CSC 2429 – Approaches to the P versus NP Question

Lecture #7 (part 2): 5 March 2014

Lecturer: Joshua A. Grochow

Scribe Notes by: Joshua A. Grochow

The main result we'll cover in this lecture and the next is:

Theorem 1 (Mulmuley [Mul99])**.** *In the parallel RAM model without bit operations, "*$\mathsf{P} \neq \mathsf{NC}$*."*

This is really what I consider the masthead result of [Mul99], but in the same paper Mulmuley also showed that $\mathsf{P} \neq \mathsf{RNC}$ in the PRAM model without bit operations, and that the NC^i and RNC^i hierarchies are strict in this model, and furthermore that all of these results holds not just for the decision problems but even for the corresponding additive approximation problems. As the proof for these other results follows the same outline as the proof for Theorem 1, we focus only on that result.

The parallel RAM without bit operations is not the same as the purely algebraic model considered in the first half of this lecture (and in most of algebraic complexity), but is rather somewhere between a purely algebraic model and the usual boolean models of computation. Although this may sound somewhat exotic, it's not; we'll say this more slowly in the next section, but for now: it's a nonuniform algebraic parallel RAM over $\mathbb{Z}$—so addition and multiplication of integers can be done at unit cost, regardless of their bit-size—in which the program and the number of processors may depend not only on the number of inputs, but also on their total bitsize. The model is useful, natural, and strong enough to express many algorithms in computational geometry.

To our knowledge this model has not been considered *per se* beyond this lower bound, so for concreteness and completeness we begin by explaining the model in full detail. We then discuss the strength and significance of this result, especially in light of what we now know fifteen years later. We then give a more-or-less complete overview of the proof of Theorem 1. Finally, we discuss how this result naturally inspired the current incarnation of the Geometric Complexity Theory (GCT) Program.

1 The model of computation

For those familiar with the algebraic RAM, or equivalently with the Blum–Shub–Smale model, the PRAM model without bit operations is *almost* exactly what you would expect of the phrase "non-uniform algebraic PRAM." The one difference is that the number of processors and the running time may be a function not only of the number of integer input values, but may also

depend on the total bitlength of the input. It is in this sense that the model is somewhere between the usual algebraic and boolean models.

For those familiar with the algebraic computation tree model (as in [BCS97, Section 4.4]), the PRAM model without bit operations is *almost* exactly what you'd expect of the phrase "parallel algebraic computation tree," except that the computation tree may depend not only on the number n of input integers, but also on the total bitlength of the input.

In the rest of this section we give the details of this model for those not familiar with the above two models, or for those readers just wanting to make sure that they got the right idea from the above two paragraphs.

As in many problems on weighted graphs, inputs in this model have two parts: one part consisting of integer parameters, and another part consisting of boolean parameters, with boolean values encoded as the integers $0, 1 \in \mathbb{Z}$. For a weighted graph problem, the structure of the graph would be encoded in the boolean parameters and the weights would be the integer parameters.

An algebraic random access machine (RAM) program over the integers is a machine consisting of one processor and infinitely many memory locations. Each memory location can store an arbitrary integer, and is marked either as an input, output, or work location. Each instruction in the program has a unique label identifying it, and is one of the following forms:

- $w = u \circ v$ where $\circ$ is one of $+, -, \times$, w is a memory location, and u, v are either memory locations or constants from $\mathbb{Z}$. For this lower bound, each arithmetic operation is assumed to have *unit cost*, regardless of the bit-size of the operands (this only makes the lower bound stronger);
- Go to instruction ℓ, where ℓ is an instruction label;
- If $u : 0$ then go to instruction ℓ otherwise go to instruction ℓ', where $:$ is one of $<, \leq, =$, u is a memory location, and ℓ, ℓ' are instruction labels. The value stored in memory location is compared against zero, and the program branches accordingly;
- Copy u to v, where u and v are memory locations;
- Indirect (pointer) reference, that is, take the value in memory location v, treat it as a memory location, then take the value in *that* location and store it in u. We add the technical, but useful, condition that indirect reference is only allowed when the value in memory location v does not depend on any of the integer inputs, though it may depend on the boolean ones. Here "depend" means syntactically: there should be no chain of instructions linking operands to value that starts at some integer input and ends at v;
- Stop.

