LARGE-SCALE SECURE COMPUTATION

MPC FOR PARALLEL RAM PROGRAMS

Elette Boyle
Technion

Based on join works with Kai-Min Chung and Rafael Pass
Multi-Party Computation (MPC)

[GMW87] – Computational Setting
[BGW88, CCD88] – Information Theoretic Setting with Secure Channels

128-bit AES
This Talk: Large-Scale MPC

Set intersection of Facebook friends

[GMW87] – Computational Setting
[BGW88, CCD88] – Information Theoretic Setting with Secure Channels
This Talk: Large-Scale MPC

Lots of Parties

Lots of Data
MPC Efficiency Metrics

Communication  Memory  Computation

How are these affected in the large-scale setting?
Costs of Communication

- # of bits communicated
- # of sequential rounds
- …Who a party is speaking to

Nearly all protocols:
Every party speaks to every party
Communication: *Locality Metric*

\[ \text{\# parties:} \quad n = 5 \]
Communication: *Locality Metric*

# parties:  
$n = 10,000$

[Not practical!]

[BGT13]
Communication: *Locality* Metric

# parties: 
\[ n = 10,000 \]

**Locality:** Total # parties each party communicates with throughout protocol lifetime
Memory: Balancing the Burden

- Combined data size is huge!
- Want: Memory requirement per party
  \[ \approx \left( \text{his input} + \text{Space}(\Pi)/n \right) \]
Computation: Going Beyond Circuits

Optimizing this transformation yields better MPC efficiency

MPC protocol for securely computing the circuit
Computation: Going Beyond Circuits

Program

Generically:
Blow up by factor of entire database size!

```
BINRY-SEARCH(a, T, p, r)
1   low = p
2   high = max(p, r + 1)
3   while low < high
4       mid = (low + high)/2
5           if x ≤ T[mid]
6               high = mid
7           else low = mid + 1
8       return high
```
Models of Computation 101

- Circuits
  - AND, OR, NOT gates

- Turing Machines

- RAM Machines

- Parallel RAM Machines
Computation: Going Beyond Circuits

Large-scale computations $f$ leverage \textit{random access} and \textit{parallelism}

\begin{itemize}
  \item Circuit (and TM) model for $f$ not appropriate!
\end{itemize}
Computation: Going Beyond Circuits

Large-scale computations $f$ leverage *random access* and *parallelism*

- Circuit (and TM) model for $f$ not appropriate!
- RAM model for $f$ loses parallelism!
- **Parallel RAM (PRAM) Model**
Rough History of Prior MPC Work

• Circuits model
  E.g.: Original protocols [GMW87, BGW88, CCD88,…], **Scalable MPC** [DI06, DN07, DIK+08, DIK10, DKMS12, ZMS14], MPC on incomplete networks [CGO10, CGO12], MPC based on FHE / Obfuscation [Gen09, AJL+12, MSS13, GGHR14], Optimized MPC for practice [BNP08, KS08, LPS08, NO09, LP11, BDOZ11, DPSZ12, NNOS12, L13, FJN+13, ALSZ13, DZ13, LR14, ZRE15,…]

• RAM model
  – **2-PC** [OS97, GKK+11, LO13, GGHJ+13, GHRW14, WHHSS14]
  – Extensions to MPC [DMN11] **don’t scale with n**

• PRAM model (nothing)

Eg: Per-party memory requirement ~ size of all parties’ inputs
The Goal: Efficient MPC for PRAM

\( n \)-party MPC for PRAMs \( \Pi \)

- **Time Steps** - \( \text{Parallel Time}(\Pi) \)
- **Per-party Computation** - \( \frac{\text{Comp}(\Pi)}{n} + \text{His input} \)
- **Per-party Memory** - \( \text{His input} + \frac{\text{Space}(\Pi)}{n} \)
- **Comm Locality** - 1

Asymptotically Needed for security
Theorem \[\text{[BCP14, BCP15]}:\]

\(n\)-party MPC for PRAMs \(\Pi\)

- Rounds: \(\tilde{O}(\text{Parallel Time}(\Pi))\)
- Per-party Computation: \(\tilde{O}(\text{Comp}(\Pi)/n)\)
- Per-party Memory: \(\tilde{O}(\text{His input} + \text{Space}(\Pi)/n)\)
- Comm Locality: \(\tilde{O}(\text{His input}) + \text{BC } /\text{party}\)

