**Parallel Algorithms for Accelerating Homomorphic Evaluation**

Krish Sunil Rohra (kr2920@rit.edu) | Dr. Peizhao Hu
Rochester Institute of Technology

---

**Introduction & Motivation**

Homomorphic encryption allows computations on ciphertext without decrypting it first. The homomorphic evaluations are computationally intensive due to various noise reduction procedures that are required after each multiplication. In this project, we explore parallel algorithms to accelerate the evaluation of a protocol for comparing numbers homomorphically.

**Secure Comparison Protocol**

The ability to compare two encrypted numbers without the evaluator knowing them has a lot of applications, such as in private evaluation of a decision tree. As an example, we investigate a protocol [1] that produces a single bit as the comparison result.

**Theorem:** Given two ℓ-bit integers in their binary representations x = (x₁, x₂, x₃, …, x₇) and y = (y₁, y₂, y₃, …, y₇), b = 1 (x < y) if and only if there exists an index j, where 0 ≤ j ≤ ℓ, such that xₗ < yₗ and all leading bits are equal xₗ = yₗ ≥ j; otherwise, b = 0.

This protocol can be represented as an arithmetic circuit.

\[
\begin{align*}
& (x₁ < y₁) \lor \\
& (x₁ = y₁) \land (x₂ < y₂) \lor \\
& \vdots \\
& (xₗ = yₗ) \land \cdots \land (xₗ = yₗ) \land (x₀ < y₀) \land \cdots
\]

where \((xₗ < yₗ) = (xₗ \land yₗ) = (1 \land xₗ)\) and \((xₗ = yₗ) = (xₗ \lor yₗ \lor 1)\).

**Sequential Evaluation Approach:** Evaluating the above protocol sequentially incurs high number of consecutive multiplications, specifically, \(3^\ell - 2\), where \(\ell\) is the bit-length. This results in selecting large parameters for the encryption scheme, which in turn affects the running time of the algorithm and generates large ciphertexts.

**Proposed Solutions**

### Approach 1: Maximize reusable computations

- Four prioritized stacks are used to give preference to computationally intensive tasks that are constrained by data dependency.
  - Accelerated by OpenMP multithread parallelization
  - Avoided repeated computations by caching
  - Did not improve on multiplicative depth
  - Required strict multithreading management

### Approach 2: Minimize multiplicative depth

- Targeting reduction of multiplicative depth by pairing AND operations that include multiplications
  - Reduced Multiplicative depth: \(\log(\ell) + \ell/2 + \ell\)
  - Allowed smaller parameters (smaller ciphertexts and faster evaluation time)
  - Supported acceleration and caching
  - Introduced marginal overhead for tree creation

**Analysis of multiplicative depth**

<table>
<thead>
<tr>
<th>Input Size</th>
<th>4-bit</th>
<th>8-bit</th>
<th>16-bit</th>
<th>32-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>10</td>
<td>22</td>
<td>46</td>
<td>94</td>
</tr>
<tr>
<td>Approach 1</td>
<td>10</td>
<td>22</td>
<td>46</td>
<td>94</td>
</tr>
<tr>
<td>Approach 2</td>
<td>8</td>
<td>15</td>
<td>28</td>
<td>53</td>
</tr>
</tbody>
</table>

This table shows the number of consecutive multiplications.

**Characteristics of the circuit:**
- The structure of the circuit is PREDICTABLE which can be leveraged for developing parallel algorithms
- Complexity increases with the bit-length of the input
- A lot of REPEATED computations
- INDIVIDUAL operations are mostly INDEPENDENT

**Results & Discussions**

<table>
<thead>
<tr>
<th>Input Size</th>
<th>4-bit</th>
<th>8-bit</th>
<th>16-bit</th>
<th>32-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>0.95s</td>
<td>11.05s</td>
<td>86.40s</td>
<td>1129.22s</td>
</tr>
<tr>
<td>Approach 1</td>
<td>0.46s</td>
<td>4.85s</td>
<td>43.80s</td>
<td>491.24s</td>
</tr>
<tr>
<td>Approach 2</td>
<td>0.35s</td>
<td>2.79s</td>
<td>20.81s</td>
<td>175.67s</td>
</tr>
</tbody>
</table>

Approach 2 outperforms the sequential approach and Approach 1 by a speedup factor up to 6.5x on a 8-core machine.

**Conclusion**

We can significantly improve the evaluation time by reducing multiplicative depth (smaller HE parameters), but careful design is needed to ensure efficiency while maintaining high security level.

**Reference**