Catapult – Lab 2: Task Scheduling/Loop Pipelining

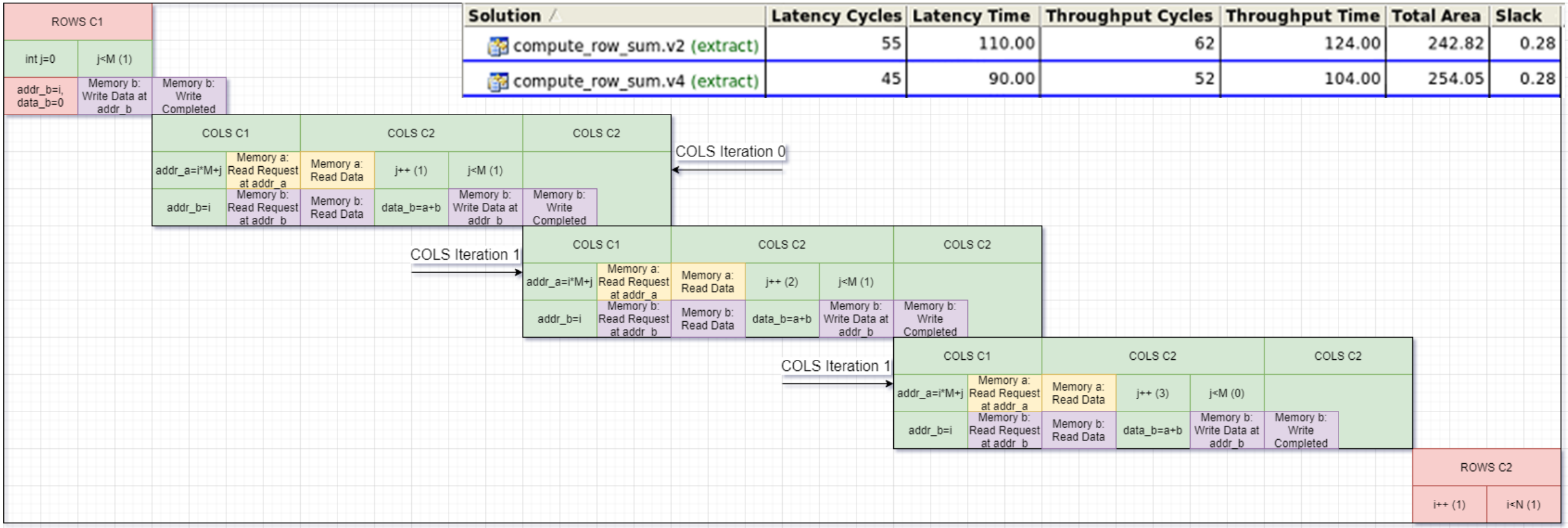
Margomenos Nikos

Initial Design – COLS Pipeline

Single-port memories:
- Only one read or write can occur per cycle
- Each read/write takes 1 cycle

Reading & writing to memory back-to-back requires at least 2 clock cycles:
- Initiation interval cannot be less than 2 (**II=2**)

This scheduling requires:
- 1 cycle to initialize row sums (**5 cycles**)
- 2 cycles to read & write an element (**30 cycles**)
- 1 additional cycle for the last element of each row (**5 cycles**)
- 1 cycle to prepare for the next ROWS iteration (**5 cycles**)



Solution	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Total Area	Slack
compute_row_sum.v2 (extract)	55	110.00	62	124.00	242.82	0.28
compute_row_sum.v4 (extract)	45	90.00	52	104.00	254.05	0.28

Initial Design – Main Pipeline

Having 2 nested loops hinders performance:

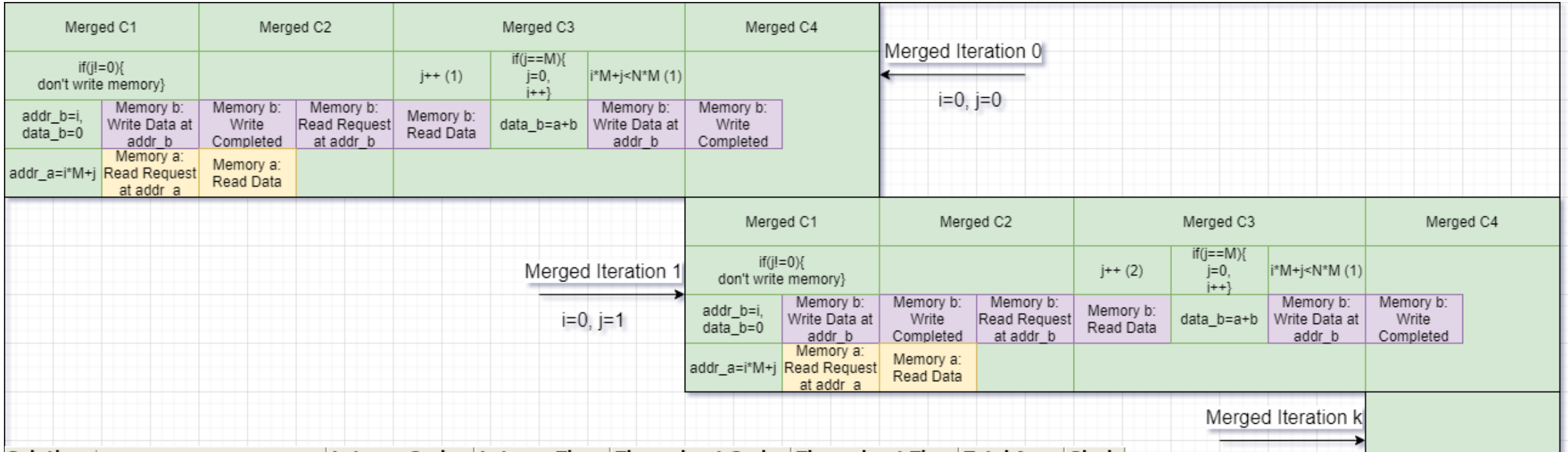
- 1 wasted cycle to prepare for the next ROWS iteration (**5 cycles**)
- 1 wasted cycle for breaking the pipeline at the last element of each row (**5 cycles**)

However, merging these two loops adds another stage in the pipeline:

- Once every 3 iterations, an extra write is required to initialize row sums
- Initiation interval increases to 3 (**II=3**)

This scheduling requires:

- 3 cycles to initialize, read & write an element (**45 cycles**)



Solution	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Total Area	Slack
compute_row_sum.v5 (extract)	55	110.00	62	124.00	242.82	0.28
compute_row_sum.v6 (extract)	**44**	**88.00**	**45**	**90.00**	**376.26**	**0.28**

Improved Design – COLS Pipeline

By observing this design, there is no need to read from the return array before writing back to it. The previous sum can be kept internally in a **local register**.

Reading from memory requires only 1 clock cycle:

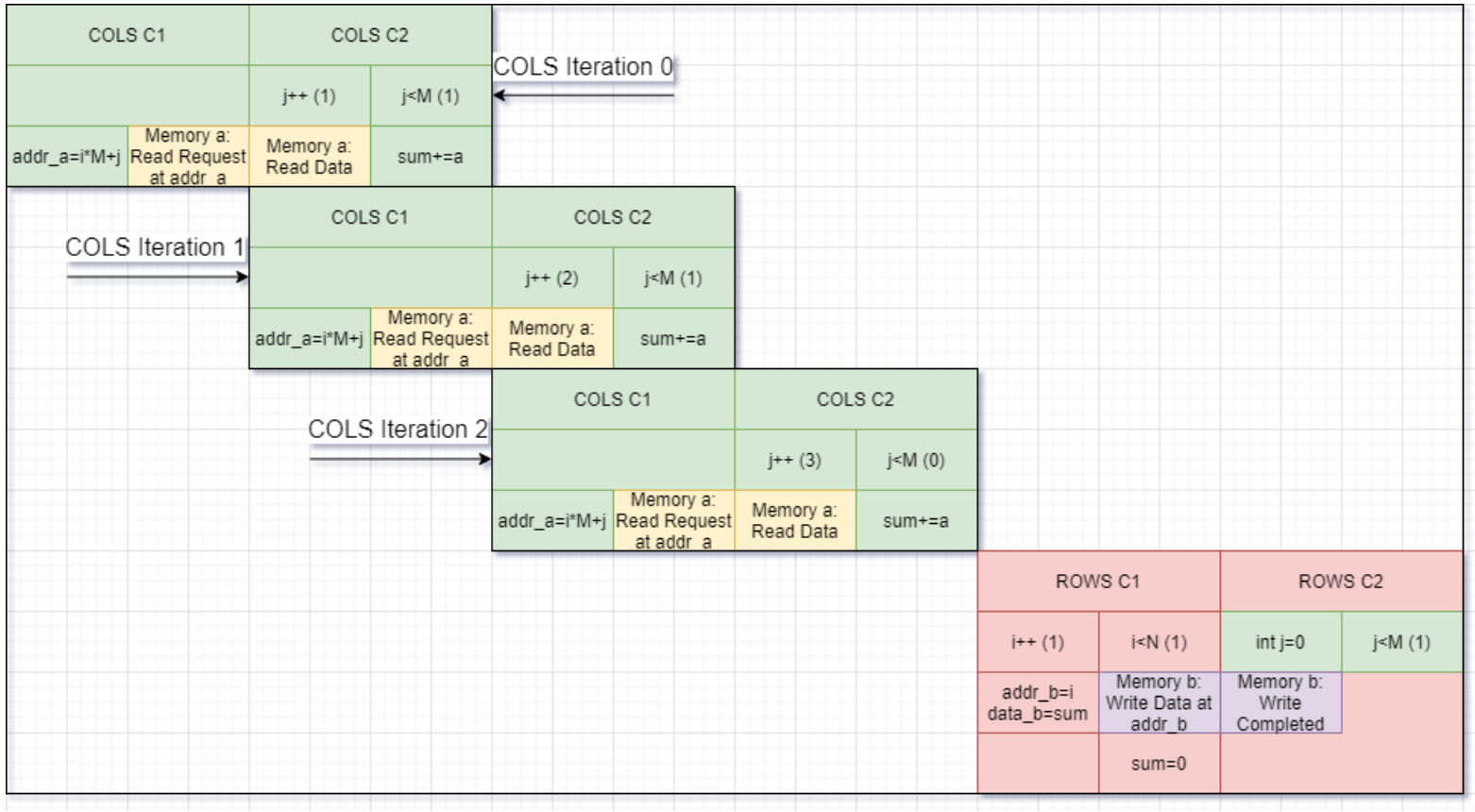
- Initiation interval can be 1 (**II=1**)

This scheduling requires:

- 1 cycle to read an element (**15 cycles**)
- 1 additional cycle for the last element of a row (**5 cycles**)
- 2 cycles to write each output & to reset local sum (**10 cycles**)

```
void CCS_BLOCK(compute_row_sum)
(short a[N][M], short row_sum[N]){
  short sum;
  ROWS:for (int i=0; i<N; i++){
    sum = 0;
    COLS:for (int j=0; j<M; j++){
      sum += a[i][j];
    }
    row_sum[i] = sum;
  }
}
```

Solution	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Total Area	Slack
compute_row_sum.v5 (extract)	55	110.00	62	124.00	242.82	0.28
compute_row_sum.v10 (extract)	38	76.00	42	84.00	334.85	0.29
compute_row_sum.v11 (extract)	**28**	**56.00**	**32**	**64.00**	**349.09**	**0.29**

