High-Frequency FOREX Trading:
Identification of Triangular Arbitrage Opportunities

Graham Gobieski, Kevin Kwan, Ziyi Zhu, Shang Liu
Demos

Demo 1

Source

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-7</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>-9</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Destination

Demo 2

Source

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-9</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>-13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Demo 3: Live Cycles
Motivation

• High-Frequency Trading: taking advantage of opportunities (inefficiencies, etc.) on very short timescales

• Triangular Arbitrage: due to market inefficiencies, exchanging a currency between three or more currencies and arriving back at the original currency might be profitable

• Timescale: ~5-20ms, data streams over network
Bellman-Ford

for each vertex x in V do
    if x is source then
        w(x) = 0
    else
        w(x) = INFINITY
        p(x) = NULL
    end if
end for

for i = 1 to v - 1 do
    for each edge(i, j) in E do
        if w(i) + w(i, j) < w(j) then //Relaxation
            w(j) = w(i) + w(i, j)
            p(j) = i
        end if
    end for
end for

for each edge(i, j) in E do
    if w(j) > w(i) + w(i, j) then
        //Found Negative-Weight Cycle
    end if
end for

Transformation
1. w1 * w2 * w3 * ... * wn > 1
2. log(w1) + log(w2) + log(w3) + ... + log(wn) < 0
3. -(log(w1) + log(w2) + log(w3) + ... + log(wn)) < 0
Hardware Design

**Storage: SRAM**
- Vert List (33 Nodes)
- Adjacency Matrix (33 Nodes)

**FOREX**
- Frame

**Container**
- Update Adjacency Matrix

**Bellman-Ford**
- Setup:
  - Src: 0
  - Other: Inf
- Read:
  - Src and Dst
- Relax:
  - \( w(src) + w(e) < w(dst) \)

**Cycle Detection**
- Read:
  - Vertex
- Test:
  - \( w(src) + w(e) < w(dst) \)
- Read Cycle:
  - \( \text{pred}(dst) \)

**Print-Cycle**
- Read:
  - Vertex
- Test:
  - High bit is 1
- Print:
  - frame_we <= 1

**Program Flow**
Software Design, VGA

Python Front-End
1. Load Data
2. Preprocess Data
3. Write Data via custom ioctl call

Kernel Module
1. Setup Memory-mapped I/O
2. Write Data to Bus

C Front-End
1. Load Data
2. Preprocess Data
3. Write Data via custom ioctl call
4. Write keyboard events

Kernel Module 2
1. Setup Memory-mapped I/O
2. Write Key Data to Bus

Option 1

Option 2

Hardware

Amba Bus
RTL Synthesis

DIAGRAM

FOREX

Container

Other Modules
Challenges

- Obviously Timing…
- Memory Accesses
- Nested Non-blocking Assignments
- Combinational Logic and Sequential Logic Working Together
- Compile Time