Given a 1-time (reusable) preprocessing stage

Static corruptions, \(2/3 + \varepsilon\) honest parties, Unconditional security
The Construction
Rough Intuition of MPC Protocols

1. **Step 1: Secret Share** inputs across parties
   Eg: evaluations of random polynomial $p(0)=s$ [Sha79]

2. **Step 2: Evaluate gate-by-gate** on shares
   (sometimes with communication)

3. **Problem 1:** Everyone talks to everyone
4. **Problem 2:** Everyone stores all inputs

5. **Problem 3:** Computation ~ Circuit Size

For Large Data, Many Parties...
Consider a Simpler Problem: Large Data, Few Parties

**Step 1:** Secret Share inputs across parties

Eg: evaluations of random polynomial st \( p(0) = s \) [Sha79]

**Step 2:** Evaluate gate-by-gate on shares

(sometimes with communication)

**Problem 1:** Everyone talks to everyone

**Problem 2:** Everyone stores all inputs

**Problem 3:** Computation ~ Circuit Size
Consider a Simpler Problem:

**Large Data, Few Parties**

- **Step 1**: Secret Share inputs across parties
  
  Eg: evaluations of random polynomial st \( p(0) = s \) \([\text{Sha79}]\)

- **Problem 1**: Everyone talks to everyone
- **Problem 2**: Everyone stores all inputs

- **Step 2**: Evaluate gate-by-gate on shares
  
  (sometimes with communication)

- **Problem 3**: Computation \(\sim\) Circuit Size

Wanted:

\(\text{Comp} \sim |\text{PRAM}|\)
How PRAM Works

PRAM Π

CPU_1

CPU_2

CPU_m

Shared Memory Database
MPC for PRAM: First Idea

Emulate CPU steps via MPCs for circuits [LO96,DMN11]
MPC for PRAM: First Idea

• **Step 1:** Secret Share inputs across parties
• **Step 2:** Emulate PRAM CPU steps via small-scale MPCs

Parties only see addresses & shares of secrets!

Addresses may leak information!
Memory Access Patterns May Leak Information!

Toy Example: Binary Search for 100

Access mem 4
Access mem 6
Stop

Reveals:

$x_4 < 100$
$x_6 = 100$
Wanted: PRAM $\rightarrow$ **Oblivious** PRAM

“Oblivious” = memory access patterns appear independent of data

**Toy Example: Binary Search for 100**

Access mem 4
Access mem 6
Stop

Reveals: $x_4 < 100$
$x_6 = 100$

Leaks Information!
Oblivious Program Compilers

History:

- Turing Machines: $\log(M)$ overhead [PF 79]
- RAM programs: $\text{polylog}(M)$ overhead [Gol86, Ost90, GO96, Ajt10, DMN11, SCSL11, CP13, GGHJ+13, SDSF+13]
- PRAM: $\text{polylog}(M)$ overhead [BCP14]
Core Problem:
Supporting Parallel Accesses!

Can’t afford for CPUs to take turns!

Storing multiple copies causes consistency issues!

Reveals lookup collision!
New Protocol: (Few-Party) \textbf{MPC for PRAM}

- **Step 1:** Secret Share inputs across parties

- **Step 2:** PRAM $\rightarrow$ Oblivious PRAM

- **Step 3:** Emulate OPRAM via small-scale MPCs
And for **Large Data** and **Many Parties**…

- **Step 1**: Secret Share inputs across parties
  
  **Problem 1**: Everyone talks to everyone
  **Problem 2**: Everyone stores all inputs

- **Step 2**: PRAM $\rightarrow$ Oblivious PRAM

- **Step 3**: Emulate OPRAM via small-scale MPCs

**Computation $\sim |PRAM|$**
Teaser of Additional Techniques

- **Electing Committees**

- **Load-Balancing via Job Passing**

- **Distributed OPRAM**

- **Load-Balanced Routing over Expander Graphs**
Future Directions

• “OPRAM is the new ORAM” - me

• Pushing Large-Scale MPC toward Practicality
  
  Leveraging computational assumptions?  Adaptive security?
  Improving broadcast with locality?  Honest minority?  Targeted protocols?
  MPC for MapReduce?  Asynchronous models?