Fully Parallelized Multi Prover Protocols for NEXP-time

(Extended Abstract)

Dror Lapidot Adi Shamir
Applied Mathematics Dept.
The Weizmann Institute of Science
Rehovot 76100, Israel
email: drorl@wisdom.bitnet, shamir@wisdom.bitnet

Abstract

A major open problem in the theory of multi-prover protocols is to characterize the languages which can be accepted by fully parallelized protocols which achieve an exponentially low probability of cheating in a single round. The problem was motivated by the surprising observation that the probability of cheating in n parallel executions of a multi-prover protocol can be exponentially higher than the probability of cheating in n sequential executions of the same protocol. In this paper we solve this problem by proving that any language in $NEXP$-time has a fully parallelized multi prover protocol. By combining this result with a fully parallelized version of the [BGKW] protocol, we can obtain a one-round perfect zero-knowledge protocol (under no cryptographic assumptions) for every $NEXP$-time language.

1 Introduction

The model of multi prover interactive proofs was introduced by Ben-Or, Goldwasser, Kilian and Wigderson [BGKW]. In these proof systems several unbounded provers (who are not allowed to communicate among themselves, and who can not see the messages exchanged between the verifier and the other provers) have to convince a polynomial time verifier that a certain common input X belongs to some language L. If $X \notin L$, then the probability that the verifier accepts should be exponentially small. Babai, Fortnow and Lund [BFL] characterized the class $MIP(poly)$ of all languages which have sequential multi prover protocols with a polynomial number of rounds by proving that this class is exactly NEXP-time.

One of the famous open problems in this area is whether such sequential interactive proofs can be fully parallelized, namely, whether they can be transformed into protocols which require just a single round, and still have an exponentially small probability of cheating. Many works were devoted to this problem achieving many partial results.

One of the preliminary parallelization problems was whether languages in NP have parallel multi prover *zero-knowledge* protocols. Ben-Or, Goldwasser, Kilian and wigderson, who proved in [BGKW] that NP has sequential two prover zero-knowledge protocols, without any cryptographic assumptions, remarked that the parallel execution of their protocol is also a zero-knowledge proof system with a single round under a weak definition which requires only a constant probability of cheating. Fortnow, Rompel and Sipser [FRS] claimed the same result under the stronger definition which requires a negligible probability of cheating, but their proof of soundness was later shown to be faulty by Fortnow [Fo], and no alternative parallel protocol was known to be sound in this strong sense. Moreover, there are some examples of protocols (see [Fo], [LS] and [Fe]) for which the probability of cheating in their parallel version is known to be exponentially higher than in their sequential version.

A more general question is whether arbitrary languages in $MIP(poly)$ can be recognized by fully parallelized protocols. Kilian [Ki] proved that anything provable by two provers can be proven in a constant number of rounds, but achieving only a constant probability of error. Independently, Feige [Fe] proved that NEXP-time languages have one round two prover protocols with $\frac{1}{2} + \epsilon$ probability of cheating. So far, no one was able to reduce the probability of cheating below a constant.

Our main result in this paper is that the sequential [BFL] protocol can be ***fully parallelized into a single round protocol with an exponentially small probability of cheating***, and thus the multi-prover round hierarchy collapses to its first level: $MIP(poly) = MIP(1)$. Our analysis is based on the new notion of a ***quasi-oracle***, which is a weaker version of the oracle implemented by the second prover in the [BFL] sequential protocol,

but suffices to prove our result.
In Section 2 we briefly review the sequential [BFL] protocol and the method developed by Cai, Condon and Lipton [CCL] to parallelize sequential single prover protocols. In Section 3 we describe our parallel protocol, based on the quasi-oracle approach introduced in Section 4. Section 5 outlines some of the details of its proof of soundness. In Section 6 we combine the recent [LS] parallelization of the [BGKW] protocol and the new parallelization of the [BFL] protocol to obtain fully parallelized perfect zero-knowledge protocols for all $NEXP$-time languages under no cryptographic assumptions.

