ and $\bar{i}$ iff $\sigma(i) = 1$. The weight of $C(\sigma)$ at any $\lambda \in I(\sigma)$ is of the form $A(\sigma)\lambda + B(\lambda)$, where $A(\sigma) = -\sum \sigma(i)w_i$. Thus the slopes of the weight functions of all 2^n cuts $C(\sigma)$, as σ ranges over all strings of 1 and -1 of length n, are distinct. This proves Theorem 3.2.1. $\square$

Bit-lengths of the edge-capacities in our construction remain $O(n^2)$ in the interval $[-T, T]$ of interest. Hence it follows that:

Theorem 3.2.3 Parametric complexity $\phi(n, O(n^2))$ of the s-t-mincut problem for weighted undirected graphs of cardinality n is $2^{\Omega(n)}$. This also holds for the dual max flow problem.

3.3 Global mincuts in weighted undirected graphs

In this section we show that the parametric complexity of the global mincut problem for weighted undirected graphs is at most polynomial in the number of vertices–this should explain why this problem has fast parallel algorithms in our model [29, 30]. By a global mincut we mean a nontrivial cut; there is no restriction that some distinguished vertices s and t be separated by the cut.

We shall make use of the following combinatorial result and its proof in [29].

Lemma 3.3.1 *In any positively weighted undirected graph, the number of cuts within a constant multiplicative factor of a global mincut is polynomial in the number of vertices.*

We shall also use a result due to Chandrasekharan and Gusfield [18] that the parametric complexity of the maximum weight spanning tree problem is $O(n^4)$; this is in fact true for any matroidal optimization problem having a greedy algorithm.

Let G be an undirected graph with n vertices, whose edge weights are linear functions of a parameter λ. It is furthermore guaranteed that in some interval $[p, q]$ all edge weights remain positive. Let $g(\lambda)$ denote the cost of a mincut in G at a given λ.

Theorem 3.3.2 *The number of breakpoints in the function graph of $g(\lambda)$ in the interval $[p, q]$ is at most polynomial in n.*

Proof. The coefficients of the linear weight functions of the edges in G may be arbitrary reals. But one can show that, without of loss of generality, these coefficients can be assumed to rationals with bit sizes at most polynomial in n. This follows from the fact that the parametric complexity of the mincut problem is the same as the maximum size of a two-dimensional projection of the corresponding combinatorial polytope, which is the convex hull of the incidence vectors of all cuts in G. Since the coefficients of the linear weight functions have polynomial size bit lengths, all vertices of the function graph of $g(\lambda)$ have rational coefficients with polynomial bit lengths. For this reason one can also assume that p and q have polynomial bit lengths.

The parametric complexity of the maximum-weight spanning tree problem is $O(n^4)$ [18]. So by dividing $[p, q]$ into at most $O(n^4)$ subintervals if necessary one can assume that a maximum weight spanning tree in G remains the same throughout $[p, q]$, i.e., as λ varies in the interval $[p, q]$. This tree has n edges, since G can be assumed to be connected. So subdividing $[p, q]$ further into at most n subintervals, one can also assume, without loss of generality, that the edge of minimum weight in this maximum-weight spanning tree remains the same throughout $[p, q]$. Let this edge be e and let its weight function be $w_e(\lambda)$. By subdividing the interval $[p, q]$ further into at most n subintervals if necessary, one can also assume that the set of edges whose weights exceed n^2 times the weight of e also remains the same throughout $[p, q]$. As in Karger [29], we can contract these edges in G without changing the mincut. So let us assume that all edges in G have weights at most n^2 times that of e throughout $[p, q]$. This means the weight of any cut at any $\lambda \in [p, q]$ is at most n^4 times the weight $w_e(\lambda)$ of e throughout $[p, q]$. It also follows, as in [29], that the weight of any cut at any $\lambda \in [p, q]$ is at least $w_e(\lambda)$.

By our assumption, both $w_e(p)$ and $w_e(q)$ must be positive. Assume, without loss of generality, that $w_e(p) < w_e(q)$, the other case being similar. Since the the coefficients of the linear weight functions, and also p and q, have polynomial size bit lengths, $\log(w_e(q)) - \log(w_e(p))$ is polynomial in n. So by subdividing $[p, q]$ further into polynomially many subintervals if necessary we can assume that $\log(w_e(q)) - \log(w_e(p)) \leq 1/n$, which means that $w_e(q) \leq 2w_e(p)$.

For any cut J, let $w_J(\lambda)$ denote its weight function. Then our interval $[p, q]$ has the following properties: First,

$$w_e(p) \leq w_e(q) \leq 2w_e(p). \tag{3.3}$$

Second, for any cut J and any $\lambda \in [p, q]$,

$$w_e(\lambda) \leq w_J(\lambda) \leq n^4 w_e(\lambda). \tag{3.4}$$

Divide $[p, q]$ further into, say, n^5 subintervals of equal length. Let $[a, b]$ denote any such subinterval.

Claim. *For any $\lambda \in [a, b]$, and any cut J, $|w_J(\lambda) - w_J(b)| \leq 2w_e(b)/n$.*
Proof of the claim: By eq.(3.4), $w_J(q) \leq n^4 w_e(q)$, and similarly $w_J(p) \leq n^4 w_e(p) \leq n^4 w_e(q)$, by eq.(3.4) and eq.(3.3). Hence, the slope of the weight function $w_J(\lambda)$ is at most $n^4 w_e(q)/(q-p)$. Therefore, for any $\lambda \in [a, b]$, $|w_J(\lambda) - w_J(b)|$ is at most

$$\frac{n^4 w_e(q)(b-\lambda)}{(q-p)} \leq \frac{w_e(q)n^4}{n^5} = \frac{w_e(q)}{n} \leq \frac{2w_e(p)}{n} \leq \frac{2w_e(b)}{n}.$$

This proves the claim.

Let K denote a mincut at b.

Claim. *A cut I whose weight becomes minimum at some $\lambda \in [a, b]$ has its weight at b at most twice that of K at b.*
Proof of the claim: First, note that $w_I(b) - w_K(b)$ is equal to

$$(w_I(b) - w_I(\lambda)) + (w_I(\lambda) - w_K(\lambda)) + (w_K(\lambda) - w_K(b)).$$

Since K is a mincut at b, $w_I(b) - w_K(b)$ is nonnegative, and similarly, $w_K(\lambda) - w_I(\lambda)$ is nonnegative. So $w_I(b) - w_K(b)$ is at most

$$(w_I(b) - w_I(\lambda)) + (w_K(\lambda) - w_K(b)) \leq 4w_e(b)/n,$$

by the previous claim. But $w_K(b)$ is at least $w_e(b)$, by eq.(3.4). So the last quantity is at most $4w_K(b)/n$. Now the claim follows.

By Lemma 3.3.1 there are only polynomially many cuts whose weights at b are at most four times the weight of the mincut K at b. Let Γ_b denote the set of such cuts. Now the preceding claim implies that any cut that can become minimum during $[a, b]$ must belong to the set Γ_b, whose size $|\Gamma_b|$ is polynomial in n. In other words, in the interval $[a, b]$ the mincut weight function is given by

$$g(\lambda) = \min\{w_K(\lambda) \mid K \in \Gamma_b\},$$

and so can have at most $|\Gamma_b|$ breakpoints within the interval $[a, b]$. This is true for each of the polynomially many subintervals into which $[p, q]$ was divided. So the theorem follows. $\square$

Chapter 4

Linear PRAM

In this chapter we prove our general lower bound (Theorem 3.1.1) for the linear PRAM model. Actually we get a stronger lower bound that takes into account processor-time tradeoff:

Theorem 4.0.3 Fix any homogeneous optimization problem. Let $\phi(n, \beta(n))$ be its parametric complexity for input cardinality n and bitsize $\beta(n)$. Then there exists a large enough constant b such that the decision version of the problem cannot be solved in the linear PRAM model in $\sqrt{\log[\phi(n, \beta(n))]}/b$ time using $2^{\sqrt{\log[\phi(n,\beta(n))]}/b}$ processors, or more generally, in $\log[\phi(n, \beta(n))]/(b \log p)$ time using p processors; this is so even if we restrict every numeric parameter in the input to be an integer with bitlength at most $a\beta(n)$ for a large enough constant a.

This also holds for approximate computation of the optimum within a small, say less than $1/8$, additive error.

In conjunction with Theorem 3.1.3 this leads to a somewhat stronger lower bound for the linear PRAM than in Theorem 1.3.1:

Theorem 4.0.4 *The mincost-flow problem for networks with k nodes cannot solved in the linear PRAM model deterministically (or with randomization) in $k/(b \log p)$ (expected) time using p processors, even assuming that every cost and capacity is an integer with bit length at most ak, for some large enough positive constants a and b. The same lower bound also applies for the max flow problem with the bit length restriction ak^2 instead of ak.*

Compare with the upper bound provided by the parallel algorithms of Shiloach and Vishkin [44], and Goldberg and Tarjan [13] (which lie in the linear PRAM model)–these work in $O(k^2 \log k)$ time with $O(k)$ processors without any restriction on the bitlengths.

Here we shall only consider deterministic setting. For randomized setting, see Section 6.1.

4.1 The number of possible branchings

In this section we describe a general technique for bounding the number of possible ways in which the processors can branch in the linear PRAM model. The technique is language-independent. So we shall describe it in a general setting.

Let $\Sigma = \Sigma(x_1, \cdots, x_n)$, $x_i \in Z$, be the language for which want to prove a lower bound. Here $x_1, \cdots, x_n$ denote the integer parameters in the input. We shall denote the total bit size of the input by N. We shall denote the integer tuple $(x_1, \cdots, x_n)$ by x. Consider any nonuniform machine for n and N. It is required to work correctly only on inputs with n parameters of total bit length at most N. Hence, we shall confine ourselves to only those values of x_i for which the total bit size of x is bounded by N. Let $p(n, N)$ be the number of processors in the machine. Let $t(n, N)$ be the maximum time taken by the machine on any input of bit length at most N. Let $x, x' \in Z^n$ be any two n-tuples of bit size at most N. For any positive integer t, we say that x and x' are *t-equivalent* if on the input x the instruction executed by any processor in our machine at any time less than or equal to t is the same as the instruction that would have been executed if the input were x' instead. We say that x and x' are *equivalent* if they are t-equivalent for all $t \leq t(n, N)$. By a t-equivalence class, we mean the set of all t-equivalent k-tuples of bit size at most N. An equivalence class is defined similarly. Let $\phi(t)$ be the number of t-equivalence classes, and ϕ the number of equivalence classes.

Ideally, we would like to get good bounds on $\phi(t)$ and ϕ. Unfortunately, this seems difficult. So we shall get around this problem by specializing the possible inputs to our machine–this corresponds to restricting one's attention to inputs that lie within some affine subspace. We shall then obtain a good bound on the number of equivalence classes among specialized inputs.

Specifically, let us imagine running our machine on the set of possible inputs defined by: $x_i = l_i(z) = l_i(z_1, \cdots, z_d)$, where, for each $i \leq n$, l_i is a certain integral linear form (possibly nonhomogeneous) in d variables, and z_j ranges over all integer values of bit length bounded by some integer r_j. When x_i is a nonnumeric parameter, we shall assume that $l_i(z)$ is constant. In other words, we assume that the nonnumeric parameters do not change as we change the parameters z_i. The forms l_i will depend on the problem under consideration. We shall turn to this issue in Section 4.2. Here we only need to assume that the integers r_j and the forms l_i are such that the total bit size of the tuple $x = (x_1, \cdots, x_n)$ remains bounded by N as we let z_i range over all permissible values. This implies that the machine must work correctly on this parametrized set of inputs within a given time bound $t(n, N)$. In what follows, we shall assume that every d-tuple $z = (z_1, \ldots, z_d)$ under consideration is *permissible.* This means each z_i is an integer of bit length at most r_i. In our applications d will be constant.

Let $\Sigma(z_1, \cdots, z_d)$ denote the set of permissible d-tuples accepted by the machine. We say that z is accepted if $x = l(z) = (\cdots, l_i(z), \cdots)$ is accepted. Two permissible d-tuples z, z' are said to be t-equivalent if $l(z)$ and $l(z')$ are

t-equivalent. The notion of equivalence is defined similarly. Let $\sigma(t)$ denote the number of t-equivalence classes among the permissible values of z. Let $\sigma = \sigma(t(n, N))$ denote the number of equivalence classes.

Theorem 4.1.1 *For all t, $\sigma(t)$ is bounded by $[2p(n, N)]^{dt}$.*

Proof. Let C denote a fixed t-equivalence class. Let z denote a (permissible) integral value in this equivalence class. In what follows, we pretend that z is a generic variable. We are told that it belongs to C. But we are not told its actual value. We imagine running our machine on the input $x = l(z)$. We have to specify what is meant by executing an arithmetic operation in this setting. Naturally, we can not expect the operands to be integers, unless they happen to be constants. Rather they will be functions of z. Since there is no general multiplication in the linear PRAM model, these functions will be linear. By an arithmetic operation $+$ or $-$, we simply mean the corresponding operation on the linear forms (functions). We are also allowed multiplication by constants. This amounts to multiplying the corresponding linear form by a constant. The nonnumeric parameters are fixed in our parametrization. Since in our model the memory pointers are allowed depend only on the nonnumeric parameters, indirect memory references are no problem. In other words, whenever an instruction involving a memory pointer is encountered during the course of execution, the value of the pointer would be completely determined at that time, because it can not depend on the generic parameter z. This in turn completely determines which memory locations are to be fetched (or stored into) during the execution of that instruction. In case several processors are planning to write into the same memory location, it is also determined which processor's writing is going to prevail (in the CRCW mode of communication). We assume that when a processor is executing a comparison it is simply told the result of that comparison, which, at any time less than or equal t, is completely determined by the class C; i.e., it is the same for all permissible values in C.

