Virtualization (Task Merging) Design & Implementation

-

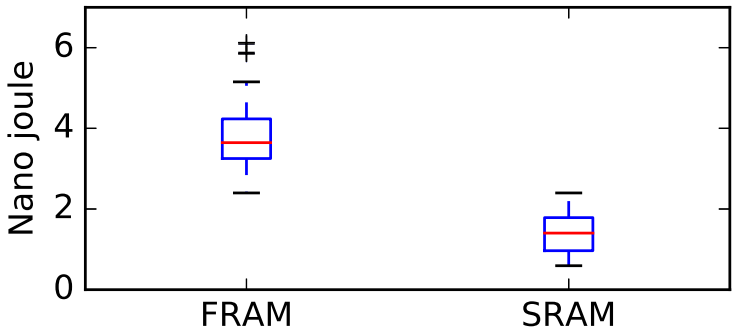
Embedded Software Group – Delft University of Technology, The Netherlands

8 February 2017

Virtualization

FRAM Write

- Almost twice more expensive than SRAM write.

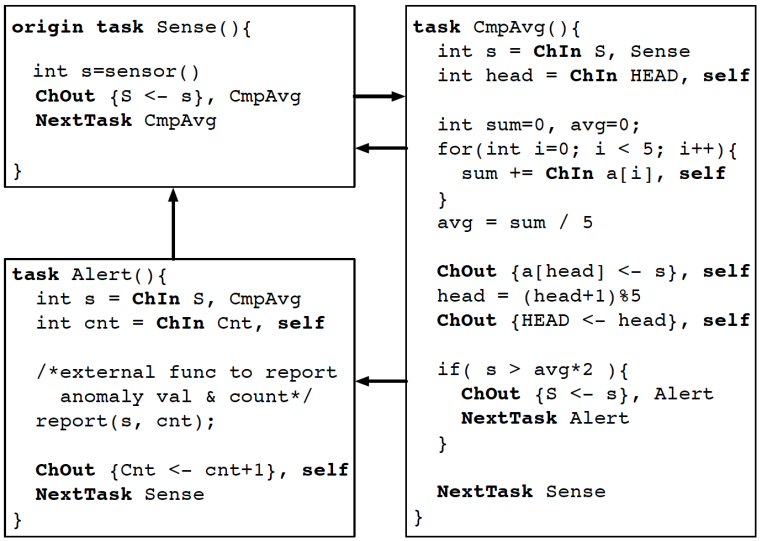


Problem Statement

- How to **postpone** commit (FRAM access) operations to reduce energy overhead?
  - The gained energy will be used for **more computation**
  - E.g. more tasks will be executed in task-based programs — in turn, the total program execution time will be reduced
    - Portability: different super-capacitor sizes will be used efficiently

Task-based Control Flow

- e.g. Chain[1]
  - no checkpointing – keep track of the current task
  - pipelined channels – maintain the consistency of FRAM
  - seperate inputs from outputs – idempotency



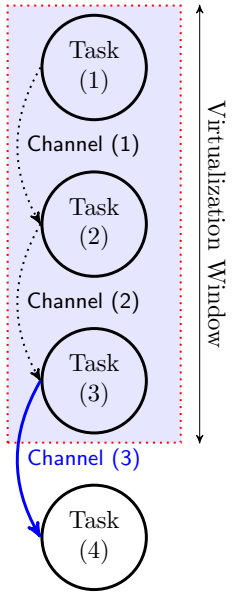
Drawbacks

- outputs are committed **at the end** of each task
  - **Commit** operation, in turn FRAM access (in particular write), is **expensive**
- **multi-versioning** of variables — memory overhead
  - increased number of FRAM operations

[1] **colin2016chain**

Details - Virtualization of Chain -I

- If all output channels are connected to and the transition path is only to the **next task**, virtualization is **easy**!
- No need to commit `Channel (1)` and `Channel (2)` since their values are already consumed to output `Channel (3)`.

