Memory-Aware Custom Processor Architecture Design and Implementation for Streaming Applications

Giulio Stramondo

25 September 2019

Outline

Internship Goal?

- Analyze tradeoff of non volatile memories.
- Optimise non volatile memory systems for broadband applications.

Research Question

Can emerging NV-RAM technologies replace SRAM technologies?

What are NV-RAMs anyway?

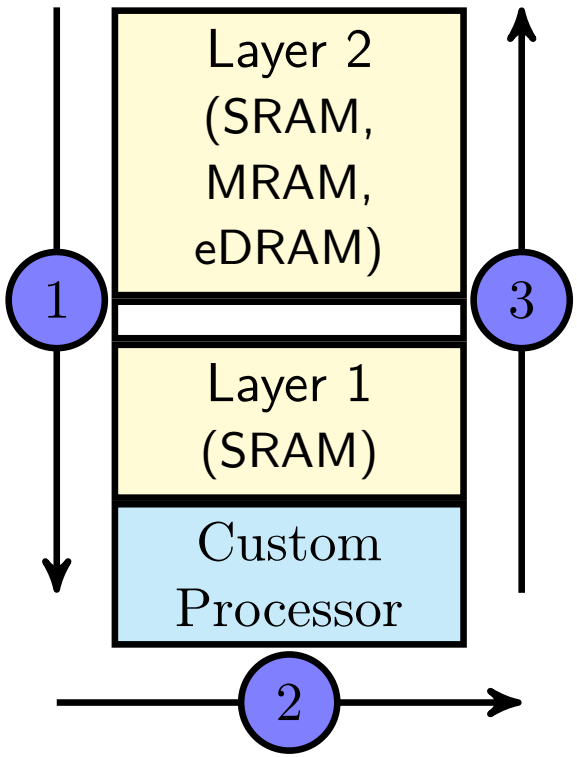
Non-Volatile Memory technologies (NV-RAM)

- Non-volatile random-access memory (NVRAM) is random-access memory that retains its information when power is turned off.
- EEPROM/FLASH is the most spread, however:
  1. write 1000 times slower than read
  2. short write-lifetimes, on the order of 10^5 writes to any particular bit.
- Emerging Non-volatile technologies try to overcome the limitation of the EEPROM/FLASH
  - Magnetoresistive RAM (MRAM) seems to be the most promising one in the short future
  - Peculiarities: different read and write latencies and power consumtion

System Under Analysis

Layer 2 larger in size (but slower in speed) than Layer 1 and fabricated on a different metal layer

1. Assume input data stored in Layer 2 (via Layer 1)
2. Transfer data and compute in the Custom Processor (via Layer 1)
3. Write back result data in Layer 2 (via Layer 1)



Prior Art vs Our Focus

Prior Art

- Generic application and generic hardware architecture
- Model NV as caches (Layer 1 or Layer 2)

Our Focus

- Start from a specified application
- Derive custom hardware
- Obtain tradeoff of different memories and process technologies

Observations

- The memory system and the processing system are interdependent therefore they should be co-designed.
- The data dependencies of the fixed application impose constraints on the design of both systems.

Ideas

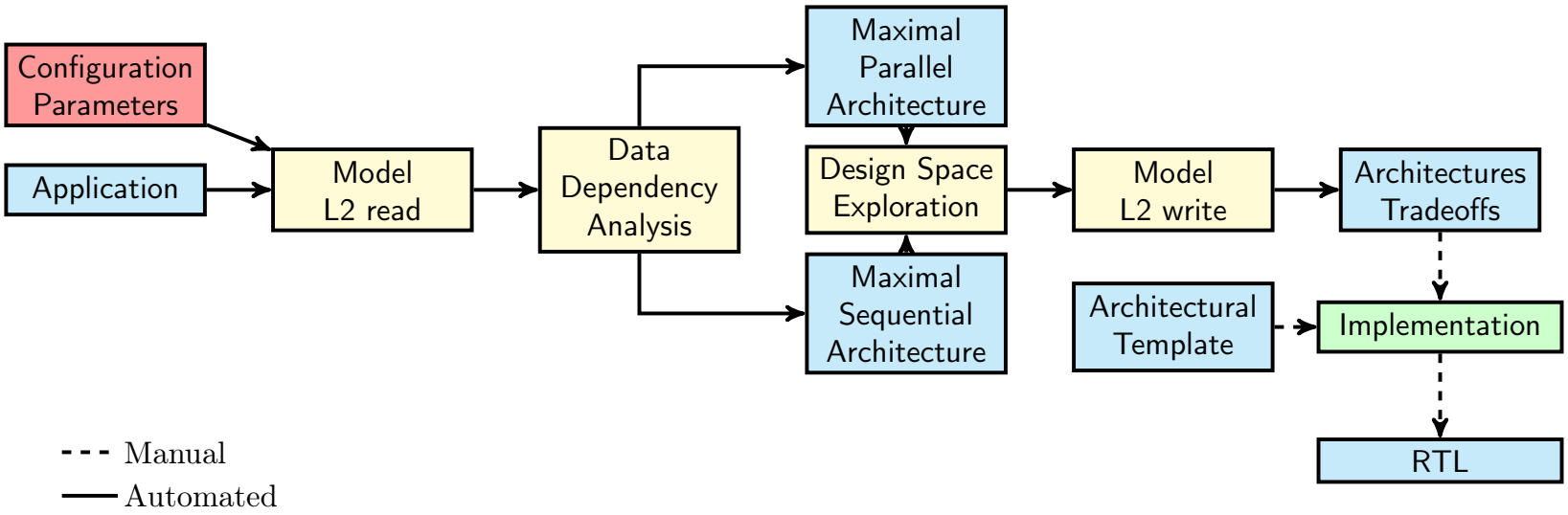
- Codesign the memory and processor systems starting from the data dependencies of an application.
- Explore suitable hardware designs with different area/latency tradeoffs.