2 A Review of the [BFL] Two Prover Sequential Protocol

2.1 Introduction to NEXP-time Protocols

Let L be a language accepted by a nondeterministic exponential time Turing Machine M, and consider the 3-CNF formula which represents the computation of M on some input X ($|X| = k$) in L. It contains an exponential number of variables and an exponential number of clauses, but it is polynomially definable. We look at the assignment A of boolean values to all the variables as an arithmetic function from the set of n-bit indices $J = \{0,1\}^n$ to $\{0,1\}$. We now extend the domain of A from J to I^n where $I = \{0, \ldots, N-1\}$ for some large integer $N > 2^n$, defining its values over I^n as a multilinear extension of its values on J, and say that $A : I^n \rightarrow Z$ satisfies the 3-CNF formula if $A|_J$ consists of 0/1 values which satisfy this formula.
Lemma 1: (Babai, Fortnow and Lund) There exists a polynomial Q_X in $n^{O(1)}$ variables such that:

1. An arithmetic expression for Q_X of degree $n^{O(1)}$ is computable in $n^{O(1)}$ time from X.

2. A function $A : I^n \rightarrow Z$ satisfies Φ_x iff

   $$\sum Q_X(z, b_1, b_2, b_3, A(b_1), A(b_2), A(b_3)) = 0$$

   where the summation extends over all the possible 0/1 substitutions to the variables $(z \in \{0,1\}^{n^{O(1)}}, b_j \in J)$.

2.2 The [BFL] Protocol

Given a common input $X \in \{0,1\}^k$, the two infinitely powerful provers (P_1, P_2) want to prove to a random polynomial time verifier V that $X \in L$. Basically the [BFL] protocol consists of three parts:

1. **Multilinearity test:** In this part V infers with exponentially high confidence that a certain function A which the provers use is approximately multilinear. This is done by sending polynomially many random inputs from each one of polynomially many random lines in I^n to P_1, and receiving his answers about the A values on these inputs. The verifier then checks that along each line, the answers are linear and bounded by $B = 2^{n^c}$ for some constant c (for details see [BFL]).

2. **Satisfiability test:** Using the protocol of Lund, Fortnow, Karloff, Nisan [LFKN] and Shamir [S] the verifier checks that:

   $$\sum Q_X(z, b_1, b_2, b_3, A(b_1), A(b_2), A(b_3)) = 0$$

   by successively eliminating all the summation symbols, where $z, b_1, b_2, b_3 \in_R I^n$ are randomly chosen by V.

3. **The final substitution:** When all the summation symbols are eliminated, the verifier substitutes the values $A(b_1), A(b_2), A(b_3) < B$ (which he receives from P_1) into the known polynomial Q_X in order to check its claimed value.

P_2 is asked just a single query in each round: either from the first part (the multilinearity test), or from the third part (the values of $A(b_1)$, $A(b_2)$ or $A(b_3)$). Therefore in each sequential round of the [BFL] protocol, P_2 can be considered as a deterministic oracle for some function A. If A is indeed approximately multilinear, then the probability of cheating in the satisfiability test is exponentially small due to the soundness of the [LFKN] and [S] protocol.

2.3 The Parallelization Problem

Cai, Condon and Lipton [CCL] developed a general methodology for parallelizing certain protocols, which include Shamir's single prover sequential protocol for PSPACE. The parallelized PSPACE protocol requires a second prover and is executed in one round as follows: One of the provers receives in one message all the queries which the verifier sends in Shamir's sequential protocol, while the other prover receives a random length prefix of this sequence. The verifier checks the answers of the first prover as the sequential verifier does, but also checks the consistency of the answers of the two provers in the common prefix. This basic protocol is carried out polynomially many times *in parallel*. The reason that the probability of cheating decreases exponentially fast is the following: In each parallel round, the first prover sends a sequence of r