The value of the machine on an input consisting of n integers is the contents of the output locations when the program evaluated at that input (in the usual manner of non-uniform computations). Loops are not needed in the instruction set—one can think of all loops as being unrolled.

A nonuniform RAM without bit operations is then a sequence $\mathcal{A}$ of algebraic RAM programs $A_{n,N}$ for $n, N \in \mathbb{N}$. The value of $\mathcal{A}$ on an input consisting of n integers of total bitlength at most N is then the output of $A_{n,N}$ evaluated on that input.

An algebraic parallel RAM (PRAM) program consists of many processors and infinitely many memory locations. Each memory location is designated either to belong to a particular processor,

or to be public, and this designation cannot be changed during the course of execution; the input and output locations are all public. Memory locations designated as belonging to processor π can only be written to or read from by π. The processors execute their instructions in lock-step, simultaneously, and a valid program is one in which any time two processors write to the same (necessarily public) memory location, they write the same value. Issues of contention and locking do not arise here; this is thus a very unrealistic model from the point of view of parallel algorithms, but proving a lower bound in this general model proves the lower bound for any of the more realistic models in which parallel algorithms are developed.

A nonuniform PRAM without bit operations $\mathcal{A}$ is then a family $\{A_{n,N} : n, N \in \mathbb{N}\}$ of algebraic PRAM programs, one for each number of inputs n and each bitlength N. The value of $\mathcal{A}$ on an input consisting of n integers of total bitlength at most N is then the value of $A_{n,N}$ evaluated at that input.

In the exposition of the proof, following the original paper [Mul99], we will also consider *linear* PRAMs without bit operations, which are the same as above, except the only multiplications allowed are those in which one of the factors is a constant that's independent of the input.

2 Significance

Despite the fact that the program and the number of processors is allowed to vary with the number of inputs *and* their total bit-length, the lower bound is stated in terms of the number of inputs only. However, unlike essentially all algebraic lower bounds, this lower bound also holds when the total bit-size of the inputs is small compared to the number of inputs, for example, $N \leq O(n)$ for the maximum flow problem on n-node graphs. This makes the lower bound significantly stronger, and is technical evidence that this lower bound is truly more than a purely algebraic lower bound.

Still today, nearly 20 years after the first publication [Mul94] that included the essential ideas of this result, this is the only known lower bound in complexity that has *all* of the following desirable properties:

- It is a formal implication and algebraic analog of a major open complexity class separation question, in this case P versus NC;
- It applies to a realistic model[1] of computation that includes a wide class of algorithms for a wide class of problems, including cuts and flows [Kar93, KM97, SV82, GT88], other

[1] Here we anticipate and rebut some possible counterexamples. Monotone or bounded depth circuits, for which super-polynomial lower bounds are known [GK98, Raz85, GKKS12], can hardly be argued to be realistic models of computation. Bounded-depth circuits, even depth-3 circuits, *are* a realistic model of bilinear computation, e. g., for matrix multiplication, but the strongest possible lower bounds on bilinear functions are quadratic in the input size (for matrix multiplication the input size is $2n^2$, so even an $\Omega(n^3)$ lower bound, which we know to be false [Str69], would have only been $\Omega(N^{3/2})$ where N is the input size). Time-space trade-offs for SAT (e. g., [For00, FLvMV05, DvMW11, Wil08, Wil06, BW12]) work with general models of computation, but the current lower bounds there have not yet reached $\Omega(n^2)$. There are exponential lower bounds on resolution in proof complexity, which realistically capture the famous and widely-used DPLL algorithm, but this is really just a model of a particular kind of backtracking search rather than a general purpose computational model. The cell probe model is a realistic and important model of data structures (see [Mil99] for a survey), but lower bounds in the cell probe model are in terms of query complexity, so a linear lower bound in that model is optimal. There have also been interesting lower bounds on models of greedy, backtracking, and dynamic-programming algorithms, the most general of which we are aware of is [ABBO+11], but by their nature these models are designed to capture a realistic but limited class of algorithms.

combinatorial optimization problems such as shortest paths and spanning trees [Lei92], linear programming [Kar84] and many other problems in computational geometry [Rei93], solving linear systems exactly over $\mathbb{Q}$ [Csa76] or numerically over $\mathbb{R}$ [PR85] and finding their rank [IMR80], and approximating complex roots of polynomials [Nef90, BOFKT88].