Problem Statement

- Model different kind of memory technologies
  - MRAM, SRAM, eDRAM, etc..
- Model different layers of memory
  - Different clock speed, read/write latency, datawidth
- Analyze different architectural implementation
  - Explore area/latency tradeoffs
- Define a custom processor architecture for hardware implementation

Framework

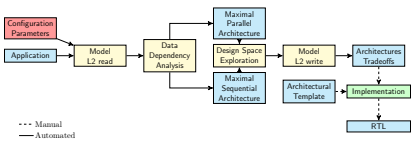
- Customize computing elements (e.g. technology, frequency, latency)
- Explore different custom architectures for a given application
- Predict area, energy, latency of possible HW architectures
- Implement custom processor architecture



Framework: Configuration Parameters

Configuration Parameters

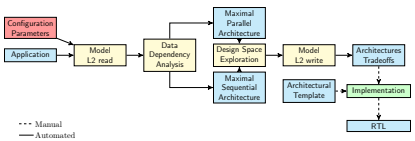
```
{"resource_database":
{ "technology": 16, "clock_frequency": 1000,
"bitwidth_adder": 128,
"bitwidth_multiplier": 64,
"bitwidth_register_file": 128,
"type_l2": "tt1v1v85c", "technology_l2": 16,
"clock_l2": 800,
"bitwidth_l2": 32, "depth_l2":2048,
"startup_write_latency_l2":2,
"startup_read_latency_l2":2 } }
```



Framework: Input Application

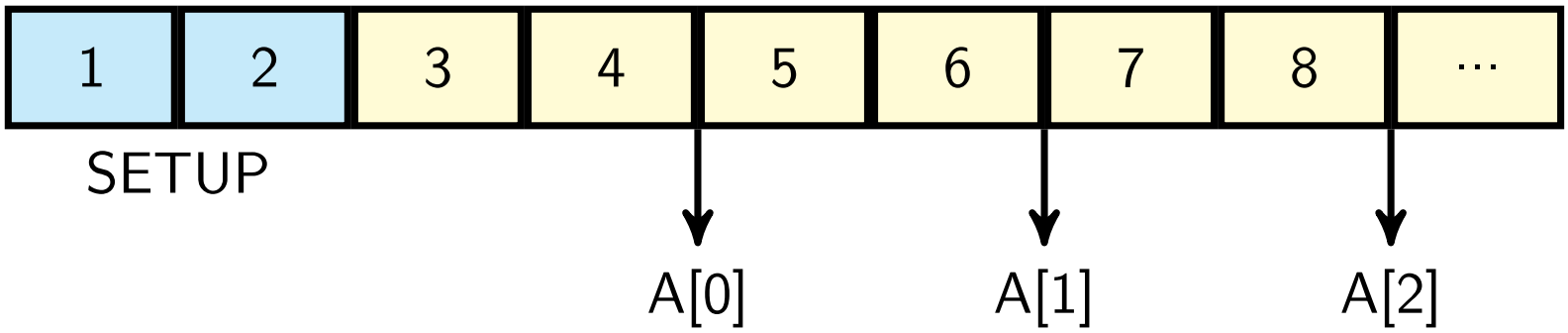
Application

```
void matrix_vec_kernel(int *A,int *B, int *C){
    int sum;
    #pragma clang loop unroll(full)
    for(int i=0;i<DIM2;i++){
        sum=0;
        #pragma clang loop unroll(full)
        for(int j=0;j<DIM1;j++){
            sum+=A[i*DIM1+j]*B[j];
        }
        C[i]=sum;
    }
}
```