polynomials as a response to r random values in some field. The second prover receives the first $1 \leq l-1 \leq r$ random values and replies with the first l polynomials. If the actual statement proven in the protocol is false but the proof succeeds, there must be $1 \leq i \leq r$ such that the i'th polynomial in the sequence sent by the first prover is incorrect and the next one is correct. Therefore the same interpolation argument as that in [LFKN] and [Sh], yields that if the choice of l is equal to this i, then with overwhelming probability (due to the ratio between the degree of the polynomials and the size of the field) the verifier will catch the provers in one of the following cases: Either the polynomial sequence sent by the first prover is incorrect (with respect to V's queries) or the l'th polynomial sent by the second prover is different from that of the first prover. The probability that $l = i$ is $1/r$ and thus if this basic protocol is carried out in parallel r^2 independent executions, then with overwhelming probability there exists a parallel execution for which $i = l$, and this particular execution by itself suffices to make the probability of cheating exponentially low.

The parallelization technique of [CCL] can be applied to the satisfiability test (the second part) in the [BFL] protocol; The main problem is how to parallelize the multilinearity test. Consider now the following naive parallel version $(\hat{P}_1, \hat{P}_2, \hat{P}_3, V)$ of the [BFL] protocol: $\hat{P}_1$ receives in parallel all the queries which are sent to the first prover in the sequential [BFL] protocol. $\hat{P}_2$ receives a random length prefix of the satisfiability test. This basic protocol is carried out $poly(n)$ times in parallel. $\hat{P}_3$ receives in parallel $poly(n)$ queries, one from each round of the multilinearity test carried out by $\hat{P}_1$ and V.

This naive approach fails since $\hat{P}_3$ receives in this single-round protocol many queries for the multilinear function, and thus he can not be considered as an oracle for each such query. Assume for example, that this protocol is executed twice, where in the first execution $\hat{P}_3$ receives in parallel the pair of queries (a_1, a_2) and in the other execution he receives (a_1, a_2') where $a_2 \neq a_2'$. $\hat{P}_3$ may reply with (b_1, b_2) in the first execution and (b_1', b_2') in the second one, where $b_1 \neq b_1'$, and thus V receives two different answers for the same question a_1, without being able to catch the prover. This contrasts the situation in the sequential [BFL], in which the second sequential prover receives a single query in each round, and thus implements some oracle in each round (the fact that this oracle may vary from round to round is irrelevant in the proof of soundness).

3 A Parallel Protocol for NEXP-time

In this extended abstract, we describe the new protocol in terms of 4 independent provers. By applying the same proof technique to a different protocol, Feige (in a private communication) recently showed that two provers actually suffice.

P_1 receives in a single message all the queries which are directed at the first prover in the sequential [BFL] protocol, namely, the queries of the multilinearity test (from I^n), the queries of the satisfiability test, and the triplet $b_1, b_2, b_3 \in_R I^n$. P_1 should reply with the same answers as those of the sequential prover, and V checks P_1's replies in the usual way. V sends to P_2 a random length prefix of the satisfiability queries sent to P_1. P_2 should reply with the same answers as those of P_1 on the common queries. This basic protocol is carried out m times *in parallel*, where $m = poly(n)$. V accepts iff all of these m basic protocols succeed, in addition to the new protocol (P_3, P_4, V) described in the next subsection. The purpose of this new protocol is to force the provers to behave as consistent oracles which answer each query x by the same value $A(x)$ regardless of the other queries x', x",.. which are asked in parallel to x.

3.1 The Oraclization Protocol

The participants in the following one-round protocol are two independent provers P_3, P_4 and the random polynomial time verifier V.

V randomly and independently chooses a single query (a vector in I^n) from each one of the m parallel sets of queries for the multilinear function (as part of the multilinearity test or the final triplet of queries) which V asks P_1. Note that the queries along each line in the multilinearity tests are not pairwise independent, but since V chooses only one query from each parallel set, he gets m random and independent vectors $x_1, \ldots, x_m \in I^n$. From now on, all the computations are carried out over the finite field Z_p, for an arbitrary prime $p > B$.