- It applies to a realistic model of computation that is strong enough to compute the determinant efficiently. Although we already mentioned this in the preceding long list, this particular problem is significant enough to warrant its own point. Proving lower bounds in models that are strong enough to compute the determinant efficiently is a long-standing "barrier" to lower bounds in complexity. Indeed, even the recent excitement over new lower bounds for the permanent [GKKS12] also give essentially the same lower bounds for the determinant.

As this is the only known result of this strength, the fact that it crucially uses algebraic geometry gives yet another argument for the utility and potential necessity of algebraic geometry for complexity lower bounds.

Although the key geometric results used in the proof of Theorem 1 have been used previously in complexity, Theorem 1 is significantly stronger than the previous lower bounds in the algebraic computation tree model (e. g., [BO83, SY82, Yao91]). The key geometric results used in the proof are the Milnor–Thom estimate on the number of connected components defined by some polynomials over $\mathbb{R}$ [Mil64, Tho65]—the same key geometric result used for classic lower bounds in the algebraic comparison tree model [BO83, SY82, Yao91]—and cylindrical decomposition in real semi-algebraic geometry [Col75]—the key geometric result used to give the first quantifier elimination algorithm over $\mathbb{R}$ of optimal (in this case, doubly-exponential) complexity.

Theorem 1 is a lower bound on universal problems such as linear programming, and represents a fundamental separation close to $\mathsf{P} \neq \mathsf{NC}$, whereas the lower bounds from the 1980s (such as [BO83, SY82, Yao91]) in the algebraic computation tree model were essentially lower bounds on the comparatively easy problem of sorting. Furthermore, those lower bounds were purely algebraic, in the sense that they required their inputs to have large bit-size relative to the number of input parameters, whereas Theorem 1 works even when the bit-size is small, say $O(n)$, compared to the number of parameters n.

The reliance of Theorem 1 on these results from *real* geometry separates it from the modern GCT program, which attempts to use more purely algebraic techniques that have a better chance of extending to the case of positive characteristic, and hence to the boolean case. Nonetheless, because the PRAM model without bit operations is somewhere between an algebraic and boolean model, this result is much closer to boolean $\mathsf{P} \neq \mathsf{NC}$ than an algebraic result of the form $\mathsf{VNC}^1 \neq \mathsf{VP}$. However, the techniques of this result are still quite far from, for example, separating the permanent from determinant; the underlying reason for this distance is what eventually led to the GCT program, as we'll discuss in the next lecture.

3 Outline of the proof

There are several key ideas in the proof of Theorem 1. These ideas are exhibited by various intermediate results along the way, and we follow the outline of the original paper [Mul99].

The basic technique is the indistinguishability technique, as exemplified but more general than in results on algebraic computation trees [BO83, SY82, Yao91] and many results in distributed computation; of course, the heart of Theorem 1 is those ideas that allow us to apply

the indistinguishability technique.

The main ideas of the proof are then:

1. To reduce dimension—even to a single input from $\mathbb{Q}$!—without losing complexity by applying ideas from *parametric complexity* (as distinct from parametr*ized* complexity);

2. To apply ideas on arrangements of hyperplanes in $d = O(1)$ dimensions, by considering a rational input as a pair of integer inputs. This transforms the graph of the optimum-value function from a subset of $\mathbb{Q}^d \times \mathbb{Q}$ to $(\mathbb{Z}^2)^d \times \mathbb{Z} \subseteq \mathbb{Q}^{2d+1}$. In particular, even when there is only one rational parameter, we have $d = 1$ but may still apply ideas from polyhedral geometry in $\mathbb{Q}^3$;

3. To apply the Milnor–Thom estimate on the number of connected components defined by real polynomials [Mil64, Tho65], the Collins cylindrical decomposition [Col75], and arbitrary small perturbations (to achieve "general position") [GG73] to extend the preceding ideas from arrangements of hyperplanes to arrangements of hypersurfaces defined by higher-degree polynomials.