Lemma 4.1.2 *Let C be a fixed t-equivalence class. For every memory location j, there exists a linear form $g_j[C, t](z)$ that gives the content of that location at time t for every $z \in C$.*

Proof. Once C is fixed, the execution path of each processor up to time t is fixed. So the lemma follows easily by induction on t. $\square$

Let us now see what happens at time t, given that the generic variable z belongs to a fixed t-equivalence class C as above. If a processor is to execute an arithmetic operation at time t, then the result of that operation will be a certain uniquely determined linear form to be stored in a certain uniquely determined location in the memory. If the operation is a branch, the branching path will be determined by a comparison between the contents of two uniquely determined memory locations, which are certain fixed linear forms in z. In other words, each branch is determined by testing a certain linear equality or inequality in z of the form $g(z) : 0$, where the comparison $:$ is $>, \geq$ or $=$. The number of such linear constraints is at most $p(n,N)$, one per processor. Let $g_1(z), g_2(z), \cdots$ be the nonzero linear forms involved in these constraints. Consider the arrangement in R^d formed by the hyperplanes $g_i(z) = 0$. The hyperplanes naturally partition R^d into a disjoint union of (open) j-faces, where each j-face of the arrangement is a j-dimensional convex polytope. When d is fixed, the total number of faces in the arrangement (of all dimensions) is $O(m^d)$, or at most $(2m)^d$, to be more precise, where m is the number of hyperplanes, i.e., the number of linear forms g_i. There are at most $p(n,N)$ forms, one per processor. Hence the number of faces in the arrangement is at most $(2p(n,N))^d$.

Now we make the following crucial observation: Any two $z, z' \in C$ belong to the same $(t+1)$-equivalence class if they belong to the same face of this partition. This is because the sign of each linear function $g_i(z)$ is the same at any point of the face (the sign is either $+$, $-$, or 0). It immediately follows that

$$\sigma(t+1) \leq (2p(n,N))^d \sigma(t).$$

Induction on t proves Theorem 4.1.1. ▯

The proof of the theorem also yields:

Lemma 4.1.3 *For every t-equivalence class C, there exists a set $\Phi(C,t)$ of at most $p(n,N)t$ linear constraints (equalities or inequalities) in d variables such that $z \in C$ iff all these constraints are satisfied.*

Proof. The constraints correspond to the branches before time t, at most t per processor, which are all fixed by the class C. ▯

By Lemma 4.1.3, each equivalence class D of permissible z-values is characterized by a set $\Phi(D) = \Phi(D, t(n,N))$ of linear constraints. Let $\Phi = \cup_D \Phi(D)$, where D ranges over all equivalence classes. By Theorem 4.1.1 and Lemma 4.1.3, it follows that Φ contains at most $[2p(n,N)]^{dt(n,N)}p(n,N)t(n,N)$ linear forms. Consider the arrangement in R^d formed by the hyperplanes defined by these linear forms. It is easy to see that each face of this arrangement can be labeled *yes* or *no* in such a way that a permissible z is accepted by the machine iff it lies in a face labeled *yes*. Thus we have proved the following.

Theorem 4.1.4 *Assume that the language $\Sigma(z_1, \cdots, z_d)$ is accepted in the linear PRAM model using $p(n,N)$ processors in $t(n,N)$ time. Then R^d can be partitioned by a set of at most*

$$[2p(n,N)]^{dt(n,N)}p(n,N)t(n,N)$$

hyperplanes and each face of the arrangement can be labeled yes or no so that a permissible $z \in Z^d$ belongs to $\Sigma(z_1, \cdots, z_d)$ iff it lies in a face labeled yes.

4.2 Parametrization

The preceding theorem suggests the following strategy for reducing a global lower bound problem to a local problem in small dimensions. Parametrize the input to the machine using a small (constant) number of parameters. Then for this parametrization prove that a partition having the properties mentioned in Theorem 4.1.4 cannot exist. As a byproduct, we shall actually end up proving a stronger lower bound (Theorem 4.2.1) that is nonuniform with respect to parametrization. This means for any parametrization $\mathcal{P}$ we get a lower bound that applies even to nonuniform machines whose programs can depend on $\mathcal{P}$ arbitrarily. Theorem 4.0.3 would be a consequence of this stronger lower bound.

So fix a general optimization problem as in Section 3.1 and any parametrization $\mathcal{P}$ for cardinality n and bit size $\beta(n)$. Let the complexity of $\mathcal{P}$ be $\rho(n)$. For a given rational parameter λ, each numeric parameter in the input $\mathcal{P}(\lambda)$ is a linear function of the form $u\lambda + v$; we can assume that u and v are integers, because if they are not, we can clear all denominators by multiplying all numeric parameters by a large enough integer, and since the problem is homogeneous, the optimum value gets multiplied by the same integer. When λ is rational, $u\lambda + v$ is in general rational too. This is not allowed, since in our optimization problem every numeric parameter has to be integral. However, with every such parametrization $\mathcal{P}$, one can associate a homogeneous integral parametrization $\tilde{\mathcal{P}}$ with two integer parameters z_1 and z_2–which can be thought of as the denominator and numerator of the parameter λ–by replacing each linear form $u\lambda + v$ in the definition of $\mathcal{P}$ by the corresponding integral form $uz_2 + vz_1$. For any integer pair (z_1, z_2), each numeric parameter of the input $\tilde{\mathcal{P}}(z_1, z_2)$ is now an integer as required. The bit size $\beta(n)$ or the complexity $\rho(n)$ of $\tilde{\mathcal{P}}$ is defined to be the same as that of $\mathcal{P}$.

Remark: Integral nature of our optimization problem–i.e., the assumption that $F(I)$ and each numeric parameter in I is an integer–is important: it is crucial in proving the lower bound for the approximate version (Corollary 3.1.2) from the one for the decision version (Theorem 3.1.1). Even if we were to allow rational values of numeric parameters, the situation would not be any easier or different, since the machine in our model can access the numerator and the denominator of a rational number separately.

Now given a large enough positive integer a, called the *permissibility constant*, let $\mathcal{I}$ be the set of inputs of the form (z_3, I) in the decision version of our optimization problem, where:

1. The integer z_3 denotes the threshold and its bit size is at most $a\beta(n)$.
2. I is of the form $\tilde{\mathcal{P}}(z_1, z_2)$, for some integers z_1 and z_2 of bit size at most $a\beta(n)$.

We shall prove that:

Theorem 4.2.1 *Assume that the permissibility constant a is large enough. Then no machine in the linear PRAM model can decide whether $w \leq F(I)$ correctly for every $I \in \mathcal{I}$ within $\sqrt{\log \rho(n)}/a'$ parallel time using $2^{\sqrt{[\log \rho(n)]/a'}}$ processors, for some large enough constant a' that does not depend on the permissibility constant a.*

More generally, no machine in the linear PRAM model can decide whether $w \leq F(I)$ *correctly for every* $I \in \mathcal{I}$ *within* $\log \rho(n)/(a' \log p)$ *parallel time using* p *processors, for some large enough constant* a' *that does not depend on the permissibility constant* a.

In the deterministic setting Theorem 4.0.3 follows from this theorem by applying it to a parametrization $\mathcal{P}$ for cardinality n and bit size $\beta(n)$ whose complexity is equal to $\phi(n, \beta(n))$, the maximum possible value. The randomized setting is considered in Section 6.1. In what follows, we shall denote $\rho(n)$ and $\beta(n)$ by simply ρ and β respectively.

Let us apply Theorem 4.1.4, with $d = 3$, to the linear parametrization of the input as in Theorem 4.2.1. It is defined by the linear map

$$(z_1, z_2, z_3) \to I(z_1, z_2, z_3) = (z_3, \tilde{\mathcal{P}}(z_1, z_2)).$$

Thus in Theorem 4.1.4 $\Sigma(z_1, z_2, z_3)$ consists of those tuples (z_1, z_2, z_3) such that $z_3 \leq \tilde{\mathcal{P}}(z_1, z_2)$. This gives:

Theorem 4.2.2 *Let* a' *be any positive constant. If a linear PRAM machine works correctly on the input* $I(z_1, z_2, z_3)$, *for every permissible* z_1, z_2 *and* z_3, *in* $t = \sqrt{\log \rho}/a'$ *time using* 2^t *processors then* R^3 *can be partitioned by at most* 2^{5t^2} *planes and each face of the resulting arrangement can be labeled* yes *or* no *so that a permissible integer point* (z_1, z_2, z_3) *lies in a face labeled* yes *iff* $I(z_1, z_2, z_3)$ *is feasible. (The constant* 5 *is just chosen to be large enough.)*

More generally, if a linear PRAM machine works correctly on the input $I(z_1, z_2, z_3)$, *for every permissible* z_1, z_2 *and* z_3, *in* $\log \rho/(a' \log p)$ *time using* p *processors then* R^3 *can be partitioned by at most* $2^{(5 \log \rho)/a'}$ *planes and each face of the resulting arrangement can be labeled* yes *or* no *so that a permissible integer point* (z_1, z_2, z_3) *lies in a face labeled* yes *iff* $I(z_1, z_2, z_3)$ *is feasible.*

4.3 A lattice problem

We are thus led to the problem of showing that an arrangement as in Theorem 4.2.2 cannot exist for large enough constants a and a'; this will prove Theorem 4.2.1. Let $F(\lambda)$ be the optimum function associated associated with $\mathcal{P}$ (as in Section 3.1) and $G = G(\mathcal{P})$ its function graph. Since our parametrization is assumed to be homogeneous, $I(z_1, z_2, z_3)$ is feasible iff $z_3 \leq \tilde{\mathcal{P}}(z_1, z_2)$, i.e., $z_3/z_1 \leq F(z_2/z_1)$. (Here, and in what follows, we assume that a permissible z_1 is always positive.) This suggests us to think of G as lying in the affine plane $z_1 = 1$ in R^3, where R^3 is now considered as the two-dimensional projective space. We let z_2, z_3 play the role of coordinates in the affine plane. Then the point (z_1, z_2, z_3) in R^3 is naturally projected onto the point in the affine plane that lies on the ray emanating from the origin and passing through (z_1, z_2, z_3). In what follows, by the *projection* onto the affine plane, we shall always mean this natural projection. Given any set ϕ of points in the affine plane, let fan(ϕ), the fan through ϕ, denote the points in R^3 that project onto the points in ϕ.

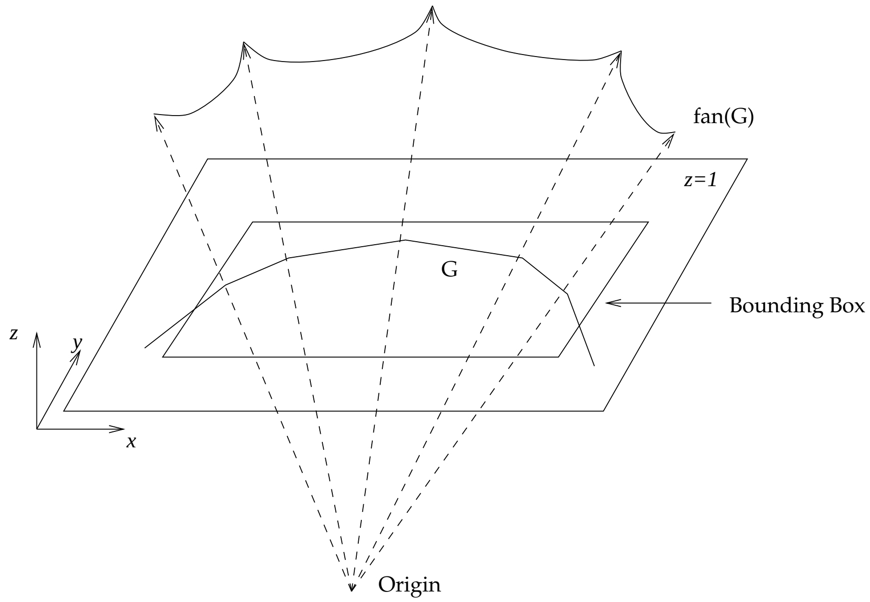

Figure 4.1: The graph G and its fan.

Then $I(z_1, z_2, z_3)$ is feasible iff (z_1, z_2, z_3) lies below fan(G) in the z_3-direction; this also includes the case when (z_1, z_2, z_3) lies on fan(G).

For the sake of simplicity, let us rename the coordinates z_1, z_2, z_3 as z, x, and y respectively. Thus $z = 1$ now serves as the affine plane in R^3. The preceding discussion implies that, in the partition of Theorem 4.2.2, a permissible integer point in Z^3 is labeled *yes* iff it lies below fan(G) in the y-direction (Figure 4.1).

