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

Feature	SRAM	DRAM	Disk	Flash	STT-MRAM	PCM	ReRAM
Cell size	$> 100F^2$	$6 - 8F^2$	$2/3F^2$	$4 - 5F^2$	$37F^2$	$8 - 16F^2$	$> 5F^2$
Read latency	< 10 ns	10-60 ns	8.5 ms	25 μs	< 10 ns	48 ns	< 10 ns
Write latency	< 10 ns	10-60 ns	9.5 ms	200 μs	12.5 ns	40-150 ns	10 ns
Energy per bit access	> 1 pJ	2 pJ	100-1,000 mJ	10 nJ	2 pJ	100 pJ	0.02 pJ
Leakage power	High	Medium	High	Low	Low	Low	Low
Endurance	$> 10^{15}$	$> 10^{15}$	$> 10^{15}$	10^4	$> 10^{15}$	10^5-10^9	10^5-10^{11}
Nonvolatility	No	No	Yes	Yes	Yes	Yes	Yes
Scalability	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Zhao, Jishen, et al. "Memory and storage system design with nonvolatile memory technologies." IPSJ Transactions on System LSI Design Methodology 8 (2015): 2-11.

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

Code motion for migration minimization in STT-RAM based hybrid cache
Qingan Li, Liang Shi, Jianhua Li, Chun Jason Xue and Yanxiang He
ISVLSI'12

- 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

MAC: migration-aware compilation for STT-RAM based hybrid cache in embedded systems
Qingan Li, Jianhua Li, Liang Shi, Chun Jason Xue, Yanxiang He
ISLPED'12

- 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

Compiler-assisted preferred caching for embedded systems with STT-RAM based hybrid cache
Qingan Li, Mengying Zhao, Chun Jason Xue, Yanxiang He
LCTES'12

- 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

MGC: Multiple graph-coloring for non-volatile memory based hybrid Scratchpad Memory
Qingan Li, Yingchao Zhao, Jingtong Hu, Chun Jason Xue, Edwin Sha, Yanxiang He
INTERACT 2012

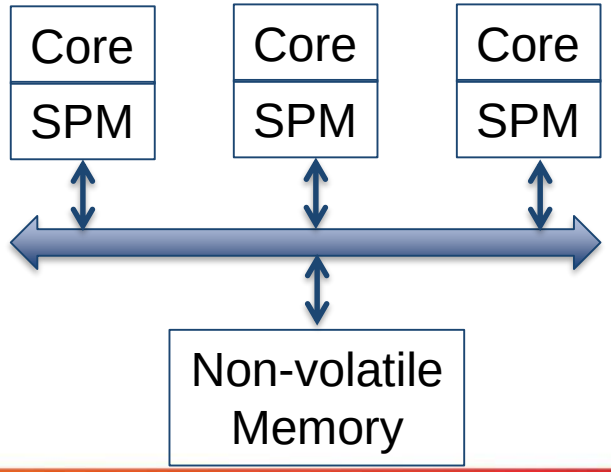
- NVM scratchpad memory
- static allocation of data to minimize “cost”
  - on-chip SRAM, on-chip NVM, off-chip DRAM
- takes into account live ranges
- Two solutions
  - ILP
  - graph coloring

Data Allocation Optimization for Hybrid Scratch Pad Memory With SRAM and Nonvolatile Memory
J. Hu, C. J. Xue, Q. Zhuge, W. C. Tseng and E. H. M. Sha
IEEE VLSI 2013

- 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

Reducing Write Activities on Non-volatile Memories in Embedded CMPs
via Data Migration and Recomputation
Jingtong Hu, Chun Jason Xue, Wei-Che Tseng, Yi He, Meikang Qiu, and Edwin H.-M. Sha
DAC'2010

- CMP + Scratchpad + NV main memory
- Scheduling of "tasks" to avoid evictions to NVM
- Recompute when cheaper
- Graph-based algorithm