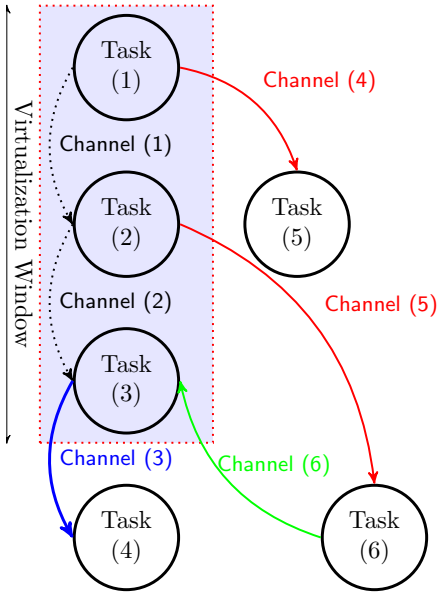


Strategy

- Postpone commit inside the **virtualization window** using corresponding **soft-channels** in SRAM
- **Hard-commit** only the channel input of the last task
- If power-interrupted, go to the first task of the window

Details - Virtualization of Chain -II

- What if the channel connections and transition paths are **divergent**?
  - `Channel (3)`, `Channel (4)` and `Channel (5)` **must** committed to FRAM —**hard-channels**
  - `Channel (1)` and `Channel (2)` are in SRAM—**soft-channels**
  - `Channel (6)` should be read from FRAM—hard-channel



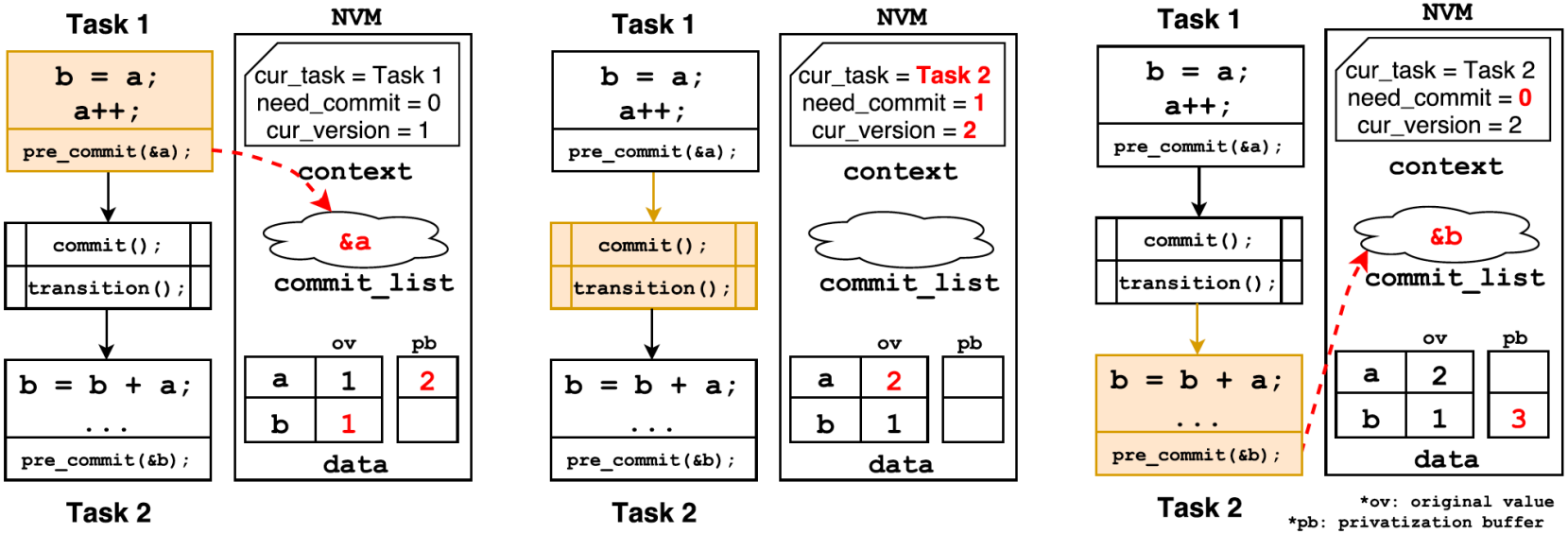
Drawbacks

- The static **task-graph** and the **dynamic** task transition path should be used at **run-time** to decide which channels need to be hard or soft

Not Feasible

- Due to loops in task graph, also multi-versioning ruins virtualization.