Since $\mathcal{P}$ has bit size $\beta = \beta(n)$ and complexity $\rho = \rho(n)$ it follows that: (1) the graph G is piecewise linear and convex with ρ bounded segments, and (2) all vertices of G have rational coordinates that can be expressed in terms of numerators and denominators of absolute value at most $\mu = 2^\beta$, or in other words, of bit size at most $\beta = \log \mu$. Trivially $\rho \leq 4\mu^2$. Moreover, all vertices of G lie in the interior of the *bounding box* defined by: $|x| \leq 2\mu$, and $|y| \leq 2\mu$ (Figure 4.1). Consider two horizontal planes, parallel to the affine plane, defined by $z = \bar{\mu}$ and $z = 2\bar{\mu}$, where $\bar{\mu}$ would be chosen to be much larger than μ. Let B denote the region (slab) between these two planes that is enclosed by the fan through the boundary of the bounding box in the affine plane. The coordinates of all integer points in this slab have bit sizes at most twice $\bar{\beta} = \log \bar{\mu}$. The following theorem would imply that the partition as in Theorem 4.2.2 cannot exist.

Theorem 4.3.1 *Let S be any set of δ planes in R^3. Assume that $\bar{\beta} = \log \bar{\mu}$ is greater than a large enough constant multiple of $\beta = \log \mu$ and that ρ is greater than a large enough constant multiple of δ^4. (Since $\rho \leq 4\mu^2$, this also means that $\log \bar{\mu}$ is greater than a sufficiently large constant multiple of $\log \delta$.) Then* *at least one face in the arrangement (partition) in B formed by the planes in S contains an integer point lying below* fan(G) *and also an integer point strictly above this fan.*

Remark. If ρ were less than δ, then one can let S be the set of planes that contain the fans through the edges of G. No face in the resulting partition can contain an integer point below fan(G) and also an integer point strictly above fan(G) (in its interior). What the theorem says is that one cannot do much better than this trivial scheme.

Remark. It is possible to strengthen the theorem slightly so that ρ is only required to be greater than a large enough constant multiple of δ^3 instead of δ^4.

In the arithmetic linear PRAM model the issue of bitlengths is ignored; i.e., the running time of the algorithm cannot depend on the bitlengths of the input parameters. In that case we only need to prove the theorem when the slab B is unbounded, i.e., when its upper bounding plane goes to infinity. This is quite easy (as the reader can verify). In fact, then S must contain every plane containing the origin and an edge of G, so δ has to be at least ρ–this completes our lower bound proof for the arithmetic linear PRAM model.

Proof of Theorem 4.2.1 from Theorem 4.3.1

Before we prove the theorem, let us see why it implies that the partition as in Theorem 4.2.2 cannot exist. This will prove Theorem 4.2.1. Let $t = \sqrt{\log \rho}/a'$, as in Theorem 4.2.2. Let S be the set of planes mentioned in its statement. Their total number δ is at most 2^{5t^2}, so $\log \delta \leq 5t^2 = 5 \log \rho / a'^2$. Choose the constant a' large enough so that $\log \rho$ is sufficiently larger than $\log \delta$ as required in Theorem 4.3.1. Next fix $\bar{\mu}$ so that $\log \bar{\mu}$ is a sufficiently large multiple of $\log \mu$, as also required in Theorem 4.3.1. If we choose the permissibility constant a (cf. the statement of Theorem 4.0.3 and 4.2.1) large enough, then all integer points in the slab B will be permissible. Now Theorem 4.3.1 implies that the arrangement as in the first part of Theorem 4.2.2 cannot exist for such constants a and a'. The proof of the second part is similar. $\square$

Proof of Theorem 4.3.1

Assume, to the contrary, that no face in the partition of B contains an integer point below fan(G) and also an integer point strictly above fan(G). In what follows, this property of the partition will be called the *separation property*. Without loss of generality, we can assume that no integer point in the slab B lies on any plane in S; otherwise, one can ensure this, without losing the separation property, by replacing each plane with two planes obtained by infinitisimal translation in the positive and negative normal directions. The regions of the partition of B are convex polytopes that can have many facets in general. So we shall refine this partition, using a well known idea in computational geometry (eg. see [34]) so that each region in the resulting decomposition is a "cylinder"

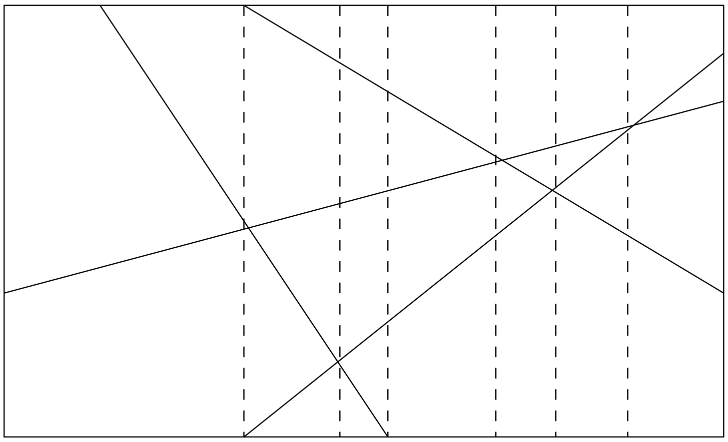

Figure 4.2: Decomposition $A(Q)$ in the affine plane

with at most six facets. Then using the pigeonhole principle we shall show that some cylinder in this decomposition must violate the separation property.

Decomposition

Let Q be the set that contains the planes in S and the planes bounding the slab B. Project onto the affine plane the intersection between every pair of planes in Q (restricted to the slab B). This yields an arrangement of lines within the bounding box in the affine plane (Figure 4.2). We refine this arrangement further by passing vertical lines (parallel to the y-axis) through all intersections among these lines and also intersections with the bounding box (Figure 4.2). Let us denote the resulting partition of the bounding box by $A(Q)$. Every region (cell) in this partition is monotone with respect to the y direction. In other words, its intersection with any line parallel to the y-axis is connected, if nonempty.

The affine partition $A(Q)$ can be lifted to the slab B in a natural fashion: We pass fans through all projected lines in the affine plane; these are the same as the fans through pairwise intersections of the planes in Q. We also pass fans through all auxiliary vertical lines that we added in the affine plane. These fans along with the planes in Q yield a decomposition of B; we shall denote it by $D(Q)$. The total number $d(Q)$ of cells (three dimensional regions) in $D(Q)$ is $O(\delta^4)$, and each cell is a convex cylinder with at most six facets. By a cylinder we mean a convex polytope which is monotone; i.e., its intersection with any ray coming from the origin is connected, if nonempty.

Sample points

Next, we choose rational sample points on all (bounded) edges of the graph G; in what follows we just ignore the two unbounded edges of G. Let c be a large enough positive integer constant to be specified later. Divide each edge of the graph into intervals of equal spans along the x-direction by placing δ^c points in its interior. These points will be called *sample points.* The x-coordinates of any two sample points differ by at least $1/(\mu\delta^c)$, because the coordinates of all vertices in G are rationals that can be expressed in terms of denominators and numerators of absolute value at most μ.

Given a sample point p on e, we say that a cell R in $D(Q)$ is *good* for p if its interior contains an integer (lattice) point lying on the ray through p (coming from the origin). We say that R is *good* for an edge e in the graph G, if it is good for $[1/d(Q)]$-th fraction of the sample points on e.

Lemma 4.3.2 *$D(Q)$ contains a cell that is good for $[1/d(Q)]$th fraction of the edges in G.*

Proof. The number of cells in $D(Q)$ is at most $d(Q)$. Hence, by the pigeon-hole principle, it suffices to show that for every edge e in the set there is at least one good cell in $D(Q)$. Applying the pigeon-hole principle once again, it suffices to show that, for every sample point p on e, there is at least one good cell in $D(Q)$. The ray through p is split into at most $\delta + 1$ intervals within B by the planes in Q. So the vertical span of at least one such interval–call its e_p–must be at least $\bar{\mu}/(\delta + 1)$. The interior of e_p must contain an integer lattice point. This follows from the last property and the following:

1. The coordinates of the sample point p can be expressed as rationals with numerators and denominators of bit length at most $\log \mu$, ignoring a constant factor.

2. By our assumption, $\log \bar{\mu}$ is greater than sufficiently large constant multiples of $\log \mu$ and $\log \delta$.

Thus the cell in $D(Q)$ containing e_p is good for p. ∎

Given an integer point lying on the fan through an edge in G, let us define its neighbour to be the unique integer point with the same x and z coordinates but whose y coordinate is greater by one. Integer points on the fan through G are all feasible whereas their neighbours are infeasible. This mean if a cell in $D(Q)$ contains an integer point on fan(G) it cannot contain its neighbour.

The graph G contains ρ edges. By the preceding lemma, there is a cell C in $D(Q)$ that is good for at least $\bar{\rho} = \rho/d(Q)$ such edges. Their number is quite large since $d(Q) = O(\delta^4)$ and ρ is larger than a sufficiently large multiple of δ^4. This means there exist a large number of edges in the graph G such that C contains a large number of integer points on the fan through each of these many edges. But it cannot contain the neighbours of these integer points. Since C is a convex cylinder with at most six facets, this is impossible (as the reader can verify).

This proves Theorem 4.3.1.

Chapter 5

PRAM without bit operations

In this chapter we shall prove our general lower bound for the PRAM model without bit operations (Theorem 3.1.1) by extending the proof in Chapter 4. As there, we shall prove a stronger nonuniform lower bound (Theorem 5.2.1) from which Theorem 3.1.1 will follow. The main difference from the linear PRAM setting is the presence of general multiplication in the instruction set. This causes some algebraic geometry to enter the proof in an essential way.

5.1 The number of possible branchings

First, we extend the general technique for bounding the number of possible ways in which the processors can branch (Section 4.1). We shall describe the technique in a language-independent setting, following the same notation as in Section 4.1.

Let $\Sigma = \Sigma(x) = \Sigma(x_1, \cdots, x_n)$, $x_i \in Z$, be the language for which want to prove a lower bound. Here $x_1, \cdots, x_n$ denote the integer parameters in the input. Fix a nonuniform machine in the PRAM model without bit operations, with $p(n, N)$ processors, which works correctly on inputs with n parameters with total bit length at most N in $t(n, N)$ time. We define the t-equivalence classes with respect to this machine just as in Section 4.1, the main difference being that the processors can now multiply. The number of t-equivalence classes, in general, does not have a good upper bound. So we get around this problem, as in Section 4.1, by parametrizing the inputs to the machine and then bounding the number of t-equivalence classes among the parametrized inputs.

So, let us imagine running our machine on the set of possible inputs defined by: $x_i = l_i(z) = l_i(z_1, \cdots, z_d)$, where, for each $i \leq n$, the parametrizing linear form l_i is exactly as in Section 4.1. As there, let $\Sigma(z_1, \cdots, z_d)$ denote the set of permissible d-tuples accepted by the machine. We say that z is accepted if $x = l(z) = (\cdots, l_i(z), \cdots)$ is accepted. Two permissible d-tuples z, z' are said to be t-equivalent if $l(z)$ and $l(z')$ are t-equivalent. The notion of equivalence is defined similarly. Let $\sigma(t)$ denote the number of t-equivalence classes among the permissible values of z. Let $\sigma = \sigma(t(n, N)))$ denote the number of equivalence classes. The following theorem generalizes Theorem 4.1.1.

Theorem 5.1.1 *For all t, $\sigma(t)$ is bounded by $[2 + 2 \cdot 2^t p(n, N)]^{dt}$.*

Proof. Let C denote a fixed t-equivalence class. Let z denote a (permissible) integral value in this equivalence class. In what follows, we pretend that z is a generic variable. We are told that it belongs to C. But we are not told its actual value. We imagine running our machine on the input $x = l(z)$. We have to specify what is meant by executing an arithmetic operation in this setting. Now the operand would be polynomials in z, rather than linear forms as in the linear PRAM setting. By an arithmetic operation $\times, +$ or $-$, we simply mean the corresponding polynomial operation. Since nonnumeric parameters are fixed by our parametrization, indirect memory references are no problem. We assume that when a processor is executing a comparison it is simply told the result of that comparison, which, at any time less than or equal t, is fixed by the class C. The following lemma generalizes Lemma 4.1.2.

Lemma 5.1.2 *Let C be a fixed t-equivalence class. Then, for every memory location j, there exists a polynomial $g_j[C,t](z)$ that gives the content of that location at time t for every $z \in C$. Moreover, the degree of g_j is at most 2^t.*

Proof. Once C is fixed, the execution path of each processor up to time t is fixed. So each memory location at time $\leq t$ is a completely determined function of $z_1, \ldots, z_d$–in fact, a polynomial function. At time $t = 0$, each memory location is either a constant or is equal to some $l_i(z)$; so it is a polynomial of degree at most one. Unlike in the linear PRAM setting, now we have general multiplication. This corresponds to multiplying two polynomials in $(z_1, \ldots, z_d)$; the degree of their product is the sum of their degrees. So the lemma follows by induction on t. ▯

Let us now see what happens at time t, given that the generic variable z belongs to a fixed t-equivalence class C as above. If a processor is to execute an arithmetic operation at time t, then the result of that operation is a certain uniquely determined polynomial to be stored in a certain uniquely determined location in the memory. If the operation is a branch, the branching path is determined by comparing contents of two uniquely determined memory locations, which are certain fixed polynomials in z. In other words, each branch is determined by testing a certain polynomial equality or inequality in z of the form $g(z) : 0$, where the comparison : is $>, \geq$ or $=$. The number of such polynomial constraints is at most $p(n, N))$, one per processor. Let $g_1(z), g_2(z), \cdots$ be the nonzero polynomials involved in these constraints. Consider the partition (stratification) of R^d formed by the hypersurfaces $g_i(z) = 0$. A sign-invariant component of this stratification is defined to be the maximal set of points such that the sign of any g_i remains the same over the whole set; it need not be connected or even simply connected. The sign of a function at a given point is defined to $+, -$ or 0 depending on whether its value at the point is positive,

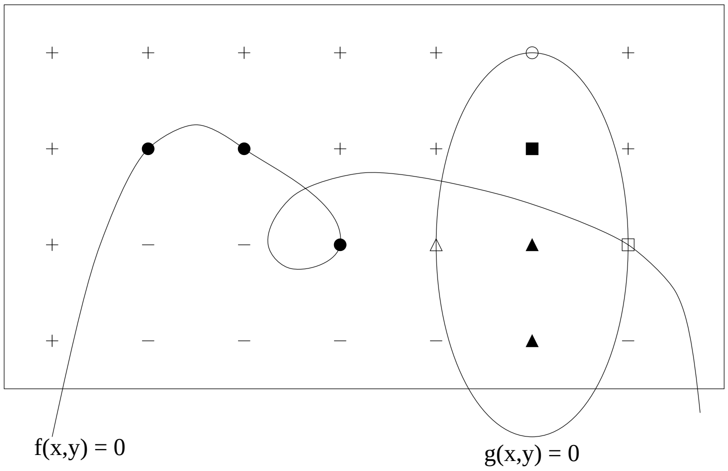

Figure 5.1: Before perturbation (integer points in different sign-invariant components are depicted differently).

negative or zero. We say that a sign-invariant component is nonempty if it contains a permissible integer point.

