NVRAM: New Opportunities for Compilers

Erven Rohou
Workshop “Raised Challenges by NVRAM”
Paris, May 29 2017
Agenda

- NVRAMs
- Who cares?
- Related work
- ANR CONTINUUM
- Thoughts
## NVRAM characteristics

<table>
<thead>
<tr>
<th>Feature</th>
<th>SRAM</th>
<th>DRAM</th>
<th>Disk</th>
<th>Flash</th>
<th>STT-MRAM</th>
<th>PCM</th>
<th>ReRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell size</td>
<td>&gt; 100F²</td>
<td>6 – 8F²</td>
<td>2/3F²</td>
<td>4 – 5F²</td>
<td>37F²</td>
<td>8 – 16F²</td>
<td>&gt; 5F²</td>
</tr>
<tr>
<td>Read latency</td>
<td>&lt; 10 ns</td>
<td>10-60 ns</td>
<td>8.5 ns</td>
<td>25µs</td>
<td>&lt; 10 ns</td>
<td>48 ns</td>
<td>&lt; 10 ns</td>
</tr>
<tr>
<td>Write latency</td>
<td>&lt; 10 ns</td>
<td>10-60 ns</td>
<td>9.5 ns</td>
<td>200µs</td>
<td>12.5 ns</td>
<td>40-150 ns</td>
<td>10 ns</td>
</tr>
<tr>
<td>Energy per bit access</td>
<td>&gt; 1 pJ</td>
<td>2 pJ</td>
<td>100-1,000 mJ</td>
<td>10 nJ</td>
<td>2 pJ</td>
<td>100 pJ</td>
<td>0.02 pJ</td>
</tr>
<tr>
<td>Leakage power</td>
<td>High</td>
<td>Medium</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Endurance</td>
<td>&gt; 10¹⁵</td>
<td>&gt; 10¹⁵</td>
<td>&gt; 10¹⁵</td>
<td>&gt; 10⁴</td>
<td>&gt; 10¹⁵</td>
<td>10⁵-10⁹</td>
<td>10⁵-10¹¹</td>
</tr>
<tr>
<td>Nonvolatility</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Scalability</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Who cares? (compiler’s point of view)

- Compilers optimize memory accesses anyway
- Different latencies
  - still ok, usually parametric

- What’s new?
  - asymmetric latencies and energy per access
  - retention time
  - endurance
Related work

• Data allocation
  – Hybrid caches
  – Scratch pad memories

• Reducing write activity
• Hybrid SRAM/STTRAM cache
  – e.g. 4-way assoc: 1 way SRAM, 3 ways STTRAM
• Migration depending on access pattern
  – causes overhead
• Instruction scheduling minimize R/W and W/R
  – group loads and stores
  – minimize transitions in a block
• Hybrid SRAM/STTRAM cache
  – e.g. 4-way assoc: 1 way SRAM, 3 ways STTRAM
• Migration depending on access pattern
  – causes overhead
• Change data layout to minimize R/W and W/R
  – group mostly written and mostly read data
  – minimize transitions in a block
- Hybrid SRAM/STTRAM cache
- Identify migration-intensive blocks
  - assign to SRAM part of cache
- Data allocation to concentrate write accesses
  - eliminate migrations
  - prefer writes on SRAM
- NVM scratchpad memory
- static allocation of data to minimize “cost”
  - on-chip SRAM, on-chip NVM, off-chip DRAM
- takes into account live ranges
- Two solutions
  - ILP
  - graph coloring
• Hybrid SRAM/PCM

• Code regions (procedure, loop)
  – Cost model to define best layout for given region

• Dynamic data management at region boundaries
  – special CPU instructions
• CMP + Scratchpad + NV main memory
• Scheduling of “tasks” to avoid evictions to NVM
• Recompute when cheaper
• Graph-based algorithm
CONTINUUM

• Goal: design energy-efficient compute nodes based on emerging paradigms

• 42 months, starting October 2015

• Coordinator: Abdoulaye Gamatié (LIRMM)

• http://www.lirmm.fr/continuum-project
CONTINUUM: challenge

Which multicore architectures?

Which task and data exploitation?

Which memory & comm. technology?

E. Rohou - NVRAM: New Opportunities for Compilers
CONTINUUM
Energy-Efficient Compute Nodes

• Considering multi-level energy optimization
  – innovative design, from technology to architecture
    • non volatile memories (NVMs)
    • power-efficient core technology
    • heterogeneous design
  – software techniques applicable to diverse platforms
    • dedicated compilation techniques
    • runtime/adaptive management
Latencies: nanoseconds vs. cycles

Word of caution
Silent-Stores to the Rescue

• Definition
  – a store is “silent” if it writes the value that is already present in memory

• Studied in the early 2000’s
  – hardware solution
  – for uniprocessor speedup, multiprocessor bus traffic

Silent-store elimination in software

- Identify likely silent stores
  - profiling
- Modify code at IR level
- Standard code generation

```
load y = @val
cmp val, y
beq next
store @x = val
next:
```
Profitability Threshold

- Replace a store by a load and possibly a store
- Beneficial iff

\[ \alpha_{\text{read}} + (1 - P_{\text{silent}})\alpha_{\text{write}} \leq \alpha_{\text{write}} \]

\[ P_{\text{silent}} \geq \frac{\alpha_{\text{read}}}{\alpha_{\text{write}}} \]

```
volatile int x;
for(i=0; i<N; i++){
    y=0;
    if (i>M)
        y=i;
    x=y; // store
}
```

- Must also consider additional instructions
## Motivational Kernel

L1 instruction: 25%
L1 data: 20%

<table>
<thead>
<tr>
<th></th>
<th>Exec. time (ms)</th>
<th>Energy (mJ)</th>
<th>EDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>full-SRAM</td>
<td>305.13</td>
<td>79.5</td>
<td>24257.83</td>
</tr>
<tr>
<td>full-STTRAM</td>
<td>305.31 (+0.06%)</td>
<td>67.8 (-14.7%)</td>
<td>20700.01</td>
</tr>
<tr>
<td>full-STTRAM + silent store opt</td>
<td>305.31 (+0.06%)</td>
<td>64.6 (-18.7%)</td>
<td>19723.02</td>
</tr>
</tbody>
</table>

```c
volatile int x = 0;
for (i=0; i<N;i++) {
    x=0; // silent
}
```

```assembly
; i in r3, N in r4
.L3
    str r2, [r1, #0]
    add r3, r3, #1 ; i++
    cmp r3, r4 ; i==N?
    bNE .L3
```

```c
volatile int x = 0;
for (i=0; i<N;i++) {
    x=0; // silent
}
```
Intermediate Results

- Rodinia benchmarks
  - your mileage may vary
Impact of architecture and compiler

- Additional instruction have a cost
  - more time $\Rightarrow$ more leakage
- Hardware
  - superscalar
  - out-of-order
  - predication
- Compiler
  - scheduling
  - “optimize” new instructions

```c
load y = @val
cmp val, y
beq next
store @x = val
next:
```
Example: instruction predication

- ARM supports predication
- LLVM back-end automatically takes advantage
  - fewer instructions
  - fewer branch mispredictions

```assembly
store @x = val
load y = @val
cmp val, y
strNE @x = val
```
Future Work

• IoT and energy-harvesting devices (*instant on-off*)
  – cannot shutdown anywhere
  – compiler
    • assesses minimum energy to reach specific points
    • inserts safe points

• Optimize for endurance?

• Re-visit “standard” optimizations?
Thank you for your attention!