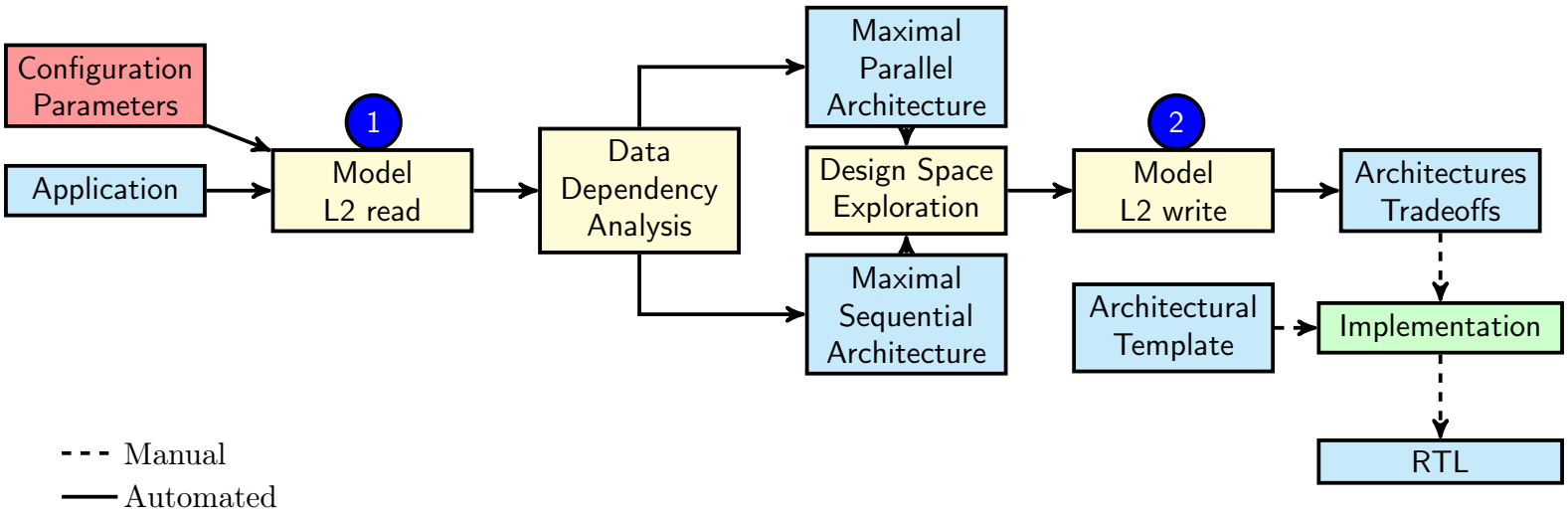
Modeling the Layer 2

- Access to L2 are performed in bursts
- All of the input data are sent in 1 burst at the beginning of the computation
- All of the output data are received in 1 burst at the end of the computation



Framework: Linking Layer 2 Model

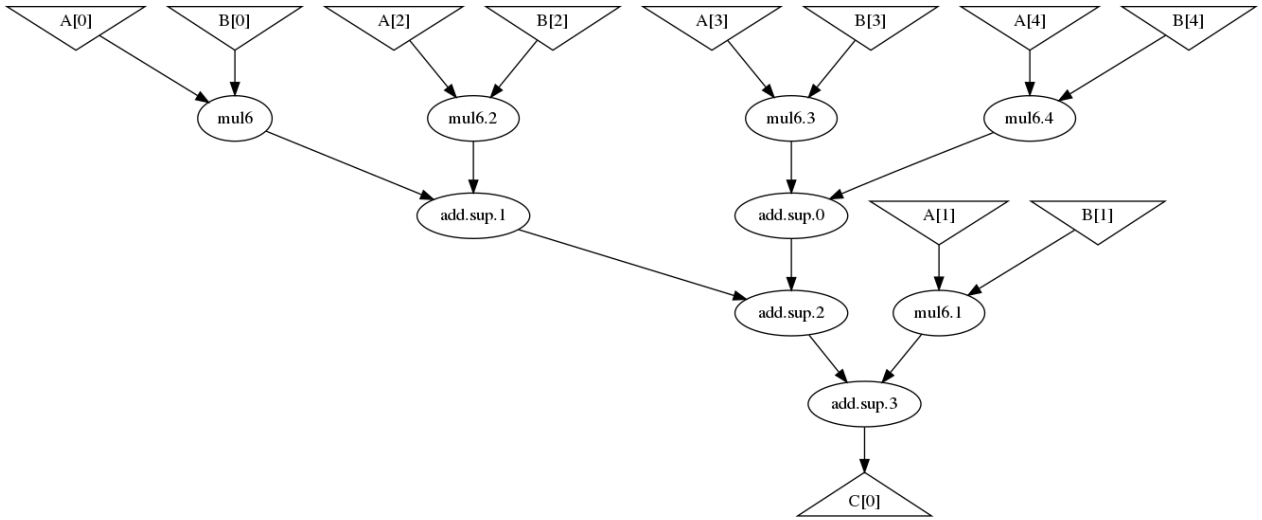
1. Before Data Dependency Analysis
   - Compute clock arrival time of elements to L1 Memory
2. After Design Space Exploration
   - Compute clock arrival time of elements to L2 Memory

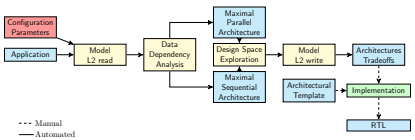


Framework: Data Dependency Analysis

1. Extract the Data Dependency Graph of the Application
2. Perform ASAP and ALAP scheduling
3. Merge the operations performed by the application in the minimum number of Functional Units