Theorem 5.1.3 (Milnor-Thom) *The number of nonempty sign-invariant components in the preceding stratification of R^d is bounded by* $[2 + 2\sum \deg(g_i)]^d$.

Proof. If no hypersurface under consideration contains any permissible integer point, then this follows directly from the Milnor-Thom bound [33, 48] on the number of connected components of the set $\bigcup_i \{g_i(z)^2 > 0\}$. Otherwise, we can replace each hypersurface $g_i(z) = 0$ by two hypersurfaces $g_i(z) = \epsilon$ and $g_i(z) = -\epsilon$, where ϵ is an infinitesimal positive real (Figures 5.1 and 5.2). Since the number of permissible integer points is finite, we can choose ϵ small enough so that no new hypersurface contains any permissible integer point and the partition of the permissible integer points induced by the sign-invariant components of the old stratification coincides with the partition induced by the sign-invariant components of the new stratification. Now we can apply the Milnor-Thom bound as before. ▯

Because the number of polynomials g_i is at most $p(n, N))$, one per processor, and $\deg(g_i)$ is at most 2^t, it follows from the preceding result that the total number of sign-invariant components in the preceding partition of R^d is at most $[2 + 2p(n, N))2^t]^d$. Now we make the crucial:

Observation 5.1.4 *Any two $z, z' \in C$ belong to the same $(t + 1)$-equivalence class if they belong to the same sign-invariant component of this partition.*

It immediately follows that

$$\sigma(t + 1) \leq [2 + 2p(n, N)2^t]^d \sigma(t).$$

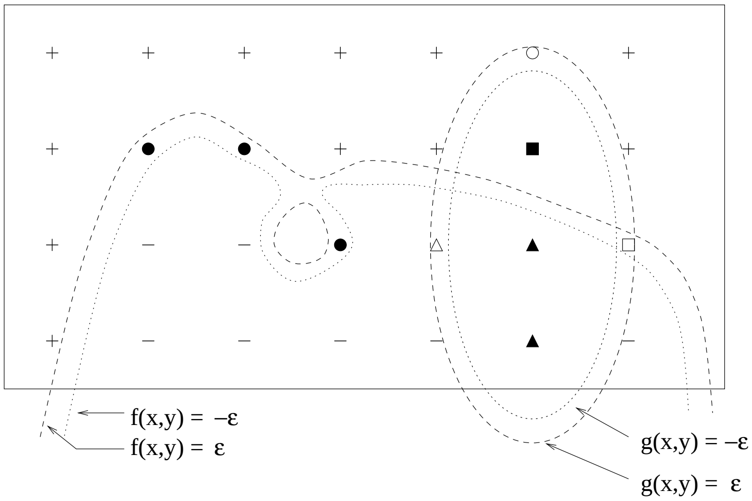

Figure 5.2: After perturbation: the sign-invariant partition of the integer points is unchanged.

Induction on t proves Theorem 5.1.1. ▯

The proof of Theorem 5.1.1 also yields:

Lemma 5.1.5 *For any t-equivalence class C, there exists a set $\Phi(C,t)$ of at most $p(n,N))t$ polynomial constraints (equalities or inequalities) in d variables such that $z \in C$ iff all these constraints are satisfied. Moreover, the degree of every polynomial occurring in $\Phi(C,t)$ is at most 2^t.*

Proof. The constraints correspond to the branches taken by all processors before time t, which are all fixed by the class C. ▯

By Lemma 5.1.5, each equivalence class D of permissible z-values is characterized by a set $\Phi(D) = \Phi(D, t(n,N)))$ of polynomial constraints. Let $\Phi = \cup_D \Phi(D)$, where D ranges over all equivalence classes. By Theorem 5.1.1 and Lemma 5.1.5, it follows that Φ contains at most

$$[2 + 2p(n,N)2^{t(n,N)}]^{dt(n,N)} p(n,N)t(n,N)$$

polynomials of degree at most $2^{t(n,N)}$. Consider the partition of R^d formed by the hypersurfaces defined by these polynomials. Each sign-invariant component of this partition can be labeled *yes* or *no* in such a way that a permissible z is accepted by the machine iff it lies in a sign-invariant component labeled *yes*. Thus we have proved the following.

Theorem 5.1.6 *Assume that the language $\Sigma(z_1, \cdots, z_d)$ is accepted in the PRAM model without bit operations using $p(n,N)$ processors in $t(n,N)$ time. Then R^d can be partitioned by a set of at most*

$$[2 + 2p(n,N)2^{t(n,N)}]^{dt(n,N)} p(n,N)t(n,N)$$

polynomials of degree at most $2^{t(n,N)}$ and each sign-invariant component can be labeled yes or no so that a permissible $z \in Z^d$ belongs to $\Sigma(z_1, \cdots, z_d)$ iff it lies in a sign-invariant component labeled yes.

In the proof of this theorem, performance of the machine on N-bit inputs that do not lie in the parameterized family $\Sigma(z_1, \cdots, z_d)$ does not matter. In other words, the theorem holds even if $t(n, N))$ is defined to be the maximum time taken on any N-bit input in $\Sigma(z_1, \cdots, z_d)$.

5.2 Parametrization

We shall now prove a general lower bound in the PRAM model without bit operations that is nonuniform with respect to parametrization. It generalizes Theorem 4.2.1, and we follow the same notation as in Section 4.2.

Theorem 5.2.1 *Assume that the permissibility constant a is large enough. Then no machine in the PRAM model without bit operations can decide whether $w \leq F(I)$ correctly for every $I \in \mathcal{I}$ in $\sqrt{\log \rho(n)}/a'$ time using $2^{\sqrt{\log \rho(n)}/a'}$ processors, for some large enough constant a' that does not depend on the permissibility constant a.*

In the deterministic setting, Theorem 3.1.1 follows from this theorem by applying it to a parametrization $\mathcal{P}$ for cardinality n with bit size $\beta(n)$, whose complexity is equal to $\phi(n, \beta(n))$, the maximum possible value. Randomized setting is considered in Section 6.1. In what follows, we shall denote $\rho(n)$ and $\beta(n)$ by simply ρ and β respectively.

Let us apply Theorem 5.1.6, with $d = 3$, to the linear parametrization of the input as in Theorem 5.2.1. It is defined (as in Section 4.2) by the linear map

$$(z_1, z_2, z_3) \to I(z_1, z_2, z_3) = (z_3, \tilde{\mathcal{P}}(z_1, z_2)).$$

Thus in Theorem 5.1.6 $\Sigma(z_1, z_2, z_3)$ consists of those tuples (z_1, z_2, z_3) such that $z_3 \leq \tilde{\mathcal{P}}(z_1, z_2)$. This gives:

Theorem 5.2.2 *Let a' be any positive constant. If a machine in the PRAM model without bit operations works correctly on the input $I(z_1, z_2, z_3)$, for every permissible z_1, z_2 and z_3, in $t = \sqrt{\log \rho}/a'$ time using 2^t processors, then R^3 can be partitioned by at most 2^{20t^2} algebraic surfaces of degree at most 2^t and each sign-invariant component of this partition can be labeled* yes *or* no *so that a permissible integer point (z_1, z_2, z_3) lies in a sign-invariant component labeled* yes *iff $I(z_1, z_2, z_3)$ is feasible.*

5.3 A lattice problem

We shall prove Theorem 5.2.1 by showing that an algebraic partition as in Theorem 5.2.2 cannot exist for large enough constants a and a'. We proved this in the linear setting in Section 4.3. We wish to generalize that proof to the algebraic setting.

Let G be the graph corresponding to the optimum function $\mathcal{P}$ as in Section 4.3 (Figure 4.1). Let $\text{fan}(G)$ be its fan. Let us also rename the coordinates z_1, z_2, z_3 as z, x, and y respectively. Thus $z = 1$ now serves as the affine plane in R^3. Then in the partition of Theorem 5.2.2, a permissible integer point in Z^3 is labeled *yes* iff it lies below $\text{fan}(G)$ in the y-direction.

Recall that (1) the graph G is piecewise linear and convex with ρ segments, and (2) all vertices of G have rational coordinates that can be expressed in terms of numerators and denominators of absolute value at most $\mu = 2^\beta$, or in other words, of bit size at most $\log \mu$. All vertices of G lie in the interior of the bounding box defined by: $|x| \leq 2\mu$, and $|y| \leq 2\mu$. Consider two horizontal planes, parallel to the affine plane, defined by $z = \bar{\mu}$ and $z = 2\bar{\mu}$, where $\bar{\mu}$ would be chosen to be much larger than μ. Let B denote the region (slab) between the two planes that is enclosed by the fan through the boundary of the bounding box in the affine plane. The coordinates of all integer points in this slab have bit sizes at most twice $\bar{\beta} = \log \bar{\mu}$. The following theorem would imply that the partition as in Theorem 5.2.2 cannot exist.

Theorem 5.3.1 *Let S be any set of surfaces in R^3 with total degree δ. Assume that $\bar{\beta} = \log \bar{\mu}$ is greater than a large enough constant multiple of $\beta = \log \mu$ and that $\log \rho$ is greater than a large enough constant multiple of $\log \delta$. Then at least one sign-invariant component in the partition of B formed by the surfaces in S contains an integer point lying below $\text{fan}(G)$ and also an integer point strictly above this fan.*

Remark. Since $\rho \leq 4\mu^2$ trivially, the assumption in the theorem also implies that $\log \bar{\mu}$ is much greater than $\log \delta$.

Before we prove the theorem, let us see why it implies that the partition as in Theorem 5.2.2 cannot exist; this will prove Theorem 5.2.1. Let S be the set of surfaces that occur in Theorem 5.2.2. Their total degree δ is at most $2^t 2^{20t^2} = 2^{21t^2}$, where $t = \log \rho / a'$. Then choose the constant a' and the permissibility constant a large enough and apply Theorem 5.3.1 to this set S exactly as we applied Theorem 4.3.1 to the set of planes occurring in Theorem 4.2.2.

In the rest of this section we shall prove Theorem 5.3.1. Assume, to the contrary, that no sign-invariant component in the partition of B contains an integer point below $\text{fan}(G)$ and also an integer point strictly above $\text{fan}(G)$. In what follows, this property of the partition will be called the *separation property*. We shall first smoothen and refine the partition of B so that the regions in the resulting partition have a simple shape. Then by the pigeonhole principle we shall show that some region in this partition must violate the separation property.

Transversality

The surfaces S in Theorem 5.3.1 depend on the nonuniform machine under consideration. In general, they can be highly singular; this cannot be helped, since we have no control over the nonuniform machine. Singularities can cause problems in our proof, but fortunately we can get rid of them as follows.

First, we can assume that no integer point in the slab B lies on any surface in S. Otherwise, one can ensure this, without losing the separation property, by replacing each surface with two surfaces obtained by perturbing the constant term in its defining equation infinitesimally–since there are only finitely many integer points in B, we can choose the perturbation small enough so that the underlying partition of the integer points in B according to sign-invariant components does not change (see Figures 5.1 and 5.2). We can also add infinitesimal reals freely to the coefficients of the polynomials defining the surfaces in S without losing the preceding or the separation property. In other words, we can assume that the surfaces in S are in *general position*–this just means we have the freedom to perturb them by adding infinitisimal reals to the coefficients of their defining equations. The general position assumption lets one handle the technical problems mentioned above in a clean fashion, because then the transversality techniques from differential geometry (Chapter 2,[16]) become fully applicable. For example, by perturbing the surfaces in S infinitisimally we can now assume that they are smooth (Sard's theorem), that they intersect transversally, that their *silhouettes* (to be defined soon) are smooth, and so forth.

Collins' Decomposition

Let Q be the set that contains (1) the surfaces in S, (2) the planes bounding the slab B, and (3) 6δ horizontal *dividing* planes that subdivide the slab B into horizontal slabs of equal height. By horizontal, we mean parallel to the affine plane.

Consider the partition of B formed by the surfaces in Q. The regions in this partition can have very complicated shapes. So we shall refine the partition further so that the regions in the resulting partition have simple shapes following a variant of the method due to Collins [10]. Before we define the partition formally, let us make a definition.