Case Study - Virtualization of ALPACA -I



(a) Executing task 1 (b) Committing task 1 (c) Executing task 2

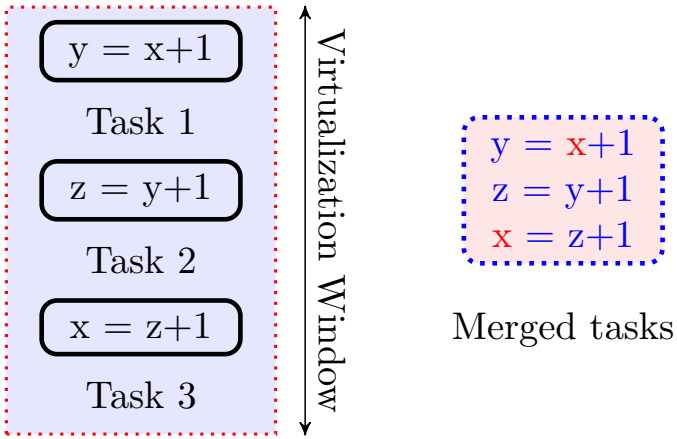
Only variables with W-A-R dependencies are protected in ALPACA.

Challanges

- Variables without W-A-R dependencies are directly read from/written to FRAM
- **Solution:** Use **SRAM buffer** and commit later—Reduced FRAM access

Case Study - Virtualization of ALPACA -II

- Merged tasks might introduce new W-A-R dependencies —at **run-time**!
- How to **dynamically** keep-track of the privitization of the W-A-R dependencies? —**too much overhead!**
  - Alpaca does not privatize x, y and z; they are directly written to FRAM.
  - We need software implementation of Clank :)