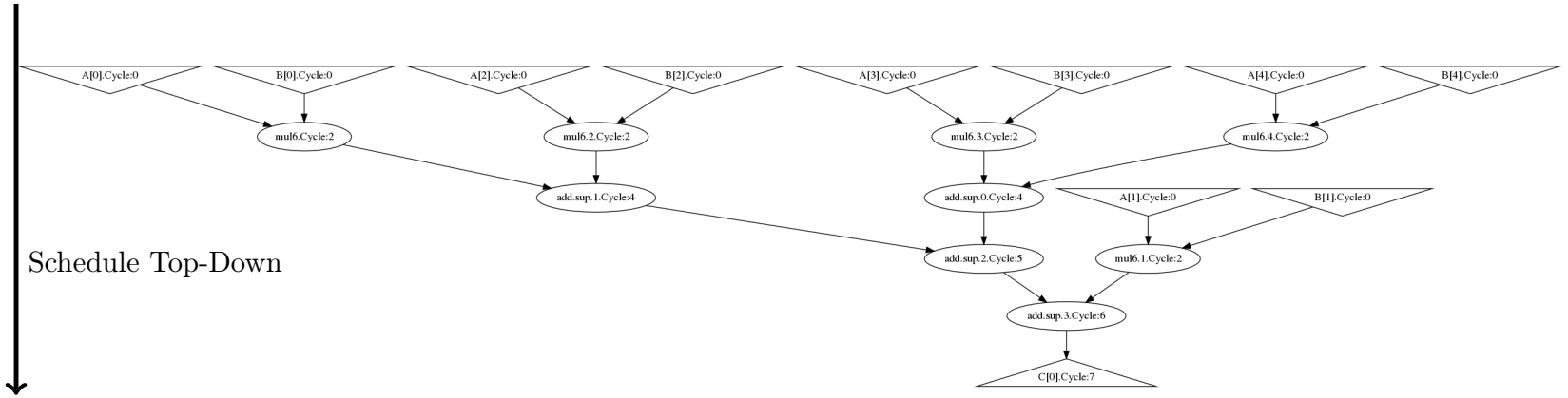
Data Dependency Analysis



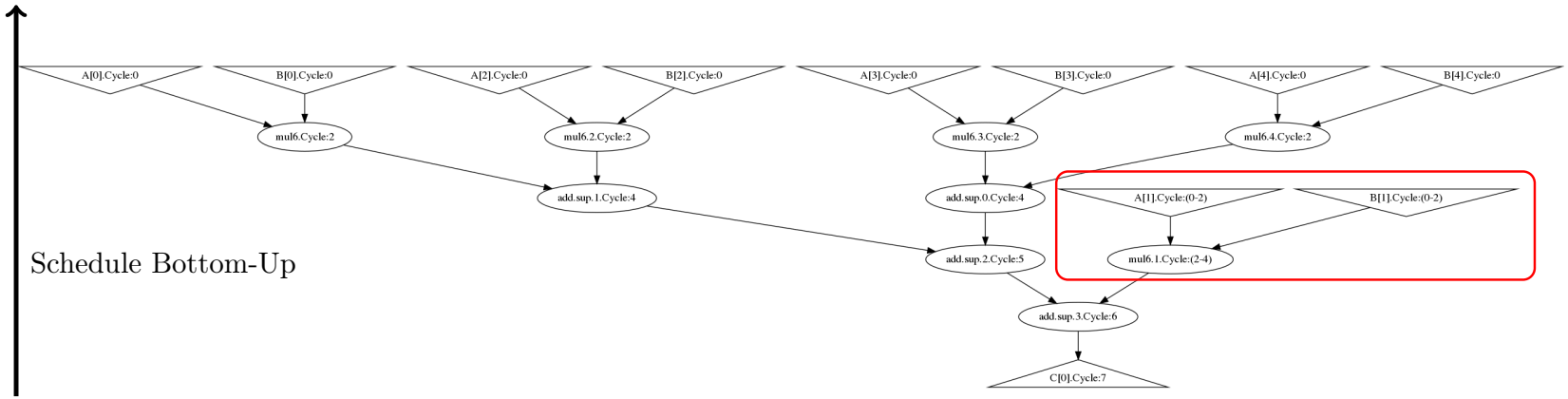


Framework: Data Dependency Analysis - 2. ASAP - ALAP

- As Soon As Possible (ASAP)



- As Late As Possible (ALAP)



Framework: Data Dependency Analysis - 3. Functional Units

- Each operation needs to be executed within its ASAP-ALAP interval
- Two operations can be computed on the same FU if their execution does not overlap

Problem

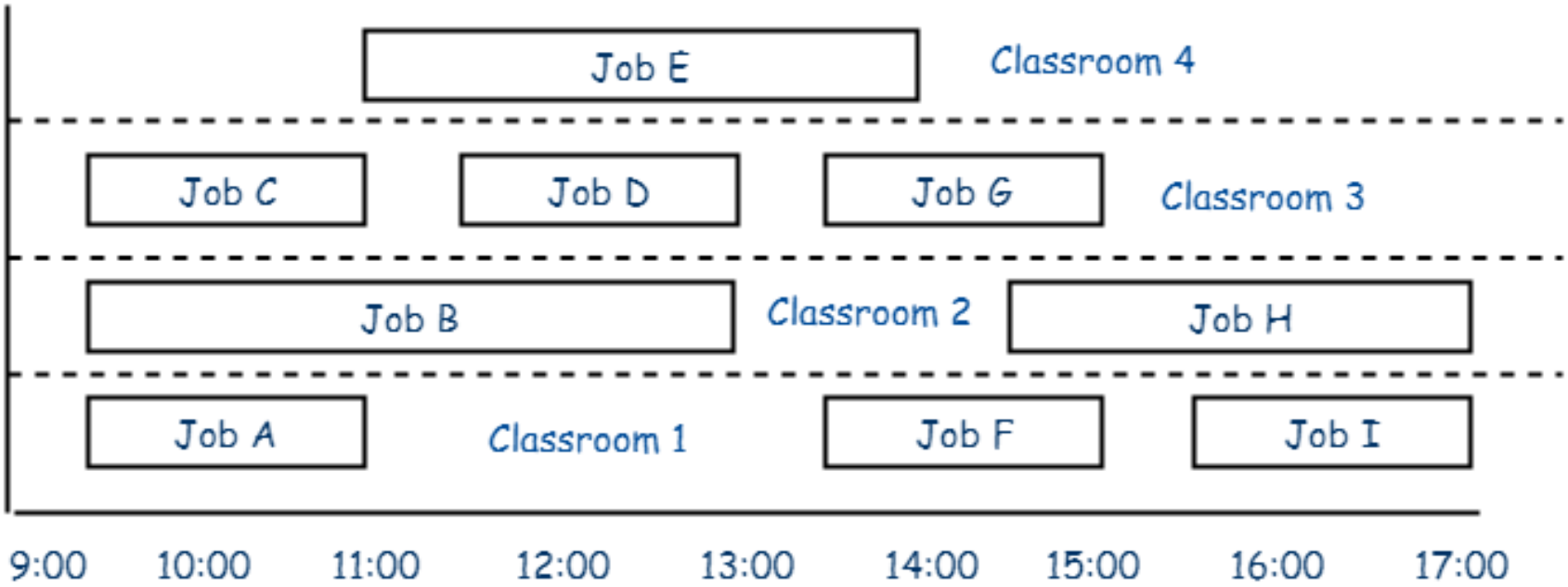
What is the minimum number of FUs such that we can compute all the operations respecting their intervals?