CONTINUUM

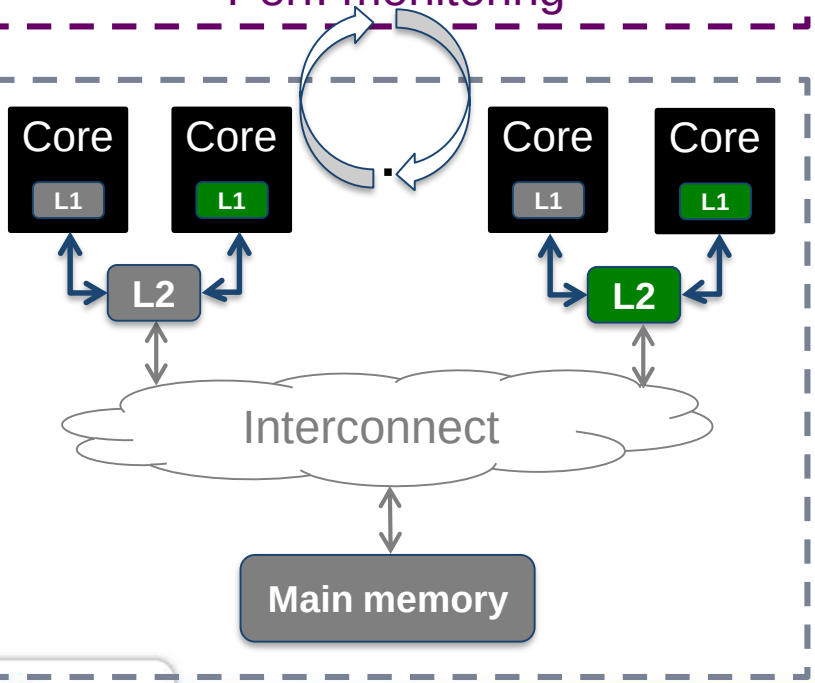
- Goal: design energy-efficient compute nodes based on emerging paradigms

- 42 months, starting October 2015
- Coordinator: Abdoulaye Gamatié (LIRMM)
- `http://www.lirmm.fr/continuum-project`

CONTINUUM: challenge

```
omp_set_num_threads(4);
#pragma omp parallel for shared(a,b,c) private(j)
for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    c[i] += a[i][j] * b[j];
/* output omitted */
```

OS / Compiler / DBT
Perf. monitoring



Which task and data exploitation?

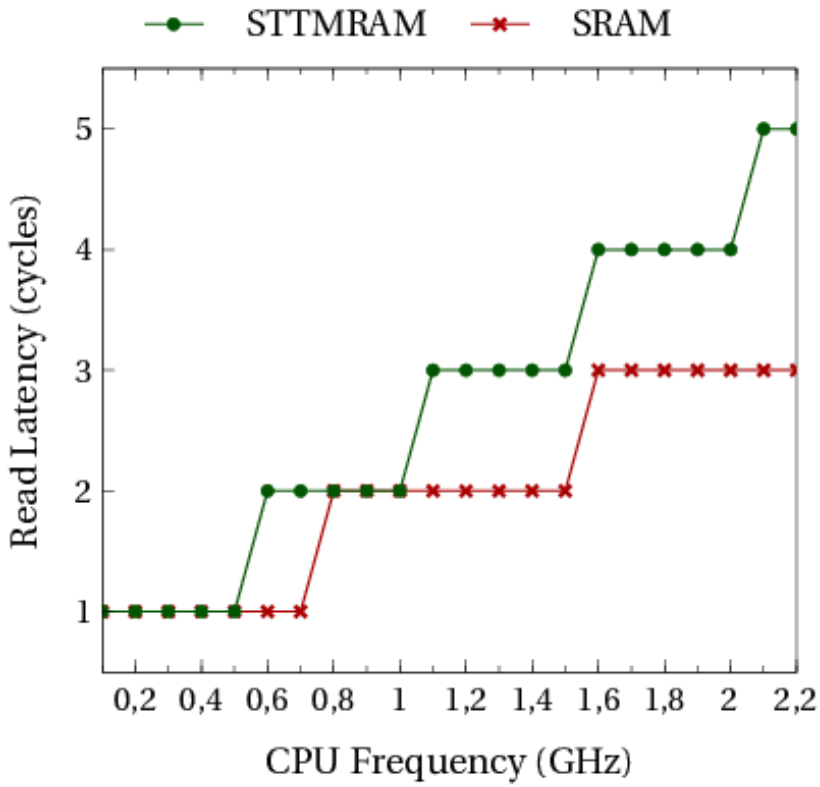
Which multicore architectures?