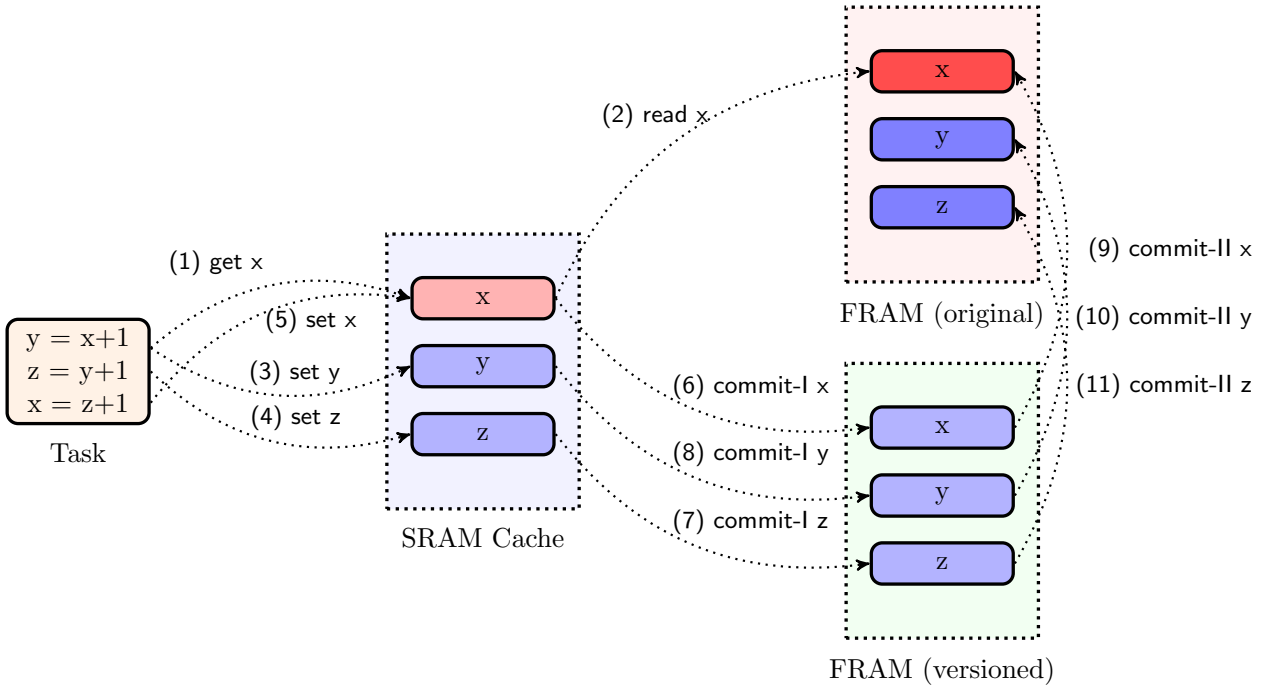


Conclusions

- We need to **eliminate** privatization so that tasks can operate on SRAM efficiently.

Virtualization with SRAM Buffer-I

- Version **task-shared** variables in FRAM
- A **fixed-sized** SRAM buffer for virtual operations
- Two-phase commit at the end of the **virtual task**.

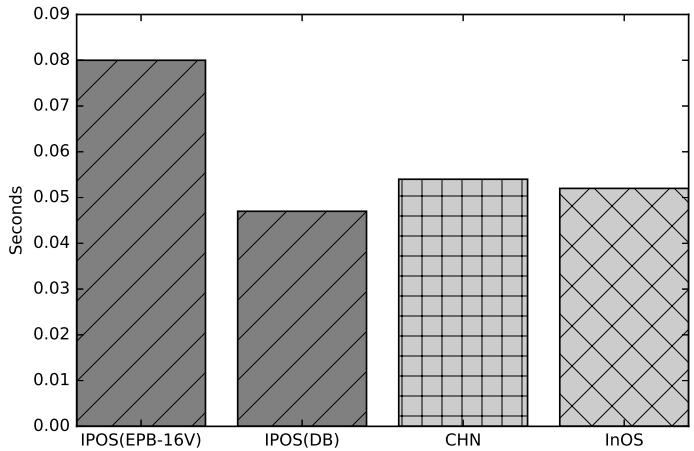


Overhead

- `get` and `set` operations require **search** in SRAM cache
- Alpaca will have 4 FRAM write operations - We will have 6 + SRAM search overhead!

Virtualization with SRAM Buffers-II

- We did not observe any benefit, even worse!

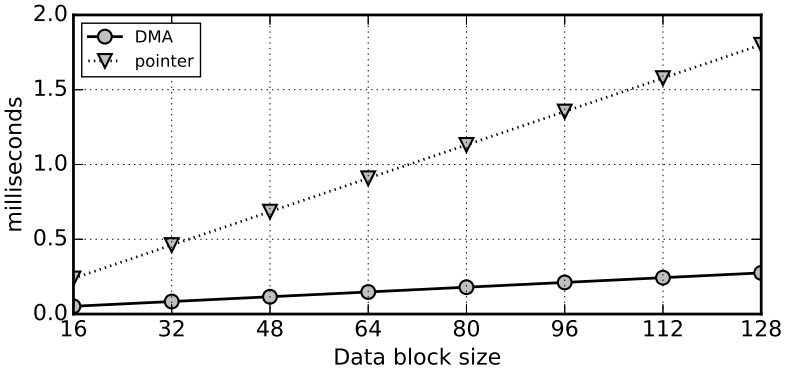


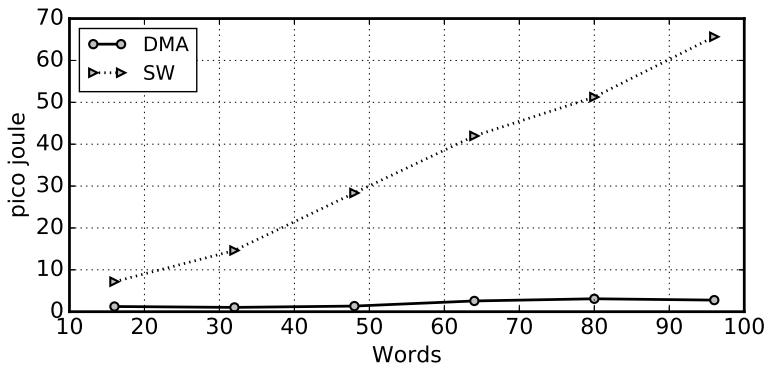
Problem

- How to eliminate SRAM Cache search operation?
- How to minimize commit overhead?