Let s be a surface in S. We say that a point p on s belongs to the *silhouette* of s, as seen from the origin, if the line joining p and the origin is tangent to s at p. This happens iff

$$f(x,y,z)=0, \text{ and } x\frac{\partial f}{\partial x}+y\frac{\partial f}{\partial y}+z\frac{\partial f}{\partial z}=0, \tag{5.1}$$

where $f(x,y,z)=0$ is the polynomial equation defining s. The silhouette of s, if nonempty, is a smooth space curve if s is in general position (Chapter 2, [16]).

Let us project onto the affine plane (1) the silhouette of every surface in S (restricted to the slab B), and (2) the intersection between every pair of surfaces

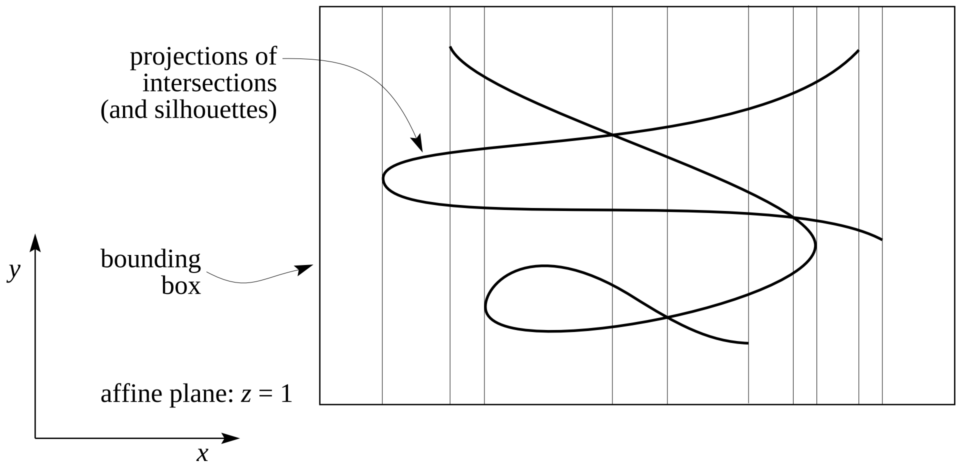

Figure 5.3: Decomposition $A(Q)$ in the affine plane

in Q; by our general position assumption, this intersection is a smooth space curve.

The preceding projection yields an arrangement of curves within the bounding box in the affine plane (Figure 5.3). We refine this arrangement further by passing vertical lines (parallel to the y-axis) through all intersections among the curves, singular[1] points on the curves, and also critical points on the curves where the tangents become parallel to the y-axis. Let us denote the resulting partition of the bounding box by $A(Q)$. It is a two-dimensional Collins' decomposition. Every region (cell) in this partition is monotone with respect to the y direction. In other words, its intersection with any line parallel to the y-axis is connected, if nonempty.

The affine partition $A(Q)$ can be lifted to the slab B in a natural fashion: We pass fans through all projected curves in the affine plane; these are the same as the fans through pairwise intersections and silhouettes of the surfaces in Q. We also pass fans through all auxiliary vertical lines that we added in the affine plane. These fans along with the surfaces in Q yield a decomposition of B. We shall denote it by $D(Q)$. It has the following properties [10]:

1. Each region, i.e., a three-dimensional cell in $D(Q)$, has at most 6 sides. It is monotone when viewed from the origin. In other words, its intersection with any ray is connected, if nonempty. By a ray, we mean here and in what follows, a ray coming from the origin. (In Collins' original construction, the regions of $D(Q)$ are monotone with respect to the z-direction.) Monotonicity allows us define the top side, the *roof*, and the bottom side, the *floor*, of each region unambiguously. Formally, the floor consists of all points on the boundary of the region that are hit first by the rays coming

[1] Transversality conditions cannot remove such singular points on projections.

from the origin; the roof is defined similarly.

2. The floor of each region is contained in just one surface in Q; the same is true of the *roof.*

3. The remaining sides of the region are contained in fan-surfaces.

4. The projection of each cell in $D(Q)$ is a cell in the affine partition $A(Q)$.

5. The total number $d(Q)$ of cells in $D(Q)$ is $O(\delta^{O(1)})$. This follows from the Milnor-Thom result [33, 48] because the total degree of the surfaces in Q, along with the fan-surfaces, is $O(\delta^{O(1)})$.

Sample points

Next, we choose rational sample points on the edges of the graph G exactly as in Section 4.3. The x-coordinates of any two sample points differ by at least $1/(\mu\delta^c)$.

Let us call a cell R in $D(Q)$ *flat* if its roof and also the floor are contained in the dividing horizontal planes. Given a sample point p on e, we say that R is *good* for p if its interior contains an integer (lattice) point lying on the ray through p. We say that R is *good* for a bounded edge e in the graph G, if it is good for $[1/d(Q)]$-th fraction of the sample points on e.

Lemma 5.3.2 *$D(Q)$ contains a flat cell that is good for $[1/d(Q)]$th fraction of the edges in G.*

Proof. The number of flat cells in $D(Q)$ is at most $d(Q)$. Hence, by the pigeon-hole principle, it suffices to show that for every bounded edge e in G there is at least one good flat cell in $D(Q)$. Applying the pigeon-hole principle once again, it suffices to show that, for every sample point p on e, there is at least one good flat cell in $D(Q)$. The ray through p is split into $6\delta+1$ intervals by the 6δ dividing planes. At most δ of these intervals are intersected by the surfaces in S. To see this, note that the surfaces in S are in general position and the sample points do not depend on S. Hence, the ray through p is intersected by the surfaces in S in a finite number of points, and this number is at most δ, the total degree of the surfaces in S, by the simplest special case of the Milnor-Thom result (which is essentially Bezout's result). It follows that there is at least one such interval in the interior of B that is not intersected by any surface in S. Denote this interval by e_p. Let P_1 and P_2 be the consecutive dividing planes adjacent to this interval. Note that:

1. The interval e_p is contained in some cell of $D(Q)$.

2. The endpoints of e_p lie on P_1 and P_2. But they do not lie on any surface in S, if the surfaces in S are in general position. The latter fact follows because the rays through the sample points on the graph G as well as their intersections with the dividing planes are finite in number and they do

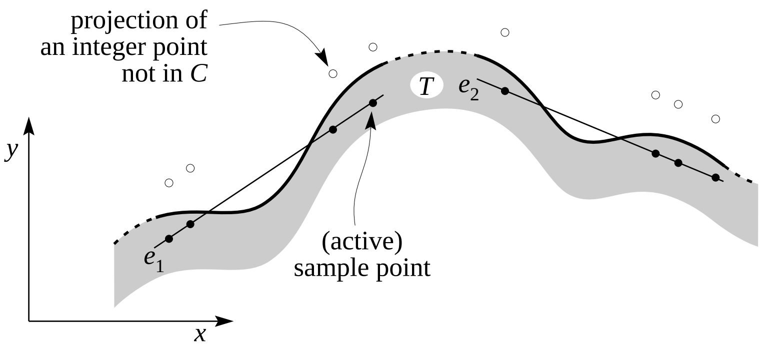

Figure 5.4: Upper side of T

not depend on S. Hence, we can perturb the surfaces in S infinitesimally before the construction of the decomposition $D(Q)$ begins and ensure that no surface in S contains any such intersection.

3. The preceding property, in conjunction with the properties of $D(Q)$, implies that the roof and the floor of the cell in $D(Q)$ containing e_p must be contained in the planes P_1 and P_2, respectively. Hence, this cell must be flat.

4. The vertical span of e_p is $\bar{\mu}/(6\delta + 1)$.

The interior of e_p must contain an integer lattice point. This follows from the last property just as in the proof of Lemma 4.3.2. Thus the flat cell containing e_p is good for p. ▯

The graph G contains ρ bounded edges of distinct slopes. By the preceding lemma, there is a flat cell C in $D(Q)$ that is good for at least $\bar{\rho} = \rho/d(Q)$ such edges. Let these $\bar{\rho}$ edges be $e_1, e_2, \ldots$ ordered from left to right in the affine plane. Such ordering is possible because the graph G, being a function graph, is monotone with respect to the y direction. Let T be the projection of C onto the affine plane. It belongs to the affine partition $A(Q)$. Hence, it is monotone in the y direction. Let g denote the curve containing the upper side of T. By the definition of $A(Q)$, the upper side of T is monotone with respect to the y direction and contains no singularities. Hence, it defines a certain smooth function $y = \bar{g}(x)$ within the span of T along the x-axis.

Lemma 5.3.3 *The second derivative of the function $\bar{g}(x)$ has at least $\bar{\rho} - 1$ extrema within the span of T along the x-axis.*

Proof. The basic idea is to show that the curve g gets very close to several sample points on each segment e_i. Since these segments are not too far apart, and their slopes are sufficiently different, g must change its direction several times (Figure 5.4).

Fix any e_i. The cell C is good for e_i. Hence, it is good for at least $[1/d(Q)]$th fraction of the sample points on e_i. Since the number of sample points on e_i is δ^c and $d(Q)$ is $O(\delta^{O(1)})$, there are at least eight such sample points in the interior of each e_i, assuming that the constant c is large enough. In what follows, these eight sample points will be the only active sample points on e_i; all other sample points are discarded. Order these remaining sample points from left to right. Fix any such sample point p. It must lie within T, because C contains an integer lattice point–call it $\bar{p}$–on the ray through p.

Claim: The difference between the y-coordinate of p and the value of the function $\bar{g}(x)$ at the x coordinate of p is at most $1/\bar{\mu}$.

Proof of the claim: Consider the integer point $\hat{p}$ with the same x and z coordinates as the integer point $\bar{p}$, but whose y coordinate is one plus the y-coordinate of $\bar{p}$. This point certainly lies in the slab B, since the bounded edge of G containing p is well within the interior of the bounding box in the affine plane. The point $\hat{p}$ does not lie on any ray through the graph G, because p lies on G and G is monotone in the y direction. Since our partition has the separation property, the cell C cannot contain $\hat{p}$, because it already contains $\bar{p}$. Let q denote the projection of $\hat{p}$ onto the affine plane. It lies above p and has the same x-coordinate as p. Moreover, its y-coordinate differs from that of p by at most $1/\bar{\mu}$. This follows because the z-coordinate of $\hat{p}$–in fact, of every point in the slab B–is at least $\bar{\mu}$. Since the cell C does not contain $\hat{p}$ and T is monotone in the y direction, the upper side of T must separate p and q (Figure 5.4). This proves the claim.

Let $a_1, a_2, \ldots$ be the slopes of $e_1, e_2, \ldots$. They are all different. The difference between any two slopes, ignoring a constant factor, is at least $1/\mu^2$, because the coordinates of all vertices in G can be expressed as rationals with numerators and denominators of absolute value at most μ. The x-coordinates of any two sample points differ by at least $1/(\mu\delta^c)$, as we remarked earlier. These facts, in conjunction with the previous claim and Roelle's mean value theorem in elementary calculus, imply the following:

1. The derivative $d\bar{g}(x)/dx$ attains a value close to a_i between the x-coordinates of the jth and $(j+1)$st sample points on e_i, for all $1 \leq j < 8$. By close we mean that the error term is $O(\mu\delta^c/\bar{\mu})$.

2. This implies further that the second derivative attains a value close to zero between the x-coordinates of, say, the third and the sixth sample points on e_i. Similarly, somewhere between the x-coordinates of the seventh sample point on e_i and the second sample point on e_{i+1}, the absolute value of the second derivative is $\Omega(|a_{i+1} - a_i|/\mu)$, ignoring an additive $O(\mu^2\delta^{2c}/\bar{\mu})$ error term; here one also needs to use the fact that the x-coordinates of any two sample points differ by at most μ.

3. The preceding fact implies that every pair (e_i, e_{i+1}) contributes at least one distinct extremum of the second derivative, because $\log\bar{\mu}$ is greater than sufficiently large constant multiples of $\log\mu$ and $\log\delta$.

The lemma follows from the last fact. $\square$

Claim: The second derivative of the function $\bar{g}(x)$ has at most $O(\delta^{O(1)})$ extrema in the span of T along the x-axis.

Proof. Let $g(x, y) = 0$ be the polynomial equation satisfied by the upper side of T. Its degree is at most $O(\delta^{O(1)})$, because the curve g containing the upper side results by projecting a silhouette or an intersection of two surfaces in S. Differentiating this equation three times in a row, we get four implicit equations satisfied by x, y and the three derivatives $y^{(1)}, y^{(2)}$ and $y^{(3)}$:

$$g(x, y) = 0, \; g_1(x, y, y^{(1)}) = 0, \ldots$$

where $y^{(k)}$ is a formal variable that denotes the kth order derivative. In addition, the extrema satisfy $y^{(3)} = 0$. Because the surfaces in S are in general position, these five equations in five variables have finite number of real solutions. This number is $O(\delta^{O(1)})$, by the Milnor-Thom result[2]. Hence, the claim follows. (Formally, finiteness of the number solutions, used above, follows from the jet-transversality results; cf. Chapter 2, [16]. Alternatively, one can forget about transversality and just bound the number of connected components of the solution set using the Milnor-Thom result. The rest of the proof then needs to be modified a bit.)

The claim contradicts the preceding lemma because $\bar{\rho} = \rho/d(Q)$, $d(Q) = \delta^{O(1)}$, and $\log \rho$ is larger than a sufficiently large multiple of $\log \delta$.