We cover each of these ideas, beginning with the basic indistinguishability technique, in each of the next subsections.

3.1 The indistinguishability technique

The basic idea of the indistinguishability technique is as follows: if the output of a program is from a discrete set (say, $\{0, 1\}$, or a permutation, rather than the evaluation of a function like $x^2 + 7$), then the output is determined by the various choices made by the branch instructions. If two inputs induce the same answers to every branch question, then the corresponding outputs must be the same. Two such inputs are called *indistinguishable*; indistinguishability is readily seen to be an equivalence relation that partitions the space of inputs. In the simplest form of the indistinguishability argument, one shows that a small (short, etc.) program has few indistinguishability classes; as the number of indistinguishability classes must at least be the number of outputs, if many distinct outputs are possible then the program cannot be too short. More sophisticated applications of the indistinguishability argument may take advantage of the structure or geometry of the indistinguishability classes, as we do here.

To give an example of the indistinguishability technique as a warm-up to Theorem 1, we briefly review the fact that extracting bits in the PRAM model without bit operations is indeed difficult. Of course, it's not difficult to extract the high-order bits: one can find the highest-order bit by repeatedly squaring 2 and comparing 2^k to a number. One can then subtract the appropriate 2^k and recurse. However, to extract, say, the $\sqrt{n}$-th high-order bit of an n-bit number would seem to require *sequential* time $\sqrt{n}$ by this method, which would be fine in the algebraic RAM model without bit operations, but is quite large in terms of parallel complexity. The following result shows that one essentially can't do better than this naive approach to extracting bits.

Proposition 2 (Proposition 2.1 of Mulmuley mulmuleyPRAM)**.** *The lowest-order bit of an n-bit integer can't be computed in the PRAM model without bit operations in time $\sqrt{n}/c$ using $2^{\sqrt{n}/c}$ processors, for sufficiently large c.*

In fact, this proof and all of the results discussed in this section can be extended to the PRAM model with limited bit operations [Mul99], in which the operations of parity and $\lfloor u/2 \rfloor$

are allowed. These operations make it easy to extract the low-order bits of a number easily. However, one can show that even in this stronger model extracting the *middle* bits is still hard. If this were not the case, the model would become equivalent to the usual boolean NC. Thus, the inability to access middle bits—which at first blush might seem like a somewhat minor restriction—is in fact quite a major restriction, as it can be viewed as the current obstacle to separating P from NC.

Proof idea. Each branch instruction is essentially of the form $f(x) : 0$ (where : is one of $>, \geq, =$) for some polynomial f of the input—note that the input here is a single integer x. The maximum degree of any polynomial computed in time t is 2^t, and such polynomial has at most 2^t real roots. The total number of branch instructions executed at time t or earlier is at most the number of processors times t. Thus, in time $t(n) = \sqrt{n}/c$ and with $p(n) = 2^{t(n)} = 2^{\sqrt{n}/c}$ processors, the total number of roots of *all* polynomials considered at all branch instructions is thus at most $p(n)t(n)2^{t(n)} \leq \sqrt{n}2^{t(n)^2}/c \leq \sqrt{n}2^{n/c^2}/c$. All of the numbers in between two consecutive roots must be indistinguishable, so there are at most $\sqrt{n}2^{n/c^2}/c$ indistinguishability classes, each of which is a union of connected sub-intervals of $[0, 2^n - 1]$ (the n-bit nonnegative numbers). For c sufficiently large, $\sqrt{n}2^{n/c^2}/c < 2^n/2$, so there must be some indistinguishability class containing a connected interval of length at least 2. Any such interval contains two integer inputs that differ by one, and hence whose lowest-order bits differ.

There are a couple issues with the above reasoning, but it's essentially right.

First, what happens if an integer point lands exactly on one of the roots? We neglected this in our counting of indistinguishability classes, but one can still upper bound the number of indistinguishability classes by $6^{n/c^2}$, and the above argument will then go through. Alternatively, one could alter the constant terms of the branch polynomials by a constant to ensure that equality never occurs, change $=$ to $>$, and possibly have to double the number of branches in order to check both $> \varepsilon$ and $< -\varepsilon$. This constant ε may be chosen small enough that, on integer inputs, none of the outputs change. We will see this same technique in the proof of Theorem 1 in Section **??**.