Virtualization with DMA Support-I

- Exploit hardware that is found in almost every CPU architecture
- Copying block of words from/to SRAM-FRAM and FRAM-FRAM is very **cheap** and **scalable** with DMA!



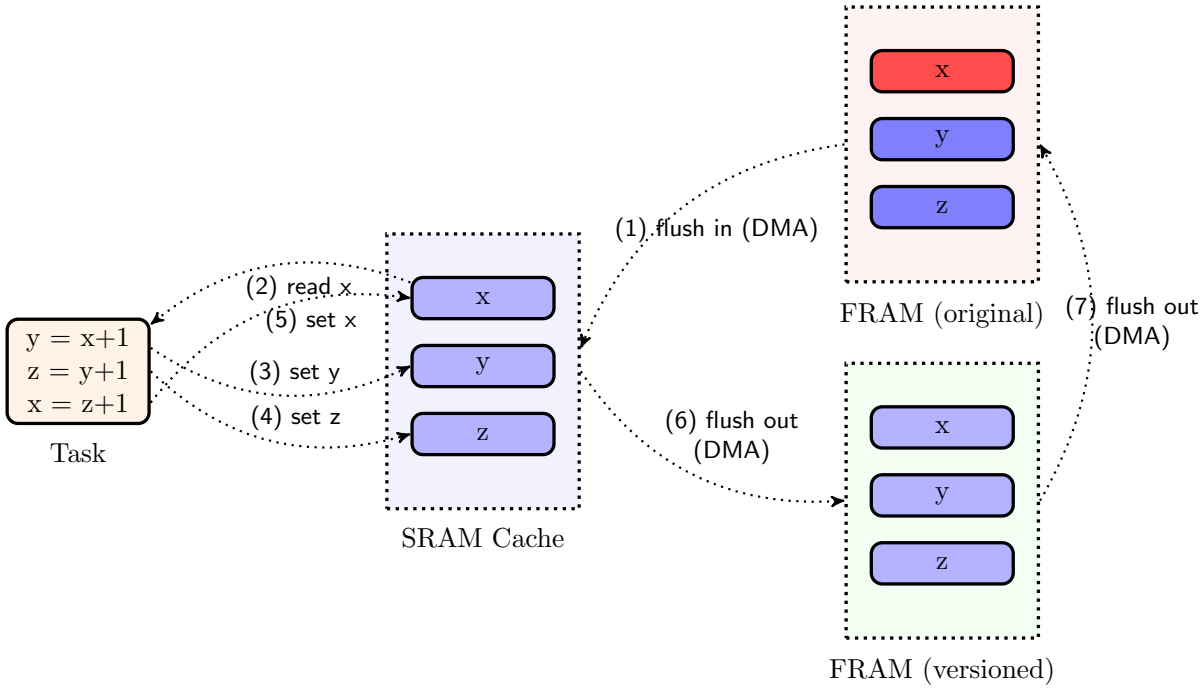


Problem

- How to eliminate SRAM Cache search operation?
- How to minimize commit overhead?