This proves Theorem 5.3.1.

[2]Or, one can also apply Bezout's theorem. But one has to be a bit careful. Bezout's theorem is applicable only if the number of complex solutions, not just the real solutions, is finite. But this technicality can be taken care of as in Lemma 1 of Milnor [33].

Chapter 6

Extensions

6.1 Randomized algorithms

All our lower bounds extend to randomized algorithms in the PRAM model without bit operations. The following result generalizes Theorem 3.1.1.

Theorem 6.1.1 Fix any homogeneous optimization problem. Let $\phi(n, \beta(n))$ be its parametric complexity for input cardinality n and bitsize $\beta(n)$. Then there exists a large enough constant b such that the decision version of the problem can not be solved in the PRAM model without bit operations using randomization in $\sqrt{\log[\phi(n, \beta(n))]}/b$ expected time using $2^{\sqrt{\log[\phi(n,\beta(n))]}/b}$ processors; this is so even if we restrict every numeric parameter in the input to be an integer with bitlength at most $a\beta(n)$ for a large enough constant a.

This also applies to the additive approximation version of the problem.

To prove it, we only need to extend Theorem 5.2.1. Its analogue in the randomized setting, following the same notation, is:

Theorem 6.1.2 *Assume that the permissibility constant a is large enough. Then no randomized machine in the PRAM model without bit operations can decide whether $w \leq F(I)$ correctly for every $I \in \mathcal{I}$ in $T = \sqrt{\log \rho(n)}/a'$ expected time using $p = 2^{\sqrt{[\log \rho(n)}/a'}$ processors with at most ϵ two-sided-error probability for some large enough constant a' that does not depend on the permissibility constant a.*

One can similarly extend Theorem 4.2.1 for the linear PRAM model; we leave this to the reader.

Now we shall prove Theorem 6.1.2. In what follows, we shall denote $\rho(n)$ and $\beta(n)$ by simply ρ and β respectively. Let $I(z_1, z_2, z_3)$ denote the total parametrized input, which includes the threshold, as in Section 5.2. Let Φ be any suitably chosen subset of the permissible integer points in Z^3. Consider a probability distribution on Z^3 that is zero outside Φ and uniform on Φ. By Yao's lemma [53], it suffices show, ignoring factors of two, that there is no deterministic machine M in the PRAM model without bit operations containing p processors that runs in expected time T, possibly erring on at most ϵ fraction of the points in Φ; here the expectation is meant to be with respect to the probability distribution on the input parameters. We can assume that ϵ is a small enough constant, say $1/100$, by increasing T suitably.

Fix another positive constant $\bar{\epsilon} = 1/100$. Let us call a permissible integer point z in Φ *satisfactory* with respect to Φ, ϵ and $\bar{\epsilon}$ if M stops on the corresponding input $I(z)$ in $t = T/\bar{\epsilon}$ time without any error. The proof of Theorem 5.2.2 can be extended *verbatim* to prove the following:

Theorem 6.1.3 *Suppose a deterministic machine M, as described above, exists in the PRAM model without bit operations. Then R^3 can be partitioned by at most 2^{20t^2} algebraic surfaces of degree at most 2^t and each sign-invariant component of this partition can be labeled* yes *or* no *so that a satisfactory permissible integer point (z_1, z_2, z_3) lies in a sign-invariant component labeled* yes *iff $I(z_1, z_2, z_3)$ is feasible.*

Now we can generalize the proof of Theorem 5.2.1 to the randomized setting. First, let us specify the set Φ that fixes the input distribution. Let G be the graph as in Section 5.3. Recall that the ray through each sample point on the graph G is split into exactly $6\delta + 1$ intervals by the horizontal dividing planes. Moreover, each such interval contains a permissible integer point (cf. the proof of Lemma 5.3.2). Fix any such integer point on every such interval; in what follows, it will be called a *distinguished* point. The *neighbour* of this distinguished point is defined to be the integer point in the slab B with the same x and z coordinates as the distinguished point, but with the y coordinate greater by plus one. The graph G contains ρ (bounded) edges with distinct slopes. We shall let Φ be the set of distinguished points, along with their neighbours, on all rays through the sample points on these edges. Observe that the neighbours of all distinguished points are disjoint; this follows because G is monotone in the y-direction. Hence, exactly half of the points in Φ are distinguished. Let us call a distinguished point *satisfactory* if it is satisfactory and so is its neighbour. The number of unsatisfactory distinguished points is at most $2(\epsilon + \bar{\epsilon})$th fraction of the number of points in Φ. Let us call a sample point on an edge of G *satisfactory* if at least half of the distinguished points on the ray through it are satisfactory. Since the rays through all sample points have exactly the same number of distinguished points, the number of such unsatisfactory sample points is at most $4(\epsilon + \bar{\epsilon})$th fraction of the total number of sample points on the edges of G. Let us call an edge of G *satisfactory* if at least half of the sample points on that edge are satisfactory. Since all edges of G have the same number of sample points, the number of unsatisfactory edges is at most $8(\epsilon + \bar{\epsilon})\rho \leq 8\rho/50$. Discard all unsatisfactory edges and all unsatisfactory sample points on the satisfactory edges. We are still left with a large number of satisfactory quantities.

Now we can translate the proof of Theorem 5.2.1, with the use of Theorem 5.2.2 substituted with that of Theorem 6.1.3, making sure that everything involved is satisfactory. Thus, we say that a flat cell R in $D(Q)$ is good for a (satisfactory) sample point p, we mean that R contains a (satisfactory, distinguished) integer point on the ray through p. The proof of Lemma 5.3.2 can be translated easily, since the number of unsatisfactory intervals, i.e., the intervals containing unsatisfactory distinguished points, on any sample ray is at most $3\delta + 1$, and they can be thrown away. Similarly, Lemma 5.3.3 holds in this extended setting, because the only integer points involved in its proof can be assumed to be satisfactory distinguished points and their satisfactory neighbours.

This proves Theorem 6.1.2.

6.2 PRAM with limited bit operations

We now indicate how our lower bounds can be extended to the PRAM model with limited bit operations.

Theorem 6.2.1 *The lower bounds in Theorem 3.1.1 and 6.1.1 also hold in the PRAM model with limited bit operations.*

For the sake of simplicity, we shall only consider deterministic setting.

We only indicate the changes that need to be made to the proof for the PRAM model without bit operations. Suppose there were to exist a machine M in this model for the problem under consideration that works in $t(n, N)$ time using $p(n, N)$ processors. Let $z_1, \ldots, z_d$ be the integer parameters that parametrize the input to this machine as in Section 5.1. Let $\sigma(t)$ denote an equivalence class of parametrized inputs at time t, where two inputs belong to the same $\sigma(t)$ iff all processors in the machine follow the same branching path up to time t when the machine is started on either of the inputs,

Claim. *Fix an equivalence class $\sigma(t)$ and the value of the parametrized input $z = (z_1, \ldots, z_d)$ modulo $2^{t(n,N)}$; i.e., z_i modulo $2^{t(n,N)}$ is fixed for every i. Then for all inputs z in this equivalence class having the same value modulo $2^{t(n,N)}$, and any time t, the value of any memory location modulo $2^{t(n,N)-t}$ at that time is the same.*

Proof: By induction on t. The base case is clear. So consider $t > 1$. Once $\sigma(t)$ is fixed, so is the corresponding $\sigma(t-1)$. By the inductions hypothesis, the values of all memory locations modulo $2^{t(n,N)-t+1}$ are fixed. Consider any memory location u. At time t, either u remains unchanged or is changed in accordance with an assignment of type:

1. $u := v\{+, -, \text{or}, *\}w$, where v and w are memory locations or constants; in this case the value of u at time t modulo $2^{t(n,N)-t}$, in fact, modulo $2^{t(n,N)-t+1}$ as well, is fixed by the values of v and w at time $t-1$ modulo $2^{t(n,N)-t+1}$,

2. or, $u := \lfloor u/2 \rfloor$; in this case the value of u at time t modulo $2^{t(n,N)-t}$ is fixed by its value at time $t-1$ modulo $2^{t(n,N)-t+1}$. $\square$

It follows that once we fix the input z modulo $2^{t(n,N)}$, and an equivalence class $\sigma(t)$, $t \leq t(n,N)$, then each assignment of type $u := \lfloor u/2 \rfloor$ that occurs on the computation path of any processor up to time t can be replaced unambiguously by a rational assignment $u := u/2$ or $u := u/2 - 1$. Now it can be checked that the proof of Theorem 5.1.6 goes through verbatim provided we concern ourselves to z-inputs having the same value modulo $2^{t(n,N)}$. For the sake of simplicity, we shall confine ourselves to z-inputs that are zero modulo $2^{t(n,N)}$; we call them *green* inputs. We are thus led to the following extension of Theorem 5.1.6 in the extended model.

Theorem 6.2.2 *Assume that the language $\Sigma(z_1, \cdots, z_d)$ is accepted in the PRAM model with limited bit operations using $p(n,N)$ processors in $t(n,N)$ time. Then R^d can be partitioned by a set of at most*

$$[2 + 2p(n,N)2^{t(n,N)}]^{dt(n,N)}p(n,N)t(n,N)$$

polynomials of degree at most $2^{t(n,N)}$ and each sign-invariant component can be labeled yes or no so that a green *permissible $z \in Z^d$ belongs to $\Sigma(z_1, \cdots, z_d)$ iff it lies in a sign-invariant component labeled yes.*

Now we are ready to extend the proof Theorem 5.2.1 to the extended model. For the rest of the proof, we simply confine ourselves to the green inputs. Note that the green z points form a sublattice of the original integer lattice. If we were change our coordinate system so as to scale the green lattice by a factor of $1/2^{t(n,N)}$ then it would look like our original lattice. What is the effect of this scaling on the rest of the proof of Theorem 5.2.1? Well, the quantities μ and $\bar{\mu}$ that occur in Section 5.3 have to be scaled by $1/2^{t(n,N)}$, but the remaining quantities such as ρ and δ are unaffected. If we choose the constant a' there large enough, $\log \mu$ and $\log \bar{\mu}$ are scaled down by at most the same constant factor, say 2. But it can be checked that the rest of the proof in Section 5.3 is robust with respect to a constant factor scaling of $\log \mu$ and $\log \bar{\mu}$. This proves Theorem 5.2.1 in the extended PRAM model with limited bit operations.

6.3 The number of iterations in sequential algorithms

Our lower bound for the mincost flow and max flow problems, which are instances of (combinatorial) linear programming, implies an interesting lower bound on the number of iterations in the sequential iterative algorithms for linear programming such as the ellipsoidal algorithm or the interior-point algorithm. These algorithms consist of a sequence of iterations, and each iteration can be accomplished—at least in theory—in $O(\log^2 n)$ parallel steps using polynomial number of processors; this can be done using fast parallel algorithms for linear algebra [11] and approximate root extraction [3, 38]. It follows from Theorem 5.2.1 that, given any parametrized family $\mathcal{I} = I(z)$ of inputs, these algorithms must require $\Omega(\sqrt{\log \rho(n)}/\log^2 n)$ iterations for some instance in the family; here $\rho(n)$ is defined for the family $\mathcal{I}$ as in Theorem 5.2.1. In the worst case–e.g. when $\mathcal{I}$ is a parametric family that attains maximum parametric complexity for the mincost flow problem–this lower bound is $\Omega(n^{1/4}/\log^2 n)$. A similar lower bound applies to any iterative algebraic algorithm, deterministic or randomized. By an iterative algorithm, we mean an algorithm whose each iteration admits efficient parallelization in a theoretical sense. In practice, such an iteration may not be actually carried out in parallel, because parallel algorithms that are efficient in theory are not always practical. Rather, we think that parallelizability of a step indicates that it may be amenable to fast special-purpose algorithms. For example, fast algorithms for solving sparse linear systems seem to be useful in speeding up the iteration in the interior-point method. Hence, the number of iterations is an important criterion in practice.

Now suppose we are given an iterative algebraic algorithm, deterministic or randomized, whose each step can be efficiently parallelized in $\tau(n)$ parallel steps. It follows similarly that, given any family $I(z)$ of inputs, this algorithm must require $\Omega(\sqrt{\log \rho(n)}/\tau(n))$ iterations for some instance in the family.

6.4 Weighted MAX-SNP problems

Once one has shown that one problem is hard to parallelize in the PRAM model without bit operations, it follows, as in the NP-completeness theory, that any problem that can be reduced to it fast in parallel in this model is also hard to parallelize. In this fashion one can show that several weighted MAX-SNP problems [40] are also hard to parallelize in the PRAM model without bit operations.

Theorem 6.4.1 *The following problems cannot be solved in the PRAM model without bit operations (or with limited bit operations) deterministically (or with randomization) in $O(N^a)$ (expected) time using 2^{N^a} processors, where N denotes the input bit length and a is a small enough positive constant: In all cases, the weights are meant to be nonnegative integers. (For the definitions of these problems see [40].)*

1. *Weighted 3-SAT-B, weighted 3-SAT-V, and weighted 2-SAT.*
2. *Weighted independent set-B.*
3. *Weighted vertex cover-B.*
4. *Weighted clique.*
5. *Weighted three dimensional matching.*
6. *Weighted Steiner trees in graphs.*
7. *Weighted not-all-equal 3SAT.*
8. *Weighted max-cut.*

9. *Traveling salesman.*

10. *Traveling salesman (metric).*

Proof.