Which memory & comm. technology?

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

Kevin M Lepak, Gordon B Bell, and Mikko H Lipasti. “Silent stores and store value locality”. In: IEEE TC 50.11 (2001).

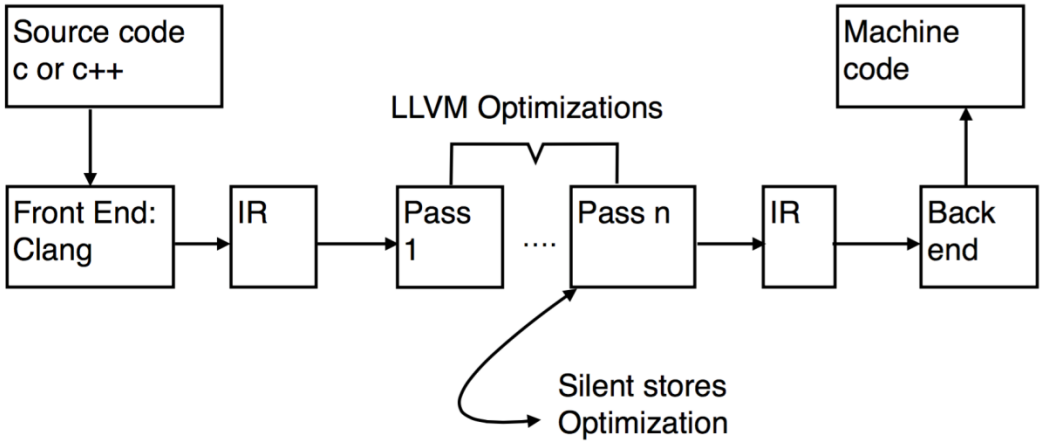
  - hardware solution
  - for uniprocessor speedup, multiprocessor bus traffic