V randomly and independently chooses m new vectors $y_1, \ldots, y_m \in Z_p^n$.

If there exists $1 \leq i \leq m$ for which $x_{i,1} = y_{i,1}$ then V sends nothing to P_3, P_4, accepts the execution and halts. Since this event has exponentially small probability ($1/p < 1/2^n$) we can ignore its negligible effect and deal just with the case where for each $1 \leq i \leq m$, x_i and y_i differ on their first coordinate (i.e. $x_{i,1} \neq y_{i,1}$). Then for each coordinate $2 \leq j \leq n$, we look at the j'th coordinate $x_{i,j}$ ($y_{i,j}$, respectively) as a linear function of the first coordinate $x_{i,1}$ ($y_{i,1}$). More precisely, we define for each i, a linear function over Z_p: $l_{i,j}(x) = a_{i,j}x + b_{i,j}$, whose value is $x_{i,j}$

at $x = x_{i,1}$, and $y_{i,j}$ at $x = y_{i,1}$. Therefore for each $1 \leq i \leq m$, the choices of x_i and y_i define a sequence of linear functions: $L_i(x) = (l_{i,2}(x), \ldots, l_{i,n}(x))$.
Due to the linear relations (L_i) between the coordinates of each of the vectors x_i, y_i we can consider the multilinear function $A(u_1, \ldots, u_n)$, when $(u_1, \ldots, u_n) \in L_i$, as a single variable polynomial $q_i \in Z_p[x]$ of degree at most n in the variable u_1.
V sends $y_1, \ldots, y_m$ to P_4 who replies with the multilinear values $A(y_1), \ldots, A(y_m)$. V sends $x_1, \ldots, x_m$ and $L_1, \ldots, L_m$ (as linear functions, without revealing the points y_i's which, together with the x_i's, defined them) to P_3, and receives from him $A(x_1), \ldots, A(x_m)$ and the polynomials $q_1, \ldots, q_m$. Now, for each $1 \leq i \leq m$, V checks that: (1)P_3's replies are identical to those of P_1 on the common queries (x_i's) (2) q_i is a polynomial of degree bounded by n in $Z_p[x]$ (3) $q_i(y_{i,1}) = P_4(y_i)$ (4) $q_i(x_{i,1}) = P_3(x_i)$. If this is the case he accepts, otherwise he rejects and halts. As proven in Section 5, acceptance implies that P_3 behaves as a weak oracle whose answers (on the x_i's) usually depend on the input but not on its context. We formalize this notion in the next section.

4 Quasi-Oracles

In our protocol P_3 gets $m = poly(n)$ simultaneous questions, and we want to prove that he either answers each one of the x_i's independently of the other x_j, $j \neq i$, or gets caught by V as a cheater with overwhelming probability. This intuitive notion is captured by the following definition:
Definition 1: (a quasi-oracle) A protocol (P_3, P_4, V) of the type described in Section 3 is a quasi-oracle if there exists a family