We have already shown that the weighted s-t-mincut problem is hard to parallelize in our model. It is easy to reduce one of the problems on the list, say the weighted 3-SAT-B, to this problem, and then one can use standard reductions among weighted MAX-SNP-complete problems [40]. One has to only check that these reductions do not manipulate bits, i.e., they work fast in parallel in the PRAM model without bit operations. $\square$

6.5 Higher order parametric complexity

The notion of parametric complexity as defined in Chapter 3 can be readily generalized to allow any fixed number of parameters. Thus instead of a linear parametrization $\mathcal{P}$ in one rational parameter λ, as defined in Section 3.1, one can consider a parametrization with any constant c number of rational parameters $\lambda_1, \ldots, \lambda_c$. The corresponding optimum function graph $G(\mathcal{P})$ should now be a convex c-dimensional simplicial complex. We define the complexity $\rho(n)$ of $\mathcal{P}$ to be the number of (bounded) facets of this complex. The bit size of $\mathcal{P}$ is defined to be the maximum among the bit sizes of the coordinates of the vertices of $G(\mathcal{P})$ and of the coefficients of the linear functions in $\mathcal{P}$ specifying numeric parameters. We associate with $\mathcal{P}$ a $(c+1)$-dimensional homogeneous integral parametrization $\tilde{\mathcal{P}}$ very much as in Section 3.1. Now given a large enough *permissibility constant* a, we can let $\mathcal{I}$ be the set of inputs of the form (z_{c+2}, I) in the decision version of our general optimization problem, where:

1. The integer z_{c+2} denotes the threshold and its bit size is at most $a\beta(n)$.
2. I is of the form $\tilde{\mathcal{P}}(z_1, \ldots, z_{c+1})$, for some integers $z_1, \ldots, z_{c+1}$ of bit size at most $a\beta(n)$.

Then the natural generalization of Theorem 5.2.1 to such parametric input families $\mathcal{I}$ holds. The proof generalizes because Collins' decomposition, Milnor-Thom result and the other techniques used in Section 5.3 work in higher dimensions as well.

Chapter 7

The P vs. NC problem

Valiant[50] conjectured that over a field of characteristic other than two the permanent of an $n \times n$ matrix cannot be expressed as a specialization–projection in his terminology–of the determinant of a matrix of poly(n) size. An evidence in support of this conjecture is #P-completeness of the permanent. Analogously we shall formulate for the mincost-flow problem a conjecture, which if true would imply that $P \neq NC$. We shall also prove its special case (Theorem 7.0.2). Evidence in support of the conjecture will be provided by this special case, our lower bound for the mincost-flow problem (Theorem 1.3.1) and the general belief that the perfect-matching problem for unweighted graphs does not belong to NC^1–it is not even known to belong to NC. The mincost-flow problem is chosen only for concreteness (see the discussion after the conjecture). In contrast to the setting in Valiant [50], which is purely algebraic, the issue of bitlengths is of paramount importance here.

Let C denote the set of complex numbers.

Definition. Given a hypersurface $H \in C^l$ and a linear map $F : C^k \to C^l$, $k \leq l$, let $F^{-1}(H)$ be the pullback (inverse image) of H under F.

Here F can be nonhomogeneous, i.e., it can have constant additive terms in its defining forms.

We are mainly interested in the case when $l = m^2$ for some m and H is the determinant hypersurface $SL_m(C)$ in C^{m^2}. It is defined as follows. Let y_{ij}, $i, j \leq m$, denote the coordinates of C^{m^2}. Let Y denote the $m \times m$ matrix whose coefficients are the coordinates y_{ij}. Let $\det(Y)$ denote its determinant. Then $SL_m(C)$ is defined by the equation $\det(Y) = 1$; it is known to be a smooth hypersurface. In this case $F^{-1}(H)$ is defined by the equation $\det(F(x_1, \ldots, x_k)) = 1$, where $x_1, \ldots, x_k$ denote the coordinates of C^k; the image $F(x_1, \ldots, x_k)$ is an $m \times m$ matrix whose coefficients are linear forms in $x_1, \ldots, x_k$, possibly nonhomogeneous.

Now consider the decision version of the mincost flow problem. Here we are given an n-vertex network, whose each arc has integer cost and nonnegative integer capacity. Given an integer flow value v and an integer threshold w, the problem is to decide if there exists a flow with value v and cost at most w. The input can be identified with an integer tuple $I \in Z^k$, with $k = 2\binom{n}{2} + 2$, consisting of the costs, capacities, flow value and threshold, where Z denotes the set of integers. Let $L(n) \subseteq Z^k$ denote the mincost-flow language, i.e., the set of tuples for which the answer is yes. For any positive constant a, let $B(a,n)$ denote the set of points in Z^k whose each coordinate has bitsize at most an, i.e., the integer points within the box defined by $|x_i| \leq 2^{an}$ for all $i \leq k$.

Definition. Given a hypersurface $H \subseteq C^l$ and a linear function $F : C^k \to C^l$, $l \geq k$, we say that the pullback $F^{-1}(H)$ separates $L(n)$ within $B(a,n)$ if the points in $B(a,n)$ that lie on it are precisely the ones in $L(n) \cap B(a,n)$.

One can then make the following:

Conjecture: Consider the hypersurface $H = SL_m(C)$, where $m \leq 2^{n/d}$ and $d > 0$ is a large enough constant ($d = 1$ may suffice). For any positive constant a and any one-to-one linear function $F : C^k \to C^{m^2}$, $k = 2\binom{n}{2} + 2$, the pullback $F^{-1}(H)$ cannot separate $L(n)$ within $B(a,n)$, assuming that n tends to infinity,

The linear function F in the conjecture is over complex numbers. This is desirable because the methods of algebraic geometry work best over algebraically closed fields. We feel that if the conjecture is true when F ranges over some special integral functions–akin to Valiant's projections [50]–then it should also be true when F ranges over all complex linear functions. This will be borne out by Theorem 7.0.2, which holds over complex numbers.

We chose the mincost-flow problem in our formulation only for concreteness. One can choose any other problem that is not believed to be in NC, as long as it is suitable for an algebraic approach. If the problem is in P, it should have a strongly polynomial time algorithm. But it need not always be in P. For example, we can choose the permanent problem. This yields a conjecture, stronger than Valiant's [50], which if true would imply that the permanent does not belong to NC.

Proposition 7.0.1 If the conjecture is true for some positive $a < 1/(2d)$ then $P \neq NC$.

Proof. The total bitlength of any tuple I in $B(a,n)$ is at most $kan = \text{poly}(n)$. Suppose, to the contrary, that $P = NC$. Then the mincost-flow problem belongs to NC. Hence there is a small arithmetic formula τ in the bits of I of size $2^{\text{polylog}(n)}$ such that the formula evaluates to one on such I in $B(a,n)$ iff it belongs to $L(n) \cap B(a,n)$. Each coordinate of I has bitsize at most an. Using Lagrange interpolation, one can construct for its every bit a formula in the coordinate of size $O(2^{2an})$: this formula evaluates to one iff that particular bit of the coordinate is one. Feeding these formulae for bits into the preceding formula τ, we get a formula of size $O(2^{2an+\text{polylog}(n)})$ in the coordinates of I; this formula evaluates to one on $L(n) \cap B(a,n)$ but on no other point in

$B(a,n)$. Using Valiant's reduction [50], one can now construct a matrix of almost the same size whose entries are linear forms in the coordinates of I and whose determinant is equal to the formula. This gives a one-to-one linear map $F : C^k \to C^{m^2}$, where $m = O(2^{2an+\text{polylog}(n)})$, such that $F^{-1}[SL_m(C)]$ separates $L(n)$ within $B(a,n)$; moreover if $a < 1/(2d)$, this $m < 2^{n/d}$. This contradicts the conjecture. $\square$

Here is the intuition behind the conjecture.

Our lower bound for the mincost flow problem (Theorem 1.3.1) is undoubtedly far from optimal and holds only in the restricted PRAM model without bit operations. But for this problem our model is fairly realistic and the lower bound quite strong–it also holds when the bitsizes of the costs and capacities are restricted to be linear in n, the number of vertices. This suggests that in the presence of this restriction one cannot probably do much better than the following well known parallel method, which we shall call *explode-and-match* method.

Let n be the number of vertices in the flow network G and let l be the maximum bitlength of a cost or capacity. The method first reduces the problem, by exploding costs and capacities, to the maximum cardinality matching problem for an unweighted graph $\bar{G}$ of size $\bar{n} = O(2^l \text{poly}(n))$. For the latter problem one can use a fast parallel algorithm in [24] or [35]. When $l = O(n)$ this method is better than the parallel algorithm of Shiloach and Vishkin [44], or of Goldberg and Tarjan [13] as far as the parallel time is concerned; though the processor count is very bad in comparison.

We are interested in the case when $l = an$ and a is the constant in the conjecture. Suppose the perfect-matching problem (and hence the maximum cardinality matching problem) for unweighted graphs belongs to NC^i for some i–so far this is open (it is known to be in RNC^2 [35]). Apply an NC^i-algorithm to the exploded graph $\bar{G}$. This gives a formula in the bits of the encoding of $\bar{G}$ which evaluates to one iff $\bar{G}$ has a matching of required cardinality–the size of this formula is $2^{c\log^i \bar{n}}$ for some constant c. In our case $\bar{n} = O(2^l \text{poly}(n)) = O(2^{an}\text{poly}(n))$ with $l = an$. Hence the explode-and-match method gives us a formula in the bits of the edge-capacities and costs of G of size $2^{c(an)^i + O(\log^i n)}$ which evaluates to one iff the mincost flow problem for the original graph G is feasible. Now one can feed into this formula the $O(4^{an})$-size formulae based on Lagrange interpolation for the bits of the costs and capacities (as in the proof of Proposition 7.0.1). This yields a formula of size

$$O(2^{c(an)^i + O(\log^i n)} 4^{an}) = O(2^{b(an)^i}),$$

where $b = c+3$. Apply Valiant's reduction [50] to it. This gives a linear function $F : C^k \to C^{m^2}$, where $m = O(2^{b(an)^i})$. The pullback $F^{-1}[SL_m(C)]$ under this linear function separates $L(n)$ within $B(a,n)$.

If $i = 1$, i.e., if the perfect-matching problem belongs to NC^1, then one can choose a small enough so that the preceding $m < 2^{n/d}$. Then the conjecture, in fact, would be false for such small a. However, it is widely believed that the perfect-matching problem does not belong NC^1–it is not even known to belong to NC.

So let us assume that $i > 1$. It need not be an integer. In other words, we are even allowing a rather unlikely possibility that the perfect matching problem belongs to NC^i where $i > 1$ is a real number arbitrarily close to one. In this case the quantity $2^{b(an)^i}$ above grows far faster $2^{n/d}$, since i occurs in the exponent of the exponent. If we believe that, when the costs and capacities have $O(n)$ bitlengths, there is no parallel algorithm for the mincost flow problem that is much better than the explode-and-match method–this is what our lower bound suggests–then, in turn, we are led to believe that one can not come up with a separating linear function $F : C^k \to C^{m^2}$ with m that is much smaller than the $O(2^{b(an)^i})$ bound in the preceding scheme. But this bound grows much faster than $2^{n/d}$. That roughly speaking is the intuition behind the conjecture.

Remark: The belief that the perfect-matching problem does not belong to NC^1 is, in turn, tied to the belief that the determinant problem does not belong to NC^1 (it belongs to NC^2 [11]). This is because the parallel algorithm of Mulmuley, Vazirani, and Vazirani [35] reduces the matching problem to a calculation of determinants.

Another evidence in support of the conjecture is the following.

Theorem 7.0.2 The conjecture is true for every large enough constant a (for the mincost-flow problem).

Proof. This can be proved using Theorem 5.3.1.

Let a be a large enough positive constant to be chosen later. Assume that the conjecture is false. Then for some linear function $F : C^k \to C^{m^2}$, where $m \leq 2^{n/d}$, the inverse image $F^{-1}(SL_m(C))$ separates $L(n)$ within $B(a, n)$. Let $SL_m^+(C)$ and $SL_m^-(C)$ be hypersurfaces infinitisimally close to $SL_m(C)$ defined by the equations $\det(Y) = 1 + \epsilon$ and $\det(Y) = 1 - \epsilon$ respectively, where ϵ is infinitisimally small. Their inverse images–call them H_1 and H_2– partition $B(a, n)$ so that all points of $L(n)$ within it are contained in the same sign-invariant component which does not contain any other point of $B(a, n)$.

Now we shall confine our attention to only those integer points of $B(a, n)$ that lie within a three dimensional affine subspace chosen as follows. We know that the parametric complexity $\phi(n, \beta(n))$ of the mincost flow language is $2^{\Omega(n)}$ for $\beta(n) = O(n)$ (Theorem 3.1.3). Let $\mathcal{P}$ be a parametrization corresponding to the maximum value of the parametric complexity. Then the corresponding graph $G = G(\mathcal{P})$ (Section 3.1) has $\rho = 2^{\Omega(n)}$ breakpoints. Let $\tilde{\mathcal{P}}$ be the corresponding homogeneous integral parametrization (Section 5.2). Consider the integral parametrization of the input as defined before Theorem 5.2.2 by the map

$$E : (z_1, z_2, z_3) \to I(z_1, z_2, z_3) = (z_3, \tilde{\mathcal{P}}(z_1, z_2)),$$

We shall confine our attention to only those integer points in $B(a, n)$ that lie in the image of E, which is a three dimensional integer lattice.