Solution

(Modified) Interval Partitioning Algorithm

Canonical Interval Partitioning Problem

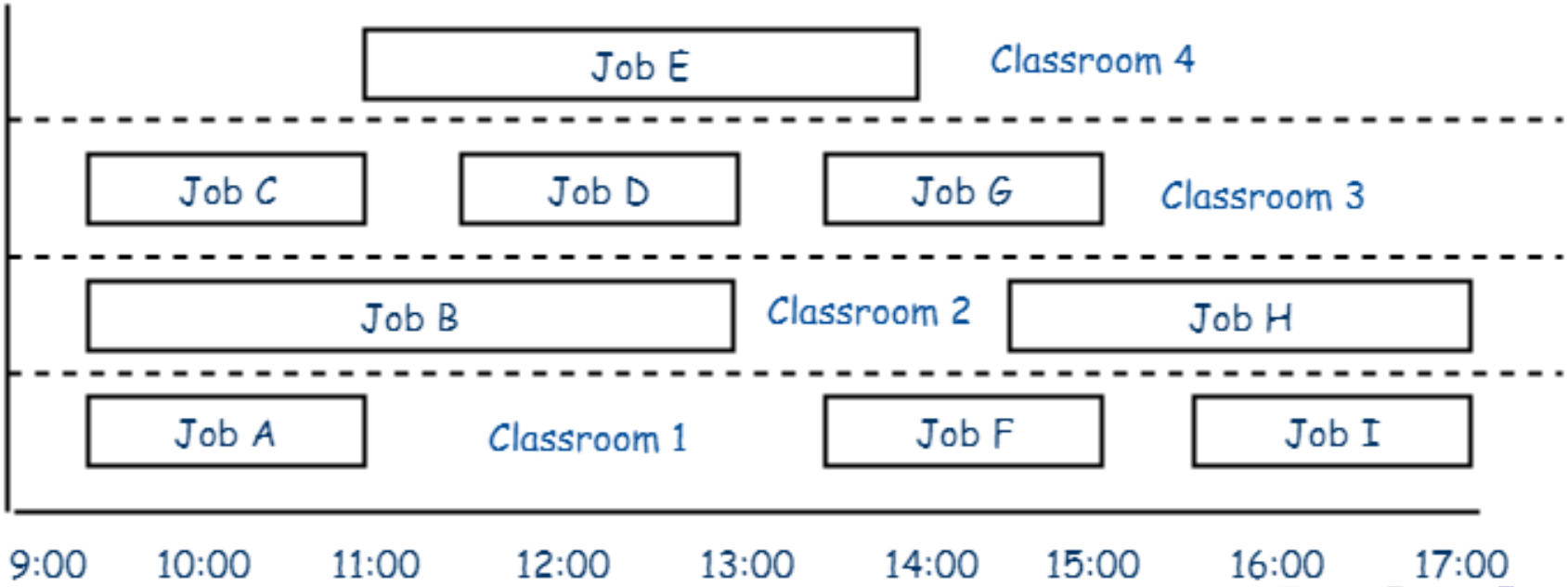
- There are n lectures to be schedules and there are certain number of classrooms.
- Each lecture has a start time S_i and finish time F_i.
- Goal is to schedule all lectures in minimum number of classes and there cannot be more than one lecture in a classroom at a given point of time.