$$\mathcal{F} = \{f_y\},\ y \in \{(Z_p^n)^m\},$$

where each f_y is a vector of functions $(f_{y,1}, \ldots, f_{y,m})$ such that: $Pr\{P_3(x_1, \ldots, x_m) =$
$(f_{y,1}(x_1), \ldots, f_{y,m}(x_m))\ or\ V\ rejects\} \geq$
$1 - 1/2^{\Omega(n)}$
where the probability space is the uniform distribution over the vector of queries $x \in (I^n)^m$ sent to P_3 and the vector of queries $y \in (Z_p^n)^m$ sent to P_4.
Remarks:
(1) To be an oracle, a prover must answer the parallel queries by applying some function f to each one of them separately. To be a quasi-oracle, we give P_3 an extra degree of freedom: the function f is allowed to depend (from V's point of view) on the queries directed at P_4. This seems to be essential since we do not know how to construct a one-round protocol which can force P_3 to use the same f in all possible scenarioes. As shown later in this paper, this extra degree of freedom has no effect on the correctness of the whole $NEXP$-time protocol.
(2)The usual definition of an oracle requires that P_3 answers all the possible parallel queries in a consistent way. Again, we do not know how to achieve such a strong condition, and replace it by a weaker condition of consistency for most random choices of x.
(3) We do not specify the queries L_i directed at P_3 since they are uniquely defined by x and y.
(4) In the proof of soundness, rejection of fraudulent provers is a desirable outcome and thus we do not care to separate between the probability of consistent behavior and of failure. The only behavior we want to exclude is an inconsistent behavior which is not caught by V.
The technique used in the sequential [BFL] protocol, is to give the second prover one random query x_i, $(1 \leq i \leq m)$ from those given to the first prover in the i-th round and to check the consistency of the answers. The vector of queries $(x_1, \ldots, x_m)$ is thus randomly chosen with uniform distribution from $(I^n)^m$. If in addition to our parallel (P_1, P_2, V) subprotocol, we have a quasi-oracle (P_3, P_4, V) with a corresponding family $\mathcal{F}$ of functions, then the (parallelized) verifier randomly chooses $f_y = (f_{y,1}, \ldots, f_{y,m}) \in_R \mathcal{F}$ and for every $1 \leq i \leq m$ compares:
$P_1(x_i) \overset{?}{=} S_i(P_3(x_1, \ldots, x_m))$, where S_i is the selector function which returns the i'th of the m values in P_3's answer. With overwhelming probability V either accepts the right hand side which is equal to $f_{y,i}(x_i)$ or rejects (according to Definition (1)). Therefore, with exponentially high probability, the parallel verifier either rejects or compares P_1's answers with the replies of m oracles giving the $f_{y,i}$'s values, and thus the oracle implemented in the sequential [BFL] protocol can be replaced by a quasi-oracle, achieving a single-round multi prover protocol for $NEXP - time$.

5 The Soundness of the Parallel Oraclization Protocol

Theorem 1: The protocol (P_3, P_4, V) defined in subsection 3.1 is a quasi-oracle.
Proof: For each $1 \leq i \leq m$, we define the following "success matrix" M_i: The names of the rows are all the possible choices of $y = (y_1, \ldots, y_m) \in (Z_p^n)^m$, and the names of the columns are all the possible choices of $x = (x_1, \ldots, x_m) \in (I^n)^m$. As noted above, for each entry the queries $L_1, \ldots, L_m$ directed at P_3 are totally determined by the names of the corresponding row and column. An entry is equal to 1 iff $S_i(P_3(x_1, \ldots, x_m))$, $S_i(P_4(y_1, \ldots, y_m))$ and $S_i(P_3(L_1, \ldots, L_m))$ (which are defined by this entry) are accepted by V (then we

say that V accepts $S_i(P_3(x_1,\dots,x_m)))$, otherwise it is 0. For the sake of convenience, we reorganize the columns of M_i in the following order: The v's block of columns consists of all the columns whose names are in $I^n\times,\dots,\times I^n \times v \times I^n\times,\dots,\times I^n$, i.e. whose i'th coordinate is the value v and whose other coordinates range over all the possible values.

Each choice of row $y = (y_1,\dots,y_m)$ in M_i defines a function $A_{y,i}$ in the following way: For each possible value of x_i consider the corresponding block of columns, rank the accepted values of $S_i(P_3(x_1,\dots,x_m))$ into decreasing order of popularity, and define $A_{y,i}(x_i)$ as the best accepted value (ties can be broken lexicographically).

For example, if for the particular value v of the i-th query, P_3 provides in 1000 contexts the value 5 (900 of which are rejected by V) and provides in 200 contexts the value 7 (50 of which are rejected by V) among all the parallel queries defining the y-th row and v-th block, then $A_{y,i}(v) = 7$.

