MP-SPDZ – A Versatile Framework for Multi-Party Computation

Marcel Keller

CSIRO's Data61

13 September 2021
Secure Multiparty Computation

Wanted: \( f(x, y, z) \)

- Computation on secret inputs
- Replace trusted third party
- Central questions in MPC
  - How many honest parties?
  - Dishonest parties still follow the protocol?
- MP-SPDZ supports > 30 protocols across all properties
Unifying MPC: Black Box

Parties
- Have handles to values
- Don’t know the values
- Can input values
- Can agree on computations creating new values
- Can agree on outputting values

Unifying MPC: Black Box

\[ a + b = c \]
Unifying MPC: Basic Operations

Communication

<table>
<thead>
<tr>
<th>Operation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>✗</td>
</tr>
<tr>
<td>Multiplication</td>
<td>✓</td>
</tr>
</tbody>
</table>
Multiplication with Random Triple
(Beaver Randomization)

Have: \( x, y \), addition in black box
Want: \( x \cdot y \)
Multiplication with Random Triple
(Beaver Randomization)

Have: \( x, y \), addition in black box

Want: \( x \cdot y \)

\[
x \cdot y = (x + a - a) \cdot (y + b - b) = (x + a) \cdot (y + b) - (y + b) \cdot a - (x + a) \cdot b + a \cdot b
\]
Multiplication with Random Triple
(Beaver Randomization)

Have: \( x, y \), addition in black box, \((a, b, a \cdot b)\) for random \( a, b \)

Want: \( x \cdot y \)

\[
x \cdot y = (x + a - a) \cdot (y + b - b)
\]

\[
= (x + a) \cdot (y + b) - (y + b) \cdot a - (x + a) \cdot b + a \cdot b
\]

Masked and revealed (one-time pad)

Random secret triple (preprocessed)
Scaling Up: I/O Parallelization

\[ z = x \cdot y \]

\[ u = z \cdot w \]

\[ z = x \cdot y \]

\[ u = v \cdot w \]
Scaling Up: I/O Parallelization

1. Compute $z$
2. Compute $u$

$$z = x \cdot y$$
$$u = z \cdot w$$

$z = x \cdot y$
$u = v \cdot w$

1. Compute $z$ and $u$

1. Compute $z$
2. Compute $u$
Goal: Automatize I/O Parallelization

Manual parallelization is tedious:

\[
\begin{align*}
x_{10} &= x_2 \cdot x_3 \\
x_{11} &= x_8 + x_4 \\
x_{12} &= x_{10} \cdot x_1 \\
x_{13} &= x_7 + x_9 \\
x_{14} &= x_7 \cdot x_1 \\
x_{15} &= x_9 + x_{12} \\
x_{16} &= x_{13} \cdot x_{14} \\
x_{17} &= x_0 + x_{11} \\
x_{18} &= x_{11} \cdot x_{15} \\
x_{19} &= x_{13} \cdot x_7 \\
x_{20} &= x_4 + x_6 \\
x_{21} &= x_{16} + x_2 \\
x_{22} &= x_0 + x_{12} \\
x_{23} &= x_{22} + x_{14} \\
x_{24} &= x_{11} + x_{19} \\
x_{25} &= x_4 \cdot x_{19} \\
x_{26} &= x_{23} \cdot x_9 \\
x_{27} &= x_7 \cdot x_5 \\
x_{28} &= x_{13} + x_{21} \\
x_{29} &= x_{14} + x_{27} \\
x_{30} &= x_{19} \cdot x_1 \\
x_{31} &= x_{16} + x_{26} \\
x_{32} &= x_0 \cdot x_{10} \\
x_{33} &= x_{26} + x_{32} \\
x_{34} &= x_7 + x_3 \\
x_{35} &= x_9 \cdot x_{29} \\
x_{36} &= x_{33} + x_{22} \\
x_{37} &= x_{29} \cdot x_{24} \\
x_{38} &= x_{16} + x_{23} \\
x_{39} &= x_{15} + x_{37} \\
x_{40} &= x_{12} \cdot x_{39} \\
x_{41} &= x_{34} + x_7 \\
x_{42} &= x_{32} + x_5 \\
x_{43} &= x_{12} + x_{26} \\
x_{44} &= x_{43} \cdot x_{38} \\
x_{45} &= x_{38} + x_{14} \\
x_{46} &= x_{44} \cdot x_{27} \\
x_{47} &= x_{22} + x_{24} \\
x_{48} &= x_{39} \cdot x_{38} \\
x_{49} &= x_{21} \cdot x_3 \\
x_{50} &= x_{28} + x_{16} \\
x_{51} &= x_{15} + x_{38} \\
x_{52} &= x_{50} \cdot x_{46} \\
x_{53} &= x_{19} + x_2 \\
x_{54} &= x_{20} \cdot x_{13} \\
x_{55} &= x_{21} + x_{22} \\
x_{56} &= x_{19} \cdot x_6 \\
x_{57} &= x_{46} + x_1 \\
x_{58} &= x_{38} \cdot x_{55} \\
x_{59} &= x_{47} + x_{29}
\end{align*}
\]
from util import max