Silent-store elimination in software

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

store @x = val


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



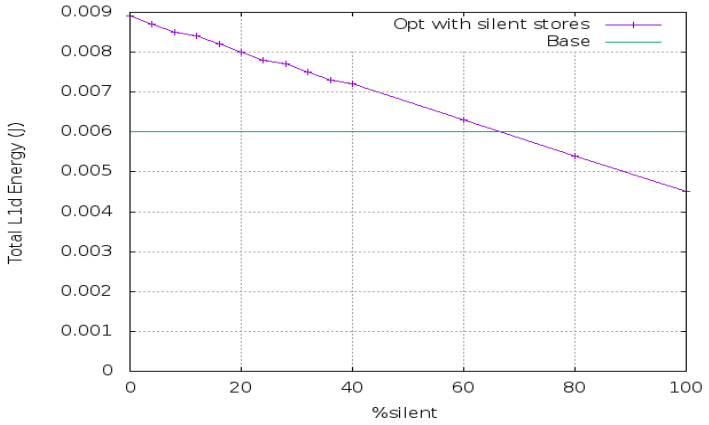
Profitability Threshold

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

$$\alpha_{read} + (1 - P_{silent})\alpha_{write} \leq \alpha_{write}$$

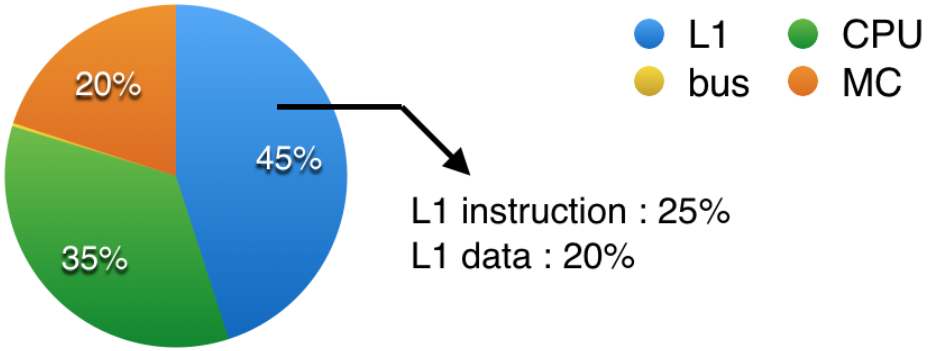
$$P_{silent} \geq \frac{\alpha_{read}}{\alpha_{write}}$$

```
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



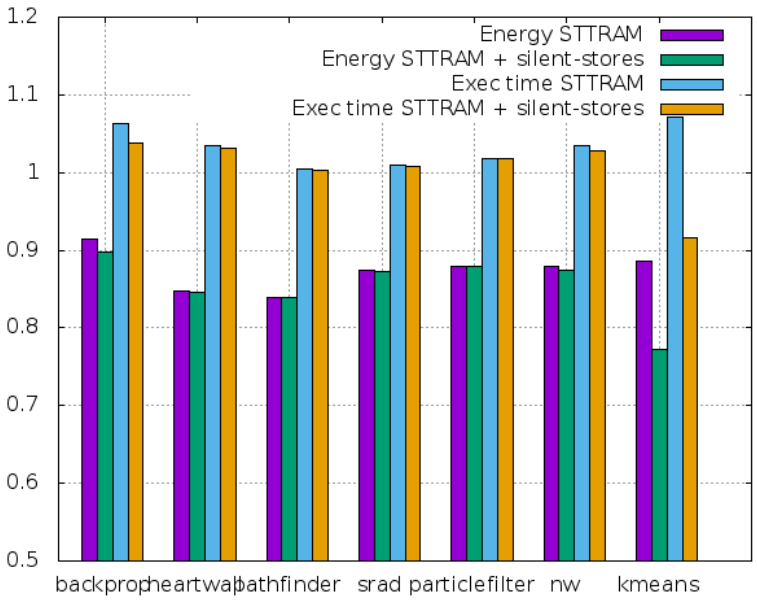

```
volatile int x = 0;
for (i=0; i<N;i++) {
  x=0; // silent
}
```

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

	Exec. time (ms)	Energy (mJ)	EDP
full-SRAM	305.13	79.5	24257.83
full-STTRAM	305.31 (+0.06%)	67.8 (-14.7%)	20700.01
full-STTRAM + silent store opt	305.31 (+0.06%)	64.6 (-18.7%)	19723.02

Intermediate Results

- Rodinia benchmarks
  - your mileage may vary



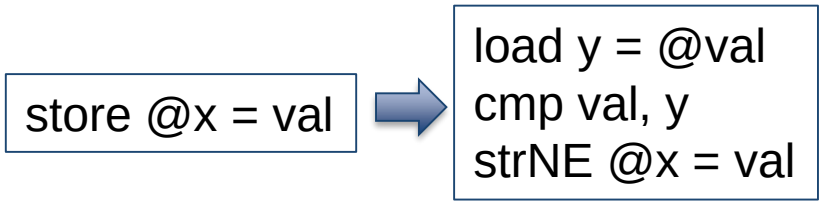
Impact of architecture and compiler

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

```
   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



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!