Computer is divided to observable and protected parts
Technologically possible: accessible memory but protected processor
Today: making interaction between processor and memory useless for learning program

Outline

What Kind of Computer Are We Going To Construct?

Basic Solutions

Hierarchial Construction

Computer Model

Two parts: Memory and Processor
Internal memory of Processor = $c \log |\text{Memory}|$
Interaction: $\text{fetch}(\text{adress}), \text{store}(\text{adress}, \text{value})$
Processor has access to random oracle
Computation starts with a program and an input in Memory
One step: fetch one cell - update value and Processor memory - store

Oblivious Execution

We want to hide: order of accesses to cells of Memory
Oblivious execution:
For all programs of size $m$ working in time $t$
order of fetch/store addresses is the same
Weaker requirement:
For all programs of size $m$ working in time $t$
order of fetch/store addresses has the same distribution

Naive Simulation

Simulation 1:
We store encrypted pairs (adress, value) in memory cells
For every fetch/store we scan through all memory
Wrong adress $\Rightarrow$ just reencrypt and store
Right adress $\Rightarrow$ do the job $\Rightarrow$ encrypt and store the result

Cost of simulation: $tm$ time, $m$ memory
We need to protect:
- Order of accesses
- Number of accesses

Memory = Main Part \( m + \sqrt{m} \) \mid Shelter \( \sqrt{m} \)

Idea:
- Divide computation in epochs of \( \sqrt{m} \) steps each
- On each original step make one fetch to the Main Part and scan through all the Shelter

Simulation 2:
- Store input in the Main Part
- Add \( \sqrt{m} \) dummy cells to the Main part
- For every epoch of \( \sqrt{m} \) steps
  - Permute all cells in the Main Part (using permutation \( \pi \) from random oracle)
  - For each process(\( i \)) scan through the shelter. If \( i \)-th element is not founded, fetch it from \( \pi(i) \), otherwise fetch next dummy cell
- Update (obliviously) the Main Part using the Shelter values

Cost of simulation: \( t \sqrt{m} \) time, \( m + 2 \sqrt{m} \) memory

Buffer Solution (1): Oblivious Hash Table

Memory of initial program: \( (a_1, v_1), \ldots, (a_m, v_m) \)

- Take a hash function \( h : [1..m] \rightarrow [1..m] \)
- Prepare \( m \times \log m \) table
- Put \( (a_i, v_i) \) to random free cell in \( h(a_i) \)-th column
- Home problem 4: Prove that the chance of overflow is less than \( 1/m \)

Buffer Solution (2): Simulation

Restricted problem: assume that every cell accessed only once

Simulation 3:
- Construct (obliviously) a hash table
- For every step fetch(\( i \)) of initial program
  - Scan through \( h(i) \)-th column
  - Update the target cell

Cost of simulation: \( t \log m \) time, \( m \log m \) memory

Outline

1. What Kind of Computer Are We Going To Construct?
2. Basic Solutions
3. Hierarchial Construction

Hierarchial Simulation

Simulation of processing cell \( i \):
- Scan through 1-buffer
- For every \( j \) scan through \( h(i,j) \)-th column in \( j \)-buffer
- Put the updated value to the first buffer

Data Structure

- \( k \)-Buffer = table \( 2^k \times k \)
- Hierarchial Buffer Structure = 1-buffer, \ldots, \log t-buffer
- Initial position: input in last buffer, all others are empty

Periodic Rehashing

Refreshing the data structure:
- Every \( 2^{l-1} \) steps unify \( j \)-th and \( j - 1 \)-th buffers
- Delete doubles
- Using new hash function put all data to \( j - \)th level

Invariant: For every moment of time for every \( l \) buffers from 1 to \( l \) all together contain at most \( 2^{2l-1} \) elements
Discussion

Comments on final solution:
- Cost: $O(t \cdot (\log t)^3)$ time, $O(m \cdot (\log m)^2)$ memory
- Omitted details: realization of oblivious hashing and random oracle
- Tamper-proofing extension

Summary

Main points:
- Theoretical model for hardware-based code protection: open memory/protected CPU
- Central problem: simulation of any program with any input by the same access pattern
- Current result: $O(t \cdot (\log t)^3)$ time, $O(m \cdot (\log m)^2)$ memory simulation

Home Problem 4

Prove that the chance of overflow in hash table construction is less than $1/m$

Reading List

O. Goldreich, R. Ostrovsky
Software protection and simulation on oblivious RAM, 1996.
http://www.wisdom.weizmann.ac.il/~oded/PS/soft.ps

Thanks for attention. Questions?