Since V randomly chooses $y = (y_1,\dots,y_m)$, he chooses a random row in the matrix to which we associate a function $A_y = (A_{y,1},\dots,A_{y,m})$. Our goal now is to prove that our two-prover one-round protocol is a quasi-oracle with respect to this family $\mathcal{A}$ of functions, and thus when V chooses a random function $A_y \in_R \mathcal{A}$, and a random parallel query $(x_1,\dots,x_m) \in_R (I^n)^m$, with overwhelming probability he either accepts the correct $A_{y,i}$ values or rejects the execution.

We denote by $\#s$ the number of occurrences of the s-most frequently accepted answer for x_i in the particular block and row of the matrix M_i, and denote by W the number of columns in that block.

The following Lemma formally states that V is very unlikely to accept any value other than the most popular one:

Lemma 2: For every $1 \le i \le m$ and for every value v of x_i, the fraction of rows in the block of columns in M_i associated with v for which:

$$\sum_{s\ge 2} \#s > \frac{W}{2^{n/5}} \qquad (1)$$

is less than $1/2^{n/4}$.

Proof: Fix $1 \le i \le m$ and consider just the block of M_i associated with v. We now divide the rows (whose names satisfy $y_{i,1} \ne v_1$) of this block of columns, into disjoint blocks of rows such that in each block of rows, all the y_j, $j \ne i$, are fixed, and all the y_i's belong to a common L_i to which v belongs. Thus each block of rows consists of exactly $p-1$ rows, and every two blocks of rows are disjoint since v is the single intersection point of the two corresponding lines L_i's. Note that P_3 (who receives the queries x and the lines L_i's) knows nothing about the actual choice of $y_{i,1}$. Now, we prove the Lemma separately for each block of rows within the v-th block of columns.

Due to the common L_i (on which all the y_i's and the v lie), and the common y_j's ($j \ne i$), once we fix a column, the queries L_i's are determined namely, they depend just on the column. Moreover, if V accepts two different replies of P_3 on v in two executions sharing the same y (which necessarily correspond to two different degrees of popularity $s' \ne s''$) then V receives two different polynomials (as answers to L_i) $q' \ne q''$ of degree bounded by n.

Assume that (1) is true for at least a fraction $1/2^{n/4}$ of the rows in the block of rows. Consider the probability distribution defined by randomly choosing one row and two columns in this block, which define two executions sharing the same y queries. The two columns define two polynomials q', q'' of degrees bounded by n in $Z_p[x]$ (provided by P_3 as answers to L_i). We derive a contradiction by trying to compute the probability of the event

$$E = \text{"}q' \ne q'' \textit{ and } V \textit{ accepts both executions}\text{"}$$

in two different ways. An easy upper bound can be obtained via:

$Pr\{E\} \le$
$Pr\{V \textit{ accepts both executions} \mid q' \ne q''\} \le$
$Pr\{q'(y_{i,1}) = q''(y_{i,1}) \mid q' \ne q''\} \le n/p$

since the non-zero n-th degree polynomial $(q' - q'')$ can have at most n possible zeroes in the field Z_p.

The computation of the lower bound is divided into two cases: In the first case, at least half of the rows in the block which satisfy (1) also satisfy:

$$\#1 > \frac{W}{2^{n/4}} \qquad (2)$$

Then:

$$Pr\{E\} \ge \frac{1}{2} * \frac{1}{2^{n/4}} * \frac{1}{2^{n/4}} * \frac{1}{2^{n/5}}$$

which is larger than n/p for $p \ge 2^n$, leading to a contradiction. In the second case, at least half of the rows in the block which satisfy (1) also satisfy:

$$\#1 \le \frac{W}{2^{n/4}} \qquad (3)$$

Then for each $s > 1$, $\#s \le \#1 \le \frac{W}{2^{n/4}}$, but the sum of all these $\#s$ is much higher $(> \frac{W}{2^{n/5}})$ and thus for each row which satisfies (1) and (3), there is a partition of the set of the s-entries (s-entry is an entry which corresponds to the s-most frequently accepted answer in this row) for $s \geq 2$, into two disjoint parts, each of which is greater than $\frac{1}{3} * \frac{W}{2^{n/5}}$. Therefore,

$$Pr\{E\} \geq \frac{1}{2} * \frac{1}{2^{n/4}} * (\frac{1}{3 * 2^{n/5}})^2$$

which is also larger than n/p. Again we get a contradiction, and the desired result follows immediately. ■