A subtler problem is that there isn't actually a fixed polynomial at each branch instruction. Rather, the value computed at a branch at time t can depend on the outcomes of the branches at times preceding t. To remedy this, one uses an inductive argument: all of the values used at the *first* branches encountered are indeed fixed polynomials of the input, and the results of these branches partition the inputs into equivalence classes defined as certain unions of intervals between the roots. If we pretend that the input is in one of these equivalence classes, then all of the values computed at the *second* branches encountered become fixed polynomials. These branches may subdivide the equivalence classes further, again by the roots of the corresponding polynomials. And so on. The resulting counting argument produces the $6^{n/c^2}$ bound mentioned above, and the above proof then goes through with all details taken care of. □

In the next lecture, we'll see that one of the key ingredients in the analog of Theorem 1 for *linear* PRAMs without bit operations (no multiplication allowed except by constants) is the extension of the idea of the preceding proof from 1 to (any) $d \geq 3$ dimensions. However, that lower bound will be on much more complicated combinatorial optimization problems, so we won't be able to use an argument like "If an indistinguishability class contains an interval of length 2..." Instead, we'll use further structure of the problem, in the form of parametric complexity. It turns out that the parametric complexity of a problem is a good measure of its (parallel) computational complexity.

References

[ABBO+11] Michael Alekhnovich, Allan Borodin, Joshua Buresh-Oppenheim, Russell Impagliazzo, Avner Magen, and Toniann Pitassi, *Toward a model for backtracking and dynamic programming*, Comput. Complexity **20** (2011), no. 4, 679–740.

[BCS97] Peter Bürgisser, Michael Clausen, and M. Amin Shokrollahi, *Algebraic complexity theory*, Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], vol. 315, Springer-Verlag, Berlin, 1997, With the collaboration of Thomas Lickteig.

[BO83] Michael Ben-Or, *Lower bounds for algebraic computation trees*, STOC '83: 15th Annual ACM Symposium on Theory of Computing, ACM, 1983, pp. 80–86.

[BOFKT88] Michael Ben-Or, Ephraim Feig, Dexter Kozen, and Prasoon Tiwari, *A fast parallel algorithm for determining all roots of a polynomial with real roots*, SIAM J. Comput **17** (1988), no. 6, 1081–1092.

[BW12] Samuel Buss and Ryan Williams, *Limits on alternation-trading proofs for time-space lower bounds*, CCC '12: 27th IEEE Conference on Computational Complexity, IEEE Computer Society, 2012, pp. 181–191.

[Col75] George E. Collins, *Quantifier elimination for real closed fields by cylindrical algebraic decomposition*, Automata theory and formal languages (Second GI Conf., Kaiserslautern, 1975), Lecture Notes in Computer Science, vol. 33, Springer, Berlin, 1975, pp. 134–183.

[Csa76] L. Csanky, *Fast parallel matrix inversion algorithms*, SIAM J. Comput **5** (1976), no. 4, 618–623.

[DvMW11] Scott Diehl, Dieter van Melkebeek, and Ryan Williams, *An improved time-space lower bound for tautologies*, J. Comb. Optim. **22** (2011), no. 3, 325–338.

[FLvMV05] Lance Fortnow, Richard Lipton, Dieter van Melkebeek, and Anastasios Viglas, *Time-space lower bounds for satisfiability*, J. Assoc. Comput. Mach. **52** (2005), no. 6, 835–865.

[For00] Lance Fortnow, *Time-space tradeoffs for satisfiability*, J. Comput. System Sci. **60** (2000), no. 2, part 2, 337–353, CCC '97: 12th IEEE Conference on Computational Complexity.

[GG73] M. Golubitsky and V. Guillemin, *Stable mappings and their singularities*, Springer-Verlag, New York, 1973, Graduate Texts in Mathematics, Vol. 14.

[GK98] Dima Grigoriev and Marek Karpinski, *An exponential lower bound for depth 3 arithmetic circuits*, STOC '98: 30th Annual ACM Symposium on Theory of Computing, ACM, 1998, pp. 577–582.

