RISC-V based Programming Model of a Computational SRAM Vector Processing Unit

Maha Kooli, Henri-Pierre Charles, Thaddée Bricout, Jean-Philippe Noel, Maria Ramirez Corrales, Bastien Giraud
Univ. Grenoble Alpes, CEA, List, F-38000 Grenoble, France
maha.kooli@cea.fr

Von Neumann Bottleneck

- Integrating computation inside or near memories
- Reduce data transfer between processor and memory
- Increase the system performance and reduce energy consumption

Computational SRAM (C-SRAM)

<table>
<thead>
<tr>
<th>Category</th>
<th>Width (bits)</th>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSRAM Line</td>
<td></td>
<td>copy</td>
<td>Copy a line into another</td>
</tr>
<tr>
<td>CSRAM Line</td>
<td>8,16,32</td>
<td>copyeq, copyeq, copyeq, copyeq, copyeq, copyeq</td>
<td>Conditional Copy</td>
</tr>
<tr>
<td>CSRAM Line</td>
<td>8,16,32</td>
<td>xor</td>
<td>Broadcast 8,16 or 32-bit value to whole line</td>
</tr>
<tr>
<td>32,64</td>
<td>xswap</td>
<td>Horizontal 32 or 64-bit word swap</td>
<td></td>
</tr>
<tr>
<td>Logical</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSRAM Line</td>
<td></td>
<td>and</td>
<td>Logical AND</td>
</tr>
<tr>
<td>CSRAM Line</td>
<td></td>
<td>or</td>
<td>Logical OR</td>
</tr>
<tr>
<td>CSRAM Line</td>
<td></td>
<td>xor</td>
<td>Logical XOR</td>
</tr>
<tr>
<td>Arithmetic</td>
<td></td>
<td>add</td>
<td>Arithmetic Addition</td>
</tr>
<tr>
<td>Arithmetic</td>
<td></td>
<td>sub</td>
<td>Arithmetic Subtraction</td>
</tr>
<tr>
<td>Arithmetic</td>
<td></td>
<td>mul</td>
<td>Arithmetic 8-bit integer multiply</td>
</tr>
<tr>
<td>Arithmetic</td>
<td></td>
<td>mac</td>
<td>Arithmetic 8-bit integer multiply-accumulate</td>
</tr>
</tbody>
</table>

RISC-V based Software Tool Chain

- A QEMU-based emulator
- A RISC-V C cross-compiler and debugger
- A source-to-source compiler to generate C-SRAM instructions

Fig 1. In/Near Memory Computing Architecture

(a) Data Transfer: 32-bit load/store conventional memory access
(b) C-SRAM Instructions: 64-bit STORE instruction encoding C-SRAM operation & operands
(c) Program Code: load/store, CPU scalar compute, C-SRAM vector compute, branch, etc

Fig 2. C-SRAM 64-bit Instruction Formats

- The communication protocol that defines the interaction between the C-SRAM and the host processor, and manages the transfer of C-SRAM instructions, helps to integrate the C-SRAM circuit into an existing system without changing all its architecture
- The host processor runs the control part of the program, while the C-SRAM runs the main workload.
- We propose to use the ‘STORE’ instruction to represent a C-SRAM instruction, where the C-SRAM operands are encoded inside the registers of the ‘STORE’ instruction

Fig 3. C-SRAM Architecture

- Compute where data is located
  - Vector computing
  - Low-latency memory
- Rethink micro-architecture and compilation
  - Specific instructions
  - Compile-time dynamic allocation
- Two design methodologies
  - Full custom solution
    - Change the memory bitcell
    - Specific in- and near-memory functions
  - Automated solution
    - Based on SRAM cut generated from SRAM compilers
    - Near memory functions only
    - Better Time-to-market

Fig 4. C-SRAM System Communication Protocol

- To support the proposed programming model, we have developed a software stack:
  - A QEMU-based emulator
  - A RISC-V C cross-compiler and debugger
  - A source-to-source compiler to generate C-SRAM instructions

Fig 5. C-SRAM generated instruction