Since $m = poly(n)$ we can combine these results for all the M_i $(1 \leq i \leq m)$ and complete the proof of theorem 1. ■

This yields the main theorem:

Theorem 3: *The protocol* (P_1, P_2, P_3, P_4, V) *is a fully parallelized multi prover protocol for NEXP-time.*

6 Zero-Knowledge Aspects

In [LS] we describe a parallelization of the Ben-Or, Goldwasser, Kilian and Wigderson protocol for NP which preserves the perfect zero-knowledge nature of this protocol. As a first step we simplify the [BGKW] protocol into one which consists of polynomially many parallel executions of a one-round basic protocol in which the verifier sends a single random bit to each of the provers. In order to upper bound the probability of cheating in this protocol, and to prove that it is exponentially small, we transform this soundness problem into an extremal graph theoretic problem. We consider the 0/1 acceptance matrix (of size $2^n \times 2^n$) in which the column names are all the possible queries to the first prover in n parallel executions, and the row names are all the possible queries to the second prover. The value of an entry is 1 iff the verifier accepts all the answers of both provers. We prove that the existence of a certain constant size pattern of 1's in the matrix is a witness for the common input. Then we prove that any density of 1's in the matrix (regardless of their locations) which is higher than some exponentially low threshold, implies the existence of such a pattern of 1's. Consequently, the probability of cheating in this single round two prover zero-knowledge protocol is exponentially small.

Ben-Or, Goldwasser, Kilian and Wigderson proved that every multi prover interactive proof can be transformed into a multi prover perfect zero-knowledge protocol. Their transformation uses a two prover zero-knowledge protocol for NP as a subprotocol, and preserves the number of rounds whenever this subprotocol requires a single round. By combining this construction of [BGKW], the one-round two-prover zero-knowledge protocol for NP described in [LS], and our main result $NEXP - time = MIP(1)$, we conclude:

Theorem 4: *every language in* $MIP(poly)$ *has a fully parallelized multi prover perfect zero-knowledge protocol under no intractability assumptions.*

References

[BFL] L. Babai, L. Fortnow and C. Lund, *Non-Deterministic Exponential Time has Two Prover Interactive Protocols,* FOCS 90.

[BGKW] M. Ben-Or, S. Goldwasser, J. Kilian and A. Wigderson, *Multi-Prover Interactive Proofs: How to Remove Intractability Assumptions.* STOC 88.

[CCL] J.Y. Cai, A. Condon and R. Lipton, *PSPACE is Provable by Two Provers in One Round,* STRUCTURE 91.

[Fe] U. Feige, *On the Success Probability of the Two Provers in One Round Proof Systems,* STRUCTURE 91.

[Fo] L. Fortnow, Ph.D. Thesis.

[FRS] L. Fortnow, J. Rompel and M. Sipser, *On the Power of Multi Prover Interactive Protocols,* STRUCTURE 88.

[Ki] J. Kilian, *Interactive Proofs Based on the Speed of Light,* e-mail announcement.

[LFKN] C. Lund, L. Fortnow, H. Karloff and N. Nisan, *Algebraic Methods for Interactive Proof System,* FOCS 90.

[LS] D. Lapidot and A. Shamir, *A One-Round, Two-Prover, Zero-Knowledge Protocol for NP,* Crypto 91.

[Sh] A. Shamir, *IP=PSPACE,* FOCS 90.