[GKKS12] Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi, *Approaching the chasm at depth four*, Tech. Report TR12-098, Electronic Colloquium on Computational Complexity, 2012.

[GT88] Andrew V. Goldberg and Robert E. Tarjan, *A new approach to the maximum-flow problem*, J. Assoc. Comput. Mach. **35** (1988), no. 4, 921–940.

[IMR80] Oscar H. Ibarra, Shlomo Moran, and Louis E. Rosier, *A note on the parallel complexity of computing the rank of order n matrices*, Inform. Process. Lett. **11** (1980), no. 4-5, 162.

[Kar84] N. Karmarkar, *A new polynomial-time algorithm for linear programming*, Combinatorica **4** (1984), no. 4, 373–395.

[Kar93] David R. Karger, *Global min-cuts in* RNC*, and other ramifications of a simple min-cut algorithm*, SODA '93: 4th ACM–SIAM Symposium on Discrete Algorithms (New York), ACM, 1993, pp. 21–30.

[KM97] David R. Karger and Rajeev Motwani, *An* NC *algorithm for minimum cuts*, SIAM J. Comput **26** (1997), no. 1, 255–272.

[Lei92] F. Thomson Leighton, *Introduction to parallel algorithms and architectures*, Morgan Kaufmann, San Mateo, CA, 1992, Arrays, trees, hypercubes.

[Mil64] John Milnor, *On the Betti numbers of real varieties*, Proc. Amer. Math. Soc. **15** (1964), 275–280.

[Mil99] Peter Bro Miltersen, *Cell probe complexity—a survey*, Invited paper/talk at *Advances in Data Structures*, a pre-conferences workshop at FSTTCS 1999, available at http://www.daimi.au.dk/~bromille/Papers/survey3.ps, 1999.

[Mul94] Ketan Mulmuley, *Lower bounds for parallel linear programming and other problems*, STOC '94: 26th Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC '94, ACM, 1994, pp. 603–614.

[Mul99] Ketan Mulmuley, *Lower bounds in a parallel model without bit operations*, SIAM J. Comput **28** (1999), no. 4, 1460–1509 (electronic).

[Nef90] C. Andrew Neff, *Specified precision polynomial root isolation is in* NC, FOCS '90: 31st Annual IEEE Symposium on Foundations of Computer Science, IEEE Comput. Soc. Press, Los Alamitos, CA, 1990, pp. 152–162.

[PR85] V. Pan and John Reif, *Efficient parallel solution of linear systems*, STOC '85: 17th Annual ACM Symposium on Theory of Computing (New York, NY, USA), ACM, 1985, pp. 143–152.

[Raz85] Alexander Razborov, *Lower bounds of monotone complexity of the logical permanent function*, Mat. Zametki **37** (1985), no. 6, 887–900, 942, English translation: Math. Notes 37 (1985), no. 5–6, 485-493.

[Rei93] John H. Reif (ed.), *Synthesis of parallel algorithms*, Morgan Kaufmann, San Mateo, CA, 1993.

[Str69] Volker Strassen, *Gaussian elimination is not optimal*, Numer. Math. **13** (1969), 354–356.

[SV82] Yossi Shiloach and Uzi Vishkin, *An $O(n^2 \log n)$ parallel MAX-FLOW algorithm*, J. Algorithms **3** (1982), no. 2, 128–146.

[SY82] J. Michael Steele and Andrew C. Yao, *Lower bounds for algebraic decision trees*, J. Algorithms **3** (1982), no. 1, 1–8.

[Tho65] René Thom, *Sur l'homologie des variétés algébriques réelles*, Differential and Combinatorial Topology (A Symposium in Honor of Marston Morse), Princeton Univ. Press, Princeton, N.J., 1965, pp. 255–265.

[Wil06] Ryan Williams, *Inductive time-space lower bounds for SAT and related problems*, Comput. Complexity **15** (2006), no. 4, 433–470.

[Wil08] Ryan Williams, *Time-space tradeoffs for counting* NP *solutions modulo integers*, Comput. Complexity **17** (2008), no. 2, 179–219.

[Yao91] Andrew Chi-Chih Yao, *Lower bounds for algebraic computation trees with integer inputs*, SIAM J. Comput **20** (1991), no. 4, 655–668.