Modified Interval Partitioning Problem

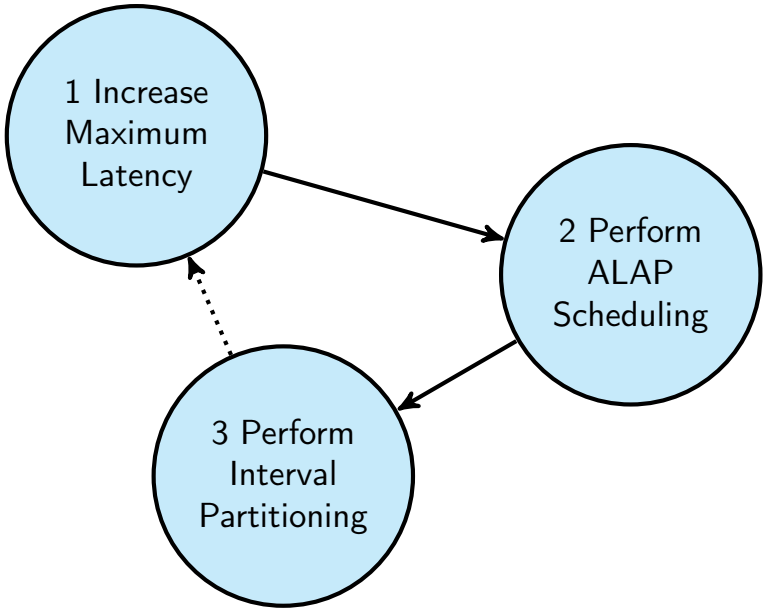
Required Modifications

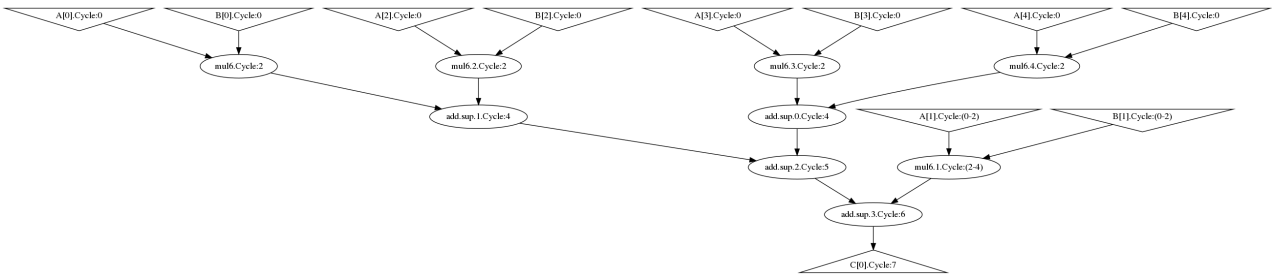
1. We perform the Interval Partitioning once per operation type
2. We give a different meaning to the intervals (related to operation binding)
3. We need to guarantee that operation dependencies are respected (related to operation compatibility)

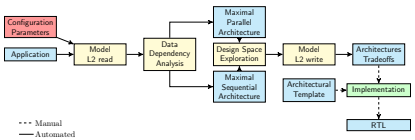


Framework: Design Space Exploration

Design Space Exploration



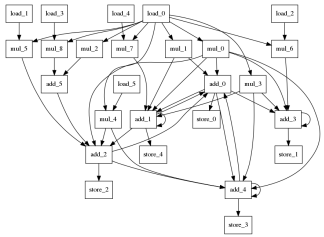


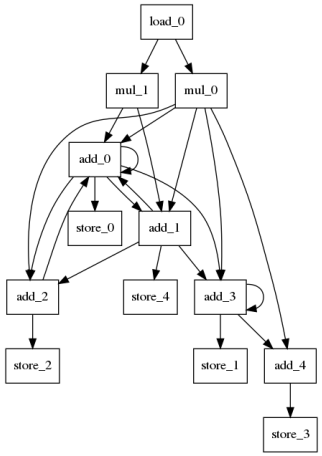


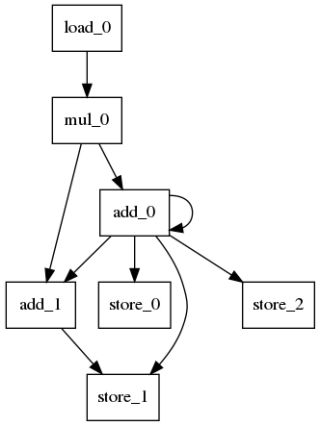
Framework: Architecture Tradeoff

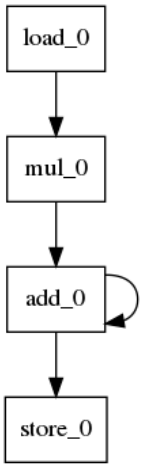
- Known (from configuration): latency, area and energy consumption of each FU
- Derived: Number, type and operation performed by all functional units
- Output: Overall area, energy and latency

