Smart Instruction Codes For In-Memory Computing
ArchitecturesCompatible With Standard Sram Interfaces
Henri-Pierre Charles, Maha Kooli, Clément Touzet, Bastien Giraud,
Jean-Philippe Noel

To cite this version:
Henri-Pierre Charles, Maha Kooli, Clément Touzet, Bastien Giraud, Jean-Philippe Noel. Smart Instruction Codes For In-Memory Computing Architectures Compatible With Standard Sram Interfaces. 2018. cea-01757665

HAL Id: cea-01757665
https://hal-cea.archives-ouvertes.fr/cea-01757665
Submitted on 3 Apr 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers. L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
SMART INSTRUCTION CODES FOR IN-MEMORY COMPUTING ARCHITECTURES COMPATIBLE WITH STANDARD SRAM INTERFACES

Maha Kooli, Henri-Pierre Charles, Clément Touzet, Bastien Giraud, Jean-Philippe Noel
Univ. Grenoble Alpes, CEA, LETI/LIST, FRANCE

DATE’18
Dresden, Germany, March 22nd, 2018
BREAK THE MEMORY WALL! BUT HOW...?

"memory wall" or "funnel effect"...

...is nowadays the main limitation for **high performance computing**
Bottleneck lies in the memory hierarchy

Memory access is still a bottleneck, even in GPUs…

Let's do multi-core processors!

Seems a good idea!

Source: Barcelona Supercomputing Center
Source: nVidia
**GPU computing model (SPMD)** need to:
- Copy/transfer data
- Group parallel instructions

**IMPACT computing model:**
- No copy / transfer
- Fine grain parallel scalar interleaving
BRING THE COMPUTATION INTO MEMORY

When data start to look like motorists in the *traffic jam* during the rush hour…
(data@memory ↔ data@comp_unit)

...it’s time to consider *teleworking*, in other word the *in-memory computing*...
VON NEUMANN ARCHITECTURE

**Von Neumann Model:**
- Data & instruction in the same memory
- i.e. instructions are data
- SoC or PCB

**Memory INSN: ld r1 = @r2**
- 1 memory access (for the instruction)
- 1 instruction cycle (Decode + RF + memory access)

**Compute INSN: add r1 = r2 + r3**
- Compute instruction
- 1 memory access (for the instruction)
- 1 computation (Decode + RF + ALU)
The largest part of power consumption of logic and arithmetic operations is due to the memory access!

- The way to perform basic operations has to be restudied
- A lot of applications should be improved in performance
# DIFFERENT APPROACHES

<table>
<thead>
<tr>
<th>Process</th>
<th>Technique</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Embedded Memories</strong>&lt;br&gt;(CMOS Process)</td>
<td><strong>In-Memory Computing</strong></td>
<td>- Additional logic in memory&lt;br&gt;- Non-destructive computing&lt;br&gt;- Non Volatile/Volatile Memories&lt;br&gt;[Akyel’16] [Aga’17] [Kooli’17]</td>
</tr>
<tr>
<td></td>
<td><strong>Logic-in-Memory</strong></td>
<td>- Non-volatile Memories (ReRAM, …)&lt;br&gt;- Destructive computing&lt;br&gt;[Matsunaga’09]</td>
</tr>
<tr>
<td><strong>Stand-alone Memories</strong>&lt;br&gt;(DRAM Process)</td>
<td><strong>Processing-in-Memory</strong>&lt;br&gt;(Off-chip Memory)</td>
<td>- Planar / 3D process&lt;br&gt;- Non-destructive computing&lt;br&gt;[Gokhale’95] [Pugsley’14] UpMem</td>
</tr>
</tbody>
</table>
OUTLINE

• Introduction & Context

• In-Memory Power Aware CompuTing (IMPACT)

• IMPACT Memory Instruction Code

• IMPACT Communication Protocol

• Conclusion & Perspectives
In-Memory Power Aware CompuTing (IMPACT)
• **Computing in dedicated units:**
  • High data transfer between the ALU & the memory
    ➢ Power hungry
    ➢ Interconnect & memory security issues