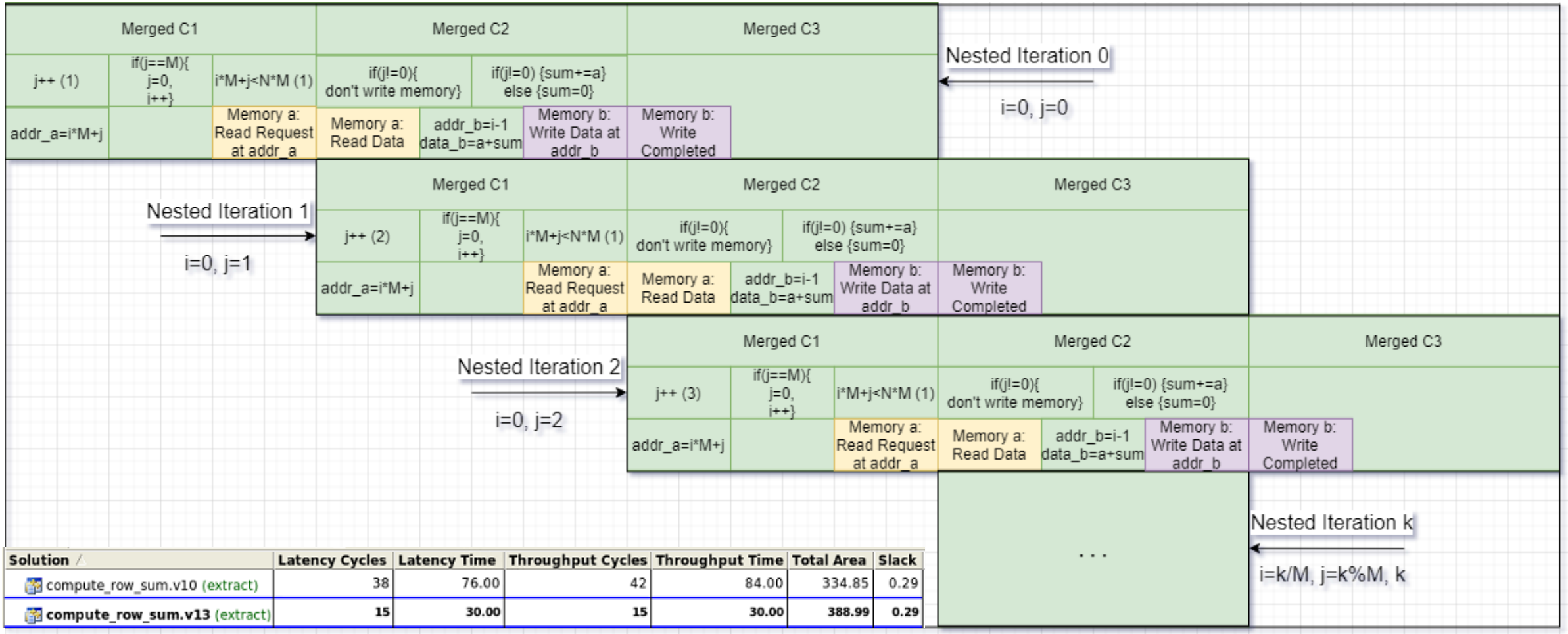


Improved Design – Main Pipeline

Merging the two loops creates the following pipe:
- Read – write
- The write occurs only once every 3 iterations

To pipeline this design, a read can be performed alongside with a write:
- Initiation interval remains 1 (**II=1**)

This scheduling requires:
- 1 cycle for each read & write (**15 cycles**)



Solution	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Total Area	Slack
compute_row_sum.v10 (extract)	38	76.00	42	84.00	334.85	0.29
compute_row_sum.v13 (extract)	**15**	**30.00**	**15**	**30.00**	**388.99**	**0.29**

Improved Design – Main Pipeline