Virtualization with DMA Support-II

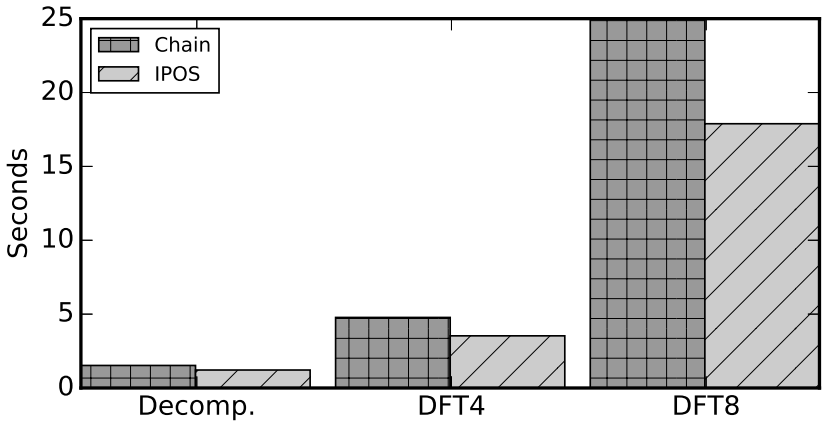
- Version all **task-shared** variables in FRAM
- Bring all to SRAM buffer for virtual operations
- Two-phase commit at the end of the **virtual task**.

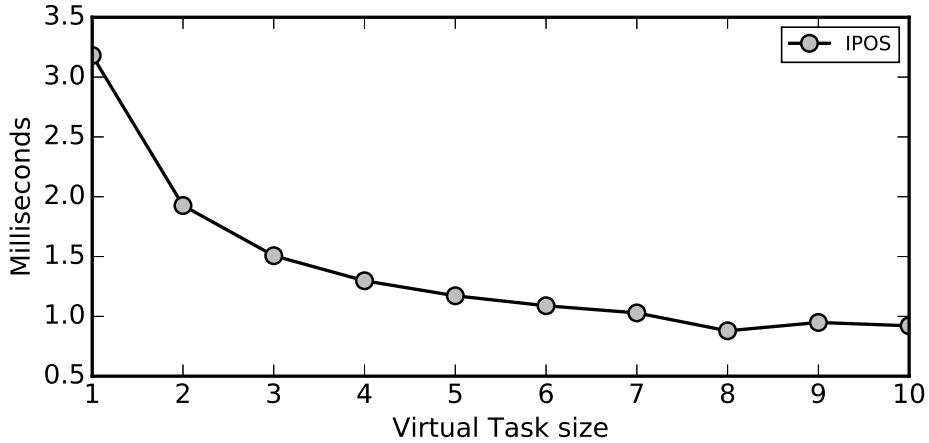


Feasibility

- Perform all these operations using DMA!

Virtualization with DMA Support-III





Virtualization with DMA Support-IV

- We have some limitations:
  - The size of the task-shared variables, in turn the size of SRAM Cache is bounded with the **SRAM size** in the platform.
    - The number of channels is directly limited with the FRAM size in Chain
    - The size of the commit list in Alpaca, i.e. the number of variables with W-A-R dependency inside a task, is limited with the FRAM.
  - The cost of start-up and commit operations increase with **the size of the task-shared variables**
    - But this comes with the elimination of the search operations in SRAM cache and selective commit
    - DMA cost increases linearly but very slowly.
- all these limitations are paid in order to benefit from virtualization.

What is next?

- How to get rid of flush in/out all task-shared global variables?
- Development and evaluation of several task-merging algorithms, i.e. **virtualization scheduler**

The structure of the article-I

- **Introduction**
  - Motivation—FRAM access, in turn commit cost, need for the virtualization
  - Problem statement
  - Challenges—Why it is difficult to modify state of the art solutions to support virtualization?
  - Our solution and contributions—Sketch of our solution and attractive results
- **Related Work and Challenges**
  - Intermittent programming with checkpointing and task-based control flow
  - Stress the problem of dynamic merging of the tasks, its feasibility with the state of the art solutions, e.g. Chain, InOS, Alpaca...
- **Virtualization: Dynamic Merging of Tasks**
  - Introduce the concept, requirements
  - Design objectives
  - Introduce the infrastructure here

The structure of the article-II

- **Virtualization Scheduler: The Algorithms**
  - Introduce the task-merging strategies that work on the infrastructre in the previous section
  - Analysis of the algorithms
  - Maybe some simulations
- **Implementation and Evaluation**
  - Introduce the implementation details
  - Test-bed setup
  - Experimental results
  - Simulations
- **Conclusions**