• **In-memory computing:**
  • Reduced data transfer
    ➢ Energy-efficient
    ➢ Execution time acceleration
    ➢ Security reinforcements (limitation of the side channel attacks (on buses))
IMPACT MEMORY

SRAM architecture

<table>
<thead>
<tr>
<th>Row decoder</th>
<th>IN/OUT DFF/LATCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL</td>
<td>IO</td>
</tr>
<tr>
<td>SRAM bitcell array</td>
<td></td>
</tr>
</tbody>
</table>

IMPACT Memory

<table>
<thead>
<tr>
<th>multi-row selector (&gt;2)</th>
<th>IN/OUT DFF/LATCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL</td>
<td>ALU-like</td>
</tr>
<tr>
<td>SRAM bitcell array</td>
<td></td>
</tr>
</tbody>
</table>

- Enable in-memory operations
  - Reduce latency & energy consumption due to data transfer
- SRAM bit cell array allows:
  - Long word arithmetic/logic operations not limited by register size, but with memory line size
  - Multi-row selection for some logic operations
  - Simultaneous storing in different addresses

Emulation Platform:

- Algorithm
- C-code + intrinsic
- LLVM IR
- Execution Trace
- Analysis
- Performance

Emulate the IMPACT system features
- Long word operations
- Multi-operand operations

LLVM
- Early design stage of the system: Not defined ISA
- Manipulate arithmetic/logic operations on large vectors

Target Applications
- Image Processing (Motion Detection)
- Cryptography (One Time Pad)

Experimental Gains
- Execution time: up to 6145x
- Energy: up to 12.9x
IMPACT Memory Instruction Code
IMPACT ROADMAP

- Initial idea: put logic operation in bitcells - done
- Added idea: add parallel arithmetic in I/O - done
- Create an high level emulation platform (based on LLVM) - done
- New idea: create an « inverted Von Neuman » protocol (aka ISA) - (this presentation) done
- Tape out april 2018 - on going
- Create a more accurate emulation platform - on going
- Create compilation toolbox - on going
- Evaluate high level benchmarks - on going
- ../..
## IMPACT OPERATIONS AKA OPCODES

<table>
<thead>
<tr>
<th>Logic &amp; Memory Operation</th>
<th>Memory</th>
<th>Shift</th>
<th>Logic</th>
<th>Arithmetic Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Not</td>
<td>Set</td>
<td>Reset</td>
<td>Additon</td>
</tr>
<tr>
<td></td>
<td>Set</td>
<td>Reset</td>
<td></td>
<td>Subtration</td>
</tr>
<tr>
<td></td>
<td>Reset</td>
<td></td>
<td></td>
<td>Increment</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Decrement</td>
</tr>
<tr>
<td>Shift</td>
<td></td>
<td>Shift Left</td>
<td></td>
<td>Comparison</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shift Right</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Xor</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Nxor</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>And</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Or</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Nor</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Nand</td>
<td></td>
</tr>
<tr>
<td>Logic</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Memory line size word
- 8 bits words
- 16 bits words
- 32 bits words
- 64 bits words

### Bit positions
- MSB
- LSB

**Notes:**
- More than two input operands
- Maximum two input operands

**Impact:**
- Logic & Memory Operations
- Shift Operations
- Logic Operations
- Arithmetic Operations
• Multi-operand operations (logic/memory operations)
• *Problematic*: encoding all the operand addresses in the instruction requires large bus size
  ➢ Propose a novel concept based on pattern construction

**IMPACT Instruction**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Pattern Code</th>
<th>Address</th>
<th>Mask</th>
<th>SP</th>
<th>Output</th>
<th>SI</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>0110</td>
<td>1100</td>
<td>1</td>
<td>1</td>
<td>1111</td>
<td>1</td>
</tr>
</tbody>
</table>

**IMPACT Memory**

Row Selector | Pattern Register | SRAM Array (N-columns, 16-rows)