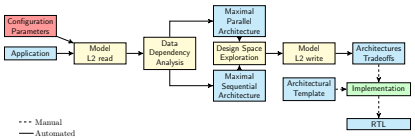
Architecture Tradeoffs





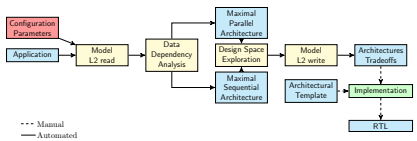




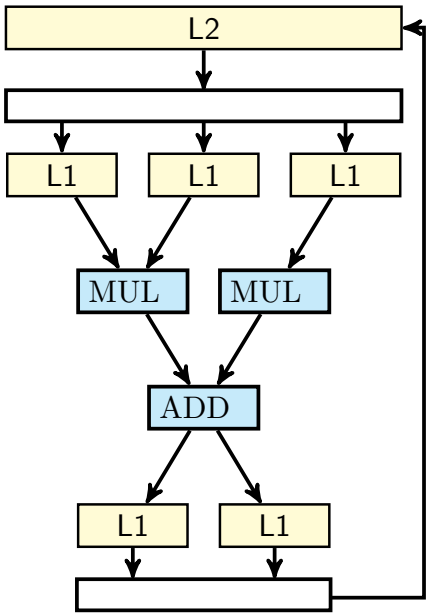


Architecture Implementation

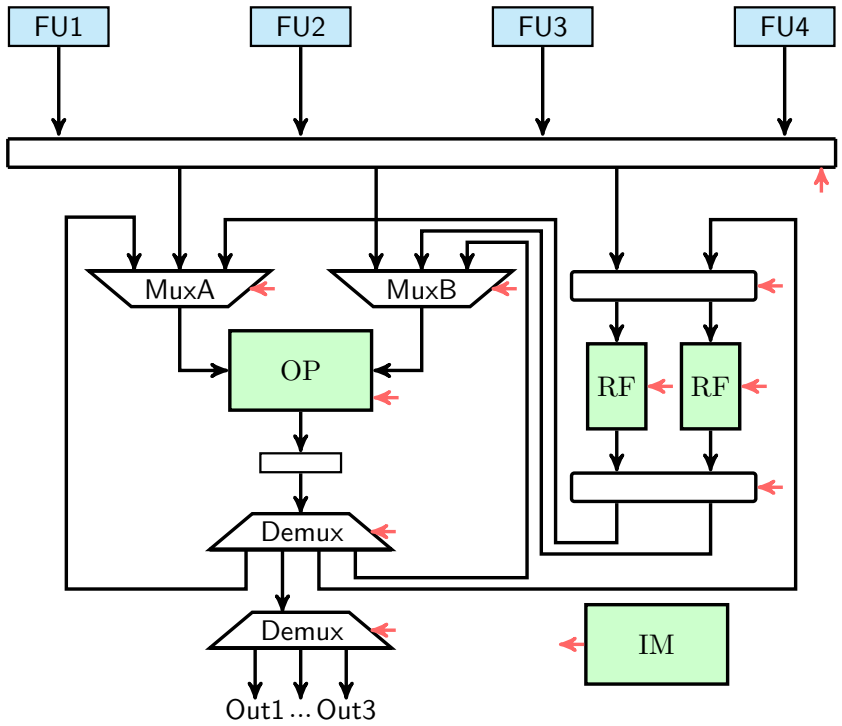
Architectural Template



Full Architecture Implementation



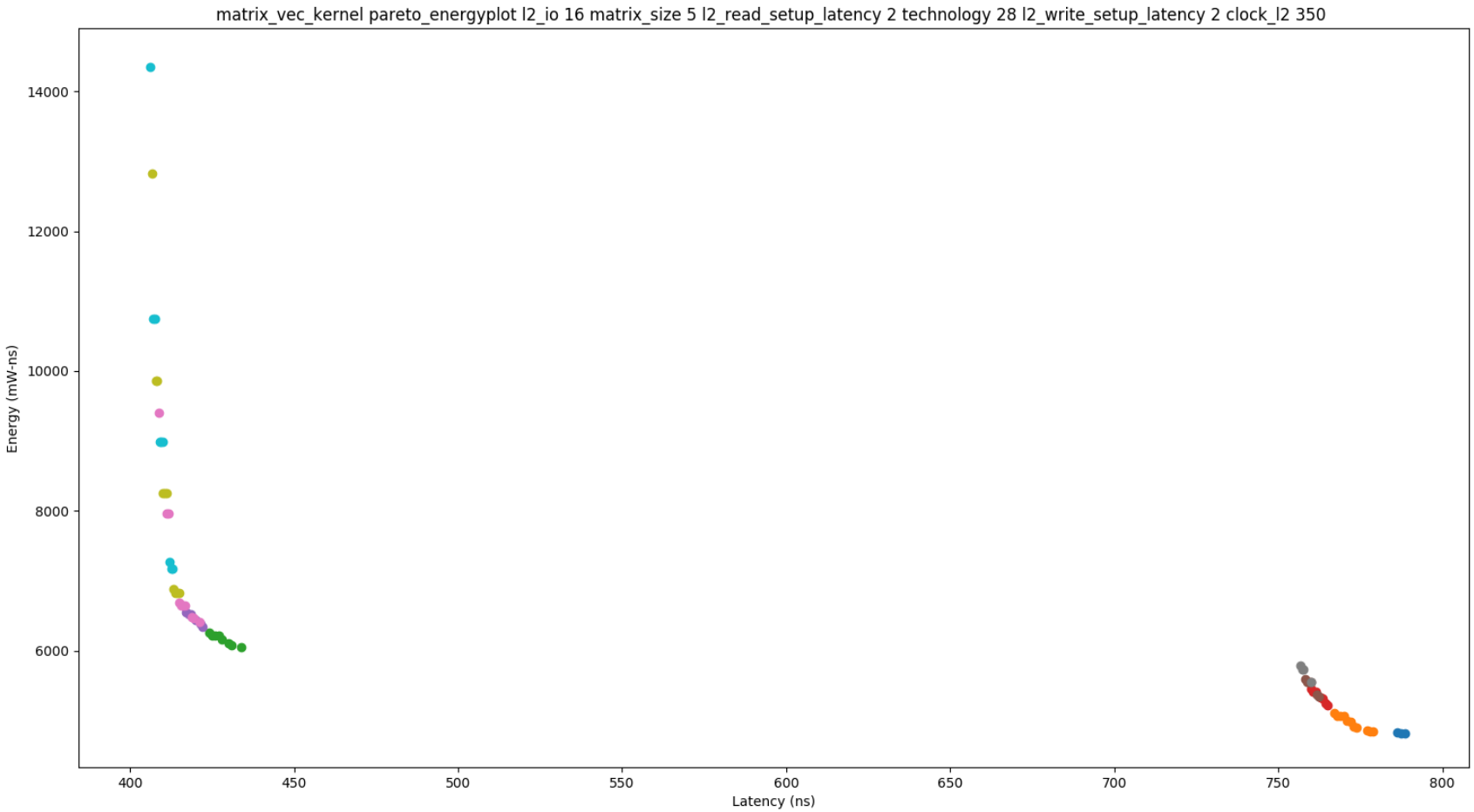
Functional Unit Template



Experiment: SRAM vs MRAM - Setup

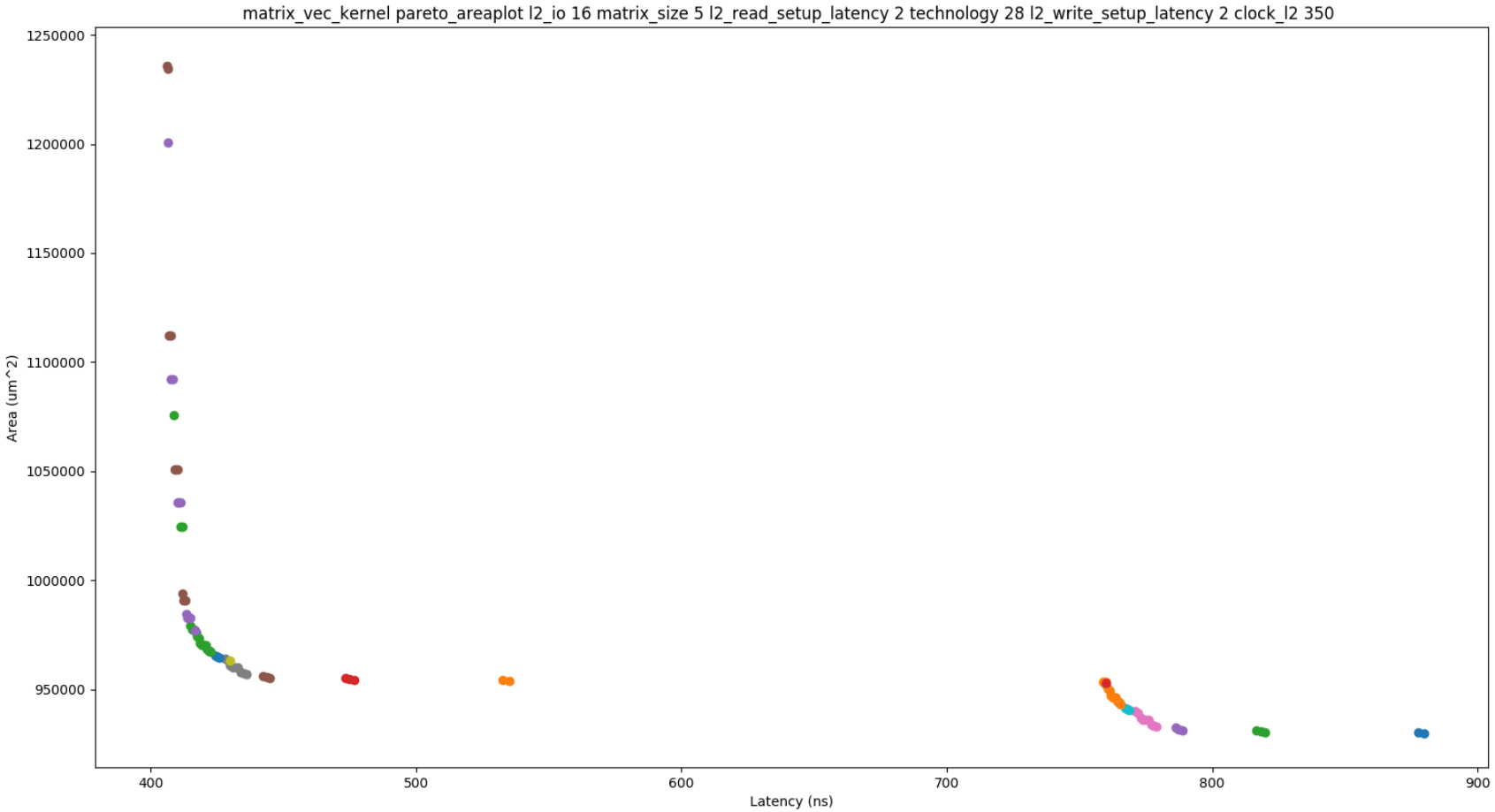
- Application: Matrix Vector Multiplication (5x5)
- L2 Clock Frequency: 350MHz
  - Maximum MRAM Clock (in our specs)
- Custom Processor Frequency: 400MHz..2000MHz step of 200MHz

Experiment: SRAM vs MRAM - Energy



- MRAM architectures consume 20.5% less energy but take 81.5% more time

Experiment: SRAM vs MRAM - Area



- MRAMs architectures consume 2,61% less area then SRAMs

Summary

1. Framework
   - Modified Interval Partitioning Algorithm
   - Design Space Exploration Methodology
   - Highly Parametrizable
2. Architecture
   - Programmable Processing Elements
3. Methodology to compare tradeoff between different technologies
4. Compared MRAM and SRAM tradeoff for a use case application
   - MRAMs architecture take 81.5% as much time, but consume 20.5% less energy and require 2.61% less area

BACKUP

Current and Future Work