Let B be the region defined as in Theorem 5.3.1 by letting $\bar{\beta} = \log \bar{\mu} = an/2$; then the coordinates of all integer points in B have bit sizes at most an, which means $B \subseteq B(a,n)$. Let S be the set of surfaces $E^{-1}(H_1)$ and $E^{-1}(H_2)$. Their total degree δ is at most $2 \cdot 2^{n/d}$. Since $B \subseteq B(a,n)$, the partition of B formed by S has the separation property as defined in Section 5.3. Choose a large enough so that $\bar{\beta} = \log \bar{\mu} = an/2$ is much larger than $\beta(n) = O(n)$, and d large enough so that $\log \rho = \Omega(n)$ is much larger than $\log \delta$, which is at most $n/d + \log 2$. Then it follows from Theorem 5.3.1 that this partition of B cannot have the separation property–a contradiction. $\square$

Our technique cannot be used to prove this theorem for the permanent problem. It also cannot be used for proving the conjecture for all positive constants a. This is because all algebro-geometric results we used depend only on the low degree of the surfaces that arise (actually the transversality results do not even depend on that). In particular, the same technique ends up proving Theorem 7.0.2 for any hypersurface $H \subseteq C^{m^2}$ of degree at most m (in place of $SL_m(C)$). This apparent strength of the technique is, in fact, its weakness, because:

Proposition 7.0.3 There exist hypersurfaces $H \subseteq C^{m^2}$, $m = 2^{n/d}$, of degree at most m, for which the conjecture is false for $a < 1/(2d)$.

Proof. Given an integer point $I \in B(a,n)$ let $\hat{I}$ denote its boolean enclo ding, which has length at most kan. Certainly there exists a (complicated) boolean formula of degree at most kan which evaluates to zero over $\hat{I}$ iff $I \in L(n)$. For any integer z of bitlength at most an, one can write down the Langrange interpolation formula of degree at most 2^{2an} for each of its bits. Feeding such formulae into the preceding boolean formula, one gets a polynomial g which evaluates to zero on $I \in B(a,n)$ iff $I \in L(n)$. Its degree is at most $kan2^{2an} < 2^{n/d}$, if a is sufficiently smaller than $1/(2d)$. Let F be the trivial embedding of C^k in C^{m^2}, and let H be defined by the polynomial equation $g = 0$. $\square$

This means if we wish to decrease the value of the constant a in Theorem 7.0.2 to something less than $1/(2d)$, as required in the conjecture, we must somehow use algebraic geometry of the *specific* hypersurface $SL_m(C)$ in the conjecture, or a similar one, in a deep way. What makes such hypersurfaces special is that they have a natural SL_m action. Such algebraic varieties have been investigated heavily in algebraic geometry, more specifically, in geometric invariant theory [36] for over a century. One source of great difficulty in our context is that the natural SL_m action on such varieties need not carry over to their linear pullbacks. So an important question here is: what is the algebraic geometry of linear pullbacks of such varieties? A good understanding of this, we feel, will greatly help in the P vs. NC problem.

Chapter 8

Conclusion

Our work shows that parametric complexity of a weighted optimization problem sets a good lower bound for its parallel complexity in the PRAM model without bit operations. In this way it may be possible to investigate parallel complexity of several other weighted optimization problems. This is especially important for problems that are neither known to be P-complete nor have fast parallel algorithms. A prime example is the minimum-weight perfect matching problem. If the weights are in unary, it has fast parallel algorithms [24, 35]. In general, its parallel complexity is open. We conjecture that the parametric complexity of this problem is $2^{\Omega(n^\epsilon)}$ for some small enough positive ϵ. It would then follow from our general lower bound that it cannot be solved in our model in $o(N^{\epsilon'})$ time using $2^{o(N^{\epsilon'})}$ processors, where N is the input bitlength and ϵ' is a small enough positive constant. Several other problems may be investigated in this way: e.g. matroid intersection problems [27], matroid parity problems [27], construction of blocking flows [13], and problems in computational geometry.

Can one prove a good lower bound in our model for computing the permanent of an integer matrix? Valiant's result [49] leads one to expect that this problem should be even harder than the problems considered in this paper. And yet, paradoxically, it seems much harder to prove a good lower bound for the permanent. Csanky [11] shows that the determinant can be computed in our model in $O(\log^2 n)$ time using polynomial number of processors. Can one show that it cannot be computed in our model in $O(\log n)$ time using polynomial number of processors?

Most importantly, the role of algebro-geometric methods in computational complexity deserves to be studied further. Such methods have been useful in proving several combinatorial results; e.g., see [28, 32]. Can the same happen for a separation question such as $P \neq NC$? Our work supports this possibility.

Acknowledgement: I wish to thank Allan Borodin, Steve Cook, Ajit Diwan, Charlie Rackoff, Donald Goldfarb, Laci Lovász, Katta Murty, Tushar Samant, L. V. Satyanarayana, and Milind Sohoni for illuminating discussions. I am also grateful to the referee for useful criticisms.

Bibliography

[1] N. Alon, R. Boppana, The monotone circuit complexity of boolean functions, Combinatorica, 7, (1987), 1–22.

[2] M. Ben-Or, Lower bounds for algebraic computation trees, 15th ACM Symposium on Theory of Computing (1983), 80–86.

[3] M. Ben-or, E. Faig, D. Kozen, P. Tiwari, A fast parallel algorithm for determining all roots of a polynomial with real roots, SIAM J. of computing, 17(1988), 1081–1092.

[4] D. Bini and V. Pan, Polynomial and matrix computations, Birkhäuser, Boston, 1994.

[5] R. Boppana, M. Sipser, The complexity of finite functions, Handbook of Theoretical Computer Science, vol. A, Edited by J. van Leeuwen, North Holland, Amsterdam, 1990, 757–804.

[6] W. Bruns, U. Vetter, Determinantal rings, Lecture notes in mathematics, Springer, 1988.

[7] P. Carstensen, Complexity of some parametric integer and network programming problems, Mathematical Programming 26 (1983), 64–75.

[8] P. Carstensen, The complexity of some problems in parametric linear and combinatorial programming, Doctoral Dissertation, The University of Michigan, Ann Arbor, 1982.

[9] N. Christofides, Worst-case analysis of a new heuristic for the traveling salesman problem, Technical report, Graduate school of industrial administration, Carnegie-Mellon university, 1976.

[10] G. Collins, Quantifier elimination for real closed fields by cylindrical algebraic decomposition, Proceedings of the second GI Conf. on Automata Theory and Formal Languages, Lecture Notes in Computer Science 33 (Springer, Berlin, 1975) 134–183.

[11] L. Csanky, Fast parallel matrix inversion algorithms, SIAM J. computing, 5, 1976, 618–623.

[12] M. Furst, J. Saxe, M. Sipser, Parity, circuits, and the polynomial time hierarchy, Math. Systems Theory 17 (1984), 13–27.

[13] A. Goldberg, R. Tarjan, A new approach to the maximum-flow problem, JACM, 35: 921–940, 1988.

[14] D. Goldfarb, Worst-case complexity of the shadow vertex simplex algorithm, report, Department of Industrial Engineering and Operations Research, Columbia University, 1983.

[15] L. Goldschlager, R. Shaw, J. Staples, The maximum flow problem is logspace complete for P, Theoretical Computer Science, 21:105-111, 1982.

[16] M. Golubitsky, V. Guillemin, Stable mappings and their singularities, Springer Verlag, 1973.

[17] M. Grotschel, L. Lovász, A. Schriver, Geometric algorithms and combinatorial optimization, Springer-Verlag, 1988.

[18] D. Gusfield, Sensitivity analysis for combinatorial otpimization, Memorandum number UCB/ERL M80/22, Electronics Research Laboratory, Berkeley, 1980.

[19] J. Hastad, Almost optimal lower bounds for small depth circuits, Advances in Computing Research 5: Randomness and Computation, S. Micali ed., JAI Press, Greenwich, CT, 1989, 143–170.

[20] O. Ibarra, C. Kim, Approximation algorithms for certain scheduling problems, Math. of Operations Research 3 (1978), 197–204.

[21] O. Ibarra, S. Moran, L. Rosier, A note on parallel complexity of computing the rank of order n matrices, Info. proc. letters, vol. 11, 1980, pp. 162.

[22] N. Karmarkar, A new polynomial-time algorithm for linear programming, Combinatorica 4(1984), 373–395.

[23] R. Karp, V. Ramachandran, A survey of parallel algorithms for shared-memory machines, Handbook of theoretical computer science, 870–941, Edited by J. van Leeuwen, Elsevier Science Publishers B.V., 1990.

[24] R. Karp, E. Upfal, A. Wigderson, Constructing a perfect matching is in random NC, Combinatorica, 6:35–48, 1986.

[25] L. Khachian, Polynomial algorithms in linear programming [English translation] U.S.S.R. Computational Mathematics and Mathematical Physics 20(1980), 53–72.

[26] V. Klee, G. Minty, How good is the simplex algorithm?, Inequalities, III (O. Shisha, ed.), Academic Press, New York, 1972, 159–175.

[27] L. Lovász, M. Plummer, Matching theory, Akadémiai Kiadó, Budapest, 1986.

[28] A. Lubotzky, R. Phillips, P. Sarnak, Ramanujan conjecture, Combinatorica 8(1988), 261–277.

[29] D. Karger, Global min-cuts in RNC and other ramifications of a simple mincut algorithm, In proceedings of the 4th ACM-SIAM Symposium on Discrete Algorithms, 21–30, 1993.

[30] D. Karger, R. Motwani. Derandomization through approximation: An NC algorithm for minimum cuts. 25th ACM Symposium on Theory Of Computing.

[31] T. Leighton, Introduction to parallel algorithms and architectures: Arrays, Trees, Hypercubes, Morgan Kauffman, 1992.

[32] G. Margulis, Arithmetic groups and graphs without short cycles, sixth Inter. Symp. on Infor. Theory, Tashkent 1984, Abstracts, Vol. I, 123–125.

[33] J. Milnor, On the betti numbers of real varieties, Proceedings of American Mathematical Society 15, 1964, 275–280.

[34] K. Mulmuley, Computational Geometry: an introduction through randomized algorithms, Prentice Hall, 1993.

[35] K. Mulmuley, U. Vazirani, V. Vazirani, Matching is as easy as matrix inversion, Combinatorica, 7(1):105–113, 1987.

[36] D. Mumford, F. Fogarty, Geometric invariant theory, second ed., Springer, New York, Berlin, Heidelberg, 1982.

[37] K. Murty, Computational complexity of parametric linear programming, Mathematical programming 19(1980), 213–219.

[38] C. Neff, Specified precision polynomial root isolation is in NC, Proceedings of the 31st annual IEEE Symp. on Foundations Of Computer Science (1990), 152–162.

[39] V. Pan and J. Reif, Efficient parallel solution of linear systems, Proceedings of the 17th annual ACM Symposium on Theory of Computing, 1985, 143-152.

[40] C. Papadimitriou and M. Yannakakis, Optimization, approximation, and complexity classes, JCSS 43, 425–440, 1991.

[41] A. Razborov, Lower bounds on the monotone complexity of some boolean functions, Dokl. Ak. Nauk. (1985), 798–801.

[42] J. Reif (Editor), Synthesis of parallel algorithms, Morgan Kaufmann, 1993.

[43] A. Razborov, S. Rudich, Natural proofs, Proceedings of the 26th ACM Symposium on Theory Of Computing, 1994.

[44] Y. Shiloach, U. Vishkin, An $O(n^2 \log n)$ parallel max-flow algorithm, J. Algorithms 3 (1982), 128–146.

[45] J. Steele and A. Yao, Lower bounds for algebraic decision trees, J. Algorithms 3, (1982), 1–8.

[46] E. Tardos, A strongly polynomial algorithm to solve combinatorial linear programs, Operations Research 34 (1986), 250–256.

[47] R. Tarjan, Algorithms for maximum network flow, Mathematical Programming Study 26 (1986) 1-11.

[48] R. Thom, Sur l'homologie des varietes algebriques reelles, Differential and combinatorial topology, edited by S. Cairns, Princeton University Press, 1965.

[49] L. Valiant, The complexity of computing the permanent, Theoretical Computer Science 8 (1979) 189–201.

[50] L. Valiant, Completeness classes in algebra, Proceedings of the ACM Symposium on Theory Of Computing (1979).

[51] A. Yao, Separating the polynomial-time hierarchy by oracles, Proceedings of the 26th annual IEEE Symp. on Foundations Of Computer Science (1985), 1–10.

[52] A. Yao, Lower bounds for algebraic computation trees with integer inputs, Proceedings of the 30th annual IEEE Symp. on Foundations Of Computer Science (1989), 1–10.

[53] A. Yao, Probabilistic computation: Toward a unified measure of complexity, Proceedings of the 18th annual IEEE Symp. on Foundations Of Computer Science (1977), 222–227.

[54] L. Valiant, S. Skyum, S. Berkowitz, C. Rackoff. Fast Parallel computation of polynomials using few processors, SIAM Journal on Computing, 12(4): 641-644, 1983.

[55] A. Wigderson, The fusion method for lower bounds in circuit complexity, Bolyai society mathematical studies, Combinatorics, Keszthely (Hungary), 453–467, 1993.

[56] N. Zadeh, A bad network problem for the simplex method and other minimum cost flow algorithms, Math. Programming, 5 (1973), 255–266.