@: 1100
Mask: 0110

[Diagram of IMPACT Memory with detailed representation of the row selector, pattern register, and SRAM array]
The proposed method allows:

- Building regular patterns
- Patterns can be refined by adding/deleting a specific line
- Patterns can be stored in the pattern register for future use
- Selecting multiple lines in the SRAM array to perform the multi-operand operation
The conventional format of instruction with maximum two source addresses
Long-word operations (logic/arithmetic operations)

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Address 1</th>
<th>Address 2</th>
<th>SP</th>
<th>Output</th>
<th>SI</th>
</tr>
</thead>
<tbody>
<tr>
<td>add, sub, …</td>
<td>@ of 1st &amp; 2nd operand</td>
<td>@ where the result is stored</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A **select pattern** bit to enable/disable the pattern construction using the row selector

A **smart instruction** bit:
- 0: if conventional instr.
- 1: if IMPACT instr.
Communicate in-memory instructions via **data & address** busses of a conventional system

- Compatible with **existing** system **architecture** (*conventional system bus*)
- Enable **interleaving** the **CPU** & the **in-memory** instruction execution

**DATE’18 | Henri-Pierre Charles | 22/03/2018 | 21**
1. Address the SRAM in conventional mode
2. Address the IMPACT memory for read/write
3. Address the IMPACT memory for computation

In-Memory Computing System

Data Bus

<table>
<thead>
<tr>
<th>Opcode</th>
<th>@1</th>
<th>@2</th>
<th>SP</th>
<th>32-bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-bits</td>
<td>12-bits</td>
<td>12-bits</td>
<td>1-bit</td>
<td></td>
</tr>
</tbody>
</table>

Address Bus

<table>
<thead>
<tr>
<th>MSB</th>
<th>INPACT</th>
<th>Output</th>
<th>SRAM</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI=1</td>
<td>@IMPACT</td>
<td>@Output</td>
<td>@SRAM</td>
<td>@Output</td>
</tr>
<tr>
<td>1-bit</td>
<td>M-bits</td>
<td>12-bits</td>
<td>N-bits</td>
<td>P-bits</td>
</tr>
<tr>
<td>[16-bits]</td>
<td>[16-bits]</td>
<td>[16-bits]</td>
<td>[16-bits]</td>
<td>[16-bits]</td>
</tr>
</tbody>
</table>

Instruction/Data Memories

SRAM

- 4k words x 2^M*32-bits

CPU clock
1. Interleave CPU & IMC instruction execution
   • Perform massive data computation inside IMC, and not optimized computation in CPU
     • For image qqVGA 160x120 (not pipelined):
       • Execution Time Speed-Up: 1376x
       • Energy Reduction Factor: 29x
   
2. Perform all the computation inside IMC
   • For image qqVGA 160x120 (not pipelined):
     • Execution Time Speed-Up: 765x
     • Energy Reduction Factor: 29x
**COMMUNICATION PROTOCOL**
**INSTRUCTION SET ARCHITECTURE**

**Source Code**

... 
R = s1 + s2 
...

**Assembly Code**

... 
mv r2, @R 
mv r1, #add 
shl r1, #6 
xor r1, #@s1 
shl r1, #6 
xor r1, #@s2 
store r1, r2 
...

**IMC Instruction:**

```
add @s1 @s2 @R
```

Do not change the actual ISA
- Overhead preparation

Architecture scenario w/o ISA modification
Conclusion & Perspectives
CONCLUSION & PERSPECTIVES

• Propose a new communication protocol between the CPU and the IMPACT memory
  ➢ Compatible with existing system architecture (conventional system bus)
  ➢ Enable interleaving the CPU & the in-memory instruction execution

• Work on the compiler:
  ➢ Generate the assembly code respecting the communication protocol
  ➢ Interleave the IMC & CPU instruction execution
    • Based on the performance evaluation
  ➢ Optimize the data set-up in the memory
    • Data alignment in the IMC
    • Data interleaving