Towards Better Code Generator Design and Unification for Stack Machine
Why do we care?

• New stack machines keep on appearing given the development of blockchain technology (EVM, TVM).

• The reason for choosing stack machines in blockchain is better code density. It allows reducing costs incurred from contract storage in the blockchain.

• Better verification is another reason to use stack machines.
### Comparing existing stack machines

<table>
<thead>
<tr>
<th>Property</th>
<th>TVM</th>
<th>EVM</th>
<th>WebAssembly</th>
<th>x87</th>
<th>.net</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Stack operations</strong></td>
<td>Materializing constants, copy, xchg, push, pop, and derivatives</td>
<td>Copy, push, pop, xchg</td>
<td>Push</td>
<td>Push, pop, index</td>
<td>Push, dup, pop</td>
</tr>
<tr>
<td><strong>Non-stack entities</strong></td>
<td>Non GP-registers</td>
<td>Memory, Storage</td>
<td>Locals, globals</td>
<td>Memory</td>
<td>Locals, globals, function arguments</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>Modelling</td>
<td>Conventional</td>
<td>Conventional</td>
<td>Conventional</td>
<td>Conventional</td>
</tr>
<tr>
<td><strong>Control flow</strong></td>
<td>Continuations</td>
<td>Branches</td>
<td>Branches + Loops</td>
<td>N/A</td>
<td>Branches</td>
</tr>
<tr>
<td><strong>Instruction set</strong></td>
<td>CISC w/o direct manipulations with registers</td>
<td>RISC</td>
<td>RISC</td>
<td>CISC</td>
<td>RISC</td>
</tr>
</tbody>
</table>
Towards Better Code Generator Design and Unification for Stack Machine

Sample TVM code

```c
int func (int a, int b) {  
    return (a + b) * (a - b);  
}  
.globl func  
.type func,@function  
func:  
    ; %bb.0:  
        PUSH2 s1, s0  
        SUB  
        XCHG s0, s2  
        ADD  
        MUL  
    .Lfunc_end0:  
```
Stack machine as an abstraction

• Each N-th operation consumes N-topmost values from the stack and records the result on top of the stack.

• Stack rearrangements related to computation have costs. The TVM architecture uses copies and exchanges, other architectures - pushes.

• The code generator goal is to minimize stack manipulation costs.
Towards Better Code Generator Design and Unification for Stack Machine

Current approach to translation

\[
\begin{align*}
a &= x_1 + x_2 \\
d &= x_3 + x_4 \\
b &= x_5 + x_6 \\
c &= a + b \\
d &= x_3 + x_4
\end{align*}
\]

Selection DAG

MIR for a register machine

PHI elimination

Stackification (WebAssembly)

Stack Modelling (TVM)

MIR for a stack VM

Global ISEL?

\[
\begin{align*}
c &= a + b; \\
xchg s1, s2; \\
c &= a + b;
\end{align*}
\]

[b, d, a] 
[b, d, a] 
[b, a, d]
What can be improved?

- In addition to ScheduleDAGFast & Schedule DAGVLIW, SelectionDAG configuration can be modified to provide better schedules for stack machines.

- Stack modeling generalization can be considered.

- Post-RA scheduling (control-flow stackification) and stack-aware reassociation are also options to consider.
Pre-RA scheduler optimizations

• LRN or RLN traversals of DAG for better stack machine scheduling.
• Passes that introduce copies of virtual registers with multiple uses.
Stack modelling potential

• While now we may seem the only users of stack machine modelling, it can be generalized to support various stack manipulation types.

• For both register and stack MIR, stack modelling can provide a helpful analysis pass to request stack configuration for a specific instruction.
Post-RA scheduling optimizations

• Output stack configurations of all predecessors must be equivalent.

• Initial stack configuration of the basic block can specify the required number of manipulations.

• The optimization goal is to minimize the number of stack manipulations in a current basic block and at the end of its predecessors.
Suggestions on stack aware reassociation

- $a + b + c + d$ with $[a, c, b, d]$ stack is better to calculate as $a + c + b + d$
- This type of reassociation cannot be performed before computation of initial stack configurations.
Thank you.

tonlabs.io