M = sint.Matrix(n_rows, n_cols)
res = sint.Array(n_rows)

# populate M
...

for i in range(n_rows):
    res[i] = M[i][0]
    for j in range(1, n_cols):
        res[i] = max(res[i], M[i][j])

Want
Maximum of every row

Without optimization
n_rows * (n_cols - 1) rounds of max

MP-SPDZ optimization
n_cols - 1 rounds of max
from util import max, tree_reduce

M = sint.Matrix(n_rows, n_cols)
res = sint.Array(n_rows)

# populate M
...

for i in range(n_rows):
    res[i] = tree_reduce(max, M[i])

Want
Maximum of every row

Without optimization
n_rows * log(n_cols) rounds of max

MP-SPDZ optimization
log(n_cols) rounds of max
Toolchain Overview

Compiler
- Implemented in Python
- Optimization (reduce network rounds)
- Library for various arithmetic: integer, fractional, mathematical
- Machine learning functionality

Virtual machine
- One per protocol
- Implemented in C++
- Optimized for speed
Section 2

Machine Learning
Privacy-Preserving Machine Learning

Outsourced training
- Data owners share their inputs among computing parties
- Computing parties train a model securely using MPC
- Output model OR use it for secure inference
Deep Learning

- Established supervised machine learning concept (known input-output combinations)
- Computation as chain of functions (layers)
- Some functions have parameters to be changed during training
- Function quantifying quality (loss)
- Chain rule allows changing of parameters toward minimizing loss (backward propagation)
Secure Deep Learning Building Blocks

Quantization
Represent $x$ as $\lfloor x \cdot 2^f \rfloor$ to use integer computation for fractional numbers.

Mathematical functions
- Comparison
- Division
- Exponentiation
- Logarithm
- Square root
MNIST – Handwritten Digit Recognition

“Hello world” of machine learning
Input: 28x28 gray-scale
Output: 0–9
Demonstrates utility of convolution (local linear function)

By Josef Steppan - Own work, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=64810040
Results for LeNet

- Convolutional neural network by LeCun et al.
- 4 linear layers
- AMSgrad optimizer (improved stochastic gradient descent)
- Co-located AWS c5.9xlarge
- Time per epoch: 9 minutes
- 1 hour for 99% accuracy
Section 3

Outlook
Secure Computation Suitability

More suitable
- Small input/output: e.g. mathematical functions
- Predictable computation path: e.g. matrix multiplication

Less suitable: data-dependent computation path
- Graph algorithms
- Dictionary data structure
More utility, less mystery
Beware of lower bounds
Tell me
Links

https://github.com/data61/MP-SPDZ
https://mp-spdz.readthedocs.io
https://ia.cr/2020/521
https://